Design and Analysis of Algorithms
Massachusetts Institute of Technology
Profs. Srini Devadas and Ronitt Rubinfeld

October 25, 2012
6.046J/18.410J
Handout 12

Problem Set 4

This problem set is due **at 11:59pm** on **Tuesday, Nov 6, 2012.**

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

Each problem must be turned in separately to stellar.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. A proof (or indication) of the correctness of the algorithm.
3. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions *which are described clearly*. Convoluted and obtuse descriptions will receive low marks.

Note: This is a 1.5-week assignment. There are only two problems, and the grading policy will not follow the previous convention of 3 points per problem.

Exercise 4-1. Do Exercise 34.3-2 in CLRS on page 1077.

Exercise 4-2. Do Exercise 34.5-2 in CLRS on page 1101.

Exercise 4-3. Do Exercise 34.5-3 in CLRS on page 1101.

Exercise 4-4. Do Exercise 34.5-5 in CLRS on page 1101.

Exercise 4-5. Do Exercise 34.5-7 in CLRS on page 1101.

Exercise 4-6. Do Exercise 29.2-3 in CLRS on page 863.

Exercise 4-7. Do Exercise 29.4-3 in CLRS on page 885.

Exercise 4-8. Do Exercise 29.4-4 in CLRS on page 885.

Problem 4-1. Dividing Acorns

Chip and Dale gather a pile of m acorns, whose weights they find to be $\{w_1, ..., w_m\}$. These weights all happen to be positive integers in Chipmunk Standard Units. Acorns are hard to crack, and therefore cannot be split into fractions.

(a) Chip and Dale wish to determine whether it is possible to divide up the acorns evenly by weight. We refer to this as the EVEN-DIVIDE problem.
Show that EVEN-DIVIDE is NP-complete, using reduction from SUBSET-SUM.

(b) They decide to save some acorns as a gift for their friend Clarice. They know that, for Clarice, there is a positive integer value v_i associated with the external appearance of each acorn $i = 1, ..., m$. They wish to determine whether it is possible to package a box of acorns with total value no less than v. The box can hold a weight of at most w.
Show that this problem, which we call VALUE-PACKAGE, is also NP-complete, using reduction from EVEN-DIVIDE.

(c) Recall that the INTEGER LINEAR-PROGRAMMING (ILP) problem determines whether there exists an integer n-dimensional vector x such that $Ax \leq b$, given an integer m-by-n matrix A and an integer m-dimensional vector b.
Show that ILP is NP-complete, using reduction from VALUE-PACKAGE.

Problem 4-2. Problem-Set Grading Time

A TA is distributing submitted problem-set papers among graders. There are m problems, each with 1 unit of workload. There are n graders, each with a different expertise in grading, and it takes t_{ij} hours for grader j to grade one unit of problem i. Moreover, grader j can only take a total workload of up to s_j units, where s_j is an integer. Assume that $\sum_{j=1}^{n} s_j \geq m$, i.e., it is possible for the graders to collectively grade all problems.

(a) Suppose the TA wishes to minimize the amount of time all graders spend in total. Formulate this as a linear program and write it in standard form. (Hint: Let f_{ij} be the fraction of problem i workload assigned to grader j.)

(b) Show that there exists a solution assigning the entire workload of every problem to a single grader each. (Hint: Use matrix properties presented in lecture.)

(c) Knowing that students would like their problem-sets to be graded as soon as possible, the TA instead wishes to minimize the time it takes for all graders to finish grading, assuming that they start at the same time and work in parallel.
Modify your formulation in part (a) to achieve this, and show by example that an integer assignment, like that in part (b), cannot be guaranteed.

(d) Write the dual of your formulation in part (a). What do the dual variables represent?

11/3
11P

P-set 4 #1

Grading policy not 3pts/problem
Is there then what?

Chip + Dale gather m acorns
weights $(w_1, \ldots w_m)$
all positive integers
can't split

a) C+D want to know if possible divide up acorns evenly by weight

Show E-D problem is NP-Complete using reduction from S-S

Well they pretty much told us how.
Follow the cook book w/ the 3 steps

(7)

p1097 in CLRS

Subset Sum

Given finite set S of positive ints

and target $t > 0$

ask if exists subset $S' \subseteq S$ whose elements sum to t

$$t = \sum_{s \in S'} s$$

input integers in binary

Steps

1. prove poly time verifier $V(x, y)$ for problem to show $B \in NP$
2. Reduce known NP-complete problem $A \Rightarrow B$
   a) give poly time alg - if input x, then input $R(x)$
   b) show if $x \in A \rightarrow R(x) \in B$
   c) " " $R(x) \in B \Rightarrow x \in A$

(3) 3. Conclude

~~(come back late)~~

~~Why decide to save~~

1. So we can clearly verify have all the same $O(n)$

2. Now fn that converts to S-S
   is it from / to ?

   A = old problem
   B = new
   x input for A
   R(x) input for B

(4)

So before have target

So do we have 2 groups?

then target = $\frac{\text{Sum}}{2}$

well $\frac{\text{Sum}}{\text{groups}}$

Then want ~~each~~ subsets to add to that

Run SS(target)

Then run SS(target) again repidly till none left

If terminates → impossible

↳ then we return impossible

b) if $x \in A \rightarrow R(x) \in B$

we are using components of running it repeditly

(5)

c) $R(x) \in B, \; x \in A$

; just a component of

We are viewing it as part of it

(; is this too obvious?)

b) save some acorns for Clarice

pos int value v_i associated w/ extra appearance of acorn

Want to see if can package $\geq V$ $< W$

reduction from Even Divide

; not in CLRS

Oh duh - the one we just did

(4)

a) Verifier

Trivial to check

add w + check $\geq$

add $\checkmark$ " $\geq$

b) Reduction from Even-Divide

hmm much harder than before

just doing at most w is tough

Since no exact target

could try diff ~~[illegible]~~ S-S starting at w

and checking each if $\geq v$

but do we have to keep it exponential?

(7)

c) ILP determines if int n-d vector x
s.t. $Ax \leq b$ given mxn matrix X
m matrix b

Show ILP is NP-complete

So show last problem is LP

And thus its LP complete

8

11/4
3PM

(covered w/ Tutor - on sep pg)

I think I can start on the write up

Did a very robot write up on a!

b) Still have not "solved"

~~I can run~~ Tutor; you can run E-D on any # of sets - a polynomial # of them

; So not everyone w/ everyone

; start w/ 2

add 2 each time until $\geq V$ / $\leq$ max w

this just seems a silly way to do this

; Or start w/ all

Then cross stuff out till it $\geq V$ / $\leq w$

but this is G+V

Perhaps should ask in OH

(9)

c) Think I can do this one anyway
let me try

Or not - should really be able to picture
V-P

(10)
1a)

11/5
OH

(going much more in depth than I ever did...)

$2t = sum$

$2t < sum$

$2t > sum$

Why #s don't end up in set
Justify that

Is a way if can just 1 #
One of even divide sets is target

want total sum $\leq 2t$

t in the set w/ the # you added
add to E-D gives you something

Have sol to E-D want to solve S-S
↑ black box

So $2t - sum_{\Sigma w}$ ← target add to sum of weights

(11)

(b) Want to reduce ED- to V-P

ED - in set of weights $\{w_1, \ldots w_n\}$

↓ reduce to

VP in set W max w

Set V min v

Input

Use engine of VP to solve ED

transconfigure input ED to VP

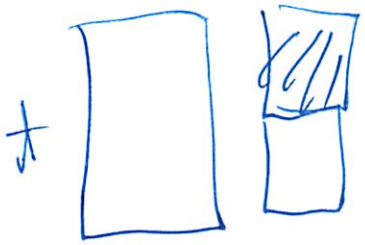
$W_i = V_i$ set $W, V = \frac{\sum w}{2}$

(12)

(a) $2t - s$

Can break original sum into size t, t

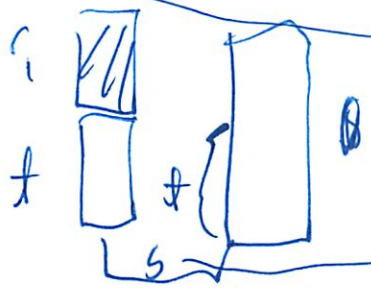
One is original target



= add el

So one is original

but only when $2t > s$



everything is not filled in is s

~~want to subtract~~

Add $s - 2t$ = the shaded area

So division of side $s - t$

$$\frac{s - 2t + s}{2}$$

Sum $-2t$ on each side

(13)

throw them in more # so can only divide in half if the subset-sum holds, sums to t in original

so add $|S-2t|$

proof by picture

Cases $2t \geq S$

$2t < S$

$2t = S$ trivial

(14)

bobby TA
kevin

if no sol → return n

1c) ~~follows trivially~~

Want some set $\geq v$
$\leq w$

A $2m+1 \times m$

$$\begin{pmatrix} W_i & \cdots \\ -V_i & \cdots \\ I_{n\times n} \\ -I_{n\times n} \end{pmatrix}$$

Only want 0 or 1

m (presence or absence
← first entry must exist no more than

$$\leq \begin{matrix} m \\ m \end{matrix} \begin{pmatrix} W \\ -V \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

~~enforce all must be 1~~
~~the solution~~

0 or 1 (what ILP returns)
indicates presence or absence

$I() \leq 1$
$-I() \leq 0$ ← all must be ≥ 0) present or not present (binary)

(15)

7p
11/6

Oh so looks like I did the direction wrong earlier

Reduction

Reduce A to B

We use easyness of B to solve A

$A \propto B$

$A \leq B$

So from

Reduction problem from which undecidable

$A \propto B$

τ from / undecidable

So $ED \propto SS$

~~Reduction~~ Reduction ED to SS

~~Use~~ Use SS to Solve ED

(that is opposite of ~~class~~ TA!)

16

Class also said we have a black box ED

want to solve SS

So SS $\leq$ ED ↑ have black box for

reduce to

"Reduce from SS to ED";

Notes

A $\leq$ B implies have poly for B

Use to solve A

For Proving P/NP

know $\leq$ want to Prove

3SAT $\leq$ Clique

---

then further down

Exact Cover $\leq$ Subset sum

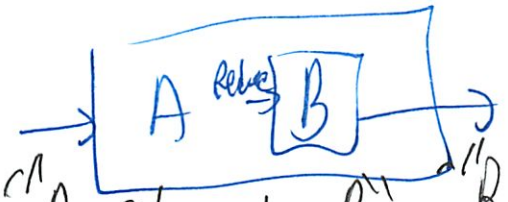
Subset Sum is known

(17)

$A \leq B$

means B is as hard of A

R maps inputs for A into B



"A reduces to B" "B is a reduction from A" correct

If have poly time for B, then have poly time for A

Also $D \leq C$

If D is NP-hard, then C is NP-hard

"Why are we all of suddy going other way?"

Show any NP-complete problem reduces to it

Ted

Please If Solve D by using C poly times

$D \leq C$

which we can't do - unless C is NP-hard

Otherwise can't reduce it

(18) Both must be of exponentially complexity

Now back to problem

$ED \propto SS$

? but this is known NP

$SS \propto ED$

D is "impossible"

If we can solve D w/ something else then that problem requires magic

Is no magic so other problem is impossible

C is like black box

If can solve ~~then we've done magic~~

D by using black box C then we know C is magic - no magic so C is impossible

then C is ~~at least as hard as D~~ NP-hard

From A To B C

19

Ok but operationally

3SAT ≤ Clique

↑ what we show to

try to reduce to Clique

So SS ≤ EP ?

So show build subset sum from even divide

Which is opposite of what I did

Wait where was that shown

Lecture took 3SAT converted to clique

3SAT ≤ Clique

Given input to 3Sat create input to Clique

So here given input to SS create input to build EP

Hmm

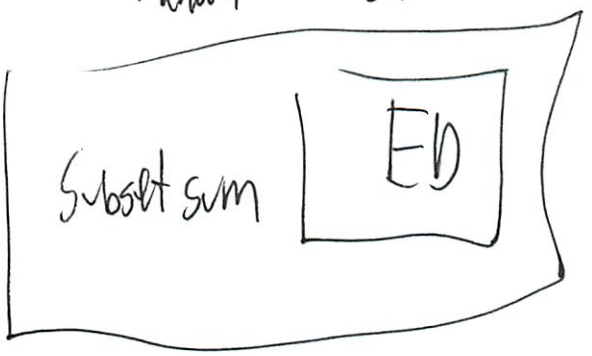
Opposite what I had

(20)

But ~~ED ∝ SS~~ ← want to prove
~~I know NP-hard~~

No want to show ED is NP complete

SS ∝ ED
I know I show



So ED is engine needed to solve subset sum!

(I really need to make a cheatsheet on this!)

Ok consistent w/ prior knowledge

(20)

Now look at their notes

We [redacted] want to know if a subset will add to a certain value — or contains those values by seeing if we can evenly divide

---

(I'm not fully seeing - stop + think)

Throw in more 1's so can only divide in half if sums to it in division

---

Ahh so each box is half
the S is the sum of both halves
We want one of the halves to be 8

(22)

$$\frac{s - 2t + s}{2}$$

$$\frac{2s - 2t}{2} = s - t$$

1. Show $x \in SS$ then $R(x)$ is in ED
2. Show $R(x)$ is in ED then x is in SS

What did we show $\Rightarrow$ #2

---

b) Forgot what I did in OH
Figured it out fast before
Oh solve ED using VP
So $W_i =$

Am I doing this yes/no proof wrong
Yeah its items which are input

(23)

Still somewhat confused on which $X \to R(x)$
$R(x) \to X$ does

Clique $\propto$ IS

Want to show that IS is same as Clique
R reduces IS to Clique ~~if~~ Given input to Clique get input to IS

C → IS

if ∃ k clique → will exist ind set

so reverse C ∈ IS

order printed of notes ∃ k ind set in G → ∃ k clique

That was confusing me!

~~R(x) ∈ B~~

$C \to IS \;\equiv\; \neg R(x) \in B^{IS} \to x \in A^{C}$

$\overline{C \in IS} \;\equiv\; x \in \overline{A^{C}} \to R(x) \in \overline{B^{IS}}$

to must be in order!
(so confused)

So we can use IS to solve Clique

(24)

Still super tired around - but I need to move on

c) Review ans from OH

1. Verify ILP

We can verify the results of the ILP by checking all the constraints. Because the constraints are linear, we can do this in $O(n)$ time.

Opps wrong paper

Michael Plasmeier
6.046 R07

P-Set 4 #1
dyim (tutor HW)

Dividing Acorns

a) Even-Divide from Subset-Sum

Even Divide can be easily built from Subset sum. We are dividing into 2 equal halfs by weight. We can run Subset-Sum once on $\frac{\Sigma \text{weight}}{2}$. This will return either yes or no. If yes, we can evenly divide because by definition the ~~other half will~~ remaining acorns will by definition also be exactly half (proved later).

(2)

More formally we can show Even-Divide is NP-complete by showing it is in NP and that Even-Divide is reducable from Subset-Sum.

1. Even-Divide is in NP because we can build a polynomial verifier that provides a certificate for a correct answer, given a proposed answer.

We verify by adding up the weights of each half independently. We then compare if the $\sum$ of the weights are =. This will take $\Theta(n)$ time.

2. Even-Divide is reducable from Subset Sum.
T_B T_A

$A \propto B$

~~Since $A \rightarrow R(B)$~~

a) if there is an input x for A
then there is an input R(x) for B

(3)

This is even more specific here. Even-divide calls subset-sum.

But we must show that a yes in A will still be a yes in B.

b) Show that if $x \in A \rightarrow R(x) \in B$

This is showing that a yes in A is still a yes in B
no " " " " " no " "

I alluded to this earlier.

If we can run subset-sum $\left(\frac{\Sigma \text{weights}}{2}\right)$ and it returns "yes" we can evenly divide the acorns. This is because subset-sum $\left(\frac{\Sigma w}{2}\right)$ will internally make a subset of acorns that add up to exactly half of the sum of the weights.

The acorns in the other half must also add up to $\frac{\Sigma w}{2}$ too in order for

(4)

the sum of the weights to $= \sum w$

$$\sum w - \frac{\sum w}{2} = \frac{\sum w}{2}$$

$$\frac{\sum w}{2} + \frac{\sum w}{2} = \sum w$$

b) If Subset-Sum$\left(\frac{\sum w}{2}\right)$ returns no, then we do not have an equal half of the acorns. There is no way to have two equal halves.

c) Show that if $R(x) \in B$ then $x \in A$

So show yes $B \Rightarrow$ yes A

no $B \Rightarrow$ no B

If we can do a Even-Divide on a set of acorns we can do a Subset-Sum$\left(\frac{\sum w}{2}\right)$. Even-Divide we have shown will give you two disjoint subsets $=$ to $\frac{\sum w}{2}$. Thus there at least 2 ways for Subset-Sum$\left(\frac{\sum w}{2}\right)$ to return

true as well.

If Even-Divide is impossible, then we do not have any sets that add to $\frac{\Sigma w}{2}$. Thus Subset-Sum will also fail.

Thus Even-Divide is NP-Complete.

(1)

b. Show if $x \in$ ED then $R(x) \in$ VP

So if we have a yes in ED then we will have a yes in VP

Our VP ~~is using~~ will come out the same because $\frac{\sum w_i}{2} \leq X \leq \frac{\sum w_i}{2}$ is $X = \frac{\sum w_i}{2}$

If this holds, then our VP will also hold

If we have a no in ED, then will have a No in VP

If we can't do a ED then we won't be able to do a VP since we won't have a set of items that is exactly $\frac{\sum w}{2}$

(8)

c) Show if $R(x)$ in VP then $x \in ED$

This is because we are using VP to solve ED, as shown on the prior page.

6

b) Show if $R(x) \in VP$, then $x \in ED$

(7)

b) Show ED → VP

if there exists a ED, there will exist a VP

if $x \in ED \rightarrow R(x) \in IS$

If

11/3
11:30P

#2) P-Set Grading Time

m problems
each w/ 1 unit of workload

n graders
takes t_{ij} for grader j to grade problem i
(unit seems weird)
each grader can take total workload up to s_j units – (integer)

$\sum_{j=1}^{n} s_j \geq n$ ie graders can grade all problems
(should that be m?)
(✓) Correction email was sent out

a) Suppose want to minimize amt of time all graders spend in total
Formulate as a LP
Write in Std form

(2)

That should be pretty trivial

f_{ij} is fraction of problem i workload assigned to grader j

Just can be multiple ~~code~~ ways

Perhaps try more than 1 as an exercise

b) Show that there exists a sol assigning workload of every problem to a single grader each.

ie a load-load of graders each doing 1 problem

Use matrix properties presented in lecture

- What property was this?
  - total unimodular
    - but what does that mean?
  - min cost flow matrix
  - or something else

(3)

c) Knowing students want stuff graded fast
instead want to minimize grade fine
assume they all start at same time fine
and work in parallel

But they have nothing else on their schedules?
; but still need a reasonable # of grades

Show by example integer assignment like b)
can't be guaranteed

So b) was about integer
When was this discussed in lecture?
; this was recitation
Ask about

(4)

d) Write the dual of a)

What does the dual represent?

This I need to study / ask about

Don't understand the whole dual thing

11/4
4p

Talked to tutor

Gave me good insights

Now let me try to actually do a)

So we want to minimize total grading time

$$\sum_{m} \sum_{i \in m} t_{ij}$$

what is diff i an m?

i is one unit of a problem m

but all problems are 1 unit?

grrrr

(5)

think of tutor's advice

units of work 5 0000

but since $t_{ij} = \frac{1}{2}$

so total time is $m \cdot t_{ij}$

or $\sum_j i \cdot t_{ij}$

$\sum_j \sum_i i \cdot A_{ij}$

$\sum_{j \in n} \sum_{i \in m} t_{ij}$

so this would be $\underbrace{\left(\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}\right)}_{5} + \underbrace{(0+0+\dots 0)}_{5} + \dots$

s.t. for each $j \in n$ $\sum_{i \in m} t_{ij} \leq s_j$

(6)

So confirm that

each problem 1 unit

so think unit = problem

So t_{ij} for 1 problem i for grader j

Constraints $\sum t_{ij} \leq S_j$

and non neg $m \geq 0$ $n \geq 0$ $t_{ij} \geq 0$

$S_{ij} \geq 0$, integer

What does f_{ij} mean here - fraction of problem i assigned to j

I am using t_{ij}

Are they equiv?

t_{ij} is fixed

f_{ij} is the variable

(7)

↙ variable ↙ given

So $f_{ij} \cdot t_{ij}$ is objective

Otherwise/before we assumed it to be 1

Which is b)

> But not assign all workload of all problems to most efficient grader - like tutor said - that what was confusing me

¿But what do we do about t_{ij}

↳ those are the variables we should be good!

---

b) So integrality means at least 1 hour

¿Can be 0-s but then eliminate them

So I think it's just going this (start w/ ans - so grader skips rest of problem :))

(8)

S_j is in units

Are there no graders w/ 0

No - but enough a non-0 graders

Oh they write $\sum_{j}^{n}$ I wrote $\sum_{j \in n}$

---

c) So now minimize largest;

Remember want them all to work the same amt of time.

So how to write this

minimize total time;

;time per person

not units - but time

---

but wait minimizing total time is same?

no - then give all to 1 person

(same trick/equiv I'm not seeing)

(9)

So want to min avg time per person

would minimize each person's time

So total time not units

divide by people

total time $\frac{\sum_j \sum_i f_{ij} t_{ij}}{n}$

But how is that diff?

It's not

Is it a constraint all spend same time

Or wait min $\sum_i f_{ij} t_{ij}$

Such that $\sum_i f_{ij} t_{ij}$ is = for all j

Could that be it?

(16)

But why not $\sum_i$ that

└ that would be total time

Ask in OH if simpler way to write

But how is

$$\sum_i f_{i0}\, t_{i0} = \sum_i f_{i1}\, t_{i1} = \text{etc in std form}$$

Pay attention to the example $\frac{10}{6}\ \frac{5}{6}\ \frac{5}{6}\ \frac{5}{6}\ \frac{5}{6}$ = 5 units

how did he get it?

Have 6 units of productive capacity / hr

~~5 units~~ 5 units at 6 units / hr

No still missing it!

Ops engineering:

Min time for 5 units:

Can do 6 units / hr

(1)

So $\frac{5}{6} = .83$

2 units $\cdot \frac{5}{6} = \frac{10}{6}$

1 unit $\cdot \frac{5}{6} = \frac{5}{6}$

So minimize $\frac{\sum \text{units}}{\sum \frac{\text{units}}{\text{hr}}}$

but that is units $\cdot \frac{\text{hr}}{\text{units}} = \text{hr}$

Grr - why am I not getting it

So obvious!

So 6 units of labor

$$\frac{1}{\frac{1}{2}+1+1+1+1} = \frac{1}{9/2}$$

No!

$\frac{1}{\text{each}}$ $\frac{1}{2} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} = 6$

(somehow different)

(12)

well addition + division ; commutative

$$\frac{1}{\frac{1}{2}+\frac{1}{1}} \stackrel{?}{=} \frac{1}{\frac{1}{2}+1}$$

no order of ops

$$3 \stackrel{?}{=} \frac{2}{3}$$

Anyway 6 hrs of work

~~With~~ with 5 units

$$\frac{5}{6} = \text{max}$$

So maximize total work per hr of entire system

~~max~~ what is that ?

$$\max \sum\sum \frac{1}{t} \;?$$

that would be $t = .5$

so $\frac{1}{.5} = 2$

$$2+1+1+1+1$$

(13)

then divide over workload
which is fixed
so it does not matter

---

but this could be it - lets try

$$\max \sum\sum f_{ij} \frac{1}{t_{ij}}$$

↑ oh how does frac play into it
leave at for now

$$\max \sum\sum \frac{1}{t_{ij}}$$

is 2+1+1+1+1
but for each problem
↳ but each is 1 hr

I assumed all problems the same

$$\max \sum\sum \frac{f_{ij}}{t_{ij}}$$

so f_{ij} is 10 for 1
5 for rest

(14)

So that is — lets try

$$\frac{f_1}{.5} + \frac{f_2}{1} + \frac{f_3}{1} + \frac{f_4}{1} + \frac{f_5}{1}$$

This might be legal

and minimize?

also $\sum f_{ij} = m$

So I have 5 units

ah this $f = \frac{10}{6}, \frac{5}{6}$

$$\frac{\frac{10}{6}}{.5} + \frac{\frac{5}{6}}{1} + \cdots$$

3.33 ? no double incentive

+ .833

f_{ij} is not that

Or is it? That is workload for each

Assuming all problems the same

(5) Grrr - I feel like I can do this - but I'm just not smart enough to see!

$$\frac{\frac{10}{6}}{2} = \frac{\frac{5}{6}}{1}$$

ah that's it

And I had this in denom - where it would be ~~#~~ 2

So minimize $\sum_j^m \sum_i^n \frac{f_{ij}}{t_{ij}}$

So here $\frac{\hat{i}}{2} + \frac{\hat{i}}{1} + \frac{\hat{i}}{1} + \frac{\hat{i}}{1} + \frac{2}{1}$

and $\sum \hat{i} = 5$

So our dist of $\hat{i}$ is $\frac{10}{6}, \frac{5}{6}, \frac{5}{6}, \frac{5}{6}, \frac{5}{6}$

Sum of $\hat{i}$ is $\frac{30}{6} = 5$ ✓

if $\frac{?}{f} \cdot 5$ then

then each $\hat{i}$ is 1

(16)

So I think I figured it out!
Lots of false starts
But did it

But how to explain
; or delete initial
that was not wrong
but should be equiv
Confuse time w/ units

Or dont $\sum_j$
but all the same → so same
But that is not what min?
Or is it

There $\sum = \frac{25}{6}$

Try rearranging $\frac{\frac{15}{6}}{2} + \frac{\frac{0}{6}}{1} + \frac{\frac{5}{6}}{1} + \frac{\frac{5}{6}}{1} + \frac{\frac{5}{6}}{1}$

(7)

That is $\frac{5}{12}$ ~~better~~ worse than $\frac{5}{6}$ better

or not $\frac{5}{6}$ ~~worse~~ better

$= 3\frac{3}{4}$ ~~no better~~

Which is worse!

So don't $\sum$

that is not in the answer!

So minimize $\sum_{i}^{n} \frac{f_{ij}}{t_{ij}}$ for $\forall j$

but that is what I had on $f_{ij}\, t_{ij}$

Did I get original wrong?

Where wanted to min total time

So if $f_{ij} = 1$ then $1 \cdot \underbrace{\frac{1}{2}}_{A \text{ is just } \frac{1}{2}}$

So we don't want $\frac{f_{ij}}{t_{ij}}$ in ours

since it is $\frac{1}{2}$ so still x

(18)

So I am back at the same ans

$$\min \sum_{i}^{m} t_{ij} f_{ij}$$

Where all $j =$

; But there must be some way to write this as one constraint!

Unless it is $t_{i0} f_{i0} = f_{i1} t_{i1} = \ldots$

trying to max throughput in problems/hr
but only have our limited problems

So our throughput is 2 problems/hr
1 problem/hr

We max throughput by using all

So $\max \frac{1}{t_{ij}} = \frac{1}{1/2} = 2$

$$\max \sum\sum \frac{f_{ij}}{t_{ij}}$$

(19)

This would be

$2f_0 + 1f_1 + 1f_2 + 1f_3 + 1f_4$

then $\frac{10}{6}$ $\frac{5}{6}$

$\frac{20}{6} + \frac{5}{6} +$

could max by putting all on left

or if f reverse

$2 \cdot \frac{6}{10} + 1 \cdot \frac{6}{5}$

but still max this?

could we max this more

$\frac{12}{10} + \frac{6}{5} = \frac{12}{5}$

if moved $\frac{6}{5}$ to first

$2 \cdot \frac{9}{5} + 0 \cdot \frac{6}{5} = \frac{18}{5}$

grr

ok give up - for 8th

very frustrated!

(20)

What did he do?
all units were same

minimized each
said had to be =

I mean what he did was minimized each
and said they were =
Any other way to reformulate!?

—

Max total throughput in $\frac{\text{problems}}{\text{hr}}$
But how to write?

$t_{ij} = \frac{1}{2}$ hrs / problem

$\frac{1}{t_{ij}} = 2$ problems / hr

Max that?

$\max \sum \frac{1}{t_{ij}}$

but how are fraction play in $\frac{\frac{10}{6}}{\frac{1}{2}} = \frac{10}{6} \cdot 2$

(21)

Or max $\frac{1}{f_{ij} t_{ij}}$ is $\frac{1}{\frac{6}{10} \cdot \frac{1}{2}} = \frac{1}{\frac{6}{20}} = \frac{1}{\frac{3}{10}} = \frac{10}{3}$

$\frac{1}{\frac{6}{5} \cdot 1} = \frac{5}{6}$

Want these to = the same

Why am I not seeing it!

Somehow $\frac{10}{6} : 2 = \frac{5}{6} : 1$

$\frac{10}{6} \cdot \frac{1}{2} = \frac{10}{12} = \frac{5}{6}$

max $t_{ij} f_{ij}$

$\frac{1}{2} \cdot \frac{10}{6} + 1 \cdot \frac{5}{6}$

no min $t_{ij} f_{ij}$ for each, and they are =!

What is total system throughput? $\frac{\text{problems}}{\text{hr}}$

$\frac{6 \text{ problems}}{5 \text{ hrs}}$ here

(27)

So $\sum \frac{1}{t_{ij}}$

is $\frac{1}{1/2} = 2 + 1 + 1 + 1 + 1 + 1$

~~$\frac{1}{t_{ij}}$~~ Or is it just that

is but still must assign?

$$\frac{\frac{1}{t_{ij}}}{f_{ij}} \quad \frac{\frac{1}{1/2}}{\frac{10}{6}} = \frac{2}{10/6} = \frac{2 \cdot 6}{10} = \frac{12}{10} = \frac{6}{5}$$

$$\frac{\frac{1}{1}}{\frac{5}{6}} = \frac{6}{5}$$

hmm that might be it

We want to max $\sum\sum \frac{\frac{1}{t_{ij}}}{f_{ij}}$

$$\frac{\frac{1}{1/2}}{\frac{10}{6}} + \frac{\frac{1}{1}}{\frac{5}{6}} = \frac{16}{6}$$

So if we reassign all to first

$$\frac{\frac{1}{1/2}}{5/2} + 0 = 2 \cdot \frac{2}{5} = \frac{4}{5} \text{ < worse}$$

(c3)

$$\frac{1}{\frac{1}{1/2}} + \frac{1}{\frac{1}{5/2}} = \frac{2}{5} \text{ more}$$

hmm - can this be it

how can we add — by 0?

$$\frac{1}{t_{ij}} \cdot \frac{1}{f_{ij}}$$

$$\frac{1}{1/2} \cdot \frac{6}{10}$$

$$2 \cdot \frac{6}{10} \text{ save}$$

but still divide by 0

$$\max \sum\sum \frac{1}{f_{ij} t_{ij}}$$

Wish I could just run...

$$\frac{1}{\frac{10}{6} \cdot \frac{1}{2}} + \frac{1}{\frac{5}{6} \cdot 1} +$$

$$\frac{12}{10} + \frac{6}{5} + \frac{6}{5} + \frac{6}{5} + \frac{6}{5}$$

No other way to maximize

Since this is throughput!

(4)

throughput = $\frac{1}{\text{speed}}$??? true

I didn't know where to put fraction
but as before fraction is related

So max. throughput

$$= \max \sum_j \sum_i \frac{1}{f_{ij} t_{ij}}$$

Then anything we have can't be improved

$$\frac{1}{\frac{1}{2} \cdot \frac{10}{6}} + \frac{1}{1 \cdot \frac{5}{6}} =$$

$$\frac{12}{10} + \frac{6}{5} = \frac{24}{10} = \frac{12}{5} \text{ a}$$

$$\frac{1}{\frac{1}{2} \cdot \frac{15}{6}} + 0 = \frac{12}{15} \text{ less}$$

$$0 + \frac{1}{1 \cdot \frac{15}{6}} = \frac{6}{15} \text{ less}$$

I like that

Could have been more strategic even

Rewrite (Should really have seen earlier - why don't act jobs)

(5)

Now try w/ 2 people 2 problems
$\frac{1}{2}$ 1

through put = 3

$$\max \frac{1}{f_{i1}\,\frac{1}{2}} + \frac{1}{f_{i2}\,1}$$

$$f_{i1} + f_{i2} = 2$$

so $\frac{2}{3}$ $\frac{1}{3}$

$$\frac{1}{\frac{2}{3}\frac{1}{2}} + \frac{1}{\frac{1}{3}1}$$

$$\frac{1}{3} + \frac{1}{3}$$

total finish in 1 hr

no can $\frac{2}{3}$ an hr

each 1 does $\frac{2}{3}$ of each grading
in $\frac{1}{3}$ hr

2 does $\frac{1}{3}$ in $\frac{1}{3}$ hr

2 problems $\frac{2}{3}$ finish, $\frac{4}{3}$ total

(26)

d) dual

Should just be mechanical process

So we had

$$\min \sum\sum f t \quad \text{actually max} \; -\sum\sum ft$$

$$\sum^{\leq} f t \leq s$$

$$\left.\begin{array}{l} t_{ij} \geq 0 \\ f_{ij} \geq 0 \\ s_j \geq 0 \end{array}\right\} \text{leave alone}$$

$$\sum\sum f_{ij} = m$$

Ah but dual more complex

t is like a_{ij}

~~x_j is t~~

f_{ij} is like x_j

(27)

i but not right dimension i

$$\min - \sum\sum \; s_j \; y_{ij}$$

transposed — other variable

$$\sum s_j \; y_{ij} \geq$$

i what abt the other equality one i

i stays

$$\sum\sum y = m$$

I'm not putting much thought into this
esp what it means in real life

But must say what y_{ij} is

? i should just be 1 D

Or original is off i

(28)

it was like f_{ij} = the fraction i spends on j

I don't deal well enough to know
- write later in OH

(2A) (2B) (29)

GH
11/5

M is a decision variable

$$\min\{f_{ij}|M\} \sum_{i,j} f_{ij}\, t_{ij} \leq M$$

$$X = \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ M \end{bmatrix}$$

2(c) (kinda missed)

Minimize M

not enough to have constraints

gives upper bound on same #

$M \geq 0$

the smallest # that is larger than ___

---

b) Something about matrices

- I didn't use
- Show totally unimodular
- det 1, 0, -1
- any sq submatrix

(33) (29) 30

OH
4/5

2a) Say each $\sum_{j}^{n} f_j = 1$ for $\forall i \in m$

↑ stronger

effects has dual formulation

2b) if its dimension better

f_{ij} does not appear in dual

↳ since like X

$X = f_{ij}$

A

~~$\in m \times n$ vector~~

$\in$ ″ ″

Same # of cols

$n \times n$

$$X = \begin{pmatrix} f_{11} \\ f_{12} \\ f_{1m} \\ f_{2m1} \\ f_{i} \\ f_{nm} \end{pmatrix}$$

(31)

$\in \mathbb{R}^{mn \times 1}$ (rows, col) $\leftarrow x = f_{ij}$

A ~~has 1 col~~

$i \times nm$

depends on # of constraints

the s_j constraint

f_j constraints (twice std form) } $\mathbb{R}^{3 \times j}$ constrs

$3n$

Cost fn and some constraints
have same coeffs
correspond to bs
if change s by little - how effect cost fn

---

r_s; more capacity to grade
or more work on fast grader $\leftarrow x_s$
overall time $\downarrow$ $\leftarrow$ in terms of cos

Whatever row + col we choose will satisfy this

Tad all problems need to be graded

How to interpret:

How to do w/ dual variables

Have more - since extra constraints

Look at obj fn

See if can make sense of it

Coeff that multiplies each variable

If b by 1 unit

How does that change optimal

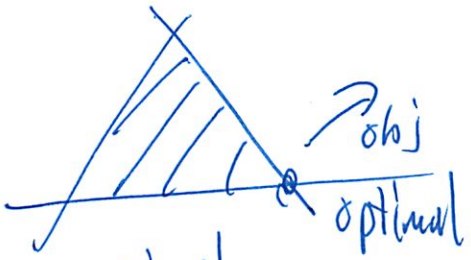
meaning of dual → Shows sensitivity to change in b

In general

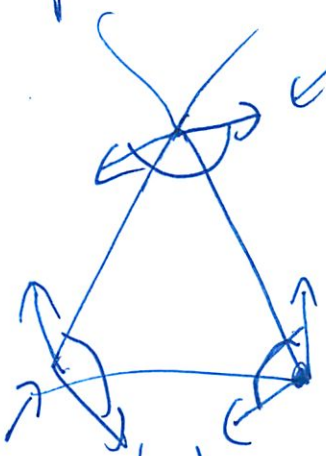
(BL) (33)

dual ll looks for lin combo of constraints
↑ cost can explain

recad res. notes



primal



not lin combo of cost - so not optimal

constraints form linear combos
opposite dir of cost

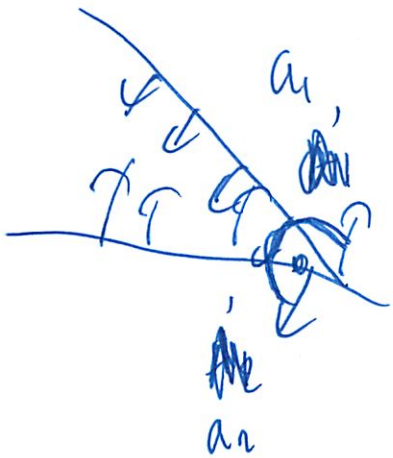
can't form lin constraints

dual

dual are weights in linear combo

a_{ij} of Matrix A

a_{ij} = each row is one constraint



lin combo

those 2 are active
uniquely show those pts

(32) (34)

$a_i' = col$

~~is a lin combo $a_1' + a_2$~~

$V = x_1 a_1' + x_2 a_2'$

↑ v is linear combo it can be expressed as

Want — of cost fn to be a linear combo

neg of it is in lin combo region

Only occurs in 1 pt + rec notes

(don't have to prove - in book)

35

Other interpretation → what is in part

Sensitive analysis

if alter b - how much does that effect optimal

(26)

11/6

Now what to revise for A

$$x = \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{1m} \\ f_{21} \\ f_{22} \\ \vdots \\ f_{nm} \end{bmatrix}$$

M is the decision variable

$$\min(f_{ij}, M) \quad \sum_{i,j}^{M} f_{ij}\, t_{ij} \leq M$$

$$x \begin{bmatrix} i \\ M \end{bmatrix}$$

(37)

; What was that
  └ some thing I read off the board

So what does that mean
M not in original problem

m is totall # of problems

So is true $\sum_{i \in Mm} f_{ij} \leq s_j$
↑ no t_{ij}

f is our variable x
t is our constants A

It was say $\sum_j f_j = 1 \quad \forall i \in M$

not

$\sum \sum f_{ij} = m$

(38)

Oh this is clever now!

