

2 table sorted crib sheets

↳ here 1 from last time
↳ Same or change

Do a quick static review

Do more problems

Focus on what I'm bad at

Interval Scheduling

Greedy but

~~Ends earliest~~

(how do you find?)

Prac w/ induction

~~Wedge~~

(2)

Proof Choosing earliest finishes first

$$\begin{array}{c} S = \emptyset \\ \uparrow \\ \text{Solution} \end{array}$$

at each each induction step

$$\begin{array}{c} \text{Base} \\ t=1 \end{array} \quad f(i_1) \leq f(j_1)$$

$$\begin{array}{c} \text{Inductive} \\ t+1 \end{array} \quad f(i_{t+1}) \leq f(j_{t+1})$$

Since request must be non conflicting

* Greedy w/ 1st to finish as criteria *

* Optimal must stay optimal

S must = Optimal

(3)

Weighted w/ DP

L: i cases DP later i

$$\text{Opt}(R) = \max (w_i + \text{opt}(R + c_i))$$

↑ all req that are compatible

$O(n^2)$

L why i

we try all n

then /
each
one tries
n-1 possibilities

$$\underbrace{n + (n-1) + (n-2) + \dots}_{n \text{ times}}$$

oh n^2 @

(4)

Multiple machines

↳ non-tractable

ways to deal w

1. Approx alg

2. Pricing heuristics

3. Greedy or other suboptimal heuristics

4. Reduce to engines
↳ ILP

(we learn so much in this class!)

(need a better short term memory!)

Divide + Conquer

Divide into subproblems

then combine merge

$$T(n) = \begin{cases} \Theta(1) \\ a T(\frac{n}{b}) + O(n^p \lg n) \end{cases}$$

" "
 divide combine

5

Convex Hull

Divide + Conquer

The highest intersection pt

than would one point

2 finger alg

Median finding

Used in the P-set...

Select middle element

Then put items on left or right

Groups of 5 elements

Divide + recurse



FFT

I suck at this

Easy way to multiply polynomials

Value Representation vs polynomial rep
How to convert forms

poly in coeff form

↓ eval

poly in pt value form

↓ multiply

result in pt value form

↓ convert

result in coeff form

7

interpolate w/ matrix ...

(I'm rushing through...)

Randomized Alg

decisions by generating random value

Las Vegas - always produce correct result
- run in poly time expected

Monte Carlo - always run poly time
 $P(\text{correct}) > \text{high}$

ie randomized quick sort

↳ just that way we avoid worst case

Skip lists

8

Checking Math w/ Randomized

- checking matrix products
- checking polynomial identities

~~Also~~

$$A \cdot B \cdot r \neq C \cdot r$$

And allow some prob of error

if pass \rightarrow okay ~~or~~ pass

not pass $\rightarrow \geq \frac{1}{2}$ time \rightarrow not pass

$\rightarrow < \frac{1}{2} \rightarrow$ pass

(I would think its the opposite...)

Alg

1. Pick n -vector r randomly

2. compute $B \cdot r$
 $A \cdot (B \cdot r)$
 $C \cdot r$

9

3. Check $A \cdot B^T r = C \cdot r$

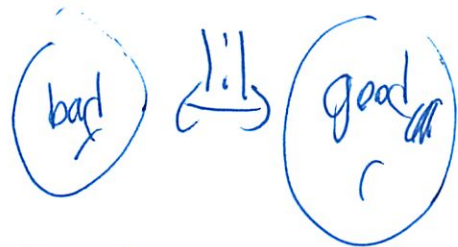
But some vectors don't work

$$- 2 \text{ Os } \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$- \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$\geq \frac{1}{2}$ r's are good

↳ but we don't know which



Friwald's Alg its called

Sometimes not pass is reported as pass
↳ false neg

If false, prob error is $\frac{1}{2^k}$
↳ exponentially ↓

(10)

Polynomial identity

Check $(x^2+1)^2 (x^2-1)^2 + (x-1)^2$
 $\stackrel{?}{=} x^8 - 2x^4 + 2x^2 - 2x + 2$

Do we need to try all x ?
↳ $\infty!$

Query w/ an oracle

↳ that ~~or~~ not pass gets ~~is~~ not pass $\geq \frac{1}{2}$
the time

0 = root

Pick x to test randomly

Asy band

$c \cdot n^3$
↑ any constant c

(10)