Now change other pg

b) ; Can I keep

Or totally unmodular

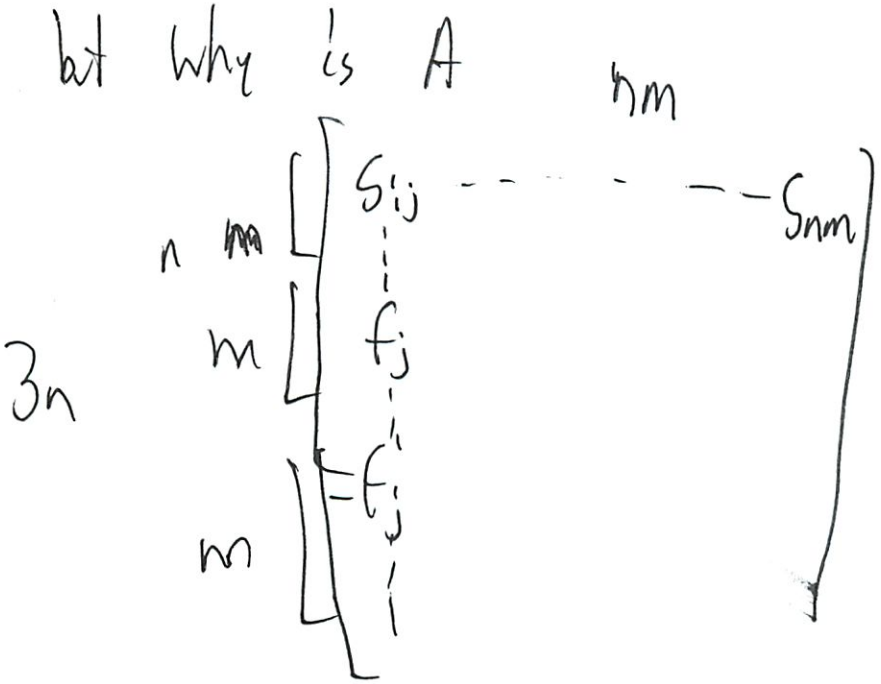
do later

d) Finding the Dual

(write)

└ pretty easy now

but why is A



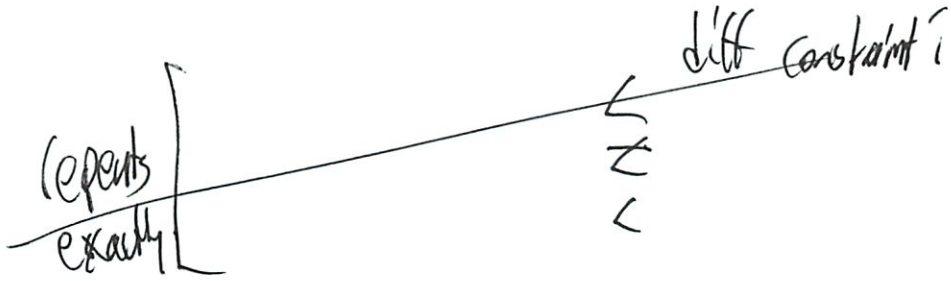
39

Well since

$$\sum_{i=1}^{m} f_{ij} \leq s_j \quad \forall j \in n \qquad \text{all } \Sigma \text{ problems for each grader}$$

$$\sum_{j}^{n} f_{ij} \leq 1 \quad \forall i \in m \qquad \Sigma \text{ work for each problem}$$

$$\sum_{j}^{n} f_{ij} \leq 1 \quad \forall i \in m$$



each row a diff j → i ↓ problems that grader grades $\leq s_j$

Only active on that js line

So if b)

$$\begin{matrix} & \\ 1 \\ 2 \\ 3 \end{matrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ but really f_{ij}

(40)

$A =$ f_{ij}

graders i \ problems j	$j=1$	$j=2$	$j=3$
$i=1$	100	000	000
$i=2$	000	010	000
$i=3$	000	000	001

each does 1 unit of their own

"then again for $f_{ai}j \leq 1$ for problems

is what I have ch

~~think~~ fixed it

think I got it now

b) Unimodular

need to research that!

not in CLRS

Sq int matrix w/ det ± 1

invertible over integers

(11)

- Identity
- inverse of unimodular
- product of two unimodular

So I grader doing 1 each will be
1s and 0s

but not all of these is it

need some property of invertable matrices
didn't ask in OH
likly not going to stumble accross

Also does 0 count?
it seems for totally unimodular)

~~X~~ important since can find integer sol
~~X~~ each col contains at most 2 non 0 coeffs
~~X~~ row of A can be partitioned into 2 sets M_1 M_2
$\sum M_1 = \sum M_2$

42

So just write this up

A bit unsure → but wing it

(3)

That means we have enough non-0 fine graders for each of our m problems.

---

c) Minimize grading time

Now our objective has changed. We want to minimize grading time, to finish.

In order to do that we want everyone to grade for the exact same time.

~~So we want to minimize $\sum_i^n f_{ij} t_{ij}$ for any and all j~~

~~Since $\forall j \sum_i^n f_{ij} t_{ij}$ are =~~

Since

This is the same as minimizing $\sum_i^n \frac{f_{ij}}{t_{ij}}$

which will be = for $\forall j$

(4)

This might no longer be integer because people may now work on a fraction of a problem

Say we have 5 problems and 5 people.

person 1 takes $\frac{1}{2}$ time/unit for all problems

" 2-5 1 " " $1\frac{1}{4}$

The least total time is to give all problems to person 1

$$\frac{5}{2} + 0 + 0 + 0 + 0 = \frac{5}{2}$$

The least time giving each one problem is 5

$$1 + 1 + 1 + 1 + 1 = 5 \text{ total}$$

1 in parallel

But optimal is for them all to work the same amount

~~$\frac{10}{6} + \frac{5}{6} + \frac{5}{6} + \frac{5}{6}$~~

(5)

We can do 6 units of work in 1 hour

$$\frac{1}{\frac{1}{2}+1+1+1+1} = \frac{1}{\frac{5}{2}}$$

(5)

This means that the the number of units each gets must be evenly distributed

In our example with have 5 people that can do 6 units per hour, So the efficent amount of units for each is

$$\frac{10}{6}, \frac{5}{6}, \frac{5}{6}, \frac{5}{6}, \frac{5}{6}$$

Remember all problems are the same

Then total time is

$$\frac{\frac{10}{6}}{2} + \frac{\frac{5}{6}}{1} + \frac{\frac{5}{6}}{1} + \frac{\frac{5}{6}}{1} + \frac{\frac{5}{6}}{1} = \frac{25}{6} \approx 4.16$$

But each works for $\frac{5}{6}$ hrs making time we can return exams in $\Rightarrow \frac{5}{6}$ hours

(2)

We must also change the equality constraint

$\sum_{j}^{n} \sum_{i}^{m} f_{ij} \leq m$ and $\sum_{j}^{n} \sum_{i}^{m} f_{ij} \geq m$

Now we want all $\leq$ so

$\sum_{j}^{n} \sum_{i}^{m} f_{ij} \leq -m$

(5)

d) Dual

The dual can be found

$$\max c^T x \quad \text{s.t. } Ax \le b,\ x \ge 0 \quad \rightarrow \quad \min b^T y \quad \text{s.t. } A^T y \ge c,\ y \ge 0$$

So

$$\min \sum_j^n \sum_i^m s_j y_{ij}$$

s_j: previous constraint; y_{ij}: other variable

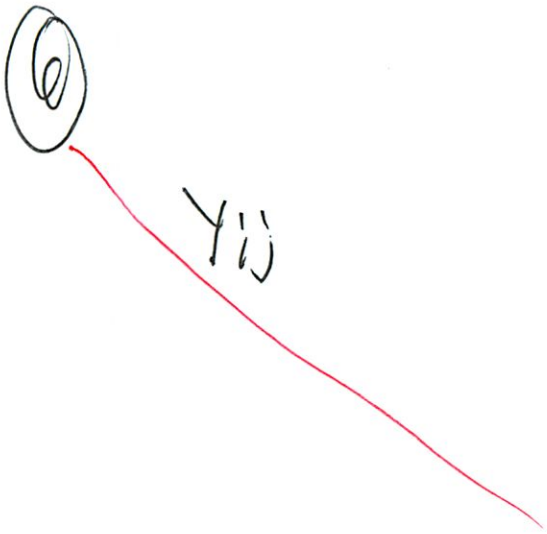
(not just s_j, add dual variable how much change the 1 the dual would reflect that — Can define Y is set of dual variables also what minimizing over)

$$\text{s.t.} \sum_i^m \text{[illegible]} \; y_{ij} \ge \text{[illegible]}$$

transposed; previous variable

$$\sum_j^n \sum_i^m y_{ij} = m \quad \} \text{ same}$$

save $\begin{bmatrix} m \neq \{\} & t_{ij} \ge 0 & s_j \ge 0 \ \} \text{ integer} \\ n \neq \{\} & f_{ij} \ge 0 & y_{ij} \ge 0 \end{bmatrix}$


Yij

Michael Plasmeier
6.046 R07

P-Set 4 #1
dyim (HKN TA)
Bobby
Kevin
OH TAs

Dividing Acorns

a) ~~Even Divide~~ Subset Sum & Even Divide

We want to show E-D is NP complete with a reduction from Subset Sum. We do this by showing we can reduce input for SS to ED.

1. First we must show that ED is $\in$ NP by showing if we have an answer to ED we can verify it. We can Σw_i's in each half to ensure they are equal.

(2)

2. Now we must show that if there is an input for SS we can reduce it to an input for ED.

First if $2t = sum$.

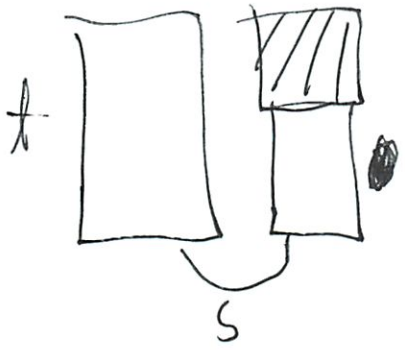
t is the target we want to hit with SS. ED will return true if there exists a sum that is exactly twice t. This is because $\frac{sum}{2} = t$. ED will return true if we have two halves that each meet target t. It only needs to be run once one one half - since if it returns true, then there will automatically be another half of exactly the same size.

Essentially our goal is to throw in more #s so we can only divide in half, if subset sum holds (ie sums to t)

(3) So we will add a certain amount more accross to get up to $2t$, so one half of originals will be t

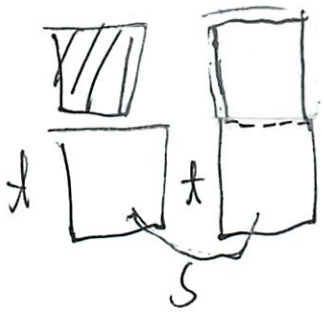
If $2t > s$

Then we shall add elements up to t



We should add $2t - s$ (the shaded region)

$2t \leq s$



We should add $s - 2t$ (the shaded region)

So division of $s - t$ is

$$\frac{(s-2t)+s}{2} = s - t$$

(4)

Basically
Both we would add $|s-2t|$ on each side

Then we can run ED on each and see if it will return true if there was up to t in the original set.

~~we just showed that R(x) in~~

1. Show $x \in SS$ then $R(x)$ in ED

   if yes in SS then we will get yes in ED. This is pretty trivial that we can run $\frac{\Sigma w}{2}$ on SS we can get ED. I commented before on this

   If no in SS then ED will also be no since we would not have 2 even halves

2. Show $R(x)$ in ED then $x \in SS$

   ~~This I showed on the last page~~

(5)

If yes in ED, then we will get a SS $(\frac{2t \text{ and } 2}{2})$
we showed this via proof by picture

If no in ED, then we will get a no in SS
because we were not able to hit the
target with values from our original sum

Thus ED is NP complete.

#6

b) ED $\propto$ VP

1. Show VP verifier

We can $\sum w_i$ and ensure its $\leq W$

We can $\sum v_i$ " " " $\geq V$

We can do this in $O(n)$ time

2. Show we can reduce an input for ED to VP, then use the engine of VP to solve ED.

a. So we can set $w_i = v_i$ and set our goal W, V to $\frac{\sum w}{2}$

This way we get a set that has weight $\leq \frac{\sum v}{2}$ and value $\geq \frac{\sum w}{2}$ and since both are the same we get a set that is exactly $\frac{\sum w}{2}$!

(7)

b.) Show if $x \in ED$, then $R(x) \in VP$

Any input we can put into ED will also be an input to VP since our VP is essentially the same as ED

Since $w_i = v_i$ then $\frac{\sum w}{2} \leq x \leq \frac{\sum w}{2}$

means $x = \frac{\sum w}{2}$ and VP will return only if it can make a set of $\frac{\sum w}{2}$

c) Show if $R(x) \notin VP$, then $x \notin ED$

Our inputs to VP will be the same as ED because we are essentially having VP do even divide, because VP will selects a set ~~with half~~ of the sum of exactly half of the elements

(8)

c) VP ∝ ILP

1. Verify ILP is in NP

We can verify the results of ILP by checking all of the constraints. Because the constraints are linear, we can do this in O(n) time.

2. Reduce VP to ILP so we can use ILP as an engine to solve VP.

Essentially we need to find a way to write VP as an ILP.

(9)

$$A \quad x \quad \leq \quad b$$

$$\begin{matrix} 2m+1 \\ \\ \\ \\ \\ \end{matrix} \begin{pmatrix} W_i \text{ ---} \\ V_i \text{ ---} \\ I_{m \times m} \\ -I_{m \times m} \end{pmatrix} \begin{pmatrix} \\ \\ \\ \\ \\ \end{pmatrix} \begin{matrix} \\ \\ m \\ m \\ \\ \end{matrix} \begin{pmatrix} W \\ -V \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

"will be 0 or 1 "bit" to indicate presence or absence in a set

The $Ix \leq 1$ enforces the 0 or 1 constraint
$-Ix \leq 0$ by only allowing them to be 0 or 1

The first two lines "do the actual work"

(10)

a) Show if $x \in$ VP then $R(x) \in$ ILP

So our VP is inside of ILP with the proper binary constraints. So the ILP should produce the proper answer.

b) Show if $R(x) \in$ ILP then $x \in$ VP

Our ILP is just another implementation of VP. The constraints of ILP enforce VP.

Michael Plasmeier
6.046 R07

P-Set 4 #2
drim(Hkw Ktas)
TAs

P-Set Grading Time

a) LP to minimize total grading time

So we want to minimize $\sum_{j \in n} \sum_{i \in m} t_{ij} t_{ij}$

Subject to. $\forall j \in n \quad \sum_{i \in m} t_{ij} \leq s_j$

$m \neq \{\}$ $\quad t_{ij} \geq 0 \quad s_j \geq 0$, integer

$n \neq \{\}$ $\quad t_{ij} \geq 0$

~~$\sum_j \sum_i t_{ij} = m$~~ $\sum_j t_{ij} = 1$ for $\forall i \in m$

This is standard form, except that its min, not max.
We can change that w/ a neg sign for the objective

maximize $-\left(\sum_j^n \sum_i^m t_{ij} t_{ij}\right)$

0

We must also change the equality constraint

$$\sum_j^n f_{ij} = 1 \quad \forall i \in m$$

to ↓

$$\sum_j^n f_{ij} \leq 1 \text{ and } \sum f_{ij} \geq 1 \quad \forall i \in m$$

& change to

$$\sum f_{ij} \leq -1 \quad \forall i \in m$$

So

$$\left.\begin{aligned} \sum_j^n f_j &\leq 1 \\ \sum_j^n f_j &\leq -1 \end{aligned}\right\} \forall i \in m$$

(3)

b) Assign every problem to a single grader each.

Since s_j must be an integer we know that it must be 0, 1, or more.

We can quickly set aside graders w/ $s_j =$ 0, since those graders aren't really graders since they have no time to grade any problems! (more late)

That means graders have ~~at least 1 hr~~ time for at least 1 unit of work. This is perfect - since we were asked to assign the work load of each problem (1 unit) to a single grader each (who have at least 1 unit of time to grade 6.046.)

We know that there are enough non-0 time graders since we know $\sum_{j=1}^{n} s_j \geq m$

(4)

That means we have enough non 0 fine graders for each of our M problems,

---

We can also use the totally unimodular property from class. This states that we will have a ~~then~~ integer solution if

- each entry of A is $\{+1, -1, 0\}$
  - which is true since we are assigning whole problems or not (1 or 0)
- each col contains at most two non 0 solutions
  - each column of A is the ~~number of~~ fraction of work for each grader for a specific problem. Only 1 of these will be non 0 since 1 grader per problem
- rows of A can be partitioned into two sets so $\sum$ rows of $M_1 = \sum$ row of M_2

(5)

Each row is one grader's work over a series of problems.

We only need 1 partition, not all partitions. Since each row will only be 1 we can't find a row that adds up to 2 halves - if we have an even # of graders

(6)

~~That means we have enough non-0 time graders for each of our m problems.~~

---

c) Minimize grading time

Now our objective is to minimize the ~~total~~ time to finish.

In order to do that we want to maximize the total throughput of the system. We want everyone working hard.
We also want everyone working the entire time, so that throughput is maximized.

So maximize $\sum_{j}^{n} \sum_{i}^{m} \frac{1}{f_{ij}\, t_{ij}}$

with the same constraints

(7)

However our solution may no longer be linear,

Proof by example, with 2 people + 2 problems

$f_{11} = 1/2 \quad f_{12} = 1$

$f_{21} = 1/2 \quad f_{22} = 1$

This means we have a ~~capacity~~ throughput of 3 problems per hour (since all problems are the same)

When we run our LP and get $f_{11} = \frac{2}{3}$

$f_{12} = \frac{1}{3}$

Then we see we finish in $\frac{2}{3}$ an hour, with $\frac{4}{3}$ total hrs of work.

However the proof is ~~now~~ it is no longer integer,

In b) we would have assigned prob 1 to person 1
prob 2 " " 2
and finished in 1 hour
with 1.5 hrs total work

In a) we would have assigned both to person one
finishing in 1 hr, with
1 hr of total work

(8)

d) Dual

The dual can be found with

$$\begin{aligned} &\max\ c^T x \\ &\text{s.t. } Ax \le b \\ &\quad x \ge 0 \end{aligned} \quad \rightarrow \quad \begin{aligned} &\min\ b^T y \\ &\text{s.t. } A^T y \ge c \\ &\quad y \ge 0 \end{aligned}$$

So in the primal

x is f_{ij} =

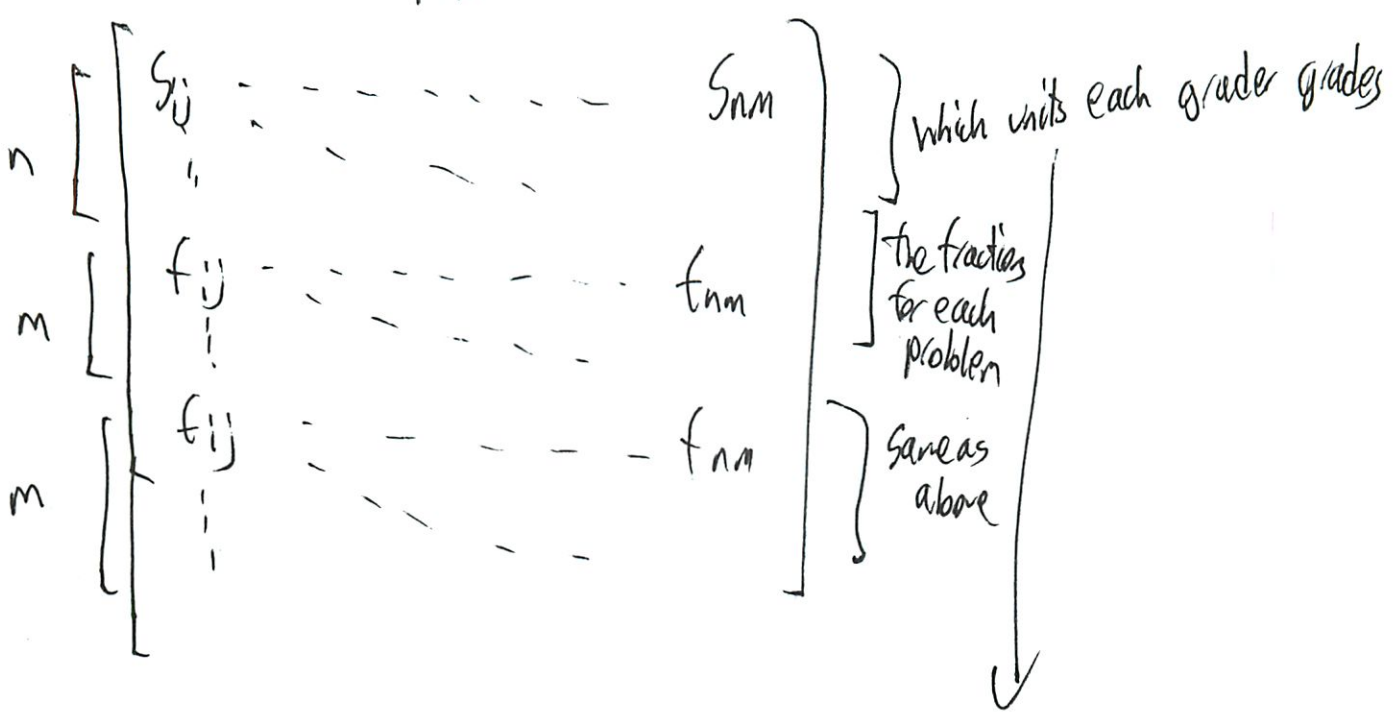
$$\begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{1m} \\ f_{21} \\ \vdots \\ f_{nm} \end{bmatrix}$$

rows: nm × cols: 1

c^T is A_{ij} = $\underbrace{[f_{11}\ f_{12} \dots f_{1m}\ f_{21} \dots f_{nm}]}_{nm}$ 1

(9)

A is the constraint matrix



for example w/ 3 problems, 3 graders

in b)

grades j \ i	i=1	i=2	i=3
1	1 0 0	0 0 0	0 0 0
2	0 0 0	0 1 0	0 0 0
3	0 0 0	0 0 0	0 0 1

(10)

b is $\begin{bmatrix} 1 \\ s_j \\ 1 \\ 1 \end{bmatrix}$ ← for each j

$3n$: 1 once, s_j each n times, 1 n times, 1 m times

This means the dual is

$$\min \begin{bmatrix} s_j & 1 & 1 \\ j \in [n] & n & n \end{bmatrix} y_{ij}$$

$$\text{s.t.} \quad \begin{bmatrix} j\in n & 3n & j\in n & j\in n \\ s_{ij} & \cdots & f_{ij} & \cdots & f_{ij} & \cdots \\ \vdots & & \vdots & & \vdots & \\ s_{nm} & & f_{nm} & & f_{nm} & \end{bmatrix} \in_{i,j}$$

nm

$$\circ\ y_{ij} \geq \begin{bmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{1m} \\ f_{21} \\ \vdots \\ f_{nm} \end{bmatrix}$$

$nm \times 1$

$y_{ij} \geq 0$

and same non neg constraints as before

(11)

The y_{ij} is a new variable that we have introduced.

It is related to the sensitivity of increasing or reducing s_j and $\sum_j^{\hat{n}} f_{ij}$ (the amount of workload for each problem)

This is what the dual lets us discover

Design and Analysis of Algorithms
Massachusetts Institute of Technology
Profs. Srini Devadas and Ronitt Rubinfeld

November 6, 2010
6.046J/18.410J
Handout 13

Problem Set 4 Solutions

This problem set is due **at 11:59pm** on **Tuesday, Nov 6, 2012.**

Exercise 4-1. Do Exercise 34.3-2 in CLRS on page 1077.

Exercise 4-2. Do Exercise 34.5-2 in CLRS on page 1101.

Exercise 4-3. Do Exercise 34.5-3 in CLRS on page 1101.

Exercise 4-4. Do Exercise 34.5-5 in CLRS on page 1101.

Exercise 4-5. Do Exercise 34.5-7 in CLRS on page 1101.

Exercise 4-6. Do Exercise 29.2-3 in CLRS on page 863.

Exercise 4-7. Do Exercise 29.4-3 in CLRS on page 885.

Exercise 4-8. Do Exercise 29.4-4 in CLRS on page 885.

Problem 4-1. Dividing Acorns

Chip and Dale gather a pile of m acorns, whose weights they find to be $\{w_1, ..., w_m\}$. These weights all happen to be positive integers in Chipmunk Standard Units. Acorns are hard to crack, and therefore cannot be split into fractions.

(a) Chip and Dale wish to determine whether it is possible to divide up the acorns evenly by weight. We refer to this as the EVEN-DIVIDE problem.
Show that EVEN-DIVIDE is NP-complete, using reduction from SUBSET-SUM.

Solution: This is also known as the SET-PARTITION problem.

Given a set of positive integers $W = \{w_1, ..., w_m\}$ as the input, EVEN-DIVIDE determines if there exists a subset $V \subset W$ such that $\sum_{w \in V} w = \frac{1}{2} \sum_{i=1}^{m} w_i$. We show its NP-completeness by showing that (i) it is in NP, and that (ii) it is NP-hard.

(i) EVEN-DIVIDE is in NP:

Recall that the number of bits required to represent a positive integer x is $b(x) = 1 + \log_2 x$. Moreover, for any two positive integers x and y, the number of bits required

to represent $x+y$ is no greater than the sum of the number of bits required to represent each of x and y, i.e., $b(x+y) \leq b(x) + b(y)$ [1].

The input size of EVEN-DIVIDE is $n = \sum_{i=1}^{m} b(w_i)$, the total number of bits required to represent positive integers in W. By the argument above, the sum of any subset of W can be calculated in polynomial time with respect to n. In particular, given a subset $W' \subset W$ whose sum is half that of W as a certificate, we can calculate the sum of both W and W' in polynomial time with respect to n. Checking that the sum of W' is indeed half the sum of W also takes polynomial time. Therefore, EVEN-DIVIDE is in NP.

(ii) EVEN-DIVIDE is NP-hard:

We reduce SUBSET-SUM to EVEN-DIVIDE. For an input (S, t) to SUBSET-SUM, where $S = \{s_1, ..., s_k\}$ is a set of positive integers and t is a positive integer target, the reduction constructs the input W to EVEN-DIVIDE as follows: Let $s_{total} = \sum_{i=1}^{k} s_i$. If $t \leq \frac{1}{2}s_{total}$, then $W = \{s_1, ..., s_k, s_{total} - 2t\}$; otherwise, $W = \{s_1, ..., s_k, 2t - s_{total}\}$. W can be constructed in polynomial time, since it only involves summation and comparison of numbers that require fewer bits than the input size to represent.

We show that $(S, t) \in$ SUBSET-SUM if and only if $W \in$ EVEN-DIVIDE:

($\Rightarrow$) If $(S, t) \in$ SUBSET-SUM, then there exists a subset $S' \subset S$ such that

$$\sum_{s \in S'} s = t.$$

Letting $\overline{S'} = S - S'$, we have

$$\sum_{s \in \overline{S'}} s = s_{total} - t.$$

If $t \leq \frac{1}{2}s_{total}$, then the set $W - \overline{S'} = S' \cup \{s_{total} - 2t\}$ and the set $\overline{S'}$ form an even partition of W, because

$$\sum_{s \in W - \overline{S'}} s = \sum_{s \in S'} s + (s_{total} - 2t) = s_{total} - t = \sum_{s \in \overline{S'}} s.$$

If $t > \frac{1}{2}s_{total}$, then the set $W - S' = \overline{S'} \cup \{2t - s_{total}\}$ and the set S' form an even partition of W, because

$$\sum_{s \in W - S'} = \sum_{s \in \overline{S'}} s + (2t - s_{total}) = t = \sum_{s \in S'} s.$$

Therefore, $W \in$ EVEN-DIVIDE.

[1] For positive integers x and y, comparing $b(x+y)$ and $b(x)+b(y)$ is equivalent to comparing $2^{b(x+y)} = 2(x+y)$ and $2^{b(x)+b(y)} = 4xy$. Since $2xy - (x+y) = 2(x - \frac{1}{2})(y - \frac{1}{2}) - \frac{1}{2} \geq 0$ for $x \geq 1$ and $y \geq 1$, we have $b(x+y) \leq b(x) + b(y)$.

($\Leftarrow$) Suppose $t \leq \frac{1}{2}s_{total}$ and $W = \{s_1, ..., s_k, s_{total} - 2t\} = S \cup \{s_{total} - 2t\}$ has an even partition. Denote the even partition as W' and $W - W'$, where the element $s_{total} - 2t$ is in W'. Then $W' - \{s_{total} - 2t\}$ is a subset of S. Moreover, the sum of W is $2s_{total} - 2t$, which implies that W' and $W - W'$ both sum up to $s_{total} - t$. As a result,

$$\sum_{s \in W' - \{s_{total} - 2t\}} s = \sum_{s \in W'} s - (s_{total} - 2t) = (s_{total} - t) - (s_{total} - 2t) = t.$$

In other words, $W' - \{s_{total} - 2t\}$ is a subset of S whose sum is t. Therefore, $(S, t) \in$ SUBSET-SUM. The case where $t > \frac{1}{2}s_{total}$ can be shown in a similar manner.

(b) They decide to save some acorns as a gift for their friend Clarice. They know that, for Clarice, there is a positive integer value v_i associated with the external appearance of each acorn $i = 1, ..., m$. They wish to determine whether it is possible to package a box of acorns with total value no less than v. The box can hold a weight of at most w. Show that this problem, which we call VALUE-PACKAGE, is also NP-complete, using reduction from EVEN-DIVIDE.

Solution: This is a version of the renowned KNAPSACK problem.

Given an input (V, v, w), where $V = \{(v_i, w_i) \mid i = 1, ..., m\}$ is a set of positive integer pairs and v, w are positive integers, VALUE-PACKAGE determines if there is a subset of indices $I \subset \{1, ..., m\}$ such that $\sum_{i \in I} v_i \geq v$ and $\sum_{i \in I} w_i \leq w$. We show its NP-completeness by showing that (i) it is in NP, and that (ii) it is NP-hard.

(i) VALUE-PACKAGE is in NP:

By a similar argument to that in (a), given the inputs (V, v, w) and a subset of indices I as a certificate, $\sum_{i \in I} v_i$ and $\sum_{i \in I} w_i$ can be calculated and compared with v and w, respectively, in polynomial time with respect to the input size. Therefore, VALUE-PACKAGE is in NP.

(ii) VALUE-PACKAGE is NP-hard:

We reduce EVEN-DIVIDE to VALUE-PACKAGE. Given an input $W = \{w_1, ..., w_m\}$ to EVEN-DIVIDE, the reduction constructs the input (V, v, w) to VALUE-PACKAGE by setting $V = \{(w_i, w_i) \mid i = 1, ..., m\}$ and $v = w = \frac{1}{2}\sum_{i=1}^{m} w_i$. This can be completed in polynomial time with respect to the size of W.

We show that $W \in$ EVEN-DIVIDE if and only if $(V, v, w) \in$ VALUE-PACKAGE. Indeed, $W \in$ EVEN-DIVIDE if and only if there exists a subset of indices $I \subset \{1, ..., m\}$ such that $\sum_{i \in I} w_i = \frac{1}{2}w_{total}$, where $w_{total} = \sum_{i=1}^{m} w_i$. This is true if and only if there exists a subset of indices I such that $\sum_{i \in I} w_i \geq \frac{1}{2}w_{total} = v$ and $\sum_{i \in I} w_i \leq \frac{1}{2}w_{total} = w$ in VALUE-PACKAGE, which is equivalent to $(V, v, w) \in$ VALUE-PACKAGE.

(c) Recall that the INTEGER LINEAR-PROGRAMMING (ILP) problem determines whether there exists an integer n-dimensional vector x such that $Ax \leq b$, given an integer m-by-n matrix A and an integer m-dimensional vector b.
Show that ILP is NP-complete, using reduction from VALUE-PACKAGE.

Solution: We show the NP-completeness of ILP by showing that (i) it is in NP, and that (ii) it is NP-hard.

(i) ILP is in NP:

Given an input (A, b) and a certificate x satisfying the requirements, the input size is $O(mn)$. Checking that x is a vector of integers take $O(n)$-time. Computing Ax and comparing it with b is also polynomial time with respect to the input size. Therefore, INTEGER LINEAR-PROGRAMMING is in NP.

(ii) ILP is NP-hard:

We reduce VALUE-PACKAGE to ILP. Given an input (V, v, w) to VALUE-PACKAGE, where $V = \{(w_i, v_i) \mid i = 1, ..., p\}$, the reduction constructs an input (A, b) to ILP from the following integer linear program, where $x_i, i = 1, ..., p$, are integers:

$$\sum_{i=1}^{m} v_i x_i \geq v$$
$$\sum_{i=1}^{m} w_i x_i \leq w$$
$$0 \leq x_i \leq 1, \quad i = 1, ..., m$$

More specifically, A is a $(2p + 2)$-by-p matrix, with $[-v_1 \cdots - v_p]$ as the first row, $[w_1 \cdots w_p]$ as the second row, and $I_{p\times p}$ and $-I_{p\times p}$ taking up the following rows, where $I_{p\times p}$ is the $p \times p$ identity matrix. The vector b is a $(2p + 2)$-dimensional vector, with $-v$ in the first entry, w in the second entry, 1 in the following p entries, and 0 in the last p entries. The running time of this reduction is polynomial with respect to the size of (V, v, w).

Next, we show that $(V, v, w) \in$ VALUE-PACKAGE if and only if the corresponding (A, b) constructed above is in ILP. This is clear from the fact that $(V, v, w) \in$ VALUE-PACKAGE if and only if there exists a subset of indices $I \subset \{1, ..., p\}$ such that $\sum_{i\in I} v_i \geq v$ and $\sum_{i\in I} w_i \leq w$. This is equivalent to the existence of a p-dimensional vector x, where $x_i = 1$ for i in some set I of indices and $x_i = 0$ otherwise, such that $\sum_{i=1}^{p} v_i x_i \geq v$ and $\sum_{i=1}^{p} w_i x_i \leq w$. Such a vector x exists if and only if $(A, b) \in$ ILP.

Problem 4-2. Problem-Set Grading Time

A TA is distributing submitted problem-set papers among graders. There are m problems, each with 1 unit of workload. There are n graders, each with a different expertise in grading, and it takes t_{ij} hours for grader j to grade one unit of problem i. Moreover, grader j can only take a total

workload of up to s_j units, where s_j is an integer. Assume that $\sum_{j=1}^{n} s_j \geq m$, i.e., it is possible for the graders to collectively grade all problems.

(a) Suppose the TA wishes to minimize the amount of time all graders spend in total. Formulate this as a linear program and write it in standard form. (Hint: Let f_{ij} be the fraction of problem i workload assigned to grader j.)

Solution: Let f_{ij} be the amount of problem i workload assigned to grader j. Then,

$$\begin{aligned}
\min_{f_{ij}} \quad & \textstyle\sum_{i=1}^{m}\sum_{j=1}^{n} t_{ij} f_{ij} & & \\
\text{subject to} \quad & \textstyle\sum_{i=1}^{m} f_{ij} \leq s_j, \quad j = 1, ..., n & & \text{(work-time constraint for each grader)} \\
& \textstyle\sum_{j=1}^{n} f_{ij} = 1, \quad i = 1, ..., m & & \text{(unit workload for each problem)} \\
& f_{ij} \geq 0, \quad i = 1, ..., m, j = 1, ..., n & & \text{(nonnegativity of assignment)}
\end{aligned}$$

Written in standard form,

$$\begin{aligned}
\max_{f_{ij}} \quad & \textstyle\sum_{i=1}^{m}\sum_{j=1}^{n} -t_{ij} f_{ij} & & \\
\text{subject to} \quad & \textstyle\sum_{i=1}^{m} f_{ij} \leq s_j, & j = 1, ..., n & \quad (1) \\
& \textstyle\sum_{j=1}^{n} f_{ij} \leq 1, & i = 1, ..., m & \quad (2) \\
& \textstyle\sum_{j=1}^{n} -f_{ij} \leq -1, & i = 1, ..., m & \quad (3) \\
& f_{ij} \geq 0, & i = 1, ..., m, j = 1, ..., n & \quad (4)
\end{aligned}$$

(b) Show that there exists a solution assigning the entire workload of every problem to a single grader each. (Hint: Use matrix properties presented in lecture.)

Solution: The constraints above can be written as $Ax \leq b, x \geq 0$, where

$$x = [\ f_{11} \quad f_{12} \quad \cdots \quad f_{1n} \quad f_{21} \quad f_{22} \quad \cdots \quad f_{mn}\ ]^T$$

is the mn-dimensional vector of workload assignments, and

$$A = \begin{bmatrix} A_1 \\ A_2 \\ -A_2 \end{bmatrix}, \quad b = \begin{bmatrix} s_1 \cdots s_n & \underbrace{1 \cdots 1}_{m} & \underbrace{-1 \cdots -1}_{m} \end{bmatrix}^T,$$

where

$$A_1 = \begin{bmatrix} \overbrace{I_{n\times n} \quad \cdots \quad I_{n\times n}}^{m \text{ identity matrices}} \end{bmatrix} \in \mathbb{R}^{n \times mn}$$

corresponds to constraints (1),

$$A_2 = \begin{bmatrix} \overbrace{1\cdots1}^{n} & 0\cdots & & \\ 0\cdots0 & 1\cdots1 & 0\cdots & \\ 0\cdots0 & 0\cdots0 & 1\cdots1 & 0\cdots \\ & & \ddots & \\ 0\cdots & & \cdots0 & 1\cdots1 \end{bmatrix} \in \mathbb{R}^{m\times mn}$$

corresponds to constraints (2), and $-A_2$ corresponds to the constraints (3).

To prove the claim, we first show that the coefficient matrix A is totally unimodular (TU) by showing that any square submatrix of A has a determinant of 0,1, or -1. Observe that the matrix $\overline{A} = \left[\begin{smallmatrix} A_1 \\ A_2 \end{smallmatrix}\right]$, the upper half of A, is TU, because it satisfies the following sufficiency conditions:

- Each entry is 0, 1, or -1;
- Each column contains at most 2 (in fact, exactly 2) nonzero coefficients;
- There exists a partition of its rows such that the sum of rows in each partition is equal. (A_1, A_2 is such a partition.)

As a result, every square submatrix of $\overline{A}$ has a determinant of 0,1, or -1.

Now let B be a square submatrix of $A = \left[\begin{smallmatrix} A_1 \\ A_2 \\ -A_2 \end{smallmatrix}\right]$, and let $I \in \{1, ..., m\}$ be the set of rows chosen from $-A_2$ to form B. Consider the following cases:

- If $I = \phi$, then no rows of B are chosen from $-A_2$, so B is a submatrix of $\overline{A}$, and its determinant is 0,1, or -1 since $\overline{A}$ is TU.
- If there is some row $i \in I$ such that the same row i is also chosen from A_2 to form B, then $det(B) = 0$ since these two rows of B are linearly dependent.
- If, for all $i \in I$, row i is not chosen from A_2, then consider the square submatrix $\overline{B}$ constructed by choosing the same rows from A_1 and A_2 as those chosen for B, and choosing rows I from A_2 (instead of from $-A_2$ as for B). Then $det(B)$ equals $det(\overline{B})$ or $-det(\overline{B})$, because the rows in B are identical to the rows in $\overline{B}$ except for the negation of each row corresponding to $i \in I$. Since $\overline{B}$ is a submatrix of the TU matrix $\overline{A}$, its determinant must be 0, 1, or -1; therefore, so is $det(B)$.

Next, we show that there exists an optimal integer assignment. Since A is TU and b is a vector of integers, there exists an optimal solution that is a vector of integers (see Theorem 5 of Recitation 8 notes, or the theorem stated in Lecture 13 found on p.5 of lecture notes). In other words, there is an optimal integer assignment $\{f^*_{ij}\}$. (Alternatively, simply observe that the formulation above is a special case of the minimum-cost flow problem, and directly apply the TU and integer optimum properties for minimum-cost flows to reach the same conclusion.)

Finally, for each i, due to constraints $\sum_{j=1}^{n} f_{ij} = 1$ and $f_{ij} \geq 0$, we have $f^*_{ij} = 1$ for some j and $f^*_{ij'} = 0$ for all $j' \neq j$, i.e., problem i is fully assigned to grader j.

(c) Knowing that students would like their problem-sets to be graded as soon as possible, the TA instead wishes to minimize the time it takes for all graders to finish grading, assuming that they start at the same time and work in parallel.
Modify your formulation in part (a) to achieve this, and show by example that an integer assignment, like that in part (b), cannot be guaranteed.

Solution: We add a decision variable T that denotes the maximum time spent among all graders. Then,

$$\begin{aligned}
\min_{f_{ij},T} \quad & T \\
\text{subject to} \quad & \textstyle\sum_{i=1}^{m} t_{ij} f_{ij} \le T, \quad j = 1, ..., n && (T \text{ is the maximum time spent among all graders}) \\
& \textstyle\sum_{i=1}^{m} f_{ij} \le s_j, \quad j = 1, ..., n && (\text{work-time constraint for each grader}) \\
& \textstyle\sum_{j=1}^{n} f_{ij} = 1, \quad i = 1, ..., m && (\text{unit workload for each problem}) \\
& f_{ij} \ge 0, \quad i = 1, ..., m, j = 1, ..., n && (\text{nonnegativity of assignment})
\end{aligned}$$

Since the constraint coefficient matrix consists of arbitrary entries t_{ij}, it is no longer totally unimodular, and thus an integer solution cannot be guaranteed.

Indeed, we can construct counterexamples for which no integer optimal solution exists. For instance, if there is only one problem and two graders with the same grading expertise, the optimal solution assigns half the workload to each grader to fully utilize parallelization.

(d) Write the dual of your formulation in part (a). What do the dual variables represent?

Solution: Associate dual variables p_j, with constraints (1), and dual variables $q_i, \overline{q}_i$ with constraints (2) and (3), respectively.

Using the standard forms of the primal (CLRS Equations(29.16)-(29.18)) and the dual (CLRS Equations(29.83)-(29.85)), the dual is

$$\begin{aligned}
\min_{p_j,q_i,\overline{q}_i} \quad & \textstyle\sum_{j=1}^{n} s_j p_j + \sum_{i=1}^{m} (q_i - \overline{q}_i) \\
\text{subject to} \quad & p_j + q_i - \overline{q}_i \ge -t_{ij}, && i = 1, ..., m, j = 1, ..., n \\
& p_j \ge 0, && j = 1, ..., n \\
& q_i \ge 0, \overline{q}_i \ge 0, && i = 1, ..., m
\end{aligned}$$

Letting $r_i = q_i - \overline{q}_i$ for all i, the problem simplifies to

$$\begin{aligned}
\min_{p_j,r_i} \quad & \textstyle\sum_{j=1}^{n} s_j p_j + \sum_{i=1}^{m} r_i \\
\text{subject to} \quad & p_j + r_i \ge -t_{ij}, && i = 1, ..., m, j = 1, ..., n \\
& p_j \ge 0, && j = 1, ..., n
\end{aligned}$$

Recall, by duality theory, that the optimal value of this dual is the optimal value of the primal in (a), which is the negative of the total grading time. From the cost function above, we can see that given an optimal solution $\{p_j^*, r_i^*\}$, each unit of increase in s_j (to the extent that the optimal solution remains the same) increases the optimal objective value (negative total grading time) by p_j^*. In other words, p_j^* indicates the "rate" at which the total grading time would decrease if the allowable workload of grader j were extended. Thus, p_j^* is a measure of how sensitive the total grading time is with respect to the workload capacity s_j of grader j. For example, a large p_j^* indicates that increasing s_j would result in a large improvement on the total grading time. This information from the dual optimal solution is helpful, for example, if we wish to identify which grader to ask first for more workload, in order to decrease the total grading time.

Similarly, for every i, the dual variable $r_i^* = q_i^* - \overline{q}_i^*$ can be interpreted as the "rate" at which the change in workload for problem i affects the total grading time; in other words, it reflects the sensitivity of the total grading time to the problem workload. In the objective function, the coefficient of r_i^* is 1 because problem i has unit workload. If, for instance, a fraction z of students didn't turn in problem i, then the total grading time would decrease by zr_i^* (to the extent that the optimal dual solution remains at $\{p_j^*, r_i^*\}$).

Direction of Proving
Reduction

8/7

C ∝ IS

C → IS vs IS → V

What is the difference?
How do you show?
Don't think we ever had a clear example in class

C → IS
If I form Ind Set for G then C=I is a k clique for G

Clique – every 2 vertices in subset connected by edge

= 4 clique
(also 4 1 vertex cliques)
6 2 vertex cliques → edges
0 3 vertex cliques

9

Ind Set Set of vertices where no 2 are adjacent
no edge connecting any vertex in the set

R | given Clique
Create input G'
↑ input to ind set

Same vertices
Only edge if $(u,v) \notin E$

~~CIS~~ | If have I - set of vertices for IS
then C=I is a k-clique
...
implies $(u,v) \notin E$ for clique
Shows C are completly connected in E

this is IS→C

(2)

C → IS| C is set of vertices that form a k clique
then I = C is a k ind set
...

Yeah they seem pretty similar

But were flipped in notes ← flipped in printed notes
That is what confused me

But things are pretty similar since
A ⟷ B

Should have looked at set cover - I got that
Clearer

Old Quiz

Design and Analysis of Algorithms
Massachusetts Institute of Technology
Profs. Piotr Indyk, Ronald Rivest and Ronitt Rubinfeld

Due November 21, 2011
6.046J/18.410J
Quiz 2

Quiz 2
Cover Sheet

Problem	Points	Grade	Initials
1	32		
2	18		
3	20		
4	30		

Name: _____

R01	R02	R03	R04	R05	R06
F10	F11	F12	F1	F2	F3
Rafael	Shaunak	Lin	Lin	Yu	Yu

R07	R08	R09	R10
F11	F12	F1	F2
Piotr	Piotr	Tom	Tom

INSTRUCTIONS

This take-home quiz contains 4 problems worth a total of 100 points. Your quiz solutions are due **at 11am on Monday, November 21, 2011.** Late quizzes will not be accepted unless you obtain a Dean's support or make prior arrangements with the course staff. You must submit your solutions electronically.

Guide to this quiz: For problems that ask you to design an efficient algorithm for a certain problem, your goal is to find the *most efficient algorithm possible*. Generally, the faster your algorithm, the more points you receive. For two asymptotically equal bounds, worst-case bounds are better than expected or amortized bounds. The best possible solution will receive full points if well written, but ample partial credit will be given for correct solutions, especially if they are well written. Bonus points may be awarded for exceptionally efficient or elegant solutions.

Plan your time wisely. Do not overwork, and get enough sleep. Your very first step should be to write up the most obvious algorithm for every problem, even if it is exponential time, and then work on improving your solutions, writing up each improved algorithm as you obtain it. In this way, at all times, you have a complete quiz that you could hand in.

Policy on academic honesty: The rules for this take-home quiz are like those for an in-class quiz, except that you may take the quiz home with you. As during an in-class quiz, you may not communicate with any person except members of the 6.046 staff about any aspect of the quiz, even if you have already handed in your quiz solutions. To communicate with the staff, you have to send an email to `6046-tas@csail.mit.edu`. We will not respond to your question if you send an email to the personal email of a staff member. In addition, you may not discuss any aspect of the quiz with anyone except the course staff **until November 30, 2011**. Note that this date is **after** the due date of the quiz.

This take-home quiz is "limited open book." You may use your course notes, the CLRS textbook, and any of the materials posted on the course web page, but *no other sources whatsoever may be consulted.* For example, you may not use notes or solutions to problem sets, exams, etc. from other times that this course or other related courses have been taught. **You may not use any materials on the World-Wide Web, including OCW.** You probably won't find information in these other sources that will help directly with these problems, but you may not use them regardless.

If at any time you feel that you may have violated this policy, it is imperative that you contact the course staff immediately. If you have any questions about what resources may or may not be used during the quiz, please send email to `6046-staff@csail.mit.edu`.

Write-ups: Type your answers up on separate files (the templates will be given), and submit all your answers electronically, as you would do in the problem set. We *strongly encourage* you to submit your problems typed in LaTeX, since this makes it easier for us to read your solutions and understand them.

Your solutions are due, electronically, at 11am on Monday the 21st of November. There is no hardcopy submission option for this test.

Your write-up for a problem should start with a topic paragraph that provides an executive summary of your solution. This executive summary should describe the problem you are solving, the techniques you use to solve it, any important assumptions you make, and the asymptotic bounds on the running time your algorithm achieves, including whether they are worst-case, expected, or amortized.

Write your solutions cleanly and concisely to maximize the chance that we understand them. When describing an algorithm, give an English description of the main idea of the algorithm. Adopt suitable notation. Use pseudocode if necessary to clarify your solution. Give examples, draw figures, and state invariants. A long-winded description of an algorithm's execution should not replace a succinct description of the algorithm itself. *Points will be taken off for overly convoluted solutions to the problems.*

Provide short and convincing arguments for the correctness of your solutions. Do not regurgitate material presented in class. Cite algorithms and theorems from CLRS, lecture, and recitation to simplify your solutions. Do not waste effort proving facts that can simply be cited.

Be explicit about running time and algorithms. For example, don't just say that you sort n numbers, state that you are using MERGE-SORT, which sorts the n numbers in $O(n \lg n)$ time in the worst case. If the problem contains multiple variables, analyze your algorithm in terms of all the variables, to the extent possible.

Part of the goal of this quiz is to test your engineering common sense. If you find that a question is unclear or ambiguous, make reasonable assumptions in order to solve the problem, and state clearly in your write-up what assumptions you have made. Be careful what you assume, however, because you will receive little credit if you make a strong assumption that renders a problem trivial.

Bugs: If you think that you've found a bug, send email to `6046-staff@csail.mit.edu`. Corrections and clarifications will be sent to the class via email and posted on the class website. Check your email and the class website daily to avoid missing potentially important announcements.

Good Luck!

Problem 1. Minimum Spanning Trees and Matchings [32 points] (4 parts)

In all four parts of this problem, $G = (V, E, w)$ is a connected, weighted, undirected graph. We define $w(e)$ to be the weight of edge e.

(a) Suppose that T is a minimum spanning tree of G. If we increase the weight of each edge of G by the same positive amount δ, is T still guaranteed to be a minimum spanning tree? Give an argument for why it will be, or a counterexample showing that it may not be.

(b) Let e be an edge of maximal weight in G. Suppose we add a single new edge e' to G to obtain the graph $G' = (V, E \cup \{e'\}, w)$. The weight function w is extended so that $w(e')$ is defined, and $w(e')$ is less than $w(e)$ – that is, the new edge does not have maximal weight. Show that the total weight of a minimum spanning tree of G' is at least $w(e') - w(e)$ plus the total weight of a minimum spanning tree of G.

(c) We now consider maximum-weight matchings in G. Suppose that we increase the weight of each edge of G by the same positive amount δ. Is it possible that a matching of maximum weight in G will no longer be maximum-weight after the change? Give an argument for why it will be, or a counterexample showing that it may not be.

(d) Finally, we consider maximum-cardinality matchings in G. A maximum-cardinality matching is a matching with the maximum possible number of edges. Suppose we add a single new edge e' to G to obtain the graph G' with edge set $E \cup \{e'\}$. Show that the size of a maximum-cardinality matching in G' is not more than one larger than the size of a maximum-cardinality matching in G.

Solution: (a) Yes. Since each tree in G has $(|V| - 1)$ edges, after the increase of weight, the total weight of each tree will be increased by the same amount $(|V| - 1)\delta$. T is therefore still the minimum spanning tree in the new graph.

(b) Let T and T' be the minimum spanning tree of G and G' respectively. By adding $\{e'\}$ to T, we create a cycle in $(T \cup \{e'\})$ that contains $\{e'\}$. The new minimum spanning tree T' in G' can be obtained by replacing the edge that has maximum weight in the cycle with $\{e'\}$. We call the edge e_m, then $w(T') = w(T) - w(e_m) + w(e') \geq w(T) - w(e) + w(e')$.

(c) Yes. it is possible. A simple example is a graph with two vertices and one edge e connecting the two vertices that has negative weight $w(e)$. Before the increase of weight, the maximum-weight matching is empty. If $\delta > -w(e)$, then after the increase of weight, the maximum weight matching is $\{e\}$.

(d) Let the the maximum-cardinality matching in G and G' be M and M' respectively. We consider two cases. First, If $e' \notin M'$, then M' is a feasible matching in G. By the definition of maximum carnality matching, $|M'| \leq |M|$. Second, if $e' \in M'$, then $M' \setminus \{e'\}$ is a feasible matching in G, $|M' \setminus \{e'\}| \leq |M|$ or equivalently $|M'| \leq |M| + 1$. In either case, the size of M' is at most one larger than the size of M.

Problem 2. Gerrymandering in LineLand [18 points]

There are n residents of LineLand, residing at the points $1, 2, \ldots, n$ of the Line which is their world. There are two political parties in LineLand: the Cardinals and the Cyans. It is known to everyone if LineLander i is a Cardinal or a Cyan, for each i, $i = 1, 2, ..., n$.

It is time to "redistrict" – i.e. to divide LineLand into m political districts. Here m is significantly smaller than n.

Each district must be a contiguous segment of LineLand: it must include LineLander i up to LineLander j, inclusive, for some i and j, $i \leq j$.

Thus, the first district goes from 1 up to a_1, the second from $a_1 + 1$ up to a_2, and so on, until the last district is from $a_{m-1} + 1$ up to $a_m = n$.

To be fair, each district must have n/m LineLanders, more or less. More precisely, by the LineLand Constitution, a district must contain at least $r = \lceil n/(2m) \rceil$ LineLanders, and may not contain more than $s = \lfloor 3n/(2m) \rfloor$ LineLanders.

Once the districts are drawn, the LineLand Parliament is created by picking one resident uniformly at random from each district. (So the Parliament has size m.)

Explain what algorithm you would use to determine the districts (i.e., to determine $a_1, \ldots, a_m$), given n, the number m of districts desired, and the political affiliation of each LineLander, in such a way that would maximize the expected number of Cardinal members in Parliament. (Also explain how efficient your algorithm is, as a function of n and m.)

Solution: We will solve this problem using dynamic programming. Our subproblem will be

$$f(i, j) = \text{most expected Cardinals from dividing } i, \cdots, n \text{ into } j \text{ regions.}$$

Thus the solution to our original problem is $f(1, m)$. Our base cases will be

$$\begin{aligned} f(n, 0) &= 0, \\ f(i, 0) &= -\infty, \ \forall i < n, \\ f(i, j) &= -\infty, \ \forall i > n - r, j > 0. \end{aligned}$$

The second and third base cases handle the cases where it is impossible to correctly divide up the region. Given these base cases, our recurrence will be

$$f(i, j) = \max_{k \in \{r, \cdots, s\}} \left(f(i + k, j - 1) + \frac{\#\text{ Cardinals in } i, \cdots, k - 1}{k} \right).$$

Essentially, the dynamic program trys all possible district sizes for the first district in the region $i, \cdots, n$ and selects the best one by evaluating the expected contribution of that district plus the most expected Cardinals achievable from the remaining districts.
The dynamic program is $O(nm)$ subproblems, and each subproblem takes $O(n/m)$ time to evaluate (since s is $O(n/m)$), so that total running time is $O(n^2)$.

Problem 3. The Ivory Tower [20 points]

A group of undergraduates, graduate students and professors work in an ivory tower. Each professor p is willing to work with a subset $U(p)$ of the undergrads and a subset $G(p)$ of the graduate students; a *research team* is comprised of an undergrad, a graduate student and a professor who is willing to work with both. No one can belong to more than one research team.

Find an efficient algorithm for determining the maximum number of research teams that can be formed given the professors' preferences.

Solution:

Maximum Flow: We will solve this problem with maximum flows. Let's first explore the following idea where a valid team is represented by a flow from undergraduate -¿ professor -¿ graduate. Like usual, we create a node for each person in P, U, G. And connect an directed edge (u_i, p_j) if professor p_j is willing to work with undergraduate u_i. Similarly, connect a directed edge (p_j, g_k) if professor p_j is willing to work with graduate student g_k. We then create a source s connected to each undergraduate as well as a sink t connected from every graduate. All edges have capacity 1. However, there is a problem with this construction so far. Notice that each professor can only be part of 1 research group, we must ensure a "vertex capacity" of 1 on each professor node. We can accomplish this by breaking each professor node into two nodes p and p', and create an edge of capacity 1 from p to p'. All the incoming edges from undergraduates will be connected to p, whereas all the out going edges to graduates will be connected from p'. Thus, a flow path from s to t will go through u, p, p', and g. The two node gadget with a capacity 1 edge for each professor ensures that there cannot be multiple flows across it.

Maximum Matching:

We will prove that the size of the maximum matching M in the graph used for maximum flow (without the source and sink) has value $P + k$, where k is the maximum number of research teams. Given this, we can thus find k in $O((P + U + G)^{2.5})$ time as the maximum matching can be found in $O(\sqrt{V}E)$.

First we will argue that if there is a valid assignment of k research teams, then there is a matching of size $P + k$. For each of the k research teams, match the undergrad to the first professor vertex and the grad student to the second professor vertex, for a total of $2k$ matched edges. For each of the professors not in any of the k research teams, neither of its vertices is matched so we can select the edge between them, for a total of $P-k$ edges. Thus we have a matching of size $2k+P-k = P+k$.

Next we will argue that if there is a maximal matching M of size $P + k$, then there is a valid assignment of k research teams. Observe that every edge in the graph is adjacent to at least one professor vertex. Thus, we can write $M = \sum_{p_i \in P} M(p_i)$ where $M(p_i)$ is the number of edges in M adjacent to the vertices corresponding to professor p_i. Observe that in any maximal matching, for a given professor, at least one of its two vertices must be matched. This is because there is an edge between the two vertices so if neither was matched then we could match them to each other. Thus, $M(p_i) \geq 1$. Since p_i corresponds to exactly two vertices, $M(p_i) \leq 2$. Therefore $M = (P - k) + 2k = P + k$ where k is the number of professors with two edges in the matching. If a professor has two edges in the matching, then one must go to an undergrad and one must go to a grad student thus the set of k professors matched to two vertices corresponds to a valid assignment of k research teams.

Problem 4. Red-Blue Network Flow [30 points] (2 parts)

You are given a flow network $G = (V, E)$ with distinguished source vertex s and distinguished sink vertex t. As usual, the edges in E are directed edges. The edges in E are now, however, of two types: blue and red. The blue edges are of the usual type: each blue edge e has a capacity $c(e)$ that is the *maximum possible* amount of flow that can pass through that edge. The red edges are built out of strange alien technology: each red edge e must have at least a certain amount $d(e)$ of flow passing through that edge. That is, $d(e)$ is the *minimum required* amount of flow that must pass through that edge.

Other that the fact that some edges are now red, every thing is as usual (e.g. flow must be conserved at all intermediate vertices, etc.). Each edge is either red or blue. We call such a network problem a "red-blue network flow problem". As usual, the goal is to find a flow of maximum possible value.

(a) Show that it is possible to compute a maximum red-blue network flow, or determine that no legal flow exists, in polynomial time.

(b) You now discover how to "turn off" red edges. If a red edge e is on, at least $d(e)$ units of flow must pass over it; if it is off, no flow can pass over it. You can turn some red edges on and others off. Show that it is now NP-complete to determine the maximum flow possible in a red-blue network.

Solution: (a) The problem can be formulated as a linear programming problem. Let $f(u, v)$ be the flow on edge $(u, v) \in E$, B and C be the sets of blue and red edges respectively. The LP is

$$\max \sum_v f(s, v) \;\; s.t.$$

$$f(u,v) \leq c(u,v) \;\; for \;\; any \;\; (u,v) \in B \;\; , f(u,v) \geq d(u,v) \;\; for \;\; any \;\; (u,v) \in C$$

$$f(u,v) \geq 0 \;\; , \sum_v f(u,v) = \sum_w f(w,u) \;\; for \;\; any \;\; u \in \{V - s, t\} \qquad (1)$$

The LP can be solved in weakly polynomial time by ellipsoid or interior point algorithm.

(b). Since the NP-complete is for decision problems, the question can be interpreted as determine if there is a legal k flow in the flow network. To prove the problem is NP-hard, we can reduce the SUBSET-SUM problem to the red-blue flow network problem. The SUBSET-SUM is the decision problem that determines if there is a subset in $S = \{x_1, x_2, ..., x_n\}$ whose sum equal to k.
We first create a red-blow flow network with source s, sink t and another n vertices. The edges in the flow network are as follows,
(s, u_i) for $i = 1, 2, ..., n$ are red edges that connects s to n vertices that represent n variables in the subset. $c(s, u_i) = x_i$
(u_i, t) for $i = 1, 2, ..., n$ are blue edges that connects u to sink t. $d(u_i, t) = x_i$.
The way we create the graph guarantees that the flow from s to t through u_i can either be x_i if the red edge (s, u_i) is on or 0 if the red edge is off. The flow in the red-blue network is therefore equal to the sum of the x_i whose corresponding red edge is on. Determine if there exists a flow k is the same as determine if there is a subset sum k. Creating the red-blue network takes polynomial running time. Since SUBSET-SUM is NP-complete, the red-blue flow network problem is NP-hard. Given the assignment of flow on each edge and the on/off assignment of red edges, it takes polynomial time to check if flow k exists, so the problem is in NP. Combining with the NP-hardness proof, the red-blue network-flow problem is NP-complete.

An example of a good soln

Alternate solutions: Many tried to reduce from 3DM, but without any gadgets. The resulting "natural" reduction is incorrect. The few who got correct solutions generally solved a different problem, in which the triangles must be vertex-disjoint instead of edge-disjoint. From the rein-à-trois perspective, this corresponds to allowing a more flexible definition of compatibility. This is a different problem, but we graded as if it were the same. In this variation, correct reductions were obtained from 3SAT, Graph 3-Coloring, and Independent Set.

Problem 5. The Producer

You are the producer for a soon-to-be cult-classic movie, *6.046 Returns*, and you need both actors and investors. You have constructed a list of all n available actors (mostly former 6.046 students), with actor $i \in \{1, 2, \ldots, n\}$ charging a_i dollars. For funding, you have found m available investors. Investor j offers b_j dollars toward your budget, but only on the condition that actors $L_j \subseteq \{1, 2, \ldots, n\}$ are included in the movie. (The offer is "all or nothing": all actors in L_j must be chosen in order to receive any funding from investor j, and then the funding is b_j.) Your profit is the sum of the payments from investors minus the sum of the payments to actors.

Design the most efficient algorithm you can to maximize your profit. (You have no limits on the number of actors or the amount of funding.)

Solution:

Executive summary: We can represent this problem as a linear program, and use a linear program solver to solve it in polynomial time. The most straightforward way to solve this problem via linear programming is to formulate this problem as an integer linear program, then relax the integrality constraints on the variables to get a linear program. One can prove that the solution to this relaxed linear program may be transformed into a solution for the original problem. A viable alternative solution, which is faster than linear programming, is to reduce this problem to maximum flow on a particular graph.

Exponential-time solutions: The simple exponential-time solution to this problem is to try all possible combinations of investors, compute the total profit from those investors, and choose the combination that maximizes this total profit. A similar exponential-time solution may iterate over all possible combinations of actors.

Linear-programming solution: Our problem can be rewritten as the following integer linear programming (ILP) problem:

$$\begin{array}{lrl}
\text{maximize} & \displaystyle\sum_{j=1}^{m} b_j y_j - \sum_{i=1}^{n} a_i x_i & \\
\text{subject to} & x_i \geq y_j & \text{for every } 1 \leq i \leq n, 1 \leq j \leq m \text{ such that } i \in L_j \\
& x_i = 0 \text{ or } 1 & \text{for each } i = 1, 2, \ldots, n \\
& y_j = 0 \text{ or } 1 & \text{for each } j = 1, 2, \ldots, m
\end{array}$$

How fast ~~must ILP be~~ Can ILP be?

In this ILP, the variable x_i corresponds to the actor i and the variable y_j corresponds to investor j. $x_i = 1$ represents the event that actor i is hired, and $x_i = 0$ means that actor i is not hired. $y_j = 1$ indicates that the offer from investor j is accepted, and $y_j = 0$ indicates that investor j's offer is not accepted.

If we relax the requirement that the solution must be integral, we will get the following linear programming (LP) problem:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{m} b_j y_j - \sum_{i=1}^{n} a_i x_i & \\
\text{subject to} \quad & x_i \geq y_j & \text{for every } 1 \leq i \leq n, 1 \leq j \leq m \text{ such that } i \in L_j \\
& x_i \leq 1 & \text{for each } i = 1, 2, \ldots, n \\
& y_j \leq 1 & \text{for each } j = 1, 2, \ldots, m \\
& x_i \geq 0 & \text{for each } i = 1, 2, \ldots, n \\
& y_j \geq 0 & \text{for each } j = 1, 2, \ldots, m
\end{aligned}$$

Clearly the optimal objective value for this LP is as least as large the optimal objective value for the ILP. Therefore, if we can show that there is an optimal solution for the LP that is integral, then this solution must also be the optimal solution for the ILP.

Let $(\vec{x^*}, \vec{y^*})$ be an optimal solution for the LP. There are two possible cases:

1. $\vec{x^*}$ **is integral** (i.e., x_i^* is either 0 or 1 for every $i = \{1, 2, \ldots, n\}$): Assume that $\vec{y}$ is not integral. There must be some y_j^* such that $0 < y_j^* < 1$. Clearly, by setting $y_j^* = 1$ we will obtain a better solution and no constraint is violated. This gives us a contradiction, and therefore $\vec{y}$ must also be integral.
2. $\vec{x^*}$ **is not integral**: Let $x_{\min}^*$ be the minimum positive value among all x_i^*'s (i.e., $x_{\min}^* = \min_{x_i^* > 0} x_i^*$). Since $\vec{x^*}$ is not integral, $x_{\min}^*$ is not integral. Let $S_{\min} = \{i : x_i^* = x_{\min}^*\}$. Clearly, $S_{\min}$ is non-empty by definition. Let $T_{\min} = \{j : y_j^* > 0 \text{ and } L_j \cap S_{\min} \neq \emptyset\}$. By the constraints in the LP, $y_j < x_{\min}^*$ for all $j \in T_{\min}$. Let $a = \sum_{i \in S_{\min}} a_i$ and $b = \sum_{j \in T_{\min}} b_j$. There are 3 possible cases:
   i. $a > b$: By setting $x_i^* = 0$ for all $i \in S_{\min}$ and $y_j^* = 0$ for all $j \in T_{\min}$, the objective value will be increased by
   $$\sum_{i \in S_{\min}} x_i a_i - \sum_{j \in T_{\min}} y_j b_j \geq a \cdot x_{\min} - b \cdot x_{\min} > 0.$$
   Therefore, we can obtain a better solution and no constraint is violated, which contradicts the optimality of our original solution.
   iii. $a < b$: Let x_1^* be the next smallest positive value in $\vec{x}$ (i.e., $x_1 = \min_{x_i^* > x_{\min}^*} x_i^*$). Then by increasing x_i^* for all $i \in S_{\min}$ and y_j^* for all $j \in T_{\min}$ by $x_1 - x_{\min}$, the objective value will be increased by
   $$(b - a)(x_1 - x_{\min}) > 0.$$

Therefore, the can obtain a better solution and no constraint is violated, which again contradicts the optimality of our original solution.

ii. $a = b$: By setting $x_i^* = 0$ for all $i \in S_{\min}$ and $y_j^* = 0$ for all $j \in T_{\min}$ we will obtain another optimal solution for the LP with smaller number positive variables and no constraint is violated.

Therefore, $(\vec{x^*}, \vec{y^*})$ is either integral or there is an algorithm to obtain another optimal solution from $(\vec{x^*}, \vec{y^*})$ with smaller number of positive variables.

Our algorithm works as follows:

```
1  Set up a LP problem based on the input.
2  Solve the LP problem (using an efficient LP solver) to obtain a solution (x*, y*).
3  while (x*, y*) is not integral
4      do
5          x*_min = min_{x*_i > 0} x*_i
6          S_min = {i : x*_i = x*_min}
7          T_min = {j : y*_j > 0 and L_j ∩ S_min ≠ ∅}
8          Set x*_i = 0 for all i ∈ S_min
9          Set y*_j = 0 for all j ∈ T_min
10 Output (x*, y*).
```

Maximum-flow solution: A faster algorithm for solving this problem uses a reduction to maximum flow. One may represent this problem using a flow network containing one vertex y_j for each investor j, one vertex x_i for each actor i, one source vertex s, and one sink vertex t. Connect s to each x_i with an edge of capacity a_i, and connect each y_j to t with an edge of capacity b_j. Finally, for each actor i and each investor j, if $i \in L_j$ then connect x_i to y_j with an infinite capacity edge. Finding the maximum flow through this flow network will expose a minimum cut of this flow network

We can find an $(S, T = V \setminus S)$ cut in this flow network that has finite capacity, so the minimum (S, T) cut of this flow network must also have finite capacity. Observe that the finiteness of the minimum cut of this flow network must enforce the all-or-nothing constraint of the problem. Suppose some actor vertex x_i is in the S portion of the minimum cut, and suppose some actor vertex y_j, where $i \in L_j$, is in the T portion of this cut. Because $i \in L_j$, by construction of this flow network, there must exist an edge of infinite capacity from x_i to y_j. Therefore, there must exist an edge of infinite capacity from S to T, and the total capacity of this cut must be infinite. This contradicts our assertion that the minimum cut must have finite capacity, so this scenario cannot happen. Therefore, if an investor vertex y_j is in the T portion of this cut, then all actor vertices x_i where $i \in L_j$ must also be in the T portion of this cut.

We can derive the optimal profit for this problem from the size of the minimum cut in this flow network. Note that the total profit from the optimal selection of actors i and investors j is

$$\sum_{j \in OPT} b_j - \sum_{i \in OPT} a_i.$$

Meanwhile, the capacity of the minimum cut is

$$\sum_{j:y_j \in S} b_j + \sum_{i:x_i \in T} a_i = \sum_j b_j - \sum_{j:y_j \in T} b_j + \sum_{i:x_i \in T} a_i.$$

The sum of all investor contributions is fixed for a given problem instance, so the capacity of the minimum cut must maximize the value of $\sum_{j:y_j \in T} b_j - \sum_{i:x_i \in T} a_i$, which is exactly the equation for the profit of the original problem. Consequently, the minimum cut of this flow network maximizes the profit of the original problem, and because maximum flow is equivalent to minimum cut, finding the maximum flow of this flow network gives us an optimal solution to the original problem.

A similar solution to this problem swaps the positions of the actor and investor nodes in this flow network, and therefore chooses the actors and investors on the S side of a minimum cut.

Alternate attempts: Many people tried to solve this problem using dynamic programming. The difficulty with such a solution is that this problem does not exhibit optimal substructure, so any feasible dynamic programming solution would be equivalent to an exponential time algorithm. Similarly, greedy strategies to this problem do not work due to this problem's lack of optimal substructure.

Ch 046
Approximation Algs 1

11/6

Defn

Vertex Cover; two algs

Partition

Set Cover (in notes)

Sometimes we need the exact ans

But often we can settle for an approximation

- Like Simplex it runs over & over again, getting better each time
- Stop at some point
- Get* some* guarantee about output
- Possibly no more than 2 worse
  - if ans 50
    - get 25 -> 100

(confirm order)

Approximation algorithm + Scheme

problem of size n

define approx ratio $\rho(n)$

if alg produces a sol'n of cost C

for any input C^* s.t. $\max\left(\frac{C}{C_{opt}}, \frac{C_{opt}}{C}\right) \leq \rho(n)$

Sometimes greater
Sometimes less than

Then we have a $\rho(n)$-approx alg

An approx scheme lets you control error

Can shrink at cost of ↑ runtime

We don't know approx alg for many np-algs!

(3)

Take as an input $\epsilon > 0$ and for any fixed ϵ
the scheme is $(1+\epsilon)$-approx alg

algo $O(n^{2/\epsilon})$
↳ polynomial in n - PTAS

algo fully PTAS: $O\left(\frac{n}{\epsilon^2}\right)$ ← ? poly-time approx scheme too large on some ϵ
poly in both n and $\frac{1}{\epsilon}$
Can get arbitrarily close in poly time w/ a given ϵ

Vertex Cover

- Heuristics
- Guarantees on optimality

Undirected graph $G(V, E)$
Find a subset $V' \subseteq V$ s.t. if (u, v) is an edge in G,

(9)

Then either $u \in V'$ or $v \in V'$

Goal: Find a V' so cardinality is minimum

Heuristic: # of edges connected to node (max degree)

Possibility: Drop Those

Then pick next one w/ max edges

(Greedy)

Pseudocode:

Approx_Vertex_Cov_Natural

$C \leftarrow \emptyset$

$E' \leftarrow E$

while $E' \neq \emptyset$

Pick V w/ max degree $C = C \cup \{V\}$

Remove V from G and all incident edges on V from E'

return C

(5) Another option

Approx Vertex Cover

$C \leftarrow \emptyset$

$E' \leftarrow E$

While $E' \neq \emptyset$

Pick $(u,v) \in E$ arbitrarily ← before was particular kind

$C \leftarrow C \cup \{u\}, u \in V$ ← add both endpoints to C

Delete from E' all edges on u, v

return C

Not going to show poly time and will produce a vertex tire (proof by induction)

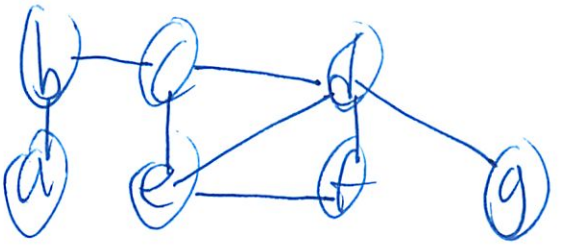
Which is better?

Class does not want to commit

Prof: You have commitment issues

(e)

Example



Natural

pick d

pick b

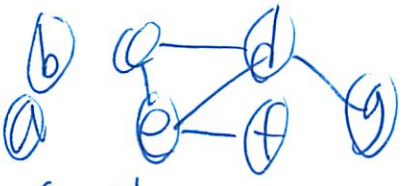
pick e

✓ optimum

Random

Say pick {b, c}

add b, c



pick {e, f}

add e, f

pick {d, g}

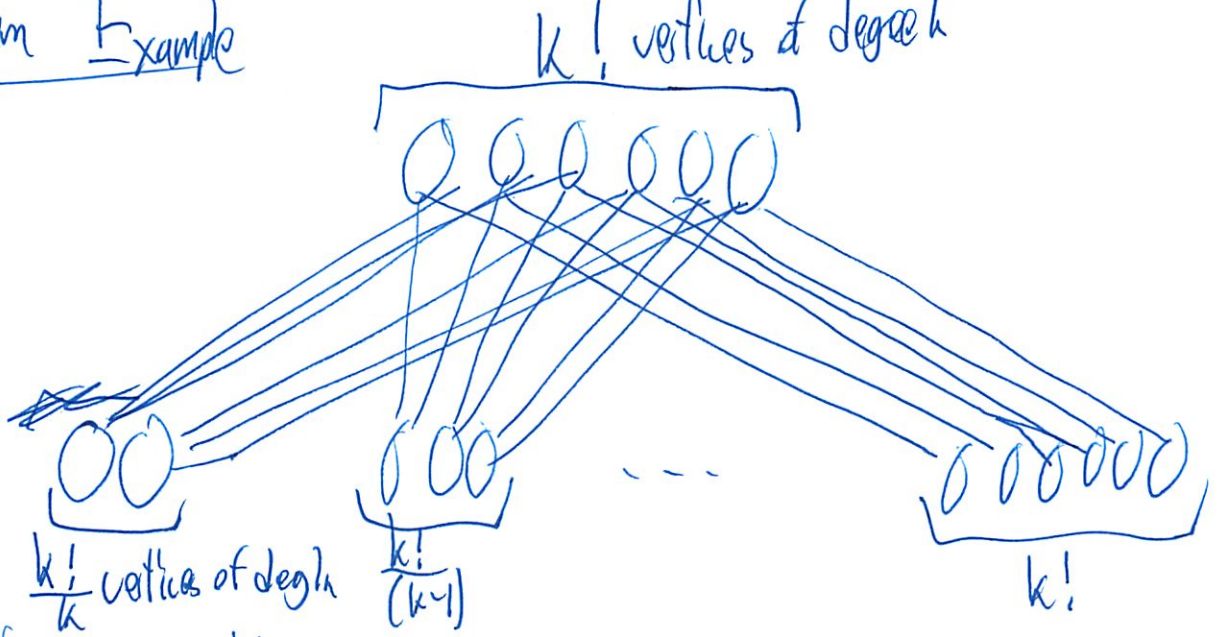
add d, g

Sooo that looks pretty bad

(7)

But second is actually better for approx alg
└ Since there we are covered w/ worst case
not best case/avg case

Extream Example



Natural will get further away from optimum as graph grows!