O asy upper O upper
 Ω asy lower Ω lower

	$f < g$ upper	both tight	$g < f$ lower
$=$	\rightarrow	Θ	\leftarrow
\leq	O	\times	Ω
$<$	O	\times	Ω

Master Theorem

$$T(n) = a T(n/b) + f(n)$$

n^c vs $f(n)$ \rightarrow $\frac{n}{b} = \# \text{ recursive calls}$
 $n^c \cdot (\lg n)^d$ for each

1. Compare c s
 ↳ bigger c wins
2. Compare d s
 $\frac{T(n)}{n^c} \cdot \lg n$
 $n^{\lg b a}$

(12)

(I get the rel size of each better...)

$$f(n) \in O(g(n)) \quad \text{is} \quad \begin{array}{c} \text{---} g \\ \text{---} f \end{array}$$

$$f(n) \in \Omega(g(n)) \quad \text{is} \quad \begin{array}{c} \text{---} f \\ \text{---} g \end{array}$$

(should do some practice problems / review solutions
(PS 1 was such a mess)

Dynamic Programming

~~Longest Palindromic Subseq~~

1. Characterize optimal sol
↳ subproblem

2. Recursively define

3. Compute

- recursive + memoize (top down)
- iterative (bottom up)

(12)

Longest Palindromic Subseq

Optimal BSTs

Finding BST minimize

$$\sum_{i=1}^n W_i \cdot (\text{depth}_T(k_i) + 1)$$

Alternating Coin Game

Shortest Path

12/15
11:30A

Single Source w/ DP
(Should look into closely...)

Unweighted \rightarrow BFS

non neg weights \rightarrow Dijkstra

General \rightarrow Bellman Fd

(14)

BFS

Queue

all start white

See neighbors \rightarrow make grey

When Finish visiting \rightarrow back

DFS

Stack

start/Completion times

↑
discovery ↑ finish

Can build parentheses structure

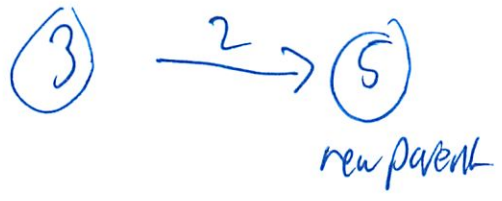
- tree edges
- back edges
- forward edges
- cross edges
- topo sort

(15)

Relaxation



↓ relax



Belman Ford

for each ~~edge~~ vertex

for each edge
relax()

check for neg weight cycles $O(V^3)$

DAG

for each vertex in top sorted order
for each adj vertex
relax

$O(V+E)$

6

Since each edge is involved once

Dijkstra

Weighted

while Queue is not empty

extract min

for each edge

relax

$O(V \lg V + E)$ w/ Fib heap min
priority queue

(should have learned all this in 6.000.06)

All Pairs

No \ominus weight cycles

↳ This applies always - correct.

Method 1 : DP1 $O(n^4)$

What is the last edge traversed?

(17)

Relaxation alg

For all j

for all u

For all v

for all x

$O(n^4)!$

Method 2: Matrix multiplication

$O(n^3 \lg n)$

A, B

$C = A \cdot B$

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj}$$

So a series of matrices

$O(n^3)$ 3 nested for loops

(12)
for i

for j

$$l_{ij} = \infty$$

for k = 1 to n

$$l_{ij} = \min(l_{ij}, l_{ik} + w_{kj})$$

I think I remember seeing an animation

for every pair it tries every intermediate

(in this n-1 times $\Theta(n^4)$)

but actually only need two squares!

L¹

L²

L⁴

L⁸

$$\Theta(n^3 \lg n)$$

↑ This is how many of that length paths
there are... ↑ at most!

Floyd Warshall

another DP formulation

larger + larger sets of ind vertices
recursive

$$d_{ij}^{(k)} = \begin{cases} w_{ij} \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) \end{cases}$$