$k = 3$

~~$k!$ vertices of degree k~~

Sol $k!\left(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1}\right)$

$\sim k! \log k$

but as you grow graph - get further + further away

8

If has a bound that grows w/ size of problem
(don't fully get at the moment...)

Random one doesn't grow!

Always within a factor of 2 of answer

Real was 3
Random was 6) ✓ within factor of 2

Will it always be within a factor of 2?

Yes

Prof A.V.C is 2-approximate

1. Correct
2. Poly time
3. Ans ~~off~~ within 2 of optimal

Let A denote the edges that are picked

Optimum Cover Copt

(9)

Let A denote the edges that are picked

Optimal size Cover C_{opt} → cover should include at least one endpoint of each edge in A

notice ~~we not repetitive~~ (missed)

no 2 edges in A share an endpoint

Then $|A|$ is a lower bound for C_{opt}

$\leq$ cardinality of A ← ? cardinality for optimum sol

So $C_{opt} \geq |A|$

# of vertices in $C = 2|A|$

$|C| \leq 2\, C_{opt}$ ⊠ □

Gave us constant approx alg

Superior to other alg in approx ratio

Since can't get constant since we've seen example where as graph size grows → get away from optimal

But we have not yet proved it (Natural) is approximate
Could be heuristic w/o guarentees

Approx VC Natural is log(n) -approx

n = # of edges in graph
k! log k + k edges in our bad example

$n \geq k!$
Worse off by log(k)
Worse then log(log(n))
|G| = n

Start w/ G = G_0
Each time we shrink G, call it G_1, G_2, etc
Stop at G_n
w/ edge selection + vertex + edge deletion
m = C_{opt} ← # of vertices in optimum cover

(11)

Use this to bound when alg will terminate

Picking max degree vertex of G_{i-1}

↑ call this degree d_i

Will delete d_i edges

from graph G_{i-1} to get G_i

Make analysis on how quickly we are shrinking the graph

Each iteration adds 1 vertex to cover

So #iterations = #vertices

$$|G_m| = |G_0| - \sum_{i=1}^{m} d_i$$

↑ #edges in G

↑ #edges that have been deleted

$$\sum_{i=1}^{m} d_i \geq \sum_{i=1}^{m} \frac{|G_{i-1}|}{m}$$ ← key step

(12)

We are assuming we have an optimal vertex cover of size m

then must be at least one vertex w/ degree $\geq \frac{m}{n}$

- lower bound on each of G_i
- That $(R+S)$ is optimum sol
- Want to show optimum sol is a certain size

We know G is shrinking

$$\sum_{i=1}^{m} d_i \geq \sum_{i=1}^{m} \frac{|G_m|}{m}$$

Since $|G_0| \geq |G_1| > |G_2| > \ldots > |G_m|$

$$= G_m$$

So $|G_m| = |G_0| - \sum_{i=1}^{m} d_i$ ← So it's bigger than before

$$|G_0| - |G_m| \geq |G_m|$$

(smaller than $\sum$ term)

(13)

So after m iterations

$|G_0| \geq |G_m|$

Cut down size of graph by a factor of 2

$m \cdot \log |G_0| = m \log n$ iterations

Size of C

Might seem specific

But there is a method here

Must use some property of optimum cover

to prove optimization bound

Look at alg

Analize execution

(14)

So 2 approx schemes for same problem

Partition

trivial constant time approx ~~schem~~ alg
2-approx

but want the PTAS version
So if we spend more + more time → we get closer to optimal

(must leave 25 min early)

L16

11/6 (1)

6.046
L16

Approximation Algorithms I

Definitions

Vertex cover
Set cover
Partition
} NP-complete problems or NP-hard

Approximation Algos & Schemes

An algorithm for a problem of size n has an approximation ratio $\rho(n)$ if for any input, algorithm produces a solution of cost C such that

$$\max\left(\frac{C}{C_{opt}}, \frac{C_{opt}}{C}\right) \leq \rho(n)$$

Algorithm is an $\rho(n)$-approximation algorithm

An approximation scheme takes as input $\varepsilon > 0$ and for any fixed ε, the scheme is a $(1+\varepsilon)$-approximation algorithm.

Probabilistic time approximation scheme (PTAS): polynomial in n

Fully PTAS: polynomial in n and $\frac{1}{\varepsilon}$

$O(n^{2/\varepsilon})$ PTAS not FPTAS. $O(n/\varepsilon^2)$ FPTAS

Vertex Cover

Undirected graph $G(V, E)$

Find a subset $V' \subseteq V$ such that if (u, v) is an edge of G, then either

$u \in V'$ or $v \in V'$ or both.

Find a V' so $|V'|$ is minimum.

Approx-Vertex-Cover

$C \leftarrow \emptyset$

$E' \leftarrow E$

while $E' \neq \emptyset$

  Pick $(u, v) \in E$ arbitrarily

  $C \leftarrow C \cup \{u\} \cup \{v\}$

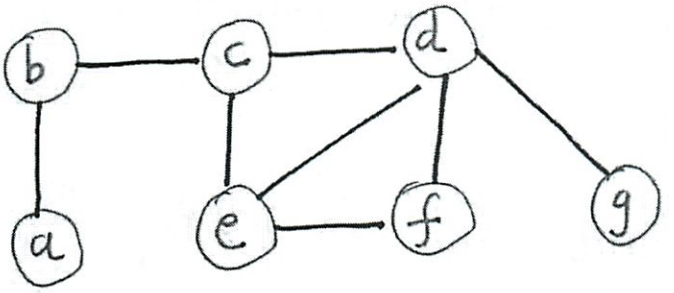
  Delete from E' all edges incident on u or v

Return C

Runs in poly time. Produces a vertex cover.

How close to optimal?

EXAMPLE



Approx_Vertex_Cover could pick (b,c), (e,f), (d,g)

$C = \{b, c, d, e, f, g\}$ $|C| = 6$

Optimal solution $C_{opt} = \{b, d, e\}$ $|C_{opt}| = 3$

Approx_Vertex_Cover is a 2-approximation algorithm

Proof: Let A denote the edges that are picked.

Optimal cover C_{opt} must include at least one endpoint of each edge in A (and other edges)

No two edges in A share an endpoint.

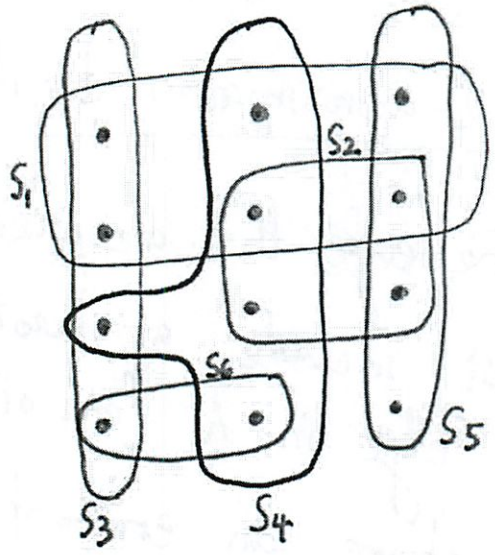
$|A|$ is a lower bound for $|C_{opt}|$, $|C_{opt}| \geq |A|$

Number of vertices in $C = 2|A|$

$|C| \leq 2|C_{opt}|$ ☒

Set-Cover

Given a set X and a family of (possibly overlapping) subsets $S_1, S_2, \ldots, S_m \subseteq X$ such that $\bigcup_{i=1}^{m} S_i = X$, find $C \subseteq \{1, 2, \ldots m\}$ such that $\bigcup_{i \in C} S_i = X$, while minimizing $|C|$.



Approx_Set_Cover (on next page) selects S_1, S_4, S_5, S_3 in that order

Optimal: S_3, S_4, S_5

Approx_Set_Cover

$|X| = n$

$C = \phi$

While elements in X remain

  Pick largest S_i; $C = C \cup \{i\}$

  Remove all elements in S_i from X and other S_j

Return C

Poly time, returns a cover

Approx_Set_Cover is a $(\ln(n)+1)$-approximation algo

Proof: Assume there is a cover C_{opt} $|C_{opt}| = t$

Let X_k be set of elements in iteration k $(X_0 = X)$

$\forall k$, X_k can be covered by t sets.

$\Rightarrow$ one of them covers at least $\frac{|X_k|}{t}$ elements.

$\Rightarrow$ algo picks a set of (current) size $\geq \frac{|X_k|}{t}$

$\Rightarrow$ $\forall k \quad |X_{k+1}| \leq \left(1 - \frac{1}{t}\right)|X_k|$

More careful analysis (see CLRS, Ch 35) relates $e(n)$ to harmonic numbers. t should shrink!

Proof (contd.)

$$\Rightarrow \forall k,\ |X_{k+1}| \leq \left(1 - \frac{1}{t}\right) |X_k|$$

$$\Rightarrow \forall k,\ |X_k| \leq \left(1 - \frac{1}{t}\right)^k \cdot n \quad \leftarrow \text{elements in } X = X_0$$

$$\leq e^{-k/t} \cdot n$$

Algorithm terminates when $|X_k| < 1$, i.e., $|X_k| = 0$ and cost = k.

$$e^{-k/t} \cdot n < 1$$

$$e^{k/t} > n$$

When $\frac{k}{t} > \ln(n)$ and algorithm terminates.

$\therefore \frac{k}{t} \leq \ln(n) + 1$

So we have an $(\ln(n)+1)$ - approximation algorithm. ☒

Approximation ratio gets worse for larger problems.

PARTITION

Set S of n items with weights $s_1, \ldots s_n$

Assume $s_1 \geq s_2 \geq \cdots \geq s_n$ WLOG

Partition into A and B to minimize

$$\max\Big(\underbrace{\sum_{i \in A} s_i}_{w(A)}, \underbrace{\sum_{i \in B} s_i}_{w(B)}\Big)$$

Define $2L = \sum_{i=1}^{n} s_i = w(S)$

Optimum solution $\geq L$.

Note: 2-approx algo trivial.

Want a PTAS.

$(1+\varepsilon)$-approximation

(FPTAS also exist for this problem)

APPROX-PARTITION

Define $m = \left\lceil \frac{1}{\varepsilon} \right\rceil - 1$ $\qquad \varepsilon \approx \frac{1}{m+1}$

First phase: Find an optimal partition A', B' of $s_1, \ldots, s_m$ ← takes $O(2^m)$ time!

Second phase: $A \leftarrow A' \qquad B \leftarrow B'$

for $i = m+1$ to n

if $w(A) \leq w(B)$

$A = A \cup \{i\}$

else $B = B \cup \{i\}$

APPROX-PARTITION IS PTAS.

WLOG, assume $w(A) \geq w(B)$

approximation ratio $= \frac{w(A)}{L}$

k is the LAST item added to A.

Could have been added in first or second phase.

PROOF (contd.)

1) k is added to A in first phase.
This means $A = A'$. We have an optimal partition since we can't do better than $w(A')$ when we have $n \geq m$ items, and we know $w(A')$ is optimal for the m items.

2) k is added to A in second phase.
We know $w(A) - s_k \leq w(B)$
This is why k was added to A. (Note $w(B)$ may have increased after this addition to A).

$$\Rightarrow w(A) - s_k \leq 2L - w(A) \qquad w(A) + w(B) = 2L$$
$$\Rightarrow w(A) \leq L + \frac{s_k}{2}$$

Since $s_1 \geq s_2 \ldots \geq s_n$ we can say that
$s_1, s_2, \ldots s_m$ all $\geq s_k$

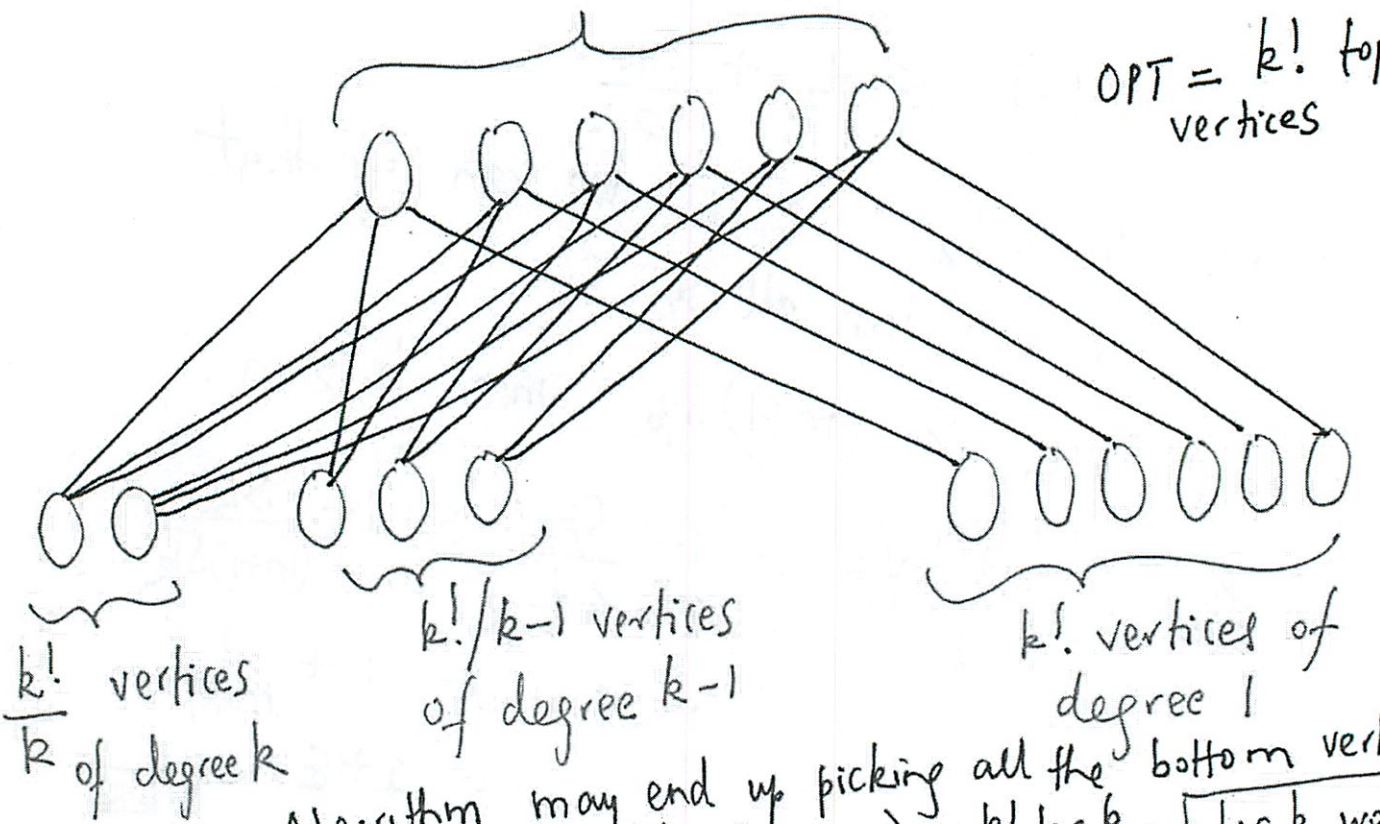
$$2L \geq (m+1)\, s_k \quad \text{since } k > m.$$

$$\frac{w(A)}{L} \leq \frac{L + s_k/2}{L} = 1 + \frac{s_k}{2L} \leq 1 + \frac{s_k}{(m+1)s_k}$$
$$= 1 + \frac{1}{m+1}$$
$$= 1 + \varepsilon. \qquad \boxtimes$$

Approx_Vertex_Cover_Natural

```
C ← ∅
E' ← E
while E' ≠ ∅
    pick v with maximum degree
    C = C ∪ {v}
    Remove v and all incident edges from E'
return C
```

A BAD EXAMPLE



Algorithm may end up picking all the bottom vertices

$SOL = k!\left(\frac{1}{k} + \frac{1}{k-1} + \cdots 1\right) \approx k! \log k.$ **log k worse**

$|G| = n$ (#edges) $\quad G \equiv G_0$

$G_0 \to G_1 \to G_2 \cdots G_m$ with vertex selection & edge deletion

$m = |C^*|$ #vertices in optimal vertex cover

Picking maximum degree vertex of G_{i-1}
→ call the degree d_i
→ delete edges incident on picked vertex to get G_i

$$|G_m| = |G_0| - \sum_{i=1}^{m} d_i$$

↗ #edges

Also, $\sum_{i=1}^{m} d_i \geq \sum_{i=1}^{m} \frac{|G_{i-1}|}{m}$

(because given $|G_{i-1}|$ edges can be covered by m vertices we know there is a vertex with degree at least $\frac{|G_{i-1}|}{m}$)

$$\geq \sum_{i=1}^{m} \frac{|G_m|}{m} \quad \text{since } |G_i| \leq |G_{i-1}|, \forall i$$

$$= |G_m|$$

$\Rightarrow |G_0| - |G_m| \geq |G_m|$ ⇒ After m iterations we have deleted half or more edges from G/G_0.

↳ smaller than $\sum_{i=1}^{m} d_i$ ⇒ $m \cdot \log|G|$ vertex cover. ☒

Approx Alg 2

11/8

6.046

Traveling Salesman w/ Δ-inequality

NP-hard

original NP complete

Special Cases

└have nice approx alg

Weighted vertex cover
- more general
- can't use what we did last time

Quiz 2 out at end

Last time 2 alg for Vertex Cover

- one looked better, but was theoretically worse

Traveling Salesman

Undirected graph $G(V, E)$ w/ non-neg integer cost $C(u, v)$ for each edge $(u, v) \in E$

Wants you to find Hamiltonian Cycle (tour) of G w/ min cost

- visits each vertex exactly once

(2)

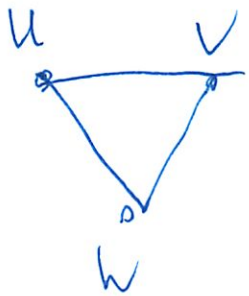
Generally can assume graph is complete

Can show NP-complete + NP-hard

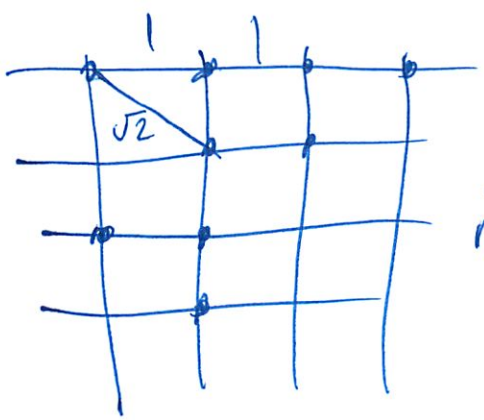
But we'll throw in some constraints

triangle inequality i.e. $c(u,w) \leq c(u,v) + c(v,w) \quad \forall u,v,w \in V$

△



If had natural way for this to arise



manhatten distance

topological version naturally causes △ inequality

Roads having tolls

Flights → direct causes more

There are many variants of TSP

Traveling Politian Go to 1 city in each state

Can get reduced to traveling salesman problem

Approximating TSP w/ Δ inequality

Strategy: Find ~~optimal~~ MST for G

Claim: MST for G has weight that is a lower bound on the lenght of the optimal tour.

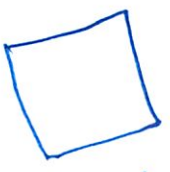
Pf. Is a cycle → goes back to started

Remove edge → is MST

Weight of tour has to $\leq$ since took edge away

if say $\underset{\text{cost}}{\text{tour}} \leq \underset{\text{cost}}{\text{MST}}$ → Contradiction!

Example



optimal tour

delete edge →

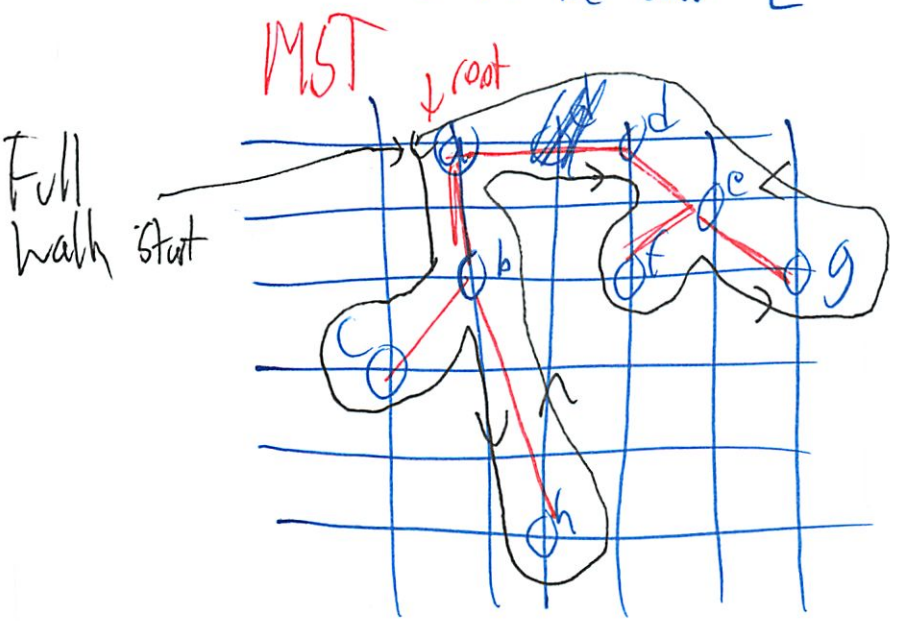
Spanning tree

(4)

If we can find a lower bound
that is 1st step for approx alg

Approx-TSP-tour

Select a vertex $r \in V$ to be root
Compute MST T for G from root r
Let list of vertices L visited in a preorder
tree walk of T (begining from r)
Return the hamiltonian cycle H that visits the
vertices in the order L



(5)

Pre-order walk

Depth 1st traversal of tree

Visits vertices in a certain order

Synthesize pre order walk from full walk

Full walk in block on previas pg

Order visited

a b c b h b a d e f e g e d a

Pre-order walk

Drop vertics you've seen a second time

Walk - not far

Drop multiple occurences

a b c h d e f g

(6)

Tour

return to a

→ a b c h d e f g)

Complexity

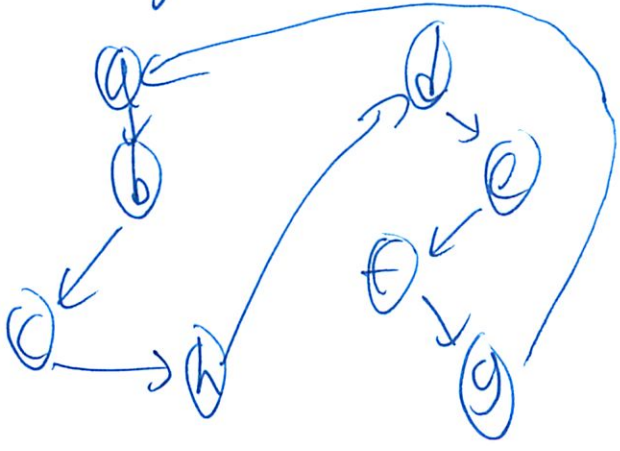
MST that tells us when first touching

"did he say linear"

Other than triangle → don't have distance

but as long as costs satisfy Triangle Ineq we're good

So returning a tour like this



(2)

Now must show is constant factor approx of actual tour
Since this is just an approximation

---

2 approximate algorithm

$$C(T) \leq C(H^*)$$

↑ Cost of optimal tour H^*

Full walk – vertices when they first visited
and when you return to them after full visit

W. Full walk → can't say visit everywhere ≥ 2
You never cross an edge more than twice
actually you cross each edge exactly twice
(good to realize!) ← traverse

$$C(W) = 2C(T)$$

So $C(W) \leq 2C(H^*)$

(8)

W is not a tour since visits same vertices more than once

So we shrunk w to get our pre-ordered walk

* this is because of our triangle inequality

Could not argue something had $\leq$ cost

$u \to \cancel{v} \to w$

$u \to w$

Need invariant to bound cost of pre order walk

This is what the triangle inequality does

So can delete from W all but the first visit to each vertex

→ preordered walk

→ make a tour H

So $C(H) \leq C(W)$ Thanks to triangle inequality

$\leq 2C(H^*)$

(don't see need to look at more)

Weighted Vertex Cover

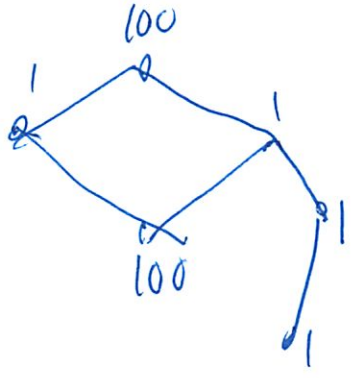
more general case of previous
Our approx from last lecture Do Not work however

Undirected graph $G(V, E)$ where each vertex $v \in V$ has an associated positive weight $w(v)$

$$w(V') = \sum_{v \in V'} w(v)$$

Find a vertex cover V' with minimum $w(V')$

Want to avoid 100-weight
Could be off big if picked wrong one



Not a simple greedy or arbitrary heuristic
↳ won't work

(10)

One way we can approximate: ILP

; but ILP is NP-hard...

Associate variable $x(v)$ w/ each vertex $v \in V$

Will require $x(v)$ to be $\{0, 1\}$ ← integer

$x(v) = 1 \rightarrow v$ is in vertex cover V'

$x(v) = 0 \rightarrow v$ is not "

Constraint

each edge (u, v) should have u or v in the vertex cover

$$\forall (u, v) \in E \quad x(u) + x(v) \geq 1$$

Minimize $\sum_{v \in V} w(v)\, x(v)$

↑ weighted sum of $x(v)$'s set to ↓

Matrix regarding Constraints

is it totally unimodular

What would it mean if matrix is TU

would get integer solution

(11)

What is implication → could solve in poly time

$P = NP$

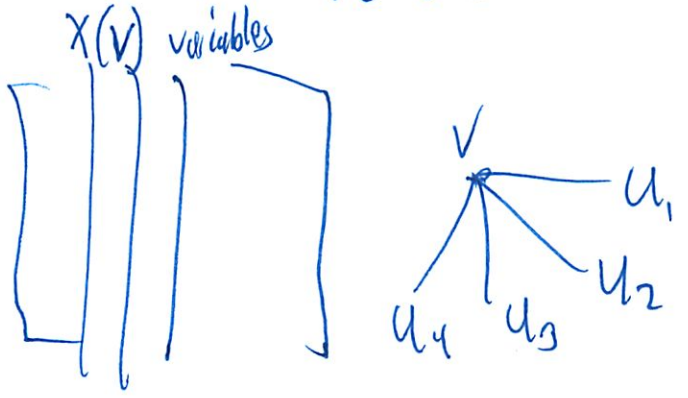
Weighted Vertex Cover → NP-hard

ILP → NP-hard

but if totally unimodular → would be poly time sol

So it must better not totally unimodular!

Could look and make sure it is not TU.



If look at constraints, could have many 1s in a column

Will violate sufficiency condition

Will get not TU matrices

Could set one up that is - but special case

(12)

We can't use TV

But we can use LP for approx sol

violate integer

but get close enough

LP relaxation

Minimize $\sum_{v \in V} w(v)\, x(v)$

Subject to $x(u) + x(v) \geq 1 \quad \forall (u,v) \in E$

$0 \leq x(v) \leq 1$ For each $v \in V$

Continuous variable

Could get fractional solutions

What is 1/2 of a vertex cover?

But if look at the value.

(13)

If 0 + 1s then value of cost fn
is optimal
lower bound - can't do better
But we're not likely to get 0s + 1s

Value of optimum LP is always lower bound for ILP
↳ for any ILP reduction

Approx-weighted VC

$C \leftarrow \emptyset$
Will get fractional values
Can round to 0 or 1
Will get vertex cover out of it
Correct sol in poly time
LP runs in poly
running in quadratic

(4)

$C \leftarrow \emptyset$

Compute $\bar{x}$ an opt sol to LP

For each $v \in V$

if $\bar{x}(v) \geq \frac{1}{2}$

$C = C \cup \{v\}$

How can we argue C is a vertex cover?

So $x(u) \geq \frac{1}{2}$ or $x(v) \geq \frac{1}{2}$

Or $x(u) = x(v) = \frac{1}{2}$

Can't have both $< \frac{1}{2}$

Rounding up gives us a cover

$x(u) + x(v) \geq 1$

$\hookrightarrow x(u) \geq \frac{1}{2}$

$x(v) \geq \frac{1}{2}$

$z^* \leq w(C^*)$

z^*: LP sol

Can round and get a vertex cover

$$z^* = \sum_{v \in V} w(v)\bar{x}(v)$$

$$\geq \sum_{\substack{v \in V \\ \bar{x}(v) \geq \frac{1}{2}}} w(v)\bar{x}(v)$$

I dropped a bunch of items
Those $\leq \frac{1}{2}$

$$\geq \frac{1}{2} \sum_{\substack{v \in V \\ \bar{x}(v) \geq \frac{1}{2}}} w(v)$$

That is our cover C

That is how we constructed our cover!

LP rounding is one of the most powerful ways to find approx alg

16

Randomized rounding is in the book
- a bit more sophisticated
- not covered in 6.046

- how to make guarantees on performance

(* is this optimum, hypothetical cover

Quiz

Rules + Strategy

not like any quiz you've done before

6 days

limited open book
- notes, CLRS
- no internet + cell phone
- no help

(17)

Same qv at level of in class quiz

Ctxes are not immediately obvious

Extrapolations of what we did in class

Must brainstorm

Can make assumptions + state them

L17

11/8 ①

6.046
L17

Approximation Algorithms II

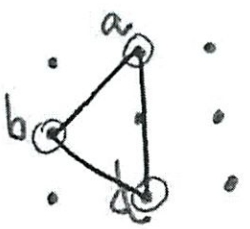
Traveling salesman with Δ-inequality
Weighted vertex cover

Traveling Salesman

Undirected graph $G(V, E)$ with non-negative integer cost $c(u,v)$ for each edge $(u,v) \in E$, and we need to find a hamiltonian cycle (tour) of G with minimum cost.

Triangle inequality: $c(u,w) \leq c(u,v) + c(v,w)$ for all $u, v, w \in V$

For example, cost of traveling between two vertices is the Euclidean distance between them satisfies Δ-inequality



$c(a,b) = \sqrt{2}$
$c(b,d) = \sqrt{2}$
$c(a,d) = 2 < 2\sqrt{2}$

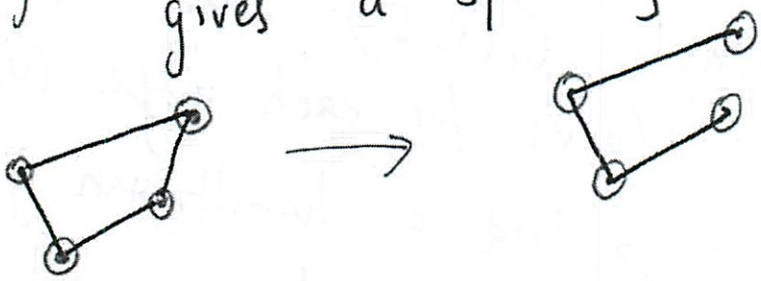
TSP still NP-complete even with Δ-inequality

Approximating TSP with Δ-inequality

Strategy: Find optimal minimum spanning tree (MST) for G

Claim: MST for G has weight that is lower bound on length of optimal TSP

Why? Deleting any edge from a tour gives a spanning tree.



Approx-TSP-Tour

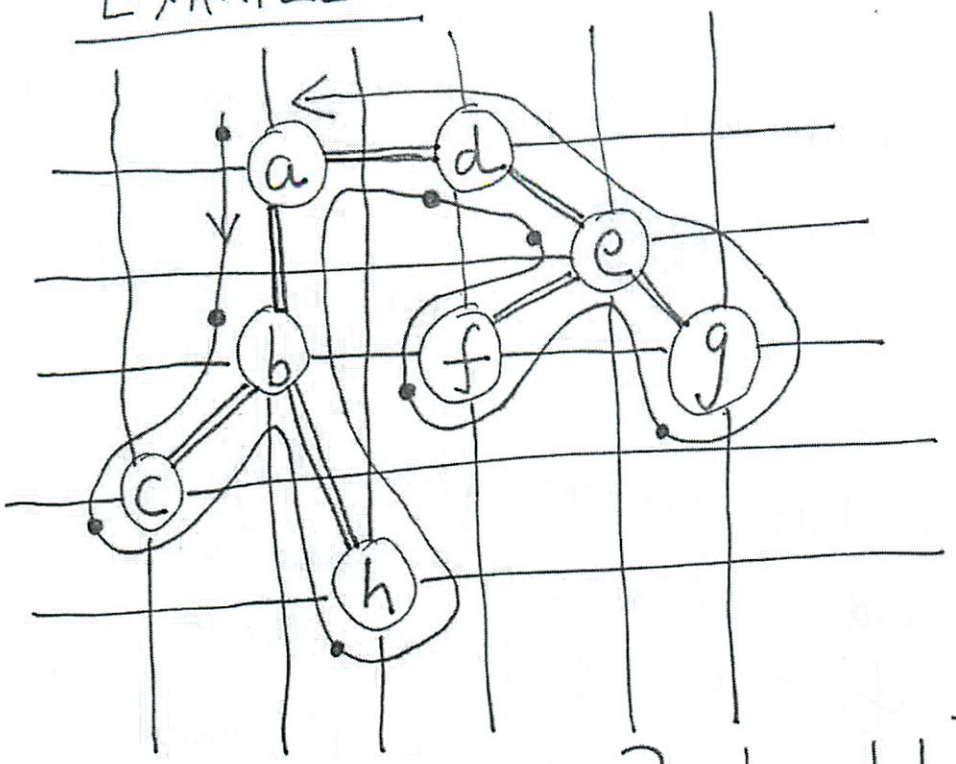
Select a vertex $r \in V$ to be root vertex

Compute MST T for G from root r

Let L be list of vertices visited in a preorder tree walk of T

Return the Hamiltonian cycle H that visits the vertices in the order L

EXAMPLE



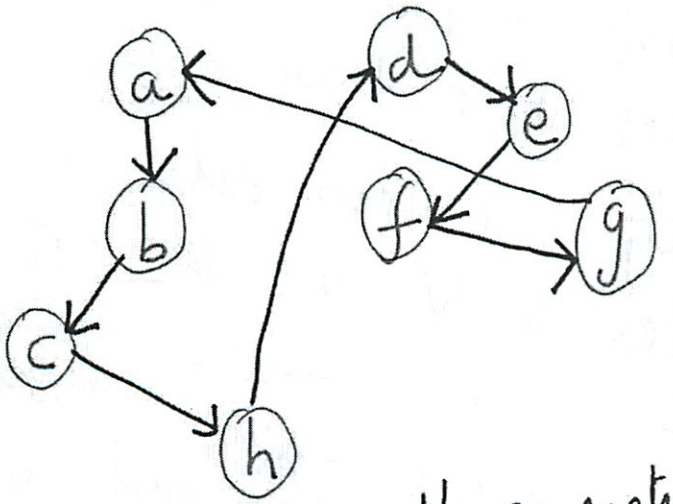
MST T shown
root a

preorder walk

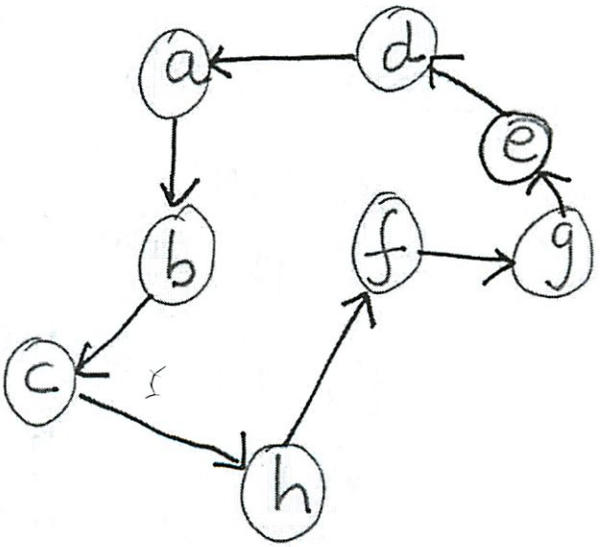
full walk:
a b c b h b a d e f e g e d a

{preorder walk derived from full walk} a b c h d e f g

preorder walk
a b c h d e f g



preorder walk generates approximately-optimal tour



optimal tour

APPROX-TSP-TOUR IS 2-APPROXIMATE

APPROX-TSP-TOUR runs in poly time & generates a tour

Proof: $c(T) \leq c(H^*)$ → cost of optimal tour H^*

Full walk of T lists vertices when they are first visited and also whenever they are returned to after a visit to a subtree.

Full walk traverses each edge of T exactly twice. Call Full Walk W.

$$c(W) = 2c(T)$$

$$\Rightarrow c(W) \leq 2c(H^*)$$

However W is not a tour since it visits some vertices more than once.

By the Δ-inequality we can delete a visit to any vertex in W and the cost will not increase. $u \to v \to w$ ⟩ no increase $u \to w$

Delete from W all but the first visit to each vertex → preorder walk, and make it a tour. H

$$c(H) \leq c(W) \leq 2c(H^*)$$ ☒

WEIGHTED VERTEX COVER

Undirected graph $G(V, E)$ where each vertex $v \in V$ has an associated positive weight $w(v)$.

$$w(V') = \sum_{v \in V'} w(v)$$

Find a vertex cover V' with minimum $w(V')$

Algorithm for unweighted vertex cover fails.

WEIGHTED COVER AS INTEGER-LINEAR PROGRAM (ILP)

Associate variable $x(v)$ with each vertex $v \in V$

Require $x(v) \in \{0, 1\}$ $\quad x(v) = 1 \Rightarrow v$ in vertex cover

$x(v) = 0 \Rightarrow$ not

Constraint: for each edge (u, v) at least one of u or v must be in the vertex cover.

$$\text{Minimize} \sum_{v \in V} w(v)\, x(v)$$

$$\text{subject to} \quad x(u) + x(v) \geq 1 \quad \text{for each } (u, v) \in E$$

$$x(v) \in \{0, 1\} \quad \text{for each } v \in V$$

ILP is NP-hard. Also the associated matrix is not totally unimodular (TU). can have lots of 1's in column for $x(u)$: one for each connecting edge.

LINEAR-PROGRAMMING RELAXATION

Minimize $\sum_{v \in V} w(v)\, x(v)$

Subject to $x(u) + x(v) \geq 1$ for each $(u,v) \in E$

$0 \leq x(v) \leq 1$ for each $v \in V$

Any feasible solution to ILP is a feasible solution for LP (but not vice versa).

∴ optimal solution for LP is a lower bound on the optimal solution to ILP, and here lower bound on weighted vertex cover.

APPROX-MIN-WEIGHT-VERTEX-COVER

$C \leftarrow \emptyset$

Compute $\bar{x}$, an optimal soln. to LP

for each $v \in V$

  if $\bar{x}(v) \geq 1/2$

    $C = C \cup \{v\}$

return C

APPROX-MIN-WEIGHT-VC IS 2-APPROXIMATE

LP is poly time and APPROX-MIN-WEIGHT is poly time

Proof: $z^* \leq w(C^*)$ → optimal vertex cover

z^* ↓ LP solution

Rounding up the fractional values produces a vertex cover.

$x(u) + x(v) \geq 1 \Rightarrow x(u) \geq 1/2$ or $x(v) \geq 1/2$

At least one of u or v will be included in the vertex cover, and every edge is covered.

$$z^* = \sum_{v \in V} w(v)\,\bar{x}(v)$$

$$\geq \sum_{v \in V : \bar{x}(v) \geq 1/2} w(v)\,\bar{x}(v) \geq \sum_{v \in V : \bar{x}(v) \geq 1/2} w(v) \cdot \frac{1}{2}$$

$$= \sum_{v \in C} w(v) \cdot \frac{1}{2}$$

(how C is created)

$$= \frac{1}{2} w(C).$$

Combining $w(C) \leq 2z^* \leq 2\,w(C^*)$

$\square$

Recitation

1/9

Canceled today

Lecture

11/13

No lecture today

Quiz 2
Cover Sheet

Problem	Title	Points	Grade	Initials
1	Red-Blue Subsets	15		
2	Broadcast Channeling	20		
3	Hashing	20		
4	Multicommodity Single-path Routing	25		
5	Minimum Spanning Tree	20		

Name: _____

INSTRUCTIONS

This take-home quiz contains 4 problems worth a total of 100 points. Your quiz solutions are due **at 11:59pm on Wednesday, November 14, 2012.** Late quizzes will not be accepted unless you obtain a Dean's support or make prior arrangements with the course staff. You must submit your solutions electronically.

Guide to this quiz: For problems that ask you to design an efficient algorithm for a certain problem, your goal is to find the *most efficient algorithm possible*. Generally, the faster your algorithm, the more points you receive. For two asymptotically equal bounds, worst-case bounds are better than expected or amortized bounds. The best possible solution will receive full points if well written, but ample partial credit will be given for correct solutions, especially if they are well written. Bonus points may be awarded for exceptionally efficient or elegant solutions.

Plan your time wisely. Do not overwork, and get enough sleep. Your very first step should be to write up the most obvious algorithm for every problem, even if it is exponential time, and then work on improving your solutions, writing up each improved algorithm as you obtain it. In this way, at all times, you have a complete quiz that you could hand in.

Policy on academic honesty: The rules for this take-home quiz are like those for an in-class quiz, except that you may take the quiz home with you. As during an in-class quiz, you may not communicate with any person except members of the 6.046 staff about any aspect of the quiz, even if you have already handed in your quiz solutions. To communicate with the staff, you have to send an email to `6046-tas@csail.mit.edu`. We will not respond to your question if you send an email to the personal email of a staff member. In addition, you may not discuss any aspect of the quiz with anyone except the course staff **until November 23, 2012**. Note that this date is **after** the due date of the quiz.

This take-home quiz is "limited open book." You may use your course notes, the CLRS textbook, and any of the materials posted on the course web page, but *no other sources whatsoever may be consulted.* For example, you may not use notes or solutions to problem sets, exams, etc. from other times that this course or other related courses have been taught. **You may not use any materials on the World-Wide Web, including OCW.** You probably won't find information in these other sources that will help directly with these problems, but you may not use them regardless.

If at any time you feel that you may have violated this policy, it is imperative that you contact the course staff immediately. If you have any questions about what resources may or may not be used during the quiz, please send email to `6046-staff@csail.mit.edu`.

Write-ups: Type your answers up on separate files (the templates will be given), and submit all your answers electronically, as you would do in the problem set. We *strongly encourage* you to submit your problems typed in LaTeX, since this makes it easier for us to read your solutions and understand them.

Your solutions are due, electronically, at 11:59pm on Wednesday the 14th of November. There is no hardcopy submission option for this test.

Your write-up for a problem should start with a topic paragraph that provides an executive summary of your solution. This executive summary should describe the problem you are solving, the techniques you use to solve it, any important assumptions you make, and the asymptotic bounds on the running time your algorithm achieves, including whether they are worst-case, expected, or amortized.

Write your solutions cleanly and concisely to maximize the chance that we understand them. When describing an algorithm, give an English description of the main idea of the algorithm. Adopt suitable notation. Use pseudocode if necessary to clarify your solution. Give examples, draw figures, and state invariants. A long-winded description of an algorithm's execution should not replace a succinct description of the algorithm itself. *Points will be taken off for overly convoluted solutions to the problems.*

Provide short and convincing arguments for the correctness of your solutions. Do not regurgitate material presented in class. Cite algorithms and theorems from CLRS, lecture, and recitation to simplify your solutions. Do not waste effort proving facts that can simply be cited.

Be explicit about running time and algorithms. For example, don't just say that you sort n numbers, state that you are using MERGE-SORT, which sorts the n numbers in $O(n \lg n)$ time in the worst case. If the problem contains multiple variables, analyze your algorithm in terms of all the variables, to the extent possible.

Part of the goal of this quiz is to test your engineering common sense. If you find that a question is unclear or ambiguous, make reasonable assumptions in order to solve the problem, and state clearly in your write-up what assumptions you have made. Be careful what you assume, however, because you will receive little credit if you make a strong assumption that renders a problem trivial.

Bugs: If you think that you've found a bug, send email to `6046-staff@csail.mit.edu`. Corrections and clarifications will be sent to the class via email and posted on the class website. Check your email and the class website daily to avoid missing potentially important announcements.

Good Luck!

Problem 1. Red-Blue Subsets [15 points] (1 parts)

Let S be a set of n elements $\{1, 2, \ldots, n\}$, and let F be a set of subsets of S. That is, for each $F_i \in F$, $F_i \subseteq S$ and $|F_i| \geq 2$. A Red-Blue coloring of S exists if each element in S can be assigned red or blue such that the set of red elements R and blue elements B satisfy:

1. R and B cover S.
2. R and B are disjoint.
3. Each subset $F_i \in F$ has at least one red and one blue element.

Prove that it is NP-complete using a reduction from 3-SAT to Red-Blue Subsets.

Problem 2. Broadcast Channel [15 points] (1 parts)

A set of up to n processors attempt to communicate over a network. The communication process is deemed successful if *any* of the processors manages to broadcast its information (since the successful processor can then lead the remainder of the communication process). However, the only means of communication is through a common broadcast channel. At any given time step (we assume the time is discrete), any subset of the processors can attempt to communicate through the channel by sending a message. The channel operates as follows:

- If *none* of the processors attempts to send a message, then all processors receive a special "none" message.
- If *only one* of the processors attempts to send a message, then all processors receive that message, and the communication process is deemed successful.
- If *two or more* processors attempt to send a message, then all processors receive a special "collision" message.

Suppose that the number of processors is at least $n/2$. Design a randomized protocol that, if followed by all processors, will result in successful communication in expected constant time, i.e., the expected number of time steps used by the protocol should be $O(1)$. Give an analysis of the expected number of time steps used by your protocol.

You can assume all processors know the upper bound n and the lower bound $n/2$on the total number of processors.

Problem 3. k**-Hashing** [20 points] (4 parts)

A k-hash function maps each key in a universe U of all possible keys to k possible slots in the given table. Suppose you have an access to a perfectly random k-hash functions, i.e. you can sample a function h_k such that for each key from U, h_k selects k slots from the table uniformly at random with replacement. Thus, when inserting a key into the table, you can choose to insert the key into the least-occupied slot out of k possible slots. When looking up the key, check all k possible slots. If k is constant, then the access time will be $O(1)$.

For a set of n keys from the universe U, we would like to have no collisions when inserting those keys into the table of size $O(n^{1+\epsilon})$, where $\epsilon > 0$ is some acceptably small constant.

(a) Show that for $\epsilon = .5$, we can achieve the goal with a good probability using a random 2-hash function. Particularly, show that when using a random 2-hash function to insert n items into the table of size $cn^{1.5}$, the probability of a collision is less than

$$\frac{1}{c^2}$$

(b) Give a better (smaller) upper bound on the probability of a collision when using a random 2-hash function to insert n items into a table of size $cn^{1.5}$

(c) We would like to have a smaller ϵ and a better chance of avoiding a collision. Show that it's possible by generalizing your result to k-hash functions. Find a function $\epsilon(k)$, monotonically decreasing in k, such that when using a k-hash function to insert n items into the table of size $cn^{1+\epsilon(k)}$, the probability of a collision is less than

$$\frac{1}{c^k}$$

(d) Conclude that you can use k-hash functions to achieve a perfect hashing. Compare it to the perfect hashing studied in class in terms of

- space usage: which one uses more space?
- use cases: give examples when one is better to use than the other;
- assumptions: how do the assumptions differ?

Problem 4. Multicommodity Single-path Routing [30 points] (2 parts)

We are given a graph $G(V, E)$ in which each edge $(u, v) \in E$ has nonnegative capacity $c(u, v) \geq 0$. We are also given a set of k different commodities, $K_1, ..., K_k$, and commodity i is specified by the triple $K_i = (s_i, t_i, d_i)$, where s_i and t_i are its source and sink, respectively, and d_i is the demand, i.e., the amount required to flow from s_i to t_i. Different commodities may share the same source or destination, but for each commodity, we assume that $s_i \neq t_i$. Moreover, each commodity i can only flow along a single path from s_i to t_i.A flow that satisifes these properties is called a *multicommodity single-path routing*.

(a) One way to model a flow network, as presented in CLRS 29.2, is to use *edge flows* $\{f_i(u, v)\}$ as decision variables, where $f_i(u, v)$ represents the flow each commodity i along the edge $(u, v) \in E$. However, we have an additional constraint that each commodity can only flow through a single path from its source s_i to t_i. Further, we are given additional constraints that the path with which to send commodity i cannot contain more than h_i edges.

The MIN-LARGEST-EDGE-LOAD problem then seeks the multicommodity single-path routing for which the largest *edge load* among all edges in the graph is minimized, where the edge load of $(u, v) \in E$ is defined as the total flow along (u, v).

Formulate MIN-LARGEST-EDGE-LOAD as an integer linear program (where some of the decision variables are constrained to be integers).

(b) Alternatively, as seen in Recitation 7, we can use *path flows* $\{f_i(p)\}$ as decision variables, where $f_i(p)$ represents the flow of commodity i along a path $p \in P_i$, and P_i is the set of all possible paths from s_i to t_i.

Show that for a multicommodity single-path routing, a set of edge flows can always be represented by a set of path flows, and the number of path flow variables required is polynomial with respect to $|V|$ and $|E|$. Do so by giving an algorithm that achieves this, and analyze its running time.

Hint: You may wish to review the algorithm for finding augmenting paths.

Problem 5. Linear time minimum spanning tree [20 points] (2 parts)

(a) Given a connected graph $G = (V, E)$ such that all weights on the edges are either 1 or 2, show that the optimal minimum spanning tree (MST) weight can be found in time linear in m (the number of edges). You may assume that $m > n$.

(b) Given a connected graph $G = (V, E)$ such that all edge weights are numbers in the range $[1/2, ..., 2]$. Show that there is an algorithm running in time linear in m that outputs a number b such that $MST(G) \leq b \leq 2MST(G)$ where $MST(G)$ is the value of the minimum spanning tree of G.

11/11
6PM

6.046 Fall 2012 Quiz 2
Q&A

General

Q: When the quiz instructs us to start our write-ups for each problem with an executive summary, does 'each problem' mean each of the five main problems, or each letter problem (e.g. 3.a.), or is it at our discretion?
A: Each letter problem! (Since some letter problems are largely unrelated.)

Q: For the take-home exam, what are the restrictions on space efficiency for our algorithms? I realize exponential-time space usage is probably a non-optimal solution, but are there restrictions, and should we be focused on finding optimal-space solutions in addition to optimal-time?
A: Try to give the solution with the best time and memory space performance you can come up with. Focus on making running time optimal; the memory efficiency usually follows closely. For example, your solutions shouldn't require exponential memory space-- that would take exponential time to construct anyway.

Q: I noticed the pset solutions were really long and detailed. For example, when showing a problem was in NP, it talked about counting the number of bits in the certificate, etc. Do we need to do this for quiz 2?
A: Yes, you should give detailed solutions like in the pset. We don't expect that in in-class quizzes, but we do for the take-home.

Q: Is it okay for me to use a graphing program to look at the shape of a function? What about writing code to get a numerical idea of how a function grows?
A: It is fine.

Q: Are the approximation topics we covered during the last two lectures fair game for this exam?
A: Yes, you can make use of the approximation topics covered in the last two lectures.

Problem 1

Q: In problem 1 on the quiz, when determining whether a given process can run in polynomial time, can we consider |F| (the number of subsets in F for a given Red-Blue Subsets problem instance) to a basic runtime variable in the same way that n is? Or do we have to treat |F| using an upper bound defined in terms of n?
A: You can treat it as part of your runtime variable.

Q: For a Red Blue subset situation, are you given an F set and an S set, or are you given only an S and a coloring may exist only if you can also show that it must be possible to construct an F that satisfies the given conditions?
A: You are given both a F set and a S set.

Q: From Problem 1, Could you please clarify what is meant by R and B cover S? Does this mean R U B = S?
A: Yes, if R and B cover S, then the union of R and B is S.

Q: Does "R and B are disjoint" mean that if an element is colored Red, it can't be colored Blue at the same time?
A: Yes.

Q: Definition of 3-SAT is not really mentioned in class materials. Is it really 3-CNFSAT?
A: Yes.

Q: The book states the Ackermann-Function(n) <= 4 for all practical purposes. Does this imply that I can assume that Kruskal's algorithm takes O(m) if the edges are passed in non-decreasing order, and no sorting is done in it?
A: The Ackermann function is slowly growing, but it is not bounded by a constant.

Problem 2

Q: I'm having trouble understanding what it means when we have "a set of up to n processors" and "the number of processors is at least n/2." So does this mean that given an input "n" in the Turing machine sense, we could have anywhere from n/2 processors to n processors?
A: There are *up to n* processors. That is, n/2 <= # processors <= n.
You can assume all processors know the upper bound *n* and the lower bound *n/2* on the total number of processors.

Q: The first sentence says that there are n processors, but the first sentence of the next paragraph says to suppose that there are at least n/2 processors. Don't we know the exact number of processors? Why are we given these upper and lower bounds? Or do these bounds simply refer to the number of processors that could be wanting to communicate at a given time step?
A:You can think of *n* as the number of processors networked together and *n/2* as a lower bound on how many computers want to communicate at once.

Q: Is there no limitation on the content of the message?
A: No limitations.

Q: Does O(1) expected runtime mean that for any time step, when any processor wants to communicate, it can do so in O(1) time? or does this mean that they can simply take turns? I'm asking if it is valid just to have processors wait to send to avoid collisions.

A: The entire process halts when *the first* processor successfully sends its message (not when all processors successfully send a message. The O(1) expected runtime is the expected run time for *any* processor to get its message successfully sent. Once 1 machine successfully sends a message, you could consider the machine elected as the "leader" and it can help coordinate from there.

Q: What causes a processor to need to send a message? question reads: "any subset of the processors can attempt to send a message." Does this mean it is possible for such a subset to attempt a message, or that randomly any such subset will at any time step request (and we have no control)?
A: The processors can store state. I'm not entirely sure how to interpret your question, but in any given time step, any subset of processors are allowed to attempt to send a message.

Q: Are the three messages that can be sent sent at every time step?
A: You have to design a protocol that determines what messages are actually sent by each of the processors. (Note that the None and Collision messages are not sent by individual processors, but may be received by the processors.)

Problem 3

Q: I just had a quick question about problem 3. It states that h_k selects k slots from the table uniformly at random with replacement. Am I correct in understanding that this could mean that the same slot is chosen multiple times? So for example, if we had k = 2, then it could be that a certain call to h_2 returned slot i twice, in which case we would be forced to insert the element into slot i. Is my reasoning here correct?
A: Yes. You're right.

Problem 4

Q: In problem 4 part a, it says that to formulate an integer linear program, where some of the decision variables are constrained to be integers. Does this mean that, in our formulation, we can have some decision variables constrained to integers, and some (the edge flows) not constrained, or does it mean that formulating the multicommodity single-path routing problem this way constraints that problem's decision variables (the edge flows) to be integers?
A: An integer linear program is a linear program that allows for the type of constraints that say a decision variable must be an integer. You can certainly make your own decision about whether or not you want all the flows to be constrained this way, based on requirements specified in the problem.

Q: In part b of problem 4 on the quiz, when it says to show that the number of path flow variables required is polynomial in |V| and |E|, does that specifically mean that those variable must represent path flows, or just that those variables are the decision variables which are used to find the path flows?

A: I'm not sure what you mean by "decision variables which are used to find the path flows". This problem asks you to show that you can use a polynomial number of path flow variables to represent a set of edge flow variables. Please refer to Recitation 7 f you are unclear about how to represent path flows.

Q: Does "polynomial in |V| and |E|" mean that k, the number of commodities, cannot appear in the number-of-variables expression?
A: Sure, k can appear in the expression, since it is included as the input. It is simply asking for something polynomial with respect to the input size, and specifically highlighting |V| and |E| as the size of the graph G(V,E).

Q: Part b of problem 4 of the quiz deals with representing a flow using path flow variables instead of edge flows variables. Is such a representation restricted to using only paths with are strictly present in the flow network, or would a representation using, say, augmenting path flows qualify?
A: Yes, it is restricted to existing paths. As already mentioned in the problem statement, please refer to the section on max-flow min-cut in Recitation 7.
Augmenting paths are defined on the residual network, not on the original network.

Q: For part a of question 4: are the d_i values necessarily integral?
A: No. The problem statement doesn't say that either.

Q: Am I allowed to define additional decision variables besides the edge flow variables? If so, is there a bound on the number of decision variables? Is it necessary to find the ILP formulation that minimizes the total number of equations or variables, or will any formulation that correctly expresses the problem suffice?
A: If you need to define additional decision variables you can, but you should try to keep the ILP formulation as simple as possible. As with other problems, we look for simplicity and clarity in the solution.

Problem 5

Q: For question 5: Does the running time depend on *m* only? do we consider *mn* linear in m? Any solution to find an MST has to depend on the number of vertices.
A: mn is not linear in m, n + m would be linear. But mn would be linear in n and m.

Q: For Problem 5, what is "n" in part (a)? I am assuming that it is the size of V.
A: Yes. Generally, you can write down at the top of your solution what assumptions (like n = |V|) you are making.

Q: For Problem 5, when you say "time linear in m", do you mean O(m)? Or would O(mn^2) also be linear in m?
A: I'll say this: you want your algorithm to run as fast as possible.

QnA 2

11/12 Middy

6.046 Fall 2012 Quiz 2
Q&A No. 2

General

Q: Would it be ok to directly copy phrases from CLRS in our solution write-up? For example, is it alright to copy the way CLRS words an instance of 3-SAT?
A: Yes, this is fine. You can also just reference material in the book without copying it all down.

Problem 1

Q: Are all F_{i}'s unique, as in, can there not be overlap between various F_{i}'s, or can there be?
A: All the F_{i}s can be *any* subset of S. If you have an element *j* in S, it can appear in any number of F_{i}s. You could potentially have two subset F_{i}s that are the same and could have that F_{i1} is a subset of F_{i2} even.

Q: Is |F| the number of subsets F_{i}s it has or the total number of elements it has, i.e. |F| = |F_1| + ... + |F_m|?
A: |F| is the number of subsets F_{i}.

Problem 2

Q: What do we know about each processor. For instance, can they instantiate random number generated, which is fitted into their cache (and thus don't share it outside)?
A: Yes, you can assume that the processor can generate random numbers for any given distribution. Other processors may see the processor's messages but do not know its internal random numbers. You can, in your solution, specify the protocol for each processor.

Q: If a processor tries to send a message at time step T, and a collision occurs, do we know for sure that the processor will reattempt to send the same exact message again on time step T+1?
A: No. It depends on your protocol. You should specify the behavior of the processors at each timestep.

Q: Can a processor be both sending and receiving a message in a single time step? Furthermore, are they able to use the information received in time step k to determine whether or not they choose to send a message in time step k, or do they have to wait until time step k + 1 to utilize that information?
A: You can assume that some processors send messages at the beginning of time step and receive messages (a real message, None, or Collision) at the end of the time step.

Problem 3

Q: There's a small typo in the statement of problem 3, "a perfectly random k-hash functions". Does it mean to say that we have access to 'a collection' of perfectly random k-hash functions, just like universal hashing involves a collection of hash functions, from which we can sample?
A: Yes. You can think of it as a collection of functions from which you can sample a function h_k and expect it to behave like a perfectly random hash function (with memory).

Q: Am I allowed to combine parts a and b if I think it will be more effective and still be clear enough?
A: It's OK to combine 3-a and 3-b, just make sure to state that clearly.
However, note that if your solution is incorrect or has some mistakes, we'll subtract points for both parts (a) and (b).

Q: Does our bound for 3b have to be only in terms of constants and c or can it also be in terms of n?
A: First, derive it in terms of n and then give the upper bound for all n, i.e. do both.

Q: When you say "you have an access to a perfectly random k-hash functions, i.e. you can sample a function h_k such that for each key from U, h_k selects k slots from the table uniformly at random with replacement", do you mean that for a given hash function h_k, every time you call it it chooses a different k slots to insert into?
A: First of all, when dealing with hash functions, the chosen hash function has to be deterministic
(so that we know where to look when accessing the items later). Thus, when we talk about probability (or randomness), we always mean the probability over possible choices of hash functions.
To make things easier, assume that you can sample a function h_k which behaves like a perfectly random hash function, i.e. for a given key, h_k selects k slots from the table uniformly at random with replacement.This means that for a given key, it selects k slots with possible repetitions (i.e. up to k distinct slots).
For the same key, it always selects the same slots so that later on we can access the item.

Q: On part d, when we "Conclude that you can use k-hash functions to achieve a perfect hashing", do you expect a mathematical proof that k-hash functions can fit the definition of perfect hashing (requiring O(1) memory accesses for a search in the worst case)?
A: Just describe in words how would you use k-hash functions to achieve a perfect hashing. Notice that we didn't say that you know all n keys in advance. How would you choose constants c and k?

Q: I have question on the k-hash function. First, when you define what h_k does, you say: "h_k selects k slots from the table uniformly at random *with replacement*". I was not sure how "with replacement" added meaning to the definition because the following sentence says that when inserting a key to the table we can choose the least-occupied slot out of k possible slots. So,

say if we feed a key to h_k, would h_k return a list of indexes and then we can choose which index to choose, depending on which is the least occupied? Furthermore, since its a k-hash function it always returns a list of size k?
A: With replacement means that some of the chosen slots could be the same. So you can think of it as up to k distinct slots or k slots with possible repetition (since k slots were chosen independently from each other).

Problem 4

Q: For Problem 4a, can we assume that a multicommodity single-path routing exists in the graph (or that multiple exist, and our goal is to find the one with the minimum largest edge load)? Or is our integer linear program responsible for determining if a multicommodity single-path routing exists in the first place?
A: The ILP you formulate should determine that.

Q: For 4b, we're supposed to "show that you can use a polynomial number of path flow variables to represent a set of edge flow variables" by giving an algorithm which converts a set of edge flow variables into path flow variables. Is that right?
A: Yes.

Q: Do we have to give the formulation in standard form?
A: No, it does not need to be in standard form. We are looking for clear and concise formulations.

IMPORTANT
Q: When problem 4a says that {f_i (u,v)} are the decision variables, does that also mean that they have to be the variables in our ILP?
A: You can define your own decision variables. This problem asks for an ILP formulation similar to the modeling approach that uses edge flows {f_i (u,v)}, but the decision variables may not necessarily be the actual edge flows {f_i(u,v)}. (The sentence "However..." in the first paragraph highlights some ways in which the ILP formulation differs from how a flow network is modeled in CLRS 29.2.)

Q: Even though Problem 4 which has a title Multicommodity Single-Path Routing, part (b) has nothing to do with single-path routing but instead is asking about the version on CLRS on page 862, i.e. the multicommodity flow problem. I was wondering if that was the intention or if the question was still rooted in single path routing?
A: That is correct-- 4(b) is independent of single-path routing, and should work for general multicommodity flows. (It was originally designed to precede more parts that would use path flows to formulate other multicommodity single-path flow problems, but we eventually decided to leave those parts out from this quiz so it won't be too long.)

Problem 5

Q: Are we only asked to return the weight of the optimal MST, not the actual MST?
A: Yes.

Q: Is the "value" of an MST the same thing as the "weight" of the MST?
A: Yes, the value of an MST is the total weights of edges.

Q: Does "numbers in the range [1/2,...,2] mean numbers in the set {.5, 1, 1.5, 2}? Or numbers between, and including, .5 and 2?
A: The second interpretation is correct.

Q: Does the m < n hold for part b) or not?
A: Actually, the assumption is m>n. It's just saying that you can assume that there are at least as many edges as there are nodes. You can assume that.

Michael E Plasmeier

11/13
8:30P

From: Srini Devadas <devadas@MIT.EDU>
Sent: Tuesday, November 13, 2012 8:34 PM
To: Srinivas Devadas
Subject: 6.046/18.410 announcement: Quiz 2: Problem 5(a) clarification

Note: This mail was sent to all students in the stellar class Design and Analysis of Algorithms

Quiz 2: Problem 5(a) clarification

Please note that for Problem 5(a), you should provide an algorithm that is linear in n (the number of vertices) and m (the number of edges).

This announcement was made in Stellar on 2012 November 13 by Srini Devadas

The announcement is also posted on the class website:
https://stellar.mit.edu/S/course/6/fa12/6.046J/index.html

Quiz 2

11/11
8P

1. Exponential alg
2. Then improve as you go
3. Write up
   inc Exec summary

---

1. Red-Blue Subsets

(is this like Red-Black trees?)

Set S $\{1, 2, \ldots n\}$

F be a set of subsets of S

For each $F_i \in F$, $F_i \subseteq S$, and $|F_i| \geq 2$

($\subseteq$ is a subset of, $|\ |$ size of)

A Red-Blue coloring of S exists if each element in S can be assigned red or blue s.t. R, B satisfies

2)

1. R and B cover S

means:
1 for 1?

2. R and B are disjoint

Well this would say 1 for 1

Every element must be in either R or B

3. Each subset $F_i \in F$ has at least one Red and 1 blue

¿Can't we just build subsets of 1 B, 1R

¿is F given (exogenous) - seems yes

↓ does not matter anyway

Prove NP-complete using reduction from 3SAT to Red Blue

(3)

3SAT ∠ Red Blue

1. Red Blue in NP

Since Can verify

Can check each condition

The P-set 4-1 way is much more formal...
Uses bits
but this problem does not use bits.

1. Check each F_i so $|\cup F_i| \geq 2$, one B_i does

F_i's are not disjoint.

2. Check each el in S is assigned to either Red or Blue

depends how mapping stored

Can cross off list in $O(n)$

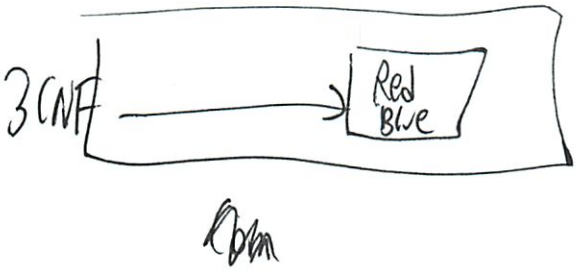
(9) QnA

F is explicitly given

Case $R \cup B = S$

3SAT = 3CNF SAT

2.a.



a. How to reduce

b. If $(S, t) \in SS$, there exists $S' \subseteq S$ s.t.

c) Suppose have event partition

$W' = \{S_{total} - 2t\}$ is a subset of S whose sum is t

Showing $(S, t) \in SS$ iff $W \in E\text{-}D$

P-Set ans 4-1a

(5)

Anyway worry about that later.

Now try to Find a way to reduce

3 CNF SAT to Red-Black

3CNF SAT

bool vars

$\wedge$ $\vee$

Can we get it to work?

Will it satisfy

Use EO to solve S-S

So its not use SAT to solve R-B

But using R-B to solve SAT

We assume we have a magic Black box for RB

Use that black box for RD to solve SAT

(6)

3CNF SAT is limited

$(x_1 \vee \bar{x}_1 \vee x_2) \wedge \ldots$

literal ∈ exactly 3

clause

$\wedge$ = AND

$\vee$ = OR

OR $\vee$ $\wedge$

A	B	OR	AND
T	T	T	T
T	F	T	F
F	T	T	F
F	F	F	F

Now how in all world will we map that onto this

(7)

Each F_i is a clause

R is T
B is F
) makes sense one or other

Each X_i is an an element (n total)

So lets try

$$(X_1 \vee \bar{X}_4 \vee \bar{X}_2) \wedge (X_3 \vee X_2 \vee X_4) \wedge (\bar{X}_1 \vee \bar{X}_3 \vee \bar{X}_4)$$

So X_1 is T or F or X_2 is B
~~X22~~ R or B

But does not mean must be at least 1 true in here

We could have all T in a ~~the~~ clause

FAQ mentions kruskal min spanning tree

"Could have multi stage"

So Clique, Vertex Cover, Subset sum

(8)

Subset sum we used previously

But how will this help us on 3SAT?

We could also SS & Red Black

Use RB as a way to Solve SS

Can #s add up to t

R = items that can add up to t
B = items that don't

But F is given exogenously
Is but through our black box...

then what is F?

Go back to pure CNFSAT

(9)

I can't think of anything besides F_i is a clause?

Could be presence in subset

But that does not make sense that only 2

SS variable + negation = auto true

each variable in at least 1 clause

two # in S for each x_i

in S C_j

t has 1 for each variable

4 Clause

if x appears in C_j, then digit v_i has a 1

$\overline{x}$ v_i' 1

All v_i v_i' unique

(10)

For each clause C_j has s_j s_j'

lables a digits

So adds to 4

Greatest sum in a digit pos is 6

So no carries

This is writing CNF so can solve w/ SS

But this is all very complex!

That got me no Fifth

---

is R = True

B = False

Well remember we don't know

Can The input is these literal + clauses

We mustry each possible assignment

(11)

Such that each clause is true
by at least 1 literal
between 1 -3 literals will be true

~~So we assign~~ 1

S colors
F subsets fixed? or black box?

I think black box

Sat if a mapping can occur
if 3SAT => will be a mapping

if we could assign at least 1 red el
but then no guarentee on blue
Unless F is all the ones that are same literal

(12)

And the literals are R, 9
But some could be mentioned or were
Unless we add fake ones
2 that seems to defeat the point

How use fact are 3 literals in a clause
Reading SAT ∠ 3SAT

$$\overline{A \wedge B} = \overline{A} \vee \overline{B}$$
and or

$$\overline{A \vee B} = \overline{A} \wedge \overline{B}$$
or and

What is ⊖

appears to be equiv to
gets a 1 when equiv to it

Oh I see what they did...

(13)

So where does that put us? — no closer!

Just more general understanding

Usually on P-Sets I've never been at a complete loss!

Reduction such that

If ~~matching~~ R-B is valid then 3-SAT is valid.

SAT means 1-3 literals in a clause valid

all clauses valid

So must be each clause is F;

each n element is a literal II always reverse order

Or how about our 3 ~~R~~ plus a B

So then to be SAT must be

at least 1 ~~R~~ R

~~If X: present~~ :)

(14)

So then a x_i is a R

not a $\overline{x_i}$ is a R

So reduction such that

$x_1 \vee \overline{x_2} \vee x_3$

if true R if $\overline{x_2}$ true R $\overline{x_2}$ false B

x_2 true B x_2 false R

So can say if x_2 false is a R

* Remember Mapping 3SAT to RB

So we can solve w/ our RB blackbox

~~Just say if~~ Given S (our literals)
Given F (our clauses)

Sat if we can satisfy and have our R-B coloring since each literal has at least one true (R)

(15)

But say write a R is true x_1
B if must be false $\overline{x_1}$

and always add a spare B

Then must prove 3SAT → R-B works
+the nominal
R-B → 3SAT
That we can do reverse
✓ ~~do later~~

Suppose we have a R-B arrangement,
Then we have a 3 SAT since
R V R V B VB 4-SAT
false
so remove
Then left w/ 3 sat
if has at least 1 R - ~~which it will~~ Then
it is 3SAT - must be true for each clause

(16)

11/14
12:15A

(write up)

I think you never *write it* as R, B

Need to clarify that as I write it up

Or do we always write the input in a fixed way?

How link literals w/ the same value?

Well really we can flip literals X_1 to either ~~yes or no~~ true or false

Or that is internal behind the scenes

X_1 is R and $\bar{X}_1$ is B

or

X_1 is B and $\bar{X}_1$ is R

No - X_1 true must be Red

but X_1 can be false

(17)

I think I need to think more about when
its mapped and possibly clarity in the ass

11/4
10:30

2. Broadcast Channel

up to n processors communicate over a network

Sucessful if any processors manages to broadcast info

Common broadcast channel

discrete time

Chanel
- none speak → "have" message
- One speak → transmitted
- two " → "collison"

$\frac{n}{2}$ processors

Randomized protocol

Want constant time $O(1)$ ← what does this mean?

all processors know $\frac{1}{2}n$ upper bound sucessful comm

$\frac{1}{2}n$ lower "

: but not exact #?

(2)

¿is this pure Ethernet contention

~~What~~ what time is that? O(1)?

or a more restricted / special case?

OnA

$n/2 \leq$ # processors $\leq n$

no limitations on message content

¿only 1 time spot?

O(1)?

entire process halts when first processor

sends a message

then it is the leader + can coordinate

that doesn't really ans qu

~~other than~~

~~Well~~

must it succeed on 1st try

well really ... just shouldn't depend on # of processors...

(3)

So processor # 1 sends
↳ don't have unique #

this is like approx alg
~~On 1st try they he~~
each sends w/ prob $\frac{1}{n}$

Well worst and best don't matter much
but on avg 1 to 1 every 2 time steps one sends
every
So more just to show prob collision is O(1)
(does not change w/ # of machines)

This is studying approx algs

(BD) (9)

11/14
1:40A

How do ya get O(1) 1 processor to speak
They can't all have a #

Not Approx
But randomized

QnA 2

Can generate random #s
but can't share

Lecture 4 + 5 Randomized
from Unit 1 though
~~Matching is the only thing we haven't covered~~
└ we did w/ max flow

Each sends w/ prob $\frac{1}{n}$
So $E[\,] = 1$ transmission

(5)

$P(\text{2 transmit collision}) = \frac{2}{n}$

$\left(\frac{1}{n}\right)^2$

↑ two pick exact same tf well the transmit

$P(\text{collission twice in a row}) = \left(\frac{1}{n}\right)^2 \cdot \left(\frac{1}{n}\right)^2$

$\left(\frac{1}{n}\right)^4$

Expected value

$$E[X] = \sum_x x \cdot P(X = x)$$

ie. flip coin heads → +3, tails → −2

$$E[X] = 3 \cdot \frac{1}{2} + (-2) \cdot \frac{1}{2}$$

$$= \frac{1}{2}$$

↑ expected to gain 50¢ each flip

(6)

¿Can we apply EV here?

expected # of collisions

$P(1 \text{ collision}) = \left(\frac{1}{n}\right)^2$

$P(2\ \ ) = \left(\frac{1}{n}\right)^4$

$P(3\ \ ) = \left(\frac{1}{n}\right)^6$

So $\sum_{i=1}^{\infty} \left(\frac{1}{n}\right)^{2i}$

$= \frac{1}{n^2-1}$

Say $2n = 100$ ¿or math error

Since its $\underbrace{\frac{1}{3} + \frac{1}{8} + \frac{1}{15} + \frac{1}{24} + \frac{1}{35}}_{\frac{221}{420}} + \cdots$

So $P(0 \text{ collisions}) = 1 - \text{That}$

$= \frac{199}{420}$ which is good

(1)

$$E[\#\text{ collisions}] = 0 \cdot \frac{199}{420} + 1 \cdot \left(\frac{1}{n}\right)^4 + \cdots$$

well really the qu is does it depend on n?

$$E[\text{collisions}) = 1\left(\frac{1}{n}\right)^2 + 2\left(\frac{1}{n}\right)^4 + \cdots$$

$$= \sum_{i=1}^{\infty} i \left(\frac{1}{n}\right)^{2i}$$

As n gets bigger it gets smaller ....