Bottom up

for $k = 1$ to n

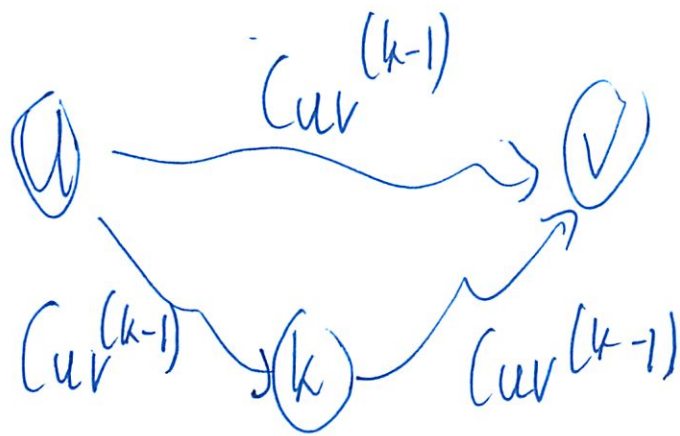
let $D^{(k)}$ be new $n \times n$ matrix

for i

for j

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

isn't this the same as before? $\Theta(n^3)$



No node visited more than k times

↳ or neg weight cycle

↳ return false

or cycle \rightarrow not shortest path
(is that the difference?)

Use node 1 as start

then 1, 2 ...

then 1, 2, 3 ...

$\Theta(V^3)$

So for my 4 \rightarrow 5 example w/ $k = 1$

$$d_{45} = \min(d_{45}, d_{41} + d_{15})$$

↳ so actually just node 1

(21)

So I think the diff is we look at a specific spot

Johnson

best alg

not DP

esp good on sparse graphs

$O(V^2 \lg V + VE)$

uses reweighting

negative weight cycle

Then uses Dijkstra

reweight w/ f_u h

$$W_h(u, v) = W(u, v) + h(u) - h(v)$$

Shortest pt is preserved

(27)

So what is this actually?

δ = shortest-path weights

$\hat{\delta}$ = after reweighting

Add new vertex s

and point to all $\delta(s, v)$

$$h(v) = \delta(s, v) \text{ for all } v$$

By triangle inequality

$$h(v) \leq h(u) + w(u, v)$$

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$$

still don't get

73

Johnson's algorithm

From Wikipedia, the free encyclopedia

Johnson's algorithm is a way to find the shortest paths between all pairs of vertices in a sparse directed graph. It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist. It works by using the Bellman–Ford algorithm to compute a transformation of the input graph that removes all negative weights, allowing Dijkstra's algorithm to be used on the transformed graph. It is named after Donald B. Johnson, who first published the technique in 1977.

A similar reweighting technique is also used in Suurballe's algorithm (1974) for finding two disjoint paths of minimum total length between the same two vertices in a graph with non-negative edge weights.

Contents

- 1 Algorithm description
- 2 Example
- 3 Correctness
- 4 Analysis
- 5 References
- 6 External links

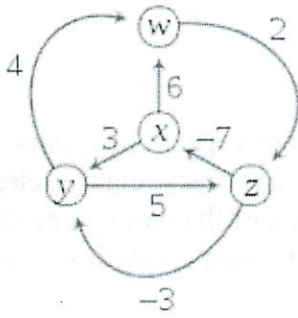
Algorithm description

Johnson's algorithm consists of the following steps:

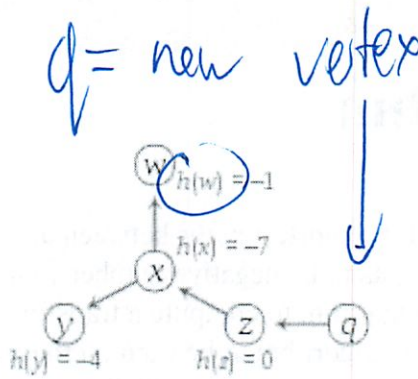
1. First, a new node q is added to the graph, connected by zero-weight edges to each of the other nodes.
2. Second, the Bellman–Ford algorithm is used, starting from the new vertex q , to find for each vertex v the minimum weight $h(v)$ of a path from q to v . If this step detects a negative cycle, the algorithm is terminated.
3. Next the edges of the original graph are reweighted using the values computed by the Bellman–Ford algorithm: an edge from u to v , having length $w(u, v)$, is given the new length $w(u, v) + h(u) - h(v)$.
4. Finally, q is removed, and Dijkstra's algorithm is used to find the shortest paths from each node s to every other vertex in the reweighted graph.

Example

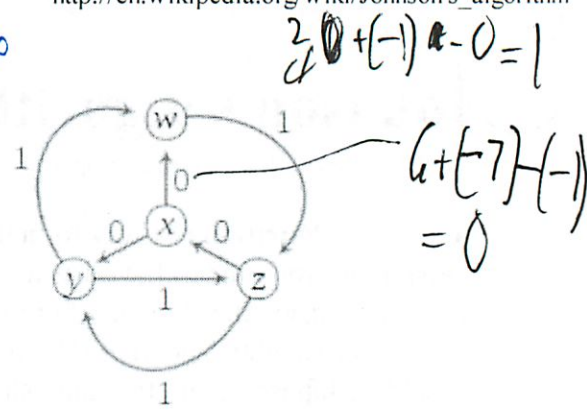
The first three stages of Johnson's algorithm are depicted in the illustration below.



original graph with negative edges



shortest path tree found by Bellman-Ford



reweighted graph with no negative edges

The graph on the left of the illustration has two negative edges, but no negative cycles. At the center is shown the new vertex q , a shortest path tree as computed by the Bellman-Ford algorithm with q as starting vertex, and the values $h(v)$ computed at each other node as the length of the shortest path from q to that node. Note that these values are all non-positive, because q has a length-zero edge to each vertex and the shortest path can be no longer than that edge. On the right is shown the reweighted graph, formed by replacing each edge weight $w(u, v)$ by $w(u, v) + h(u) - h(v)$. In this reweighted graph, all edge weights are non-negative, but the shortest path between any two nodes uses the same sequence of edges as the shortest path between the same two nodes in the original graph. The algorithm concludes by applying Dijkstra's algorithm to each of the four starting nodes in the reweighted graph.

Correctness

then delete s

In the reweighted graph, all paths between a pair s and t of nodes have the same quantity $h(s) - h(t)$ added to them. The previous statement can be proven as follows: Let p be an s - t path. Its weight W in the reweighted graph is given by the following expression:

$$(w(s, p_1) + h(s) - h(p_1)) + (w(p_1, p_2) + h(p_1) - h(p_2)) + \dots + (w(p_n, t) + h(p_n) - h(t))$$

Notice that every $+h(p_i)$ is cancelled by $-h(p_i)$ in the previous bracketed expression; therefore, we are left with the following expression for W :

$$(w(s, p_1) + w(p_1, p_2) + \dots + w(p_n, t)) + h(s) - h(t)$$

Notice that the bracketed expression is the weight of p in the original weighting.

Since the reweighting adds the same amount to the weight of every s - t path, a path is a shortest path in the original weighting if and only if it is a shortest path after reweighting. The weight of edges that belong to a shortest path from q to any node is zero, and therefore the lengths of the shortest paths from q to every node become zero in the reweighted graph; however, they still remain shortest paths. Therefore, there can be no negative edges: if edge uv had a negative weight after the reweighting, then the zero-length path from q to u together with this edge would form a negative-length path from q to v , contradicting the fact that all vertices have zero distance from q . The non-existence of negative edges ensures the optimality of the paths found by Dijkstra's algorithm. The distances in the original graph may be calculated from the distances calculated by Dijkstra's algorithm in the reweighted graph by reversing the reweighting transformation.

236

WP

Old Have neg edges
But not neg cycles

add new vertex q

build a shortest path tree w/ Bellman-Ford
w/ q as start

↳ call $h(\text{node})$

↳ shortest path (Bellman-Ford) $h \rightarrow \text{node}$

↑ these are all negative (non-positive)

Since q has a 0 to each!
as a fall back...

reweight all

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$

(how in all world does this work?)

(24)

Then delete S

← can't do neg

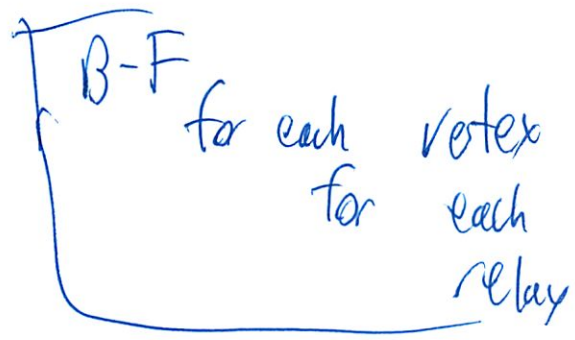
Then Dijkstra

↳ extract min (really need to know off hand!)