This is worst case → all n processors

will argue since E[# steps] ↓ as n ↑, its constant

(write)

Is this right?

$$P(\text{any collision}) \quad \sum_{i=1}^{n} \left(\frac{1}{n}\right)^i$$

$$\text{is} \quad \frac{1}{n-1}$$

(8)

No lets say i=2

I think my prob is wrong

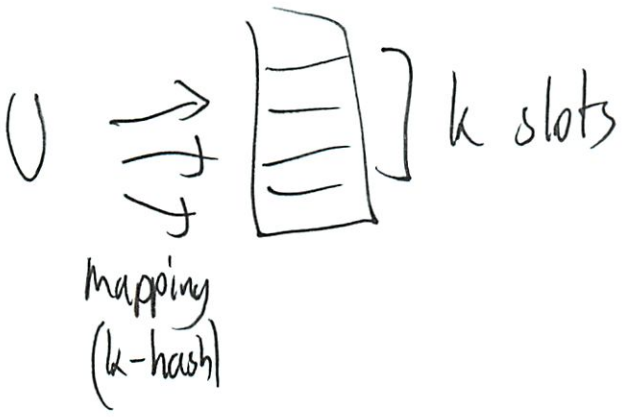
need to review 6.04

and closed sums 6.042

11/1
10:50

3. k-Hashing

k-hash function maps each key in a universe U of all possible keys to k possible slots



Suppose perfectly random k-hash

ie. Can sample a random h_k s.t. for each key from U, h_k selects k slots uniformly at random

Ohhh k is $<<$ size of table

NOT k = size of table!

(2)

So selects k slots

Then we choose least occupied slots

When looking up, check all k possible slots

If k is constant, access time is O(1)

Reduces to constant

For a set of n keys from U, we would like to have no collisions while inserting those keys into table of size $O(n^{1+\epsilon})$

? Small constant $\epsilon > 0$

~~Same~~ So randomized

Or trying to pick size of each k

So no collisions

What is a collision?

(3)

a) Show that for every $\epsilon = .5$ we can achieve the goal w/ a good prob using random $k=2$-hash fn

Show that when using a random 2-has fn to insert n items into table size $cn^{1.5}$ prob collision $< \frac{1}{c2}$

This is just a show qu ...

11/12
12:15ish

But how do we do it?

So we randomly pick 2

~~Then pick the one w/~~

P(that both don't work)

Table size $cn^{1.5}$

"How full is the current table"

Wait is table size $cn^{1.5}$ or $O(n^{1.5})$

I see - same

(4)

So $c=1$ $\quad n^{1.5}$ $\quad p \leq \frac{1}{1^2} = 1$

2 $\quad 2n^{1.5}$ $\quad p \leq \frac{1}{4}$

3 $\quad 3n^{1.5}$ $\quad p \leq \frac{1}{8}$

As table gets bigger, $p <$

"How do we prove this" induction

Open addressing $\quad P(A_i) = \frac{n}{m}$

P jth prob is to occupied is $\frac{(n-j+1)}{(m-j+1)}$

Since can find $(n-(i-1))$ els in $(m-(j-1))$ unexamined slots

$$P(X \geq i) = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdot \dots \frac{n-i+2}{m-i+2}$$

$$\leq \left(\frac{n}{m}\right)^{i-1}$$

$$= \alpha^{i-1}$$

(6)

$$E[x] = \sum_{i=1}^{\infty} P(x \geq i)$$

$$\leq \sum \alpha^{i-1}$$

$$= \sum \alpha^{i}$$

$$= \frac{1}{1-\alpha}$$

if α is constant $\rightarrow$ then $O(1)$

50% full $\rightarrow$ 2

90% full $\rightarrow$ 10

What does that tell us here?

We don't know how full the table is

$P(\text{collision normally}) = \frac{1}{m}$ for n items, m slots, last item

$= \frac{n}{m}$ $\$$/b

So if 6 slots, w/ 2 full

$P(\text{collision}) = \frac{2}{6} = \frac{1}{3}$

(6)

P(Select 2) then only chance that both have collision

$$\frac{n}{m} \cdot \frac{n}{m} = \left(\frac{n}{m}\right)^2$$

So again

$$\frac{2}{6} \cdot \frac{2}{6}$$

↑ could select same one

? with replacement
? think that means

$$\frac{4}{\text{Out of } 36} = \frac{1}{9}$$ shot ya get both

Since $\frac{2}{6} \cdot \frac{4}{6} + \frac{4}{6} \cdot \frac{2}{6}$ shot you pick one

$$\frac{16}{36} = \frac{4}{9}$$

$\frac{4}{6} \cdot \frac{4}{6}$ pick both good

$$\frac{16}{36} = \frac{4}{9}$$

So $\frac{1}{9} + \frac{4}{9} + \frac{4}{9} = \frac{9}{9} = 1$ ✓ verified

(7)

So $\left(\frac{n}{m}\right)^2$

$m = cn^{1.5}$ = size of table

but then what is n?

Or keep a variable

$$\left(\frac{n}{cn^{1.5}}\right)^2 < \frac{1}{c^2}$$

$c=1$

$$\left(\frac{n}{n^{1.5}}\right)^2 < 1$$

$$\left(\frac{1}{n^{0.5}}\right)^2 < 1$$

$$\frac{1}{n} < 1$$

$n \geq 0$ so < 1 well not $n=0$

c)

$$\left(\frac{kn}{cn^{1.5}}\right)^2 < \frac{1}{c^2}$$

(8)

$$\left(\frac{1}{cn^{1.5}}\right)^2 < \frac{1}{c^2}$$

$$\frac{1}{c^2 n} < \frac{1}{c^2}$$

Since $n \geq 0$

ie $n = 4$, $c = 4$ $\quad \frac{1}{4(4^2)} < \frac{1}{4^2}$

$$\frac{1}{64} < \frac{1}{16} \quad ☺$$

I think that is a).

Was not that complicated

but I over complicated it...

b) Give a better (smaller upper bound) for 2-hash fn

So ideally something = to

So we had $\frac{1}{c^2 n} < \frac{1}{c^2}$

$c = 1$ $\quad \frac{1}{n} < 1$

$c = 2$ $\quad \frac{1}{4n} < \frac{1}{4}$

(9)

; how do we go about doing this?

Well prob of collision $= \frac{1}{cn}$

So that would be a perfect lower bound

Can we involve n?

n is given ergonomsly - So sure!

c) We want a smaller ϵ and better chance of collision ? this just relates to table size

↪ so how are these 2 related

Show it is possible by generalizing to k-hash fn

Find a fn $\epsilon(k)$ monotonically ↓ in k

s.t. when using a k-hash fn to insert n items into table of size $cn^{1+\epsilon(k)}$

the prob of collision is $< \frac{1}{c^k}$

(10)

So what does "find a εn $\varepsilon(k)$ mean"

Say what that εn is?

So Prob collision $\left(\frac{n}{m}\right)^k$

and $m = cn^{1+\varepsilon(k)}$

So $\left(\frac{n}{cn^{1+\varepsilon(k)}}\right)^k < \frac{1}{c^k}$

$$\left(\frac{1}{cn^{\varepsilon(k)}}\right)^k < \frac{1}{c^k}$$

$$\frac{1}{c^k n^{\varepsilon(k)k}} < \frac{1}{c^k}$$

as long as $n^{\varepsilon(k)k} < 1$ Wait a minute! > 1

So $k \geq 1$ we know

So $\varepsilon(k) < \frac{1}{k}$ must be > 0

(11)

So this means $\varepsilon(k)$ can be a wide range of stiff we know $k \geq 1$

But what does monotically decreasing in k mean?

↳ always goes down / flat never up even for a moment

But then $\varepsilon(k)$ is anything as long as < 1

So then to keep $\varepsilon(k) \cdot k$ fixed call d,

$\varepsilon(k) = \frac{1}{k}$ so $\frac{k}{k} = 1 \rightarrow n' = n$

But is that monotonically ↓?

Well

k	ε
1	1
2	$\frac{1}{2}$
3	$\frac{1}{3}$
4	$\frac{1}{4}$

↓ So ↓ always

(12)

Then we have

$$\frac{1}{C^k n} < \frac{1}{C^k}$$

and since $n > 0$ by defn this will hold

(is this too easy?)

d) Conclude that can use k-hash fn to achieve perfect hashing.

What is that?

hashing can be worst-case if static

Perfect hashing O(1) memory accesses worst case

example: 2 levels of hashing

? but how does this provide a guarentee

by choosing hash fns carefully → no collisions

how

it seems to relate closly to how the hash functi

(13)

Proof no collisions

$\binom{n}{2}$ pairs of keys may collide

each pair $\frac{1}{m}$ prob of collision

$m = n^2$

$$E[x] = \binom{n}{2} \cdot \frac{1}{n^2}$$

expected # of collisions $= \frac{n^2 - n}{2} \cdot \frac{1}{n^2}$

$< \frac{1}{2}$

?: this is what theorm showed

does that mean "perfect"

More likly than not to have no collisions

So can find w/ few trials

∴ So perfect hashing is $E[\text{collision}] < \frac{1}{2}$

∴ Or if guaranteed 1:1 matching

(14)

What was in class one? 11/1

↳ skipped that day?

Who was there

Universal if $P_{h \in H}[\text{collision}] \leq \frac{1}{m}$

uniformly

So lines  $ax+b \pmod p$

$a \neq 0$

$0 \leq b < p$

So $(p-1) \cdot p$ = size of H

(p-1): Choices a

p: Choices b

↳ So will never overlap

→ if so this is the first time I really understand universal hash fns

But we showed $1 \ldots n$ can map to $1 \ldots m$

Snooze!

keys are from much bigger space

$1 \ldots |u|$ over $1 \ldots m$

prob collision $\simeq \frac{1}{m}$

Perfect Hashing

keys static (don't change)

└ how did they change before?

time/op → $O(1)$

So also 2 level example

Some $\leq \frac{1}{2}$ h_i are collision free

↑ can test right away since els of dict are static + given

↑ what does that mean? how is it special?

(6)

? How do the keys change?

Oh up front.

Items are not added / removed

Why does that matter?

Seems more from a prod pov

"Since you can test right away"

Oh well don't worry about

Compare to one presented in class

(forgot original qu!)

$$\left(\frac{n}{cn^{1+\varepsilon(n)}}\right)^k < \frac{1}{c^k}$$

$$\frac{1}{c^k n} < \frac{1}{c^k}$$

$\blacksquare$

(17)

Space Usage

This uses table size $n^{1+\epsilon}$

and I said $\epsilon = \frac{1}{k}$

So for $k = 2$

$n^{1.5}$ vs $2n$

$k=10$ $n^{1.1}$ vs $2n$? better

Run time both $O(1)$ for accesses

Use cases ~~2~~ 2 look ups ...

hmm

Original good on when taking mod of

here works for other non-numeric inputs

Oh $n^{1.0001}$ vs $2n$

~~better (smaller)~~ better/smaller up to ~~1.0~~ 2^{100}

(18)

So besides size and when ~~table/spread~~ can't take mod

T; can always reduce to #?

We still need to choose same way for same input
So don't think that matters

assumptions

(, still static (don't get)

What assumptions were made in either case?

Well prob collision $\leq \frac{1}{2}$ before

mine $< \frac{1}{c^2}$

So can change it by changing c

~~its also likely~~

That is another way they are diff

(19)

Before $2n$ is expected amt of storage

$p < \frac{1}{2}$ that $\geq 4n$ storage

which it still can

= assumption

Third thats it

(20)

11/13
11:30p

(write up a)
(write up b)
(write up c)

wait what are constraints on $n^{\epsilon(k)k}$?

$$\frac{1}{c^k n^{\epsilon(k)k}} < \frac{1}{c^k}$$

if $c^k = 2$

$n^{\epsilon(k)k} = \frac{1}{2}$

$$\frac{1}{2 \cdot \frac{1}{2}} < \frac{1}{2} \quad (\times)$$

$n = 2$ $\quad \frac{1}{4} < \frac{1}{2}$ (✓)

that makes it more interesting

(2)

We know $k \geq 1$

So $\epsilon(k)$ must also be ≥ 1

And want it monotonically $\downarrow$

hmm

(am I sure about these constraints?)

$\epsilon(k)$

$\frac{1}{k}$ would not be ≥ 1

$1 + \frac{1}{k}$

So then $\epsilon(1) = 2$

$\epsilon(2) = 1.5$

$\epsilon(10) = 1.1$

$\epsilon(100) = 1.01$

(22)

$n^{(1+1/k)k}$

n^{k+1}

n^{k+1} still ~~$\neq 0$~~ so will always be smaller

sorry > 1

nice!

(write d)

I still don't really get

11/2
2A

4. Multicommodity Single Path Routing

Given a graph $G(V, E)$ in which edge $(u, v) \in E$ has non neg capacity $c(u, v) \geq 0$

We also have a set of k diff commodites $k_1, \ldots k_k$ and codmity i is specified by $k_i = (s_i, t_i, d_i)$ ~~where $s_i \to t_i$~~

source sink demand = amt to flow $s_i \to t_i$

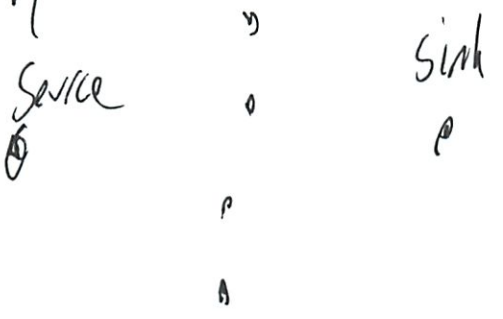
Diff commodites may share source or dest but we assume $s_i \neq t_i$

Each commodity i can only flow along a single path s_i to t_i

Is a bunch of mini routing problems?

(2)

Normally



Fad Fulkerson

Augment paths

~~Edmond~~ max flow over the min cut

(what is min cut?

Edmond Karp

Max bipute matching

So we could do mega Source to mega sink

but must overlay on graph

Single path this is weird - how do we do that?

(3)

11/13
10:30A

Graph

Kommodities

go from a source to a sink

all must flow along a single path

? could we add them one at a time
but that's typical optimization
LP

How do we normally do it?
initialize all to 0
while there is aug. path, increase that
? from s to t in aug flow
? which we find through DFS?

So F-F its arbitrary
E-K it is BFS - so shortest (that is the difference!)

(13) (4)

4) One way to model a flow network is with edge flows $\{f_i(u,v)\}$ as decision variables

CI was thinking LP!

CLRS 29.2 Is Formulating problems as linear problems

Once we can make something a LP (poly-sized) we can solve in poly time

1. Single source shortest path
2. Max flow
3. Min cost flow ← normal is poly time
4. Multic commodity
   - ↳ only known soln is LP

d_v = shortest path weight for v

f_{uv} = flow from u to v

(5)

Shortest path

Single pair

Want d_t = weight shortest path $s \to t$

Bellman Ford does this

Makes for each vertex v, d_v

S.t. for each edge $d_v \leq d_u + w(u,v)$

add the weight of previous

Max d_t

S.t. $d_v \leq d_u + w(u,v)$ for each edge

$d_s = 0$

note is maximizing

Since each $d_v = \min_{u:(u,v) \in E} \{d_u + w(u,v)\}$

Since d_v is largest value that is less than or equal to all values in set $\{d_u + w(u,v)\}$

Maximize d_v for all vertices $s \to t$

(6)

So let me make sure I understand this

First off Bellman Ford

initialize $d = \infty$ ← shortest path
$\pi = nil$ ← parent

for each vertex
  for each edge
    relax → check if $v.d > u.d + w(u,v)$
      if so $v.d = u.d + w(u,v)$
        $v.\pi = u$

Then this

look back at matrix stuff

Objective

Constraints

Standard form

ILP?

integral if totally unimodular

7

Did min cost Network flows in lecture

$G(V, E)$ c_{ij} for $(i,j) \in E$

want b_{ij} at each $i \in V$

Cost h_{ij}

min $h_{ij} f_{ij}$ for $i \in V$

$$\sum f_{ik} - \sum f_{ki} = b_i$$

$$0 \leq f_{ij} \leq c_{ij} \text{ for } (i,j)$$

all integral optimal sols for integer capacities

$$\underbrace{A}\, x < b$$

totally unimodular

(oh there is TU criteria in notes!)

(8)

f_{ij}

each col has 2 entries for +1, -1

iso each col is a line

each row is a node?

Think so

Book) min cost flow

often the actual alg is better

like if we add a cost to flow

same as lecture

$$\sum f_{ru} - \sum f_{uv} = 0$$

~~flow in = flow out~~?

no flow for each dir on line =

$$\sum f_{sv} - \sum f_{vs} = d$$

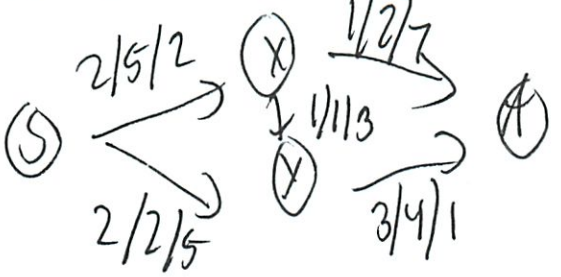
↑ value of the flow

* flow demand *

for s & t

(9)

So in the example want 4 units



flow/cap/cost

So $\sum_{sx} - \sum_{xs}$

$2 - 0 = 4$?

$\sum_{sy} - \sum_{ys}$

$3 - 0 =$? 4

$\sum_{st} - \sum_{ts}$

$4 - 0 =$? 4

q should be for

Ohh for all $v \in V$

$2 + 3 + 4 - 0 =$? 4

(unless include self arcs?

I think I am reading it wrong...

(10)

flow from s to every node v

$f_{ss} = 0$ - right

$f_{sx} = 2$

$f_{sy} = 3$

$f_{st} = 4$ ← node incoming

---

look again at

$$\sum_{v \in V} f_{uv} - \sum_{v \in V} f_{uv} = 0 \text{ for each } u \in V - \{s, t\}$$

So $u = x, y$

$v = s, x, y, t$

for x:

$$f_{sx} + f_{yx} + f_{tx} - (f_{xs} + f_{xy} + f_{xt})$$

$$2 + 0 + 0 - (0 + 1 + 1) = 0?$$

or neg flows

$$2 + \underset{0}{-1} + \underset{}{-1} - (2 + 1 + 1)$$

a trivial

(11)

So again at d

$$\sum_{s,x,y,t} f_{sv} - \sum f_{vs}$$

$$f_{ss} + f_{sx} + f_{sy} + f_{st} - (f_{ss} + f_{xs} + f_{ys} + f_{ts}) = 4$$

$$0 + 2 + 2 + 4 - (0 + -2 + -2 +$$

Can we only have f_{uv} if line?

I think so

Yeah

$0 + 2 + 2 + 0 - ($ then here should be 0 $) = 4$

Since no backward edges? meaning

This was from before

$$|f| = \sum_{v \in V} f(s, v) - \sum f(v, s)$$

total flow (directly) out of source − flow into source

(usually 0)

So reasve max flow w/ flow out of source

(12)

Oh enough bg work

Settled f_{uv} is only if arrow in that direction

$f_{uv} = 3$

$f_{vu} = 0$

* flow can't be (−)

So $\underbrace{\sum f_{vu}}_{\text{edges incoming}} - \underbrace{\sum f_{uv}}_{\text{edges outgoing}} = 0$

Would have been so fast to realize if could have someone point it at!

Multicommodity flow

hockey pucks, sticks helmets

each has own factory and warehouse

must share shipping network

note → no antiparallel edges

(13)

f_{uv} = aggregate flow

$$\sum_{i=1}^{k} f_{iuv}$$

Still must be subject to capacity of edge

Note: not trying to minimize any objective fn
just does it exist (in textbook)

Now back to problem

$f_i(u,v)$ = decision variable

represent flow of each commodity on each edge

Find the Min Largest-Edge-Load

the largest edge load is minimized

Write as a ILP

L = the total flow on edge (all commodities)

(4)

Ok the task is to write this as an ILP

Which appeared pretty non standard

Why did we maximize before

Well we did minimize in max cost

minimize max f_{uv}

$\sum_i f_{i,uv}$

Can we do that?

Well we are trying to minimize

max( ) is just a std function

But how to constrain to single path

extra constraint I'm pretty sure

$f_{i,uv} = \{0, 0\}$

how do you say that? or

(15)

What did we do before - on p-set?
that was 0 or 1
when it was an ILP

In example in book it's just multicommodity
That is the real challenge here...

Qn A

d_i is not necessarily integer
Does not have to be std form *

* Don't forget can't have more than h_i edges in a commodity path

So since not in std form, can I say $\{d_i, 0\}$
that seems like a cop-out but could be an easy problem

16

And for the h_i

$$\sum_v \sum_v f_{uv} \quad \text{for each } i$$

$$\text{should} \leq d_i h_i$$

as a way of constraining

Since d_i is always the same

So $\underbrace{5+5+5}_{\text{have 3 steps}} + \underbrace{0+0+0}_{\text{and a bunch of doesn't matter}} \leq 5 \cdot 3$ (max steps)

So that should be it...

Starting write up in LaTex

it lets me be less creative

write out on paper 1st!

(17)

What is Exec summary?

Problem you are solving
techniques used
assumptions made
running time

Is a normal LP - but w/ integral optima?
Or just say normal LP - poly time
↑ think this -
other constraints

Max
X, u, v

don't want sum
- no d,

(18)

11/13
3:15P

b) As seen in Rec 7

Can use path flows $\{f_i(p)\}$ as decision variables where $f_i(p)$ represents flow along a path $p \in P_i$ and P_i is the set of all paths s_i to t_i

So aggregating at the path level now

Show that for multicommodity single path routing, a set of edge flows can always be represented by a set of path flows

And the # of path flows is polynomial w/ r to $|V|$ and $|E|$

Hint: Review alg to find augmenting paths.

Augmenting paths

path from s to t in residual network G_f

Increase flow on edge (u, v) of aug. path by upto c_f

~~So 1. find residual capacity~~

Find residual network first G_f

$c_f(u, v) = c(u, v) - f(u, v)$

edges that can contain more flow

Add negitive edge

$c(v, u) = f(u, v)$

Since can cancle out flow

Aug path is path $s \to t$ in G_f

E-k we choose residual path BFS

So its the shortest path

Now what was Res 7 Path flows

was this through taking the dual?

$Y_{u,v}$ corresponds to edges

So path $s \to t$ must include at least One edge from each path $s \rightsquigarrow t$

Want to minimize this sum

let me think this through...

Primal

A has +1 for each path that uses an edge

Dual rows have +1 for every edge along a path

Primal A is capacity constraints

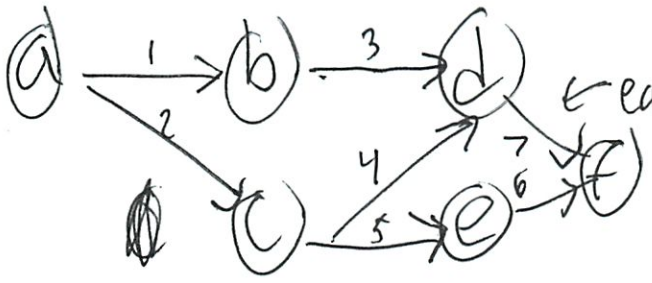
$$\sum \underbrace{x_{ij}}_{\text{sum of in}} - \underbrace{x_{ji}}_{\text{sum of out}} = 0$$

(2)

So then that is

A:

a node → [↓ an edge



← each has capacity 1 ~~and flow 1~~

~~A ends up being the flow~~

no the capacity!

don't include a bf →

	edge 1	edge 2	3	4	5	6	7
A	-1	-1					
B	+1		-1				
C		+1		-1	-1		
D			+1	+1		-1	
E					+1	-1	
F						+1	+1

] adds to 0 wellsarp sink except

adds to 0 each col

f_{ij} are X these are constraints A

Transpose

	A	B	C	D	E	F
edge 1	−1	+1				
edge 2	−1		+1			
edge 3		−1		+1		
edge 4			−1	+1		
edge 5			−1		+1	
edge 6					−1	+1
edge 7				−1		+1

So what does this mean?

x_p flow on each path p in set of paths P from s to t

$$\max \sum_p x_p$$

Subject to $\sum_p x_p \leq c_{ij} \; \forall (i,j) \in E$

$x_p \geq 0$

(23)

Now taking the dual is simpler

A has +1 for each path that uses an edge

So dual has +1 for each row along a path

So its something else we defined!

We just claim to have a set of paths P
- which perhaps we find w/ BFS

Then take the dual of that
- which idk why we do

"So cons have +1 for each edge along a path"

how does that help?

Do we just use this fact

Well I guess show we can do this

29

So we want to show that it is done

- Why do we pick shortest path in augmenting?
  - proves there is a possible path?
  - maintains our current state
    - don't start fresh each time
    - work off what we have

So for each commodity G_i in certain order

1. find residual flow G_f
2. BFS augmenting path to (no cost data...)
3. Add that to our network
4. Repeat

But must find all...

~~Q~~ second primal is exponential

~~But~~ what about dual

(the notes are imprecise!)

So I can find by taking dual like in notes
(don't really get...)

$$\min \sum_{ij} c_{ij} y_{ij}$$

$$\text{s.t} \sum_{u,v \in p} y_{u,v} \geq 1 \quad \forall p \in P$$

(but this is not operationl)

Still need to find p though

Or just show it is possible to represent

Must ~~show~~ give an alg to convert a set of edge flows into path flows.

---

* Not single path

(26)

So said algorithm to convert

¿ Need all

No - only need 1 representation.... I think

So what I attied before

try each w/ augmenting flow

To prevent retry / backoff

High demand → low demand

Since say we have

cap 5 — and want to route 3 and 2 through

if rate .3 through → 2 could fit

no bad example

cap 4 → could rate 3 thrugh closer to goal

(27)

The question is can greedy work

~~Never~~ Could we get 2,2 to go through as 4
Instead 3 goes first + clogs it up

But must be greedy for poly time

Oh in a) Super source / sink

Or no -> just # paths is polynomial
(ie one per commodity)
But running time * could be * exponential
But I have a feeling it is not

Or # LP that maximizes
that backoff is chariotic of LP

(28)

So maximize $\sum$ Throughput

w/ the multicommodity constraints

(any others)

Prove it can be represented by

Saying if max flow is possible w/ edge ~~vals~~ weights

We could also do it w/ paths

Since same underlying ans

Just 2 diff ways to represent

Underlying ans: a max flow

(29)

11/13
10:15p

(a write up)

11/13
11:45p

(b write up)

Did I end up solving this?
How to do assignment?
Would be exponential.
I don't think there is a better way to do this...

What is the LP?

$\min \sum_i f_i u_v$ total cost

Same as before?
Yeah then write each path as a variable
(very handwavy)

(26)

maximize total flow

Number of path flow variables

We need k for each possible path

but must be polynomial

You never know if you are right in this class

limited # of possible paths

Since ~~V~~ E^E exponential

put that in the proof as a subpart

Also dual as in R7 notes

just mention

(21)

I made up that last part

Is oh well...

11/18
4:30p

#5 Linear time MST

a) Given a connected graph $G=(V,E)$

"point from anywhere to everywhere

such that all weights are 1 or 2 show optimum MST weight can be found in linear time (# of edges)

$m > n$

edges > ? nodes

? of course

MST

Greedy

Each time we add a node so still MST

find minimum one

Cut a partition

edges cross

cut respects edges if they don't cross cut

light edge is min crossing cut

(7)

light edge is the safe edge

A = MST

So Cut can't cross MST (must respect)

but parts of A could be in sep sides

↳ when do ya have more than 1 part of A?

|V| − 1 times

? but this is O(n)

and since m > n

isn't this better?

Kruskal

finds all edges that connect any two trees in forest the edge (u, v) that is least weight

So this finds disjoint trees!

goes up by weight

edges

- if already in same set → skip

Sorting edges O(E lg E)

(3)

for $O(E)$ Find-Set and Union

~~Kbst~~

So $O((V+E)\alpha(V))$

?Slow growing fn

$|E| \geq |V| - 1$

So disjoint set ops takes $O(E\alpha(V))$

$\alpha(|V|) = O(\lg V) = O(\lg E)$

$O(E \lg E)$

Since $|E| < |V|^2$

$\lg|E| = O(\lg V)$

$O(E \lg V)$

(4)

Prim:

like Dijstra

Starts at arbitrary V

never disjoint

greedy

need a fast way to select a new edge

Lso min-priority queue

V.key = min weight of any edge in tree

So ~~for each node~~ until done

extract min node

for each edge, check if $w(u,v) < v.key$

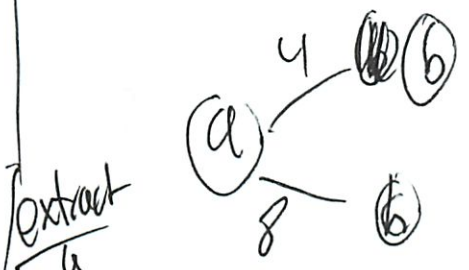
if so - add them

determines the light edge crossing the graph

So each step it has the min cost edge for each node it has seen

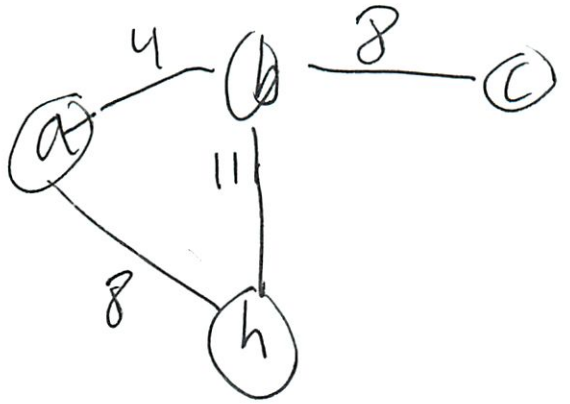
(5)

In their example



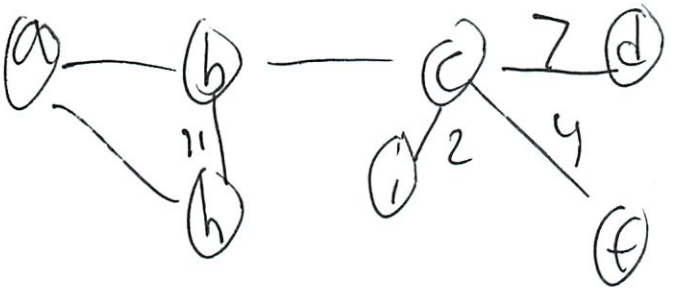
extract a

a.lowest = 4 ← had extract
b.lowest = 8 ← extract next
h.lowest = 11



extract b

~~a.lowest~~
b.lowest = 8 ← had extract
c.lowest = 2 ← extract next
h.lowest = 11



c.lowest = 2
h.lowest = 11
i.lowest

When does it look for lowest
Checks when it extracts

so extract a = u
for each v : b, h
if w(a,b) < b.key
set b.key = w(a,b)

(6)

if $w(a,h) < h.key$

set $h.key = w(a,h)$

So sets $b.lowest = 4$

$h.lowest = 8$

So what I had was wrong!

→ Then picks b

now check b, c → set c.lowest = 8

b, h → set h.lowest = 8 (better than 11)

→ Randomly pick c or h

Now c, d = d.lowest = 7

c, f = f.lowest = 4

e, i = i.lowest = 2

So he lowest to get there

Then it picks the next ~~lowest~~ cheapest node to get to

(7)

Makes sense

IDK why I never studied these

So min heap

$O(V \lg V)$

(no skipping)

$O(E \lg V)$

Fib heaps better

Anyway the point is both of these suck (non linear)

Can we do something special for our special case of weights = 1, 2

<u>And</u> fully connected

We could do all the 1s first

Then the 2s

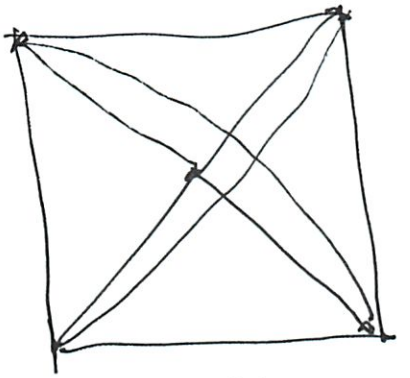
(8)

build list of 1s and 2s separate

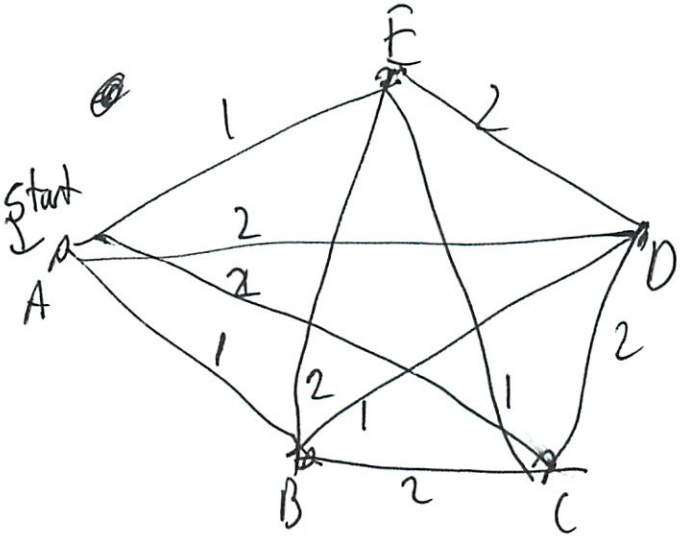
So not priority queue - simple queue

visit all of those Prim style

Then visit the 2s



That was silly



So 1s: E B

At A 2s: D C

(9)

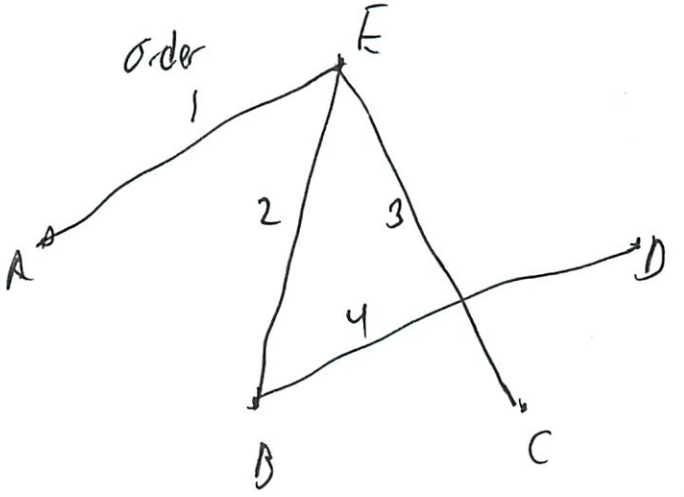
At E ls: A B C ← from prev
(↓ already there)
2s D C
(stays for now)

At B ls: C D
2s D C

At C; ls: D
2s: DC

At D ls - Done!

So order



(✓) MST
I like it

O(E) since checks each edge once

(10)

b) Given a connected graph $G=(V,E)$ s.t. all edge weights are numbers in the range $[1/2, \ldots, 2]$

"what does that mean?
any # (fraction or int)
both $\geq \frac{1}{2}$ and ≤ 2 ?

Show there is an alg that runs in linear time (m) that outputs b s.t. $MST(G) \leq b \leq 2MST(G)$ where $MST(G)$ is the value of the min spanning tree of G.

What is the value of a MST?
I can not seem to find in MST chap

value of a function $f: A \to B$

$b = f(a)$
b value a argument

not helpful...

(11)

QnA

Value = total weight of edges

Again may assume $m \geq n$

Ah ok -> total weight of edges

So in our example before

weight = [illegible] 4

all edges were 1

So we can output a # b

that is b/w [illegible] 4 and 8

Approx alg!

I wanted to study those as well!

Lecture | +Book

Approx alg for problem of size n has approx ratio $\rho(n)$ if for any inst, alg produces a sol of cost C s.t. $\max\left(\frac{C}{C_{opt}}, \frac{C_{opt}}{C}\right) \leq \rho(n)$

↳ then is $\rho(n)$-approximate

Maximization $0 < C \leq C^*$

$\frac{C^*}{C}$ gives factor by which cost optimal sol is larger than the cost of the approx soln

So $C^* = C_{opt}$ = cost of optimal soln

C = cost of approx soln

So these are ~~problems~~ algs that give near-optimal sols for NP-hard problems

13

For minimization $0 < C^* \leq C$

ratio $\frac{C}{C^*}$

Approx ratio is never less than 1

So a $\rho = 1$ means perfect/optimal

Can trade computational time for quality of approx

So for fixed ϵ → have $(1+\epsilon)$-approx scheme

Then poly time approx

What does that look like in ratio?

actual 2^n

this approx n^2

$$\max\left(\frac{2^n}{n^2}, \frac{n^2}{2^n}\right)$$

much bigger

but still exponential - So how does this help?

Super confused

Fully poly time approx if approx and running time is poly in both $\frac{1}{\epsilon}$ and size n of inst

L ie $O\left(\left(\frac{1}{\epsilon}\right)^2 n^3\right)$ huh?

? So constant factor decrease in ϵ Comes w/ corresponding constant factor ↑ in running time

Ok don't get all that

Prob don't need to for problem

But spend same time

Probabilistic time approx scheme (PTAS) = polynomial in n

Fully PTAS: polynomial in n and $\frac{1}{\epsilon}$

$O(n^{2/\epsilon})$ PTAS only

$O(n^{n/\epsilon^2})$ FPTAS

How can something be polynomial in something?

(15)

So in lecture speech said if 50

25 → 100

Can we get arbitrarily close in poly time w/ a given ϵ?

ask tutor after exam

I think approx ratio should be numbers?

like for 2-approx

$$\max\left(\frac{4}{2}, \frac{2}{4}\right) \leq 2$$

largest

So last $C = 4$

$C^* = 2$

---

well $C^* = 4$, $C = 2$ should be

$$\max\left(\frac{2}{4}, \frac{4}{2}\right) \leq 2$$ ☺ works

↑ max

16

But then $\rho(n)$ I think is a function of n

So $O(n^{2/\varepsilon})$ means ε...

well ε is fixed

but then why do they say $\varepsilon \downarrow$ by constant factor

Lets try ~~some~~ more

Vertex cover

- 2 approx methods
- natural (max degree)
- random

Random is a 2-~~ish~~ approx alg

$|A|$ = edges picked

C_{opt} must include at least 1 endpoint of (optimal cover)

each edge in A (and other edges)

No two edges in A share endpoint

$|A|$ is lower bound for $|C_{opt}|$

$|C_{opt}| \geq |A|$

(17)

# of vertices in $C = 2|A|$

$|C| \leq 2|C_{opt}|$

So what are these?

$C_{opt} = \{b, d, e\}$ $|C_{opt}| = 3$

So actual cost of optimal sol

not the cost to find it

Our C was $\{b, c, d, e, f, g\}$ $|C| = 6$

A = edges picked

> remember vertex cover = exactly 1 endpoint of each edge included

So $|C_{opt}| \geq |A|$

Taller band

~~if Copt faces~~

(if Copt was perfect)

there $C = 2|A|$ ← how do we know

so $|C| \leq 2|C_{opt}|$

What how do we know the 2
Or know that is that is the worst case?

Unless that is how we find it...

11/13
7:30P

~~Approx Set Cover~~

Textbook look at best case

$|C^*| \geq |A|$

Then see our approx worst case (upper bound)

$|C| = 2|A|$

* Since each execution of 4 picks an edge for which either of endpoints is in C

A = edges picked

I kinda see this now!

(19)

Set Cover

Each element belongs to at least one X

Find min size subset whos members cover all of X

└ # of sets ?

So we get a bunch of candidate sets up front

And we need to pick which

real one is NP-Complete

But greedy one (tries to cover as much as possible)

$O(|X||F|)$

predicted result is not that much larger than real result

$p(n) = H(\max\{|S| : S \in F\})$

So need to find cost

S_i = ith subset selected

each subset = cost = 1

(20)

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup \dots)|}$$

each:

$$|\text{alg}| = \sum_{x \in X} c_x$$

⋮

So S_i chooses larger one 1st

$$\sum_{x \in S} c_x \leq \sum_{i=1}^{k} (u_{i-1} - u_i) \frac{1}{u_{i-1}}$$

Can bound

⋮

$$= H(|S|)$$

↑ harmonic #

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$$

$$= \sum_{k=1}^{n} \frac{1}{k}$$

$$= \ln(n) + O(1)$$

(21)

So $\rho(n) = H(\max\{|S| : S \in F\})$

^ So max of what?

Size of S or the S in F

Harmonic in # of sets provided

$H(6)$ here

$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6}$

$= 2.45$

So within 2.45

But here it changes as it gets bigger

$2 \to 2.59$

$8 \to 2.71$

$4 \to 2.82$

$10 \to 2.92$

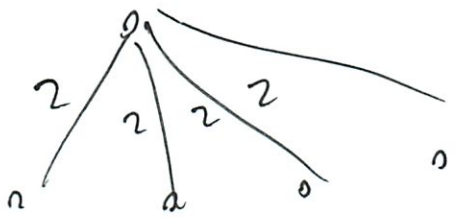
etc

but diff gets <u>smaller</u> as it gets <u>bigger</u>

(22)

So we can now turn our attention to our approx

Worst case (w/ 5 elements)



MST = 8

2 MST = 16

Best case (w/ 5 els)

MST = 2

2 MST = 4

So can't be constant

New scheme

(this is kinda fun!)

(23)

Average

Would work here

but what if $4\frac{1}{2}$ and 20 2s

? may here

for 5 nodes

~~min edges = 4~~

No Connected so

2 nodes = 1 edge —— 2

3 nodes = 3 "  3

4 nodes = 6  4+2

5 = 10  5+5

Well first node Connects

in 6 Connects to 5

next 4

5 · 4 · 3 · 2 · 1

Or +

4 + 3 + 2 + 1

10 (3 / 6) ✓ works

(24)

So nodes $= \sum_{i=1}^{n-1} i$

but anyway MST is (nodes -1) edges

So $\frac{1}{2} \cdot (\text{nodes} - 1) + \sum_{i=1}^{n-1} i - (\text{nodes}-1) \cdot 2$ is what?

for 2 nodes $\frac{1}{2} + (1-1)\cdot 2 = \frac{1}{2}$ MST $\frac{1}{2} = \frac{1}{2}$

3 $2 \cdot \frac{1}{2} + (3-2)2 = 3$ $2 \cdot \frac{1}{2} = 2$

4 $3 \cdot \frac{1}{2} + (6-3)\cdot 2 = \frac{1}{2}$ $3\frac{1}{2} = \frac{3}{2}$

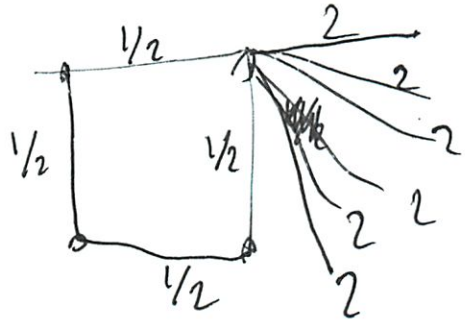
5 $4 \cdot \frac{1}{2} + (10-4)\cdot 2 = 14$ $4 \cdot \frac{1}{2} = 4$

(but must stumble across right sol...)

(25)

Take the $\sum_{\text{nodes}-1} \min(\text{edges})$

Since Kruscal does this
but skips if already part of it
So how to prove?
Anti example:

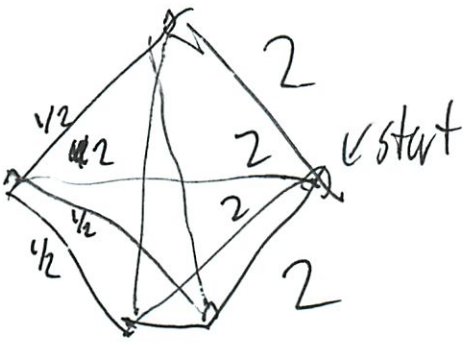


here mst = $3\frac{1}{2} + 6 \cdot 2 = 13\frac{1}{2}$
but our ans is $-\infty$ since # of edges

*And must be connected

(26)

So I am feeling better about this!



rest 2

So $2 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2}$

ours would return that

So I am happy w/
But how to prove?
Something about connected

I think we can actually show worst-case

(21)

i what I shoed
first tip somewhere is 2
but rest is to lowest pt

Oh it could have avoided
for 2+2 but then falls to a $\frac{1}{2}+\frac{1}{2}+\frac{1}{2}$

prediction $2+4\cdot\frac{1}{2}=4$
actual $5\frac{1}{2}$

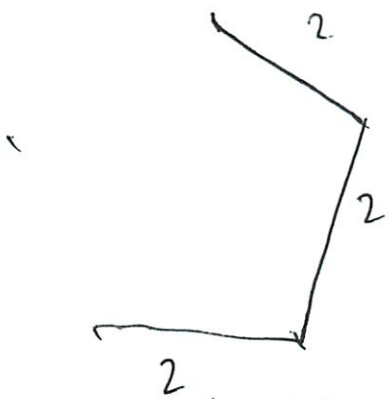
Oh we never want to be <u>under</u>
So double 8 is b/w $5\frac{1}{2}$ and 11

Since predict =2
act = 4
then 4 b/w 4+8 since =4

but if all 2 then correct
if 3 ~~the~~ 2s and $2\frac{1}{2}$s
Could $2+2+2+\frac{1}{2}+\frac{1}{2}=7$ actual

(28)

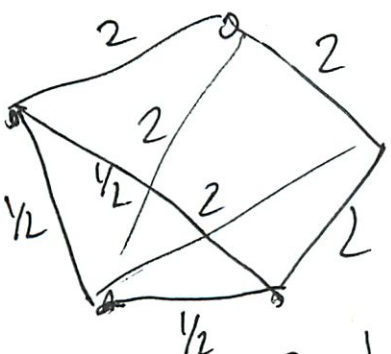
wait would have to have bad luck



bad luck
all must be 2

So that 1 point must only have $1\frac{1}{2}$
then ~~xxx~~ predicted = actual

2 2s



$3\frac{1}{2}$s actual $2+2+\frac{1}{2}+\frac{1}{2}$

Saw earlier

Im convinced but I don't know how to actually show...

(29)

11/13
9p

(Write up for b)

How to do the proof

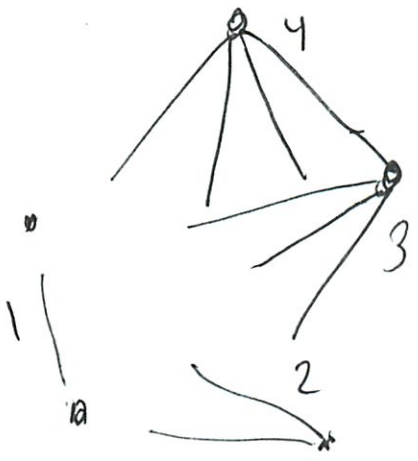
In a fully connected graph, you can only go so far before each decision

5 nodes

1st observe 4 edges

2nd observe 3 new ~~edges~~ edges to new node

3rd 2 new



So now what?

Eventually you will find a minimum point

(30)

They are not ness. connected

Need a certain amount to be large

Some # of large and smalls!

(21)

11/13
10p

(write up a)

Dating my routine
Think about it later

Michael Plasmeier
6.046 R07

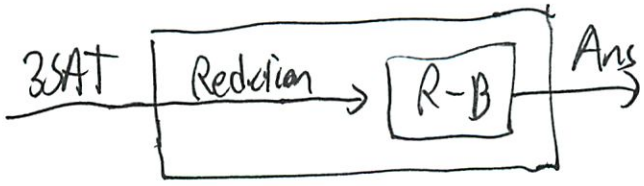
Quiz 2 #1

Red-Blue Subsets

Exec Summary

We can prove Red Blue Subsets (R-B) is NP-complete by reducing 3SAT $\propto$ R-B
↑ 3CNF SAT

We do this by
1. Showing R-B is in N-P
2. Praiding a redction for 3SAT to R-B



a) Show if $x \in$ 3SAT then R(x) in R-B
b) Show if R(x) $\in$ R-B then $x \in$ 3-SAT

(2)

Detailed Solution

1. R-B is in NP; since we can build a verifier $V(x, y)$ where x is the input and y is the potential solution.

We check each criteria individually.

1. We can go through the list of nodes in R and B to make sure we "cover" each element in S once - by crossing each off and not having a remainder $O(S)$

2. We can go through R, S to ensure none is duplicate by creating a tempary 3rd table (size S) where we check off each element in R and see in B. $O(S)$

(3)

3. We can scan through the list of subsets to insure there is at least 1 R and 1 B in each. O(FBS)

Thus we can provide a certificate in polynomial time

---

2. We can provide a reduction from 3SAT to R-B.

(We are pretending we have a black box for R-B and are using our R-B black box to solve 3SAT.)

Well first we notice that S is like our set of literals

We notice that F is like our set of clauses.

(9)

We can treat R as true and B as False.

Then for each clause we can write it as a series of elements as a Fi

For example $X_1 \vee \overline{X_2} \vee X_3$

Can be written as

$R \vee B \vee R \vee B$

We write true as R

false (or negated as B

We also add an extra $\underline{\vee B}$, at the end.

This is to satisfy condition 3.

This is because each clause must have between 1 - 3 trues. (it can't have 0)

(5)

So we add this extra $\vee B$ which helps us achive the condition if the other 3 are rcd.

We then have a 3SAT <u>if</u> we can write the series of R, Bs such that the conditions hold.

Of course we must make sure we keep our items internally consistent behind the scenes

X_1 is R and $\bar{X}_1$ is B

Or

X_1 is B and $\bar{X}_1$ is R

(6)

Proof

1. An input to 3SAT is an input to R-B

This is what we just explained. Each input to 3SAT can be reduced to a R-B problem.

The answer to the R-B problem is an answer to 3-SAT because the conditions are the same.

1. An item must be true or false.
2. " " can not be both true and false.
3. Each clause requires at least 1 true (and we've added a free false) All clauses must be true.

(7)

2. An input to R-B could be an input to 3SAT.

We can take our R-B and write it as a 3SAT largly reversing our process.

Write them as literals

$(x_1 \vee \bar{x_1} \vee x_2)$ (drop the always false fake literal

Then we are left w/ a 3SAT.

If we had a R-B that worked, then it would have had at least 1 R, so at least one of our literals would evaluate to true in each clause, so our 3SAT would be true as well.

The converse is true as well (if happens not to be an R in R-B) the clause is false and 3SAT would be False.

Michael Plasmeier
6.046 R07

Quiz 2 #2

Broadcast Channel

Executive Summary

We can use a randomized algorithm to solve this. Each server can independently decide to transmit or not. We can show that the chance of collision decreases as we add machines.

This allows the first message to transmit in O(1) time steps

(2)

Detailed Solution

We will have a randomized protocol.
On each time stamp, each machine will independently decide if it should transmit

$$P(\text{transmission}) = \frac{1}{n}$$

$$P(\text{no transmission}) = 1 - \frac{1}{n}$$

The prior history of collisons is not a factor.

Let us assume we have all n machines
(less than n machines is just a weaker case where the $E[\,]$ time steps is lower)

The probability of a collission is the probability that two or more machines transmit at once.

(3)

For 2 machines this is

$$\frac{1}{n} \cdot \frac{1}{n} = \left(\frac{1}{n}\right)^2$$

We could have 3 machines transmit at once

$$\left(\frac{1}{n}\right)^3$$

So we must add all of these chances together

$$P(\text{any collision in 1 time step}) = \sum_{i=2}^{n} \left(\frac{1}{n}\right)^i$$

This is

$$\frac{1}{(n-1)n}$$

n	2	3	4	5
P(collision)	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{20}$

Now we must consider the probability of 2 collisions in a row.

This is $P(\text{any collision in 1 time step}) \cdot P(\text{any collision in 1 time step})$

or $P(\text{any collision in 1 time step})^2$

(4)

We must consider the probability of any # of collisions

$$\sum_{i=1}^{\infty} P\left(\begin{array}{c}\text{any collision} \\ \text{1 time spot}\end{array}\right)^i$$

This is

$$\frac{1}{n^2 - n - 1}$$

↑ ~~its con~~ I'm confusing what is n

n	3	4	5
P(any collisions)	$\frac{1}{5}$	$\frac{1}{11}$	$\frac{1}{19}$

So as n grows, our P(collisions) shrinks. Thus our protocol is O(1)

(5)

The expected # of steps is

$$E[\#\text{ steps}] = 0 \cdot P(0 \text{ collisions}) + 1 \cdot P(1 \text{ collision}) + 2 \cdot P(2 \text{ collisions}) + \cdots$$

$$= 0 + 1 \cdot \sum_{i=2}^{\infty} \left(\frac{1}{n}\right)^i + 2 \cdot \left[\sum_{i=2}^{\infty} \left(\frac{1}{n}\right)^i\right]^2 + \cdots$$

$$= 0 + \underline{1} \cdot \left(\left(\frac{1}{n}\right)^2 + \left(\frac{1}{n}\right)^3 + \left(\frac{1}{n}\right)^4\right) + 2\left(\left(\frac{1}{n}\right)^2 + \left(\frac{1}{n}\right)^3 + \left(\frac{1}{n}\right)^4\right)^2 + \cdots$$

with $n = 4$

$$= 0 + 1 \cdot \frac{21}{256} + 2 \cdot \frac{441}{6536} + \cdots$$

$$= .0954$$

↑ rounding appropriately

↑ add 1 for 1.0954

1 step should be good

Michael Plasmeier
6.046 R07

Quiz 2 #3

k-Hashing

a) Exec Summary

We can prove to have good randomness by proving an upper bound (given) on our collision probability.

Detailed Solution

Normally the probability of a collision is $\frac{1}{m}$

If we have n items it is $\frac{n}{m}$

If we try twice, the probability both have a collision is $\frac{n}{m} \cdot \frac{n}{m} = \left(\frac{n}{m}\right)^2$

We are told $m = cn^{1.5}$

So let's prove

$$\left(\frac{n}{cn^{1.5}}\right)^2 < \frac{1}{c^2}$$

(2)

$$\left(\frac{1}{cn^{.5}}\right)^2 < \frac{1}{c^2}$$

$$\frac{1}{c^2 n} < \frac{1}{c^2}$$

Since $n > 0$, it will always make the left side smaller so proved ▨

b) Exec Summary

We are asked to give a better lower bound. We can just answer what the probability is exactly.

Detailed Solution

The probability of collision is $\frac{1}{c^2 n}$

This can also be our upper and lower bound on the probability of collision.

Note n is given exogenously, so we can do this.

(3)

c) Executive Summary

We can find a function $\varepsilon(k)$ so that k is monotonically decreasing so the probability of a collision is less than $\frac{1}{c^k}$

Detailed Solution

Generally our probability of collision is $\left(\frac{n}{m}\right)^k$ and $m = cn^{1+\varepsilon(k)}$

So $\left(\frac{n}{cn^{1+\varepsilon(k)}}\right)^k < \frac{1}{c^k}$

$\left(\frac{1}{cn^{\varepsilon(k)}}\right)^k < \frac{1}{c^k}$

$\frac{1}{c^k n^{\varepsilon(k)k}} < \frac{1}{c^k}$

So $n^{\varepsilon(k)k} > 1$ must be

(4)

We know $n > 0$

$k \geq 1$

So $\varepsilon(k)$ must be > 1

and we want it to be monotonically decreasing.

So $\varepsilon(k) = 1 + \frac{1}{k}$ meets that criteria

As $k \uparrow$, $\varepsilon(k) \downarrow$ monotonically

So back to our constraint

$$\frac{1}{c^k n^{\varepsilon(k) k}} < \frac{1}{c^k}$$

$$\frac{1}{c^k n^{(1+1/k)k}} < \frac{1}{c^k}$$

$$\frac{1}{c^k n^{(k+1)}} < \frac{1}{c^k}$$

$? n^{k+1} > 1$ so bound holds

(5)

d) Executive Summary

Compared to the two stage hashing shown in lecture, they both have the same runtime, they depend on case for space usage, and make different assumptions

Detailed Solution

Perfect hashing means $O(1)$ memory accesses to look up a value.

We can have that when $E[\text{collision}] < \frac{1}{2}$

Universal is if $P[\text{collision}] \leq \frac{1}{m}$

Space Usage

Class : $2n$ Though it varies

This : $n^{1+\epsilon}$ fixed

(6)

For a very small ε, $n^{1+\varepsilon}$ is actually smaller.

For example $n^{1.01}$ is better up to 2^{100} entries

Run time

both are $O(1)$ for accessing w/ 2 look ups

(7)

Use cases

Original requires you to take the mod of
this does not - allowing one to be
more flexible on the input type

assumptions

Before: prob of collision $\leq \frac{1}{2}$

Here: " " " $\leq \frac{1}{c^k}$

Before $2n$ is expected storage and
$p < \frac{1}{2}$ that storage $\geq 4n$

Michael Plasmeier
6.046 R07

Quiz 2 #4a

Multicommodity Single Path Rating

Exec Summary

This answer builds upon the multi-commodity flow Linear Program (LP) shown in CLRSv3 p863 in section 29.2 by adding an objective and the constraints that only a single path be used and we only use h_i edges.

We can represent it as an ILP which is defined to be NP-hard.

(2)

Detailed Solution

Start with multicommodity flow in CLRS03 p863.

Add the objective

$$\text{minimize} \quad \max\left(\sum_{i}^{k} f_{i\,u,v}\right) \text{ for } \forall \overset{E}{(u,v)}$$

We are instructing our LP to minimize.
Max( ) is a function which returns the maximum of all of the edges.

(3)

We also need to add constraints.

To constrain to single path

$$f_{i,u,v} = \{d_i, 0\}$$

To constrain to at most h_i edges

$$\sum_u \sum_v f_{iuv} \leq d_i h_i \text{ for } \forall\ i \in k$$

This is because $d_i h_i$ is the sum of the flow over all edges, which is either d_i or 0 for each edge.

Michael Plasmeier
6.046 R07

Quiz 2 #4 ab

Exec Summary

We want an algorithm which finds a way to create a path flow representation for our multicomodity single path routing. I first try to use an augmenting path method, but determine that it is exponential.

I also propose a LP solution which can run in polynomial time.

(2)

Detailed Solution

Traditionally we find paths from s to t by first finding the residual network of G (and a flow f) called G_f (CLRS v3 p 716)

We can then find an augmenting path from s to t (CLRS v3 p 719). Ideally we use BFS (CLRS v3 p 727) to get the shortest augmenting path.

We extend that method for our multicommodity single path routing.

(3)

However which commodity to do first?

I don't think we can establish an order, since greedy does not seem to be acceptable.

This means we need to try each in a different order → which would be exponential.

LP As an alternative we could use LP to solve the problem.

max $\sum_{v \in V} f_{i,sv} - \sum_{v \in V} f_{i,vs}$ for $\forall i$ in k

With the same constraints of the multicomodity flow in CLRS v3 p 863 and the single path constraint from Ya[illegible] (but not the h_i constraint)

(9)

This will give us a path which we can give path variables to

Proof

We can prove that a set of edge flows can always be represented by a set of path flows since they both refer to the same underlying thing.

We just showed two methods for finding a set of edge flows.

Because they both have the same underlying representation we can transform one to another.

(5)

Because the answer we found is polynomial in respect to $|V|$ and $|E|$ since we have shown 1 ~~representation~~ possible edge flow.

Also, there is a limit to the number of possible flows which is polynomial in $|V|$ and $|E|$

In addition, we can use the Recitation 7 notes to take the dual of the LP using path flow variables.

Since we are representing these path flows in the matrix A, we can use the fact ~~that~~ that the constraint from the R7 notes is for each $(u,v) \in E$
└ this one can for each edge

(6)

And one column per path variable

For the LP to be polynomial, the number of columns must be polynomial.

Michael Plasmeier
6.046 R07

Quiz 2 #5a

5a) Exec Summary

Our goal is to make a MST in linear time (# of edges) when we have a fully connected graph where the weights are 1 or 2.

We do this by building two seperate queues one for the 1s and one for the 2s.

This allows us to avoid the prioritized queue which slows down Prim's Algorithm and let us be linear in the number of edges and nodes.

(2)

Detailed solution

Create two queues 1s and 2s and a visited table
Start at the start node, mark in visited table
Add all of the children to the respective queues,
Pull the next node from the 1s, visit, dequeue
If none avilable pull from 2s, visit, dequeue
If ya have already visited, skip it

Psudo code

```
SPECIAL-MST(V, E[redacted], start) {
    1s = {start}
    2s = { }
    visited = {[redacted]}
    while (1s.notempty() and 2s.notempty()) {
        try {
            node=1s.pop() [redacted]while not in visited
        } else {
            node=2s.pop() while not in visited
```

(3)

```
        visited.enqueue(node)
        for each node.children not in visited {
            if child = 1 → 1s.enqueue(child)
            if child = 2 → 2s.enqueue(child)
        }
    }
    return visited;
}
```

Runtime

Creating queues	O(1)
Mark visited/enqueue	O(1)
Add all children edges	O(E)
Pop	O(1) for V nodes
Check in visited	O(1)

since table-takes O(E) memory

Doing above for each edge O(V+E)

Each edge will be added or processed 1 time each

Michael Plasmeier
6.046 R07

Quiz 2 #5b

5b | Exec Summary

We want to be able to estimate the MST of a fully connected graph where all edge weights are $\geq \frac{1}{2}$ and ≤ 2.

I propose a method to take the least $n-1$ edge weights, sum them, and then double that to be above MST(G) but below 2MST(G)

This as allows us to produce an estimate in linear time (# of edges).

(2)

Detailed Solution

We are able to produce an approximation of the answer by selecting the minimum (# of nodes −1) edge weights and summing those.

We can do this in linear time (# of edges) by storing the maximum of the (# nodes −1) minimum points. When we scan the list of edge weights, we will add it to our minimum list if our new point is < stored maximum. This takes $O(1)$, to update the list. Our entire scan takes $O(1)$ for E edges so our estimate is $O(E)$. Summing up the edges takes $O(V)$ but since $E > V$, we have $\boxed{O(E)}$ which meets our criteria. The sum is a.

(3)

To meet the criteria that our estimate b is $MST(G) \leq b \leq 2MST(G)$ we double our estimate a to be b.

$$2a = b.$$

Pseudo code

```
ESTIMATE-MST(V,E) {
    minimum = {}
    for each e ∈ E {
        if e < max(minimum):
            minimum.add(e)
            minimum.remove(max(minimum))
    }
    return (Σ minimum) * 2
```

(4)

Proof

In a fully connected graph, the number of edges is $\sum_{i=1}^{n-1} i$

For example

n	e	
2	1	= 1
3	3	= 1+2
4	6	= 1+2+3
5	10	= 1+2+3+4
6	15	= 1+2+3+4+5

On the first observation, you will observe (n-1) edges that you have not seen before

On the 2nd observation, you will observe (n-2) edges → since there n-1 outgoing edges and 1 you have just seen (since you traveled over it)

(5)

Remember you will take the lowest weight edge that you have not seen before.

But by definition you will not have seen it before, because you would have taken it otherwise since this a is a fully connected graph.

So you need a certain amount to be "large" to avoid leading to a low one.

↑ worst case the low ones will be clustered

At any point we do not have enough "large" (L_2) and "small" ($L_{1/2}$) to allow the prediction to deviate from actual by too large a distance. (L by $\frac{1}{2}$)

This is because the low values will quickly be discovered and used, we can not "save" them for too long by hoping they are not randomly discovered.

Example $\underline{n=5}$

edges = 10 edges in MST = 4

On the first node, we observe 4 edges
2nd node " 3 unique edges

By definition one of these must be in MST.

Then the next 3 must be in the MST.

we will "happen against" the low values too easily"

(2)

The estimate will not be less than $\frac{1}{2}$ actual.

Then when we double the estimate, it will be between MST(G) and 2 MST(G)

Example: if Estimate = 2
Actual = 4

then $4 \leq 4 \leq 8$ ✓

Michael Plasmeier
6.046 R07

10/15

Quiz 2 #1

Red-Blue Subsets

Total $\frac{10}{15} + \frac{10}{20} + \frac{2}{20} + \frac{7}{25} + \frac{15}{20}$

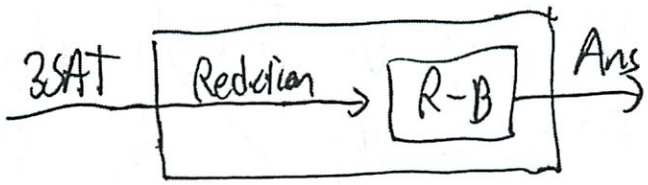
$= \frac{44}{100}$ Avg 66

Exec Summary

We can prove Red Blue Subsets (R-B) is NP-complete by reducing 3SAT $\propto$ R-B
↑ 3CNF SAT

We do this by

1. Showing R-B is in N-P
2. Praiding a redction for 3SAT to R-B



a) Show if $x \in$ 3SAT then R(x) in R-B
b) Show if R(x) $\in$ R-B then $x \in$ 3-SAT

2)

Detailed Solution

1. R-B is in NP; since we can build a verifier V(x, y) where x is the input and y is the potential solution

We check each criteria individually.

1. We can go through the list of nodes in R and B to make sure we "cover" each element in S once - by crossing each off and not having a remainder O(S)

2. We can go through R, S to ensure none is duplicate by creating a tempary 3rd table (size S) where we check off each element in R and ~~an~~ in B. O(S)

(3)

3. We can scan through the list of subsets to insure there is at least 1 R and 1 B in each. O(FBS)

Thus we can provide a certificate in polynomial time

---

2. We can provide a reduction from 3SAT to R-

(We are pretending we have a black box for R-B and are using our R-B black box to solve 3SAT.)

Well first we notice that S is like our set of literals

We notice that F is like our set of clauses.

d)

We can treat R as True and B as False.

Then for each clause we can write it as a series of elements as a Fi

For example $X_1 \vee \overline{X_2} \vee X_3$

Can be written as

$R \vee B \vee R \vee B$

We write true as R

false (or negated as B

We also add an extra $\underline{\vee B}$ at the end.

This is to satisfy condition 3.

This is because each clause must have between 1 → 3 trues. (it can't have 0)

5)

So we add this extra $\vee B$ which helps us achive the condition if the other 3 are red.

We then have a 3SAT <u>if</u> we can write the series of R, Bs such that the conditions hold.

Of course we must make sure we keep our items internally consistent behind the scenes

X_1 is R and $\bar{X}_1$ is B

Or

X_1 is B and $\bar{X}_1$ is R

6)

Proof

1. An input to 3SAT is an input to R-B

This is what we just explained. Each input to 3SAT can be reduced to a R-B problem.

The answer to the R-B problem is an answer to 3-SAT because the conditions are the same.

1. An item must be true or false.
2. " " can not be both true and false.
3. Each clause requires at least 1 true (and we've added a free fals
All clauses must be true.

(7)

2. An input to R-B could be an input to 3SAT.

We can take our R-B and write it as a 3SAT largely reversing our process.

Write them as literals

$(x_1 \vee \bar{x}_1 \vee x_2)$ (drop the always false fake literal

Then we are left w/ a 3SAT.

If we had a R-B that worked, then it would have had at least 1 R, so at least one of our literals would evaluate to true in each clause, so our 3SAT would be true as well.

The converse is true as well (if happens not to be an R in R-B) the clause is false and 3SAT will be false

not quite how reductions work.
see solutions

Michael Plasmeier
6.046 R07

10/20

Quiz 2 #2

Broadcast Channel

Executive Summary

We can use a randomized algorithm to solve this. Each server can independently decide to transmit or not. We can show that the chance of collision decreases as we add machines.

This allows the first message to transmit in O(1) time steps

3)

Detailed Solution

We will have a randomized protocol.
On each time stemp, each machine will independently decide if it should transmit

$$P(\text{transmission}) = \frac{1}{n}$$

$$P(\text{no transmission}) = 1 - \frac{1}{n}$$

The prior history of collisons is not a factor.

Let us assume we have all n machines
(less than n machines is just a weaker case where the E[ ] time steps is lower)

The probability of a collission is the probability that two or more machines transmit at once.

(3)

For 2 machines this is

$$\frac{1}{n} \cdot \frac{1}{n} = \left(\frac{1}{n}\right)^2$$

We could have 3 machines transmit at once

$$\left(\frac{1}{n}\right)^3$$

So we must ~~read~~ add all of these chances together

$$P(\text{any collision in 1 time step}) = \sum_{i=2}^{n} \left(\frac{1}{n}\right)^i$$

This is

$$\frac{1}{(n-1)n}$$

n	2	3	4	5
P(collision)	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{20}$

Now we must consider the probability of 2 collisions in a row.

This is $P(\text{any collision in 1 time step}) \cdot P(\text{any collision in 1 time step})$

or $P(\text{any collision in 1 time step})^2$

(4)

We must consider the probability of any # of collisions

$$\sum_{i=1}^{\infty} i \cdot P(\text{any collision at the } i\text{th spot})$$

This is

$$\frac{1}{n^2 - n - 1}$$

n	3	4	5
$P(\text{any collisions})$	$\frac{1}{5}$	$\frac{1}{11}$	$\frac{1}{19}$

~~It's Con~~

I'm confusing what is n

So as n grows, our $P(\text{collisions})$ shrinks. Thus our protocol is $O(1)$

(5)

The expected # of steps is

$$E[\#\text{ steps}] = 0 \cdot P(0 \text{ collisions}) + 1 \cdot P(1 \text{ collision})$$

why stop here with 2 collisions? ← 3, 4, ...

$$+ 2 \cdot P(2 \text{ collisions}) =$$

$$= 0 + 1 \cdot \sum_{i=2}^{\infty} \left(\frac{1}{n}\right)^i + 2 \cdot \left[\sum_{i=2}^{\infty} \left(\frac{1}{n}\right)^i\right]^2 + \cdots$$

This is not the prob of 3 collisions

$$= 0 + \underline{1} \cdot \left[\left(\frac{1}{n}\right)^2 + \left(\frac{1}{n}\right)^3 + \left(\frac{1}{n}\right)^4\right)$$

$$+ 2 \left(\left(\frac{1}{n}\right)^2 + \left(\frac{1}{n}\right)^3 + \left(\frac{1}{n}\right)^4\right)^2 + \cdots$$

with $n = 4$

$$= 0 + 1 \cdot \frac{21}{256} + 2 \cdot \frac{441}{6536} + \cdots$$

$$= .0954$$

↑ rounding approx

↑ ~~approx~~ add 1 for 1.0954

1 step should be good

Michael Plasmeier
6.046 R07

Quiz 2 #3

a. 1/3 b: 0/6 c: 0/6 d: 1/5

---

k-Hashing

$\frac{1}{3} + \frac{0}{6} + \frac{0}{6} + \frac{1}{5} = \frac{2}{20}$

a) Exec Summary

We can prove to have good randomness by proving an upper bound (given) on a collision probability.

Detailed Solution

Normally the probability of a collision is $\frac{1}{m}$

If we have n items it is $\frac{n}{m}$

If we try twice, the probability both have a collision is $\frac{n}{m} \cdot \frac{n}{m} = \left(\frac{n}{m}\right)^2$

We are told $m = cn^{1.5}$

So let's prove

$$\left(\frac{n}{cn^{1.5}}\right)^2 < \frac{1}{c^2}$$

2)

$$\left(\frac{1}{cn^{.5}}\right)^2 < \frac{1}{c^2}$$

see solution.

$$\frac{1}{c^2 n} < \frac{1}{c^2}$$

Since $n > 0$, it will always make the left side smaller so proved ▨

b) Exec Summary

We are asked to give a better lower bound. We can just answer what the probability is exactly.

Detailed Solution

The probability of collision is $\frac{1}{c^2 n}$

This can also be our upper and lower bound on the probability of collision.

Note n is given exogenously, so we can do this.

(3)

c) Exactive Summary

We can find a function $\varepsilon(k)$ so that k is monotioncally decreasing so the probablity of a collision is less than $\frac{1}{c^k}$

Detailed Solution

Generally our probablity of collision is $\left(\frac{n}{m}\right)^k$ and $m = cn^{1+\varepsilon(k)}$

So $\left(\frac{n}{cn^{1+\varepsilon(k)}}\right)^k < \frac{1}{c^k}$

$\left(\frac{1}{cn^{\varepsilon(k)}}\right)^k < \frac{1}{c^k}$

$\frac{1}{c^k n^{\varepsilon(k)k}} < \frac{1}{c^k}$

So $n^{\varepsilon(k)k} > 1$ must be

(4)

We know $n > 0$

$k \geq 1$

So $\varepsilon(k)$ must be > 1

and we want it to be monotonically decreasing

So $\varepsilon(k) = 1 + \frac{1}{k}$ meets that criteria

As $k \uparrow$, $\varepsilon(k) \downarrow$ monotonically

So back to our constraint

$$\frac{1}{c^k n^{\varepsilon(k)k}} < \frac{1}{c^k}$$

$$\frac{1}{c^k n^{(1+1/k)k}} < \frac{1}{c^k}$$

$$\frac{1}{c^k n^{(k+1)}} < \frac{1}{c^k}$$

$\forall n^{k+1} > 1$ so bound holds

5)

d) Executive Summary

Compared to the two stage hashing shown in lecture, they both have the same runtime, they depend on case for space usage, and make different assumptions

Detailed Solution

Perfect hashing means $O(1)$ memory accesses to look up a value.

We can have that when $E[\text{collision}] < \frac{1}{2}$

Universal is if $P[\text{collision}] \leq \frac{1}{m}$

Space usage

Class : $2n$ Though it varies

This : $n^{1+\epsilon}$ fixed

2)

For a very small ϵ, $n^{1+\epsilon}$ is actually smaller.

For example $n^{1.01}$ is better up to 2^{100} entries

run time

both are $O(1)$ for accessing w/ 2 look ups

7

Use cases

Original requires ya to take the mod of
this does not - allowing one to be
more flexible on the input type

assumptions

Before: prob of collision $\leq \frac{1}{2}$

Here: " " " $\leq \frac{1}{c^k}$

Before $2n$ is expected storage and
$p < \frac{1}{2}$ that storage $\geq 4n$

Michael Plasmeier
6.046 R07

Quiz 2 #4a

7/25

Multicommodity Single Path Rating

6 Exec Summary

This answer builds upon the multi-commodity flow Linear Program (LP) shown in CLRSv3 p863 in section 29.2 by adding an objective and the constraints that only a single path be used and we only use h_i edges.

We can represent it as an ILP which is defined to be NP-hard.

2)

Detailed Solution

Write these constraints out so we know what has to change

Start with multicommodity flow in CLRS v3 p863.

Add the objective → Not a valid LP fn. → 2

minimize $\max\left(\sum_{i}^{k} f_{i\,u,v}\right)$ for $\forall (u,v) \in E$

We are instructing our LP to minimize. Max( ) is a function which returns the maximum of all of the edges.

3)

We also need to add constraints.

To constrain to single path

$$f_{i,u,v} = \{d_i, 0\}$$

Need to use variables $\in \{0, 1\}$ to make this into an explicit constraint. -1

To constrain to at most h_i edges

$$\sum_u \sum_v f_{iuv} \leq d_i h_i \quad \forall\, i \in k$$

This is because $d_i h_i$ is the sum of the flow over all edges, which is either d_i or 0 for each edge.

Michael Plasmeier
6.046 R07

Quiz 2 #4 a b

Not really what 4(b) is asking for...

1 Exec Summary

We want an algorithm which finds a way to create a path flow representation for our multicommodity single path routing. Where I first try to use an augmenting path method, but determine that it is exponential.

I also propose a LP solution which can run in polynomial time.

2)

Detailed Solution

Traditionally we find paths from s to t by first finding the residual network of G (and a flow f) called G_f (CLRS v3 p 716)

We can then find an augmenting path from s to t (CLRS v3 p 719). Ideally we use BFS (CLRS v3 p 727) to get the shortest augmenting path.

We extend that method for our multicommodity Single path routing.

3)

However which commodity to do first?

I don't think we can establish an order, since greedy does not seem to be acceptable.

This means we need to try each in a different order → which would be exponential.

LP As an alternative we could use LP to solve the problem.

max $\sum_{v \in V} f_{is} - \sum_{v \in V} f_{ivs}$ for $\forall i$ in k

With the same constraints of the multicomodity flow in CLRS v3 p 863 and the single path constraint from 4a [but not the h_i constraint]

4)

This will give us a path which we can give path variables to

Proof

We can prove that a set of edge flows can always be represented by a set of path flows since they both refer to the same underlying thing.

We just showed two methods for finding a set of edge flows.

Because they both have the same underlying representation we can transform one to another.

(5)

Because the answer we found is polynomial in respect to $|V|$ and $|E|$ since we have shown 1 ~~representation~~ possible edge flow.

Also, there is a limit to the number of possible flows which is polynomial in $|V|$ and $|E|$

In addition, we can use the Recitation 7 notes to take the dual of the LP using path flow variables.

Since we are representing these path flows in the matrix A, we can use the fact ~~that~~ that the constraint from the R7 notes is for each $(u,v) \in E$
Thus so can for each edge

(6)

And one column per path variable

For the LP to be polynomial, the number of columns must be polynomial.

Michael Plasnoie
6.046 R07

15/20

Quiz 2 #5a

5a) Exec Summary

Our goal is to make a MST in linear time (# of edges) when we have a fully connected graph where the weights are 1 or 2.

We do this by bilding two seperate queues one for the 1s and one for the 2s.