Oh Dijkstra for each node

w/ a diff node as the source



← can do neg weights

Both Matrix, F-W, Johnson allow ⊕ weights
but not ⊖ cycles

Greedy Alg + MSTs

Spanning tree has $n - 1$ edges

Want min weighted

just add them on 1 by 1

↳ greedy

(25)

Kruskal

Uses a specific rule to add a safe edge

finds the edge of least weight
of any set

light edge

if ~~an~~ its weight is the min of
any edge crossing the cut

Checks if that edge ~~is~~ connect nodes that
are in diff sets

if they are diff \rightarrow Union them

$O(E \lg E) \rightarrow O(E \lg V)$

\hookrightarrow since time to sort the edges

26

Prim

like dijkstra

edges are always a single tree

adds isolated vertices

Use a min priority queue

[kernal sorted up front once]

$O(E \lg V)$ same

$O(E + V \lg V)$ w/ fib heap

In lecture titled DP

↳ but if its greedy → why do i

Network Flow

S source = s

sink = t

no notion of neg

27

flow \leq capacity

flow in = flow out

flow out of source = flow into sink

want max flow

transform so no antiparallel edges

if multiple sources and/or sinks, add supersource and/or
super sink

Ford-Fulkerson

initialize flow to 0

while there exists an augmenting path p in
residual network G_f

augment flow f along p

return f

(28)

Residual Network

G_f consists of edges w/ capacity,

$$C_f(u, v) = c(u, v) - f(u, v)$$

represent current flow as

$$C_f(v, u) = f(u, v)$$

Since we can cancel out current flow

Augmentation

$$f \uparrow f'(u, v) = f(u, v) + f'(u, v) - f'(v, u)$$

↑ same as C_f ?

Residual Capacity

$$C_f(p) = \min \{ C_f(u, v) : (u, v) \text{ is on } p \}$$

Capacity

Cuts net flow ← flow is always the same
Capacity of cut

find min cut of capacity
flow would do no good!

24

Max-flow min-cut theory

equivalent

- 1. f is max flow
- 2. G_f no aug paths
- 3. $|f| = c(S, T)$ for some cut

* the max possible flow = the min possible cuts over all possible cuts

∴ So find min possible cut
↳ that is our max flow!

Ford Fulkerson Theory Alg

~~While there exists a path~~

find some augmenting path p

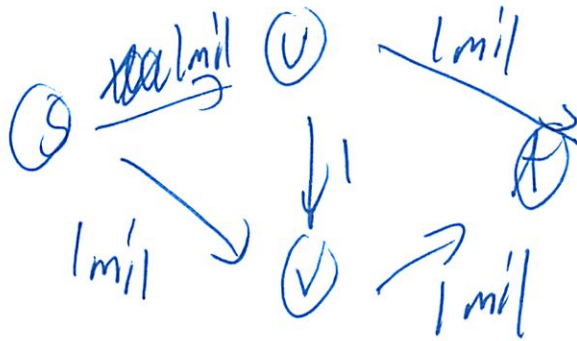
Use to update

$$O(E/f^*)$$

↑ increases at least 1 unit each iteration

30

Oh yeah that super stupid example



grows 1 at a time

$S \rightarrow U \rightarrow V \rightarrow T$ odd } iterations
 $S \rightarrow V \rightarrow U \rightarrow T$ even }

Edmonds - karp

Find the augmenting path w/ BFS

$O(VE^2)$

instead of stupid current way

What did I do wrong on my cheat sheet last time?

↳ Did I ever write it down?

Rules of alg don't matter super much in this class.

36

Unit
~~Unit~~ 2 staff

~~Mr~~ Upward critical edge when τ capacity
the total max possible flow τ

downward - ie if saturated

U = set of all nodes reachable from s in residual network
↳ do w/ BFS
 V = reachable from t

all upward critical 'if' in both U and V

Normally s, t disconnected in G_f if max flow

So say



↳ that should be τ crit

What is G_f ?



$U = s$
 $V = s, t$

(32)

Ah so $S \rightarrow T$
 $\uparrow \quad \uparrow$
 'is in' 'is in'
 $U \quad V$

So 'is' is 'upward' critical

$O(V+E)$ same 2 BFSes

Downward Critical

it there 'is' no path from U to V in G

yeah was not $(S) \xleftarrow{(U)} (T)$
 path $S \rightarrow T$

(don't know why table 'is' in there?
 are they computable?)

33

Scheduling Meeting

n students

n faculty

meet A faculty member

only 1:1 meetings

min # time slots ?

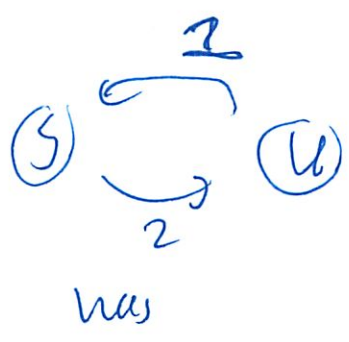