10

This allows us to avoid the prioritized queue which slows down Prim's Algorithm and let us be linear in the number of edges and nodes.

(2)

Detailed solution

Create two queues 1s and 2s and a visited table
Start at the start node, mark in visited table
Add all of the children to the respective queues,
Pull the next node from the 1s, visit, dequeue
If none avilable pull from 2s, visit, dequeue
If you have already visited, skip it

Psudo code

```
SPECIAL-MST(V, E[illegible], start) {
    1s = {start}
    2s = { }
    visited = {[illegible]}
    while (1s.notempty() and 2s.notempty()){
        try {
         node=1s.pop() [illegible]while not in visited
        } else {
         node=2s.pop() while not in visited
```

(3)

```
        visited.enqueue(node)
        for each node.children not in visited {
            if child = 1 → 1s.enqueue(child)
            if child = 2 → 2s.enqueue(child)
        }
    }
    return visited;
}
```

Runtime

Creating queues	O(1)
Mark visited/enqueue	O(1)
Add all children edges	O(E)
Pop	O(1) for V nodes
Check in visited	O(1)

since table-takes O(V) memory

Doing above for each edge O(V+E)

Each edge will be added or processed 1 time each

Michael Plasmeier
6.046 R07

Quiz 2 #5 6b

5b | Exec Summary

We want to be able to estimate the MST of a fully connected graph where all edge weights are $\geq \frac{1}{2}$ and ≤ 2.

I propose a method to take the lowest $n-1$ edge weights, sum them, and then double that to be above MST(G) but below 2MST(G)

(5)

This is allows us to produce an estimate in linear time (# of edges).

(2)

Detailed Solution

We are able to produce an approximation of the answer by selecting the minimum (# of nodes −1) edge weights and summing those.

We can do this in linear time (# of edges) by storing the maximum of the (# nodes −1) minimum points. When we scan the list of edge weights, we will add it to our minimum list if our new point is > stored maximum. This takes $O(1)$, to update the list. Our entire scan takes $O(1)$ for E edges so our estimate is $O(E)$. Summing up the edges takes $O(V)$ but since $E > V$, we have $\boxed{O(E)}$ which meets our criteria. The sum is a.

(3)

To meet the criteria that our estimate b is $MST(G) \leq b \leq 2MST(G)$ we double our estimate a to be b.

$$2a = b.$$

Pseudo code

```
ESTIMATE-MST(V,E) {
    minimum = {}
    for each e ∈ E {
        if e < max(minimum):
            minimum.add(e)
            minimum.remove(max(minimum))
    }
    return (∑ minimum) · 2
```

What if graph looks like this:

$n-\sqrt{n}$ nodes connected by cycle where each edge has wt 2

$\sqrt{n}$ nodes with all $\binom{\sqrt{n}}{2}$ edges of wt $1/2$

This is off by almost a factor of 4!

(4)

Proof

In a fully connected graph, the number of edges is $\sum_{i=1}^{n-1} i$

For example

n	e	
2	1	= 1
3	3	= 1+2
4	6	= 1+2+3
5	10	= 1+2+3+4
6	15	= 1+2+3+4+5

On the first observation, you will observe (n-1) edges that you have not seen before

On the 2nd observation, you will observe (n-2) edges → since there n-1 outgoing edges and 1 you have just seen (since you traveled over it)

(5)

Remember you will take the lowest weight edge that you have not seen before.

But by definition you will not have seen it before, because you would have taken it otherwise since this a is a fully connected graph.

So you need a certain amount to be "large" to avoid leading to a low one.

↑ worst case the low ones will be clustered

At any point we do not have enough "large" and "small" to allow the prediction
L_2 $L_{1/2}$

to deviate from actual by too large a distance.
$L_{by\ \frac{1}{2}}$

This is because the low values will quickly be discovered and used, we can not "save" them for too long by hoping they are not randomly discovered.

Example $n=5$

edges = 10 edges in MST = 4

On the first node, we observe 4 edges
2nd node " 3 unique edges

By definition one of these must be in MST.
Then the next 3 must be in the MST.

we will "happen against" the low values too easily"

(2)

The estimate will not be less than $\frac{1}{2}$ actual.

Then when we double the estimate, it will be between MST(G) and 2 MST(G)

Example: if Estimate = 2
Actual = 4

then $4 \leq 4 \leq 8$ ✓

Solutions

Design and Analysis of Algorithms
Massachusetts Institute of Technology
Profs. Srini Devadas and Ronitt Rubinfeld

Due November 14, 2012
6.046J/18.410J
Quiz 2

Quiz 2
Cover Sheet

Problem	Title	Points	Grade	Initials
1	Red-Blue Subsets	15		
2	Broadcast Channeling	20		
3	Hashing	20		
4	Multicommodity Single-path Routing	25		
5	Minimum Spanning Tree	20		

Name: _____

INSTRUCTIONS

This take-home quiz contains 4 problems worth a total of 100 points. Your quiz solutions are due **at 11:59pm on Wednesday, November 14, 2012.** Late quizzes will not be accepted unless you obtain a Dean's support or make prior arrangements with the course staff. You must submit your solutions electronically.

Guide to this quiz: For problems that ask you to design an efficient algorithm for a certain problem, your goal is to find the *most efficient algorithm possible*. Generally, the faster your algorithm, the more points you receive. For two asymptotically equal bounds, worst-case bounds are better than expected or amortized bounds. The best possible solution will receive full points if well written, but ample partial credit will be given for correct solutions, especially if they are well written. Bonus points may be awarded for exceptionally efficient or elegant solutions.

Plan your time wisely. Do not overwork, and get enough sleep. Your very first step should be to write up the most obvious algorithm for every problem, even if it is exponential time, and then work on improving your solutions, writing up each improved algorithm as you obtain it. In this way, at all times, you have a complete quiz that you could hand in.

Policy on academic honesty: The rules for this take-home quiz are like those for an in-class quiz, except that you may take the quiz home with you. As during an in-class quiz, you may not communicate with any person except members of the 6.046 staff about any aspect of the quiz, even if you have already handed in your quiz solutions. To communicate with the staff, you have to send an email to `6046-tas@csail.mit.edu`. We will not respond to your question if you send an email to the personal email of a staff member. In addition, you may not discuss any aspect of the quiz with anyone except the course staff **until November 23, 2012**. Note that this date is **after** the due date of the quiz.

This take-home quiz is "limited open book." You may use your course notes, the CLRS textbook, and any of the materials posted on the course web page, but *no other sources whatsoever may be consulted.* For example, you may not use notes or solutions to problem sets, exams, etc. from other times that this course or other related courses have been taught. **You may not use any materials on the World-Wide Web, including OCW.** You probably won't find information in these other sources that will help directly with these problems, but you may not use them regardless.

If at any time you feel that you may have violated this policy, it is imperative that you contact the course staff immediately. If you have any questions about what resources may or may not be used during the quiz, please send email to `6046-staff@csail.mit.edu`.

Write-ups: Type your answers up on separate files (the templates will be given), and submit all your answers electronically, as you would do in the problem set. We *strongly encourage* you to submit your problems typed in LaTeX, since this makes it easier for us to read your solutions and understand them.

Your solutions are due, electronically, at 11:59pm on Wednesday the 14th of November. There is no hardcopy submission option for this test.

Your write-up for a problem should start with a topic paragraph that provides an executive summary of your solution. This executive summary should describe the problem you are solving, the techniques you use to solve it, any important assumptions you make, and the asymptotic bounds on the running time your algorithm achieves, including whether they are worst-case, expected, or amortized.

Write your solutions cleanly and concisely to maximize the chance that we understand them. When describing an algorithm, give an English description of the main idea of the algorithm. Adopt suitable notation. Use pseudocode if necessary to clarify your solution. Give examples, draw figures, and state invariants. A long-winded description of an algorithm's execution should not replace a succinct description of the algorithm itself. *Points will be taken off for overly convoluted solutions to the problems.*

Provide short and convincing arguments for the correctness of your solutions. Do not regurgitate material presented in class. Cite algorithms and theorems from CLRS, lecture, and recitation to simplify your solutions. Do not waste effort proving facts that can simply be cited.

Be explicit about running time and algorithms. For example, don't just say that you sort n numbers, state that you are using MERGE-SORT, which sorts the n numbers in $O(n \lg n)$ time in the worst case. If the problem contains multiple variables, analyze your algorithm in terms of all the variables, to the extent possible.

Part of the goal of this quiz is to test your engineering common sense. If you find that a question is unclear or ambiguous, make reasonable assumptions in order to solve the problem, and state clearly in your write-up what assumptions you have made. Be careful what you assume, however, because you will receive little credit if you make a strong assumption that renders a problem trivial.

Bugs: If you think that you've found a bug, send email to `6046-staff@csail.mit.edu`. Corrections and clarifications will be sent to the class via email and posted on the class website. Check your email and the class website daily to avoid missing potentially important announcements.

Good Luck!

Problem 1. Red-Blue Subsets [15 points] (1 parts)

Let S be a set of n elements $\{1, 2, \ldots, n\}$, and let F be a set of subsets of S. That is, for each $F_i \in F$, $F_i \subseteq S$ and $|F_i| \geq 2$. A Red-Blue coloring of S exists if each element in S can be assigned red or blue such that the set of red elements R and blue elements B satisfy:

1. R and B cover S.
2. R and B are disjoint.
3. Each subset $F_i \in F$ has at least one red and one blue element.

Prove that it is NP-complete using a reduction from 3-SAT to Red-Blue Subsets.

Solution:

In NP: We first prove it's in NP by giving a verifier $V(x, y)$.

Let $x = (S, F)$ and let $y = (R, B)$. First, check that $|R|$ and $|B|$ are at most n and check that each element in R and B are in S. This takes $O(n^2)$ time. Then, check that R and B cover S and R and B are disjoint by looping over the elements in S and looking for each element in R and B. If the element is not in either, then R and B do not cover S. If the element is in both, then the sets are not disjoint. This check takes $O(n^2)$ time. Finally, loop over each $F_i \in F$ and determine if there is an element $r \in F_i$ such that $r \in R$ and an element $b \in F_i$ such that $b \in B$. Each subset takes linear time in the size of the subset to check $|F_i| = O(n)$, so total, this takes $O(n|F|)$ time. $O(n^2 + n|F|)$ is polynomial in the size of the input, and thi proves that Red-Blue Subsets is in NP.

In NP-Hard: We prove that it is NP-hard by reducing 3-SAT to Red-Blue Subsets.

Let X be the set of variables and C be the set of clauses in the 3-SAT problem. Build S such that $S = \{x_i | x_i \in X\} \cup \{\overline{x_i} | x_i \in X\} \cup \{f\}$. This takes $O(|X|)$ time.

For each clause $C_j \in C$, add F_j that contains the corresponding literals and the element f. For example, the clause $C_j = x_1 \vee x_2 \vee \overline{x_3}$ produces the subset $F_j = \{x_1, x_2, \overline{x_3}, f\}$. This takes $O(|C|)$ time.

For each variable $x_i \in X$, create the subset $F_i = \{x_i, \overline{x_i}\}$. This takes $O(|X|)$ time. Total, this construction takes $O(2|X| + |C|)$ time, which is polynomial in the size of the 3-SAT instance.

Now we prove the following claim: there is a solution to the 3-SAT problem if and only if there is a solution to the Red-Blue Subset problem.

($\Rightarrow$) Prove there is a solution to 3-SAT if there is a solution to Red-Ble Subset. Let all the literals that are colored the same color as f be set to false. Let all the other literals be set to true. Each x_i will only have one assignment since the set $\{x_i, \overline{x_i}\}$ means that the two elements will be colored differently. Each F_j that corresponds with a clause C_j must have at least one element with color opposite of that of f, which means that the corresponding literal would be true. Thus, each clause is satisfied, which means that the 3-SAT is satisfied.

($\Leftarrow$) Prove there is a solution to 3-SAT only if there is a solution to Red-Blue Subset. Let R be the set containing f, x_i if x_i is set to false, and $\overline{x_i}$ if x_i is set to true. Let B contain the rest. In general, this corresponds to the R set being false and the B set being true. Since each clause is satisfied, each F_j that corresponds to a C_j will have at least one blue element (so that the clause evaluates to true) as well as at least one red element (f). Each F_i that corresponds to a variable x_i will have one red and one blue element since x_i and $\overline{x_i}$ cannot have the same truth value. Since no literal can be both true and false at the same time and each literal must have a truth value, the sets R and B cover S and are disjoint. Thus, this is a valid instance of the Red-Blue Subset problem.

This proves 3-SAT reduces to Red-Blue Subsets. Therefore, Red-Blue Subsets is in NP-Hard. Since Red-Blue Subsets is in NP and NP-Hard, it is in NP-Complete.

Problem 2. Broadcast Channel [20 points] (1 parts)

A set of up to n processors attempt to communicate over a network. The communication process is deemed successful if *any* of the processors manages to broadcast its information (since the successful processor can then lead the remainder of the communication process). However, the only means of communication is through a common broadcast channel. At any given time step (we assume the time is discrete), any subset of the processors can attempt to communicate through the channel by sending a message. The channel operates as follows:

- If *none* of the processors attempts to send a message, then all processors receive a special "none" message.
- If *only one* of the processors attempts to send a message, then all processors receive that message, and the communication process is deemed successful.
- If *two or more* processors attempt to send a message, then all processors receive a special "collision" message.

Suppose that the number of processors is at least $n/2$. Design a randomized protocol that, if followed by all processors, will result in successful communication in expected constant time, i.e., the expected number of time steps used by the protocol should be $O(1)$. Give an analysis of the expected number of time steps used by your protocol.

You can assume all processors know the upper bound n and the lower bound $n/2$ on the total number of processors.

Solution: We assume $n > 1$, since otherwise the problem is trivial. The algorithm is as follows: at each time step, each processor sends its message with probability $\frac{1}{n}$. If exactly one of the processors manages to broadcast its message, the whole process stops. Otherwise, the protocol is repeated in the next time step.

The analysis is as follows: we will prove that the expected number of time steps used by the protocol is constant. Since each processor sends a message with probability $\frac{1}{n}$, the number of messages sent in each time step follows the binomial distribution. In particular, if there are k processors, the probability that exactly one message is sent at a given time step is

$$P = \binom{k}{1}\frac{1}{n}(\frac{n-1}{n})^{k-1} \geq 1/2 \cdot (1-1/n)^{k-1} \geq 1/2 \cdot (1-1/n)^{n},$$

Since $(1-1/n)^n \to 1/e$ as $n \to \infty$ and $(1-1/n)^n > 0$ for $n > 1$, it follows that $(1-1/n)^n \geq \delta$ for some absolute constant $\delta > 0$ (in fact, we can take $\delta = 1/4$). This implies that $P \geq \delta/2$. The expected number of time steps used by the protocol is at most $1/P \leq 2/\delta = O(1)$.

Problem 3. k**-Hashing** [20 points] (2 parts)

A k-hash function maps each key in a universe U of all possible keys to k possible slots in the given table. Suppose you have an access to a perfectly random k-hash functions, i.e. you can sample a function h_k such that for each key from U, h_k selects k slots from the table uniformly at random with replacement. Thus, when inserting a key into the table, you can choose to insert the key into the least-occupied slot out of k possible slots. When looking up the key, check all k possible slots. If k is constant, then the access time will be $O(1)$.

For a set of n keys from the universe U, we would like to have no collisions when inserting those keys into the table of size $O(n^{1+\epsilon})$, where $\epsilon > 0$ is some acceptably small constant.

(a)Show that for $\epsilon = .5$, we can achieve the goal with a good probability using a random 2-hash function. Particularly, show that when using a random 2-hash function to insert n items into the table of size $cn^{1.5}$, the probability of a collision is less than

$$\frac{1}{c^2}$$

Solution: [3 points] Let P_i equal the probability that there is a collision when inserting the ith item. Since there are maximum $i-1$ slots that are occupied, the probability that both candidate locations are occupied is at most

$$(\frac{i-1}{cn^{1.5}})^2 = \frac{(i-1)^2}{c^2n^3}$$

Notice that probability of any collision occurring while inserting n items equals to

$$Pr[P_1 \cup P_2 \cup \cdots \cup P_n] \leq \sum_{i=1}^{n} Pr[P_i]$$

by the union bound (Boole's inequality). We get that the probability of any collision occurring is bounded by

$$\sum_{i=1}^{n} \frac{(i-1)^2}{c^2n^3} \leq \sum_{i=1}^{n} \frac{n^2}{c^2n^3} = \frac{1}{c^2}$$

(b)Give a better (smaller) upper bound on the probability of a collision when using a random 2-hash function to insert n items into a table of size $cn^{1.5}$

Solution: [6 points] To get a tighter bound, it was enough to use the sum of squares formula given in CLRS, Appendix A, formula A.3, p 1147. We showed that the probability of a collision occurring is bounded by

$$\sum_{i=1}^{n} \frac{(i-1)^2}{c^2n^3} = \frac{1}{c^2n^3} \sum_{i=1}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6c^2n^3} \leq \frac{2n^3}{6n^3c^2} = \frac{1}{3c^2}$$

Students who showed this bound got full points. Students who showed worse bounds got less points. Students who showed bounds that involved n got less points even if the bound was tighter. Students who didn't provide sufficiently clear and detailed explanation got less points.

(c)We would like to have a smaller ϵ and a better chance of avoiding a collision. Show that it's possible by generalizing your result to k-hash functions. Find a function $\epsilon(k)$, monotonically decreasing in k, such that when using a k-hash function to insert n items into the table of size $cn^{1+\epsilon(k)}$, the probability of a collision is less than

$$\frac{1}{c^k}$$

Solution: [6 points] Consider function

$$\epsilon(k) = \frac{1}{k}$$

It is monotonically decreasing in k. As in previous parts, let P_i equal the probability that there is a collision when inserting the ith item. Since there are maximum $i-1$ slots that are occupied, the probability that a single slot chosen by k-hash function is occupied is as most

$$\frac{i-1}{cn^{1+\frac{1}{k}}}$$

k-hash function choses k slots independently with replacement. Thus,

$$Pr[P_i] \leq \left(\frac{i-1}{cn^{1+\frac{1}{k}}}\right)^k \leq \frac{n^k}{c^k n^{k+1}} = \frac{1}{nc^k}$$

The probability of any collision occurring while inserting n items equals to

$$Pr[P_1 \cup P_2 \cup \cdots \cup P_n] \leq \sum_{i=1}^{n} Pr[P_i]$$

by the union bound (Boole's inequality). We get that the probability of any collision occurring is bounded by

$$\sum_{i=1}^{n} \frac{1}{c^k n} = \frac{1}{c^k}$$

Students who showed this or a better bound got full points. Students who didn't provide sufficiently clear and detailed explanation got less points.

(d)Conclude that you can use k-hash functions to achieve a perfect hashing. Compare it to the perfect hashing studied in class in terms of

–space usage: which one uses more space?

–use cases: give examples when one is better to use than the other;

–assumptions: how do the assumptions differ?

Solution: [5 points] There are many things that can be said about k-hashing. At the very least, you had to mention that $\frac{1}{c^k}$ is minuscule even for small values of c and k. For example, if $k = 100$ and $c = 2$, then $\frac{1}{2^{100}}$ is practically zero. Therefore, if the keys are not known in advance, in practice, it is possible to achieve the perfect hashing with a very high probability. Since the search takes $O(k)$ time, it's important to keep k constant. Note that in k-hashing, we didn't make the assumption that the keys are static.

When comparing to the perfect hashing studied in class, you had to mention the following differences:

–space usage: The total space used by the in class perfect hashing is $O(n)$ while the space used by k-hashing is $O(n^{1+\frac{1}{k}})$ where k is a positive integer constant. Therefore, k-hashing uses more space.

–use cases: The in class perfect hashing works only if the keys are static. If the keys are known in advance, use the in-class perfect hashing. If the keys are not known, use k-hashing. When keys are known, the in class hashing is much better in terms of the access time and the space usage.

–assumptions: The in-class perfect hashing assumes universal hash family functions while k-hash functions assume a random hash function family.

Here are other good points one can make.

The in class perfect hashing needed $O(n)$ different hash functions for the second level hashing.

For k-hashing, the expected number of collisions is

$$\sum_{i=1}^{n} Pr[P_i] < \frac{1}{c^k}$$

If we use chaining to resolve the collisions, then the expected access time will be $O(1)$ when k is constant because the expected number of collisions is $O(1)$. This strategy is better than rehashing everything when a collision occurs.

Problem 4. Multicommodity Single-path Routing [25 points] (2 parts)

We are given a graph $G(V, E)$ in which each edge $(u, v) \in E$ has nonnegative capacity $c(u, v) \geq 0$. We are also given a set of k different commodities, $K_1, ..., K_k$, and commodity i is specified by the triple $K_i = (s_i, t_i, d_i)$, where s_i and t_i are its source and sink, respectively, and d_i is the demand, i.e., the amount required to flow from s_i to t_i. Different commodities may share the same source or destination, but for each commodity, we assume that $s_i \neq t_i$. Moreover, each commodity i can only flow along a single path from s_i to t_i.A flow that satisifes these properties is called a *multicommodity single-path routing*.

(a) One way to model a flow network, as presented in CLRS 29.2, is to use *edge flows* $\{f_i(u, v)\}$ as decision variables, where $f_i(u, v)$ represents the flow each commodity i along the edge $(u, v) \in E$. However, we have an additional constraint that each commodity can only flow through a single path from its source s_i to t_i. Further, we are given additional constraints that the path with which to send commodity i cannot contain more than h_i edges.

The MIN-LARGEST-EDGE-LOAD problem then seeks the multicommodity single-path routing for which the largest *edge load* among all edges in the graph is minimized, where the edge load of $(u, v) \in E$ is defined as the total flow along (u, v).

Formulate MIN-LARGEST-EDGE-LOAD as an integer linear program (where some of the decision variables are constrained to be integers).

Solution:

$$\min_{L, x_i(u,v)} \quad L$$

$$\text{subject to} \quad L \geq \sum_{i=1}^{k} d_i x_i(u, v) \quad \forall u, v \in V \tag{1}$$

$$\sum_{i=1}^{k} d_i x_i(u, v) \leq c(u, v) \quad \forall u, v \in V \tag{2}$$

$$\sum_{v \in V} x_i(u, v) - \sum_{v \in V} x_i(v, u) = 0 \quad \forall i = 1, ..., k, u \in V - \{s_i, t_i\} \tag{3}$$

$$\sum_{v \in V} x_i(s_i, v) = 1 \quad \forall i = 1, ..., k \tag{4}$$

$$\sum_{v \in V} x_i(v, s_i) = 0 \quad \forall i = 1, ..., k \tag{5}$$

$$\sum_{u,v \in V} x_i(u, v) \leq h_i \quad \forall i = 1, ..., k \tag{6}$$

$$x_i(u, v) \in \{0, 1\} \quad \forall i = 1, ..., k, u, v \in V \tag{7}$$

The decision variables are $\{L, x_i(u, v)\}$. Each $x_i(u, v)$ is constrained to be binary in (7), and is 1 if and only if the edge (u, v) if (u, v) is used in the path from s_i to t_i,

i.e., if u, v bears flow d_i. The set of constraints (1), along with the objective function, ensures that L is the maximum edge load.

Edge capacity constraints are given in (2).

Flow conservation constraints (3) ensure that the number of incoming edges with flow match that of outgoing edges with flow at each intermediate node. Constraints (4) and (5) restrict each s_i to having only one outgoing edge with flow, and no incoming edges with flow. Together, (3)-(5) ensure that exactly one path from s_i to t_i is chosen.

Finally, (6) constrains the number of edges on the path from s_i to t_i.

Comments on grading:

- 1 point for recognizing this was a flow maximization problem (everybody got this if they at least tried)
- 2 points for correctly defining the variables with constraints (i.e. ILP)
- 2 points for the the objective function
- 5 points for each of the following constraints:
  - 3 flow requirements (positive flows, flow in = flow out, capacity)
  - 1 for path length
  - 1 for demands met

Therefore, most people got at least 3 points, and usually 5 points for copying the multicommodity flow from the book (so a free 5 points for reading the problem). The other 5 points were gained from defining binary indicator variables, defining a maximum load variable, and adding the two extra constraints.

Note that because there were two ways to do this (using only binary variables or using a mix of edge flows and binary variables) then there may be a few more constraints that you needed to get right (namely that the flow on an edge is equal to the demand times the indicator variables).

(b) Alternatively, as seen in Recitation 7, we can use *path flows* $\{f_i(p)\}$ as decision variables, where $f_i(p)$ represents the flow of commodity i along a path $p \in P_i$, and P_i is the set of all possible paths from s_i to t_i.

Show that for a multicommodity flow, a set of edge flows can always be represented by a set of path flows, and the number of path flow variables required is polynomial with respect to $|V|$ and $|E|$. Do so by giving an algorithm that achieves this, and analyze its running time.

Hint: You may wish to review the algorithm for finding augmenting paths.

Solution:

Executive Summary:

We give an algorithm that constructs path flow variables $f_P = \{f_i(p)\}$ from edge flow variables $f_E = \{f_i(u,v)\}$ in polynomial time, producing a polynomial number of variables in f_P. Here, $f_i(u,v)$ represents the flow of commodity i on edge (u,v), as in CLRS.

Our proposed algorithm, which we call EDGE-TO-PATH, iteratively searches for a path p from s_i to t_i for each i, sets $f_i(p)$ to the smallest flow along that path, and decreases f_E along each edge on this path by $f_i(p)$. Since each iteration sets the flow of at least one edge in f_E to zero, the number of path variables required is $O(kE)$. Remaining cycles can be accounted for by these selected paths. As we shall see, this algorithm runs in $O(k|V||E|^2)$ time.

Detailed Explanation:

The pseudocode for EDGE-TO-PATH is given as follows.

```
EDGE-TO-PATH(f_E)
 1  f_P = φ
 2  for i = 1 to k
 3      while FIND-NEXT(f_E, i, s_i, φ) ≠ φ // There exists a path from s_i to t_i
 4          p = φ    // p will be the path added
 5          x = ∞    // x will represent the amount of flow along path p
 6          u = s_i
 7          v = FIND-NEXT(f_E, i, u, p)
 8          while v ≠ t_i
 9              p = p ∪ {(u, v)}
10              x = min{x, f_i(u, v)}
11              u = v
12              v = FIND-NEXT(f_E, i, u, p)
13          f_P = f_P ∪ {(i, p, x)}    // (i, p, x) is an alternative representation of f_i(p) = x
14          for each (u, v) ∈ p
15              f_i(u, v) = f_i(u, v) − x
16      f_P = CLEAR-CYCLES(i, f_E, f_P)
17  return f_P
```

Lines 3 to 12 finds a path p from s_i to t_i by a depth-first approach (except it always succeeds and never backtracks as in a general DFS). This is achieved using the subroutine FIND-NEXT(f_E, i, u, p), which finds and returns (if exists) a node that has some edge flow from u that is not yet designated to any paths, while avoiding edges that have already been selected to be in p, the path that is currently being constructed (so as to prevent endlessly cycling). In other words, it returns a node v such that $f_i(u, v) \neq 0$ and $(u, v) \notin p$, or returns ϕ if such a node does not exist.

Note that each iteration of the for loop in line 14 sets at least one edge to 0, since x is equal to the flow of at least one edge (namely, the edge(s) with the smallest flow). Therefore, the total number of iterations of the while loop in 3, per commodity, cannot exceed the number of edges $|E|$. The number of path variables in f_P is thus $O(|E|)$.

Upon exiting the while loop of line 3, there are no more edges in f_E that carry flow out of s_i. Moreover, there are no (non-cycle) paths in f_E, since lines 14-15 preserves flow conservation. However, there may still be cycles remaining in f_E, as illustrated in Figures 1 and 2.

To take care of this, we use the subroutine CLEAR-CYCLES, which looks for remaining cycles in f_E, and appends those cycles to paths in f_P. More specifically, we can implement CLEAR-CYCLES as follows:

CLEAR-CYCLES(i, f_E, f_P)

```
1  Find the set C = {(c, y) | c is a collection of edges that form a cycle in f_E with flow y > 0}
2  for each (c, y) ∈ C
3      for each node u on cycle c
4          while u is on path p for some (i, p, x) ∈ f_P
5              if y ≥ x
6                  p = p ∪ c
7                  y = y − x
8                  if y = 0 goto line 2
9              else
10                 f_P = f_P ∪ {(i, p ∪ c, y)}
11                 x = x − y
12                 goto line 2
13 return f_P
```

CLEAR-CYCLES looks for paths in f_P that can incorporate each of the remaining cycles, so as to account for the flow on cycles. The first line finds all remaining cycles, and can be implemented in a way similar to lines 3-12 in EDGE-TO-PATH. Lines 3-4 looks for a path p that shares a common node with the c, and thus c can be connected to p. If the flow on c exceeds that on p, then p is updated to contain that cycle, and p now accounts for some amount of flow on c. If the flow on c is equal to that of p, then the p is updated to contain that cycle and account for all the flow on c. If the flow on c is less than that on p, then a new path is created by connecting c to p, and this new path

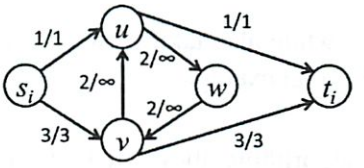
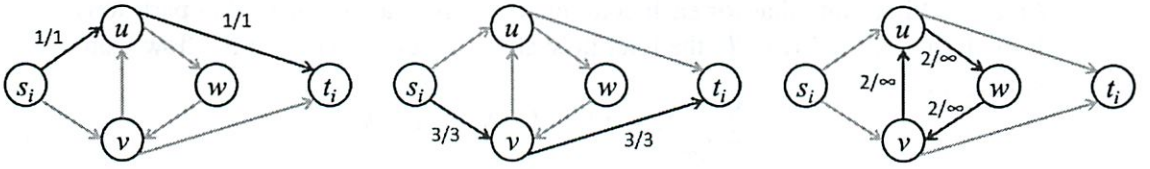
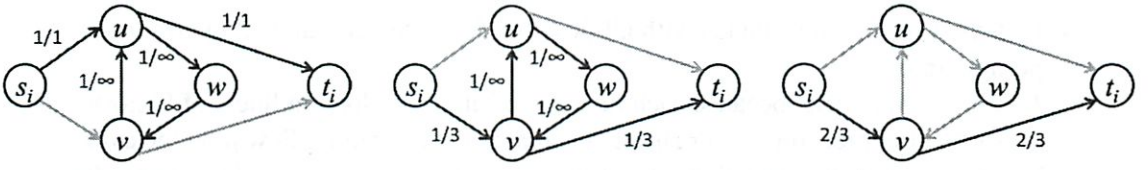


Figure 1: Example of a set of edge flows.



(a) Path p_1 (b) Path p_2 (c) Cycle c

Figure 2: Example of EDGE-TO-PATH, executed on the edge flows in Figure 1.
If FIND-NEXT$(i, u, f_E, f_P) = t_i$ and FIND-NEXT$(i, v, f_E, f_P) = t_i$ in line 12, then upon exiting the while loop of line 3, there will be two paths from s_i to t_i, along with a cycle. (However, if FIND-NEXT$(i, u, f_E, f_P) = w$, then the found path may contain the cycle.)



(a) Path p_1, generated by connecting c to p_1 in Figure 2

(b) Path p_2', generated by appending c to p_2 in Figure 2

(c) Path p_2, updated by subtracting the flow given to p_2'

Figure 3: Example of CLEAR-CYCLES, executed after Figure 2.
The cycle c is first appended to path p_1, and since the flow in c is greater than that in p_1, path p_1 claims a proportion of c equal to its own flow (which is 1 in this case). The remainder of the flow in c (which is 1) is then appended to p_2, and since the flow in p_2 is greater, it is split into two paths:p_2', which has the cycle (and has a flow of 1), and p_2, which is the original path without the cycle (and has a flow of $3 - 1 = 2$). (Note that this is only an example; it is not the only way of implementing CLEAR-CYCLES).

accounts for all the flow on c, while also taking away some flow that was originally designated to p. See Figure 3 for an example.

Proof of Correctness:

From the description of the algorithms above, it is clear that flow conservation is preserved, and that the resulting $f_P = \{f_i(p)\}$ represents the same flow as $f_E = \{f_i(u,v)\}$, because in each iteration, flow along some edges in f_E is taken from it and added as path flow to f_P.

It remains to show that the algorithm terminates with the desired results. In particular:

- The while loop in line 8 of EDGE-TO-PATH terminates with $v = t_i$.
  Proof. First, note that for each commodity i, given a cut (S,T) that partitions V such that $s_i \in S, t_i \in T$, the total flow along the cut is equal to the flow from s_i to t_i, i.e.,
  $$\sum_{u \in S, v \in T} f_i(u,v) = d_i \quad \forall i = 1, ...k.$$
  The while loop in line 8 looks for a path from s_i to t_i that still has positive flow. As long as the total flow on the paths leaving s_i found so far is not equal to d_i, there is always a path from s_i to t_i, because otherwise there would be a cut (S,T) where the total flow from S to T is less than d_i, a contradiction. Therefore, the while terminates only when $v = t_i$ and a path from s_i to t_i is found.
- The while loop in line 3 of EDGE-TO-PATH terminates with no more edge flows coming out from s_i, i.e., right before the execution of line 16, the set of edge flows f_E contain only cycles that do not involve s_i.
  Proof. As mentioned previously, this is true because flow conservation is preserved with the removal of each path flow from f_E.
- CLEAR-CYCLES terminates with all flows in f_E set to zero, and all cycles incorporated into f_P.
  Proof. This is true because each iteration of the while loop in line 4 of CLEAR-CYCLES either terminates, or strictly decreases the remaining flow y on the cycle by a value equal to some flow of a path in f_P. Moreover, each cycle can only be incorporated into a finite number of paths. Therefore, CLEAR-CYCLES sets the flow of cycles in f_E to 0 in a finite number of iterations.

Complexity Analysis:

Each iteration of the while loop in line 3 of EDGE-TO-PATH takes $O(|E|)$ time, because each path cannot contain more than $|E|$ edges, and FIND-NEXT can be implemented to take constant time (for example, the set of outgoing edges from u with positive flow can be a list). Also, since each iteration of the while loop in line 3 sets the flow of at least one edge in f_E to zero, it cannot be executed more than $|E|$ times. Therefore, lines 3-12 take $O(|E|^2)$ time, and the number of path variables generated is $O(|E|)$, per commodity.

For the same reason, the first line of CLEAR-CYCLES also takes $O(|E|^2)$ time, and the total number of cycles is $O(|E|)$. The remainder of CLEAR-CYCLES is executed

in $O(|V||E|^2)$ time, by matching the $O(|V|)$ nodes of each of the $O(|E|)$ cycles to the $O(|E|)$ paths found.

In summary, the algorithm EDGE-TO-PATH is executed in $O(k|V||E|^2)$ time, and the number of path flow variables in f_P is $O(|E|)$.

Comments on grading:

- This part of the problem is independent of whether or not the flow is a single-path routing, and should work for general multicommodity flows. (There was a correction issued for this problem. If it were a single-path routing, as in the incorrect statement, the problem would have been trivial, since the flow of each commodity can be represented by a single path.)
  This problem was originally designed to precede another problem part that models another multi-commodity single-path routing LP in terms of path flows. The result of this problem would allow us to use path flows in the model, despite the fact that the number of possible $s_i - t_i$ paths is exponential.
- The *Executive Summary* should not merely be a restatement of the problem. It should contain the high-level idea of your algorithm, briefly summarizing why and how it works, and its relevance to the problem you are asked to solve.
- It is insufficient to simply use the term "augmenting paths" directly without explanation. Augmenting paths are defined for residual graphs and are used to find the max flow. To use a similar idea in this context, it needs to be well-defined. Here is a good example of how to explain this relationship:
  "We can construct graph G', which can be seen as a kind of residual graph, where the capacities on each edge is the original flow values. Then our algorithm will be similar to that of finding augmenting paths in the residual graph of G'."
- Many students directly quoted the complexity analysis for Edmonds-Karp. This approach is suboptimal in three major ways:
  - First, it fails to observe that the flow on at least one edge is set to zero for the addition of each path, so the number of path variables is only $O(|E|)$.
  - Secondly, breadth-first search (or even depth-first search) is unnecessary because we are not looking for the *shortest* path, or any particular path; we are simply looking for *a* path, and we can always do so in $O(|E|)$ iterations.
  - Finally, formulating the given edge flows as capacities may result in a max flow that does not use the full capacities in cycles.
- Failure to define augmenting paths for a solution based on Ford-Fulkerson/Edmonds-Karp results in up to 10 points, depending on the completeness of the description. Solutions with complete definition and description of an Edmonds-Karp-based algorithm received 11 points. Solutions observing that the number of path variables is $O(|E|)$ were given 13 points. If, on top of that, cycles are taken into account, the algorithm is complete and correct, and 15 points were rewarded.

Problem 5. Linear time minimum spanning tree [20 points] (2 parts)

(a) Given a connected graph $G = (V, E)$ such that all weights on the edges are either 1 or 2, show that the optimal minimum spanning tree (MST) weight can be found in time linear in m (the number of edges).

Solution:

Consider the graph G_1 which contains only the weight 1 edges of G. Computer c, the number of connected components in G_1. Then the size of the MST in G is $n - 2 + c$.

Why? Note that the MST of G with use exactly $c - 1$ many weight 2 edges and the rest of the $n - 1$ edges of the MST will be weight 1 edges. Thus the total value of the MST is $(c - 1) \cdot 2 + (n - 1 - (c - 1)) \cdot 1 = n + c - 2$.

An alternative solution is to notice that one can get constant time implementations of the data structures required for Prim's algorithm when there are only two possible edge weights.

(b) Given a connected graph $G = (V, E)$ such that all edge weights are numbers in the range $[1/2, ..., 2]$. Show that there is an algorithm running in time linear in m that outputs a number b such that $MST(G) \leq b \leq 2MST(G)$ where $MST(G)$ is the value of the minimum spanning tree of G.

Solution:

Given G, convert all edges with weights in the range $[1/2, ...1]$ to weight 1 and all edges with weights in the range $(1, ..., 2]$ to weight 2. Call the new graph G'. Run the linear time MST algorithm from the previous part on G' and output the result. Given any spanning tree in G, each of its edges gets mapped to an edge of weight at most twice its weight in the new graph. Thus the weight of the same spanning tree in G' is at most twice the weight of the spanning tree in G. Note also that since all of the edge weights in G' are at least as big as the corresponding edge weights in G, $MST(G') \geq MST(G)$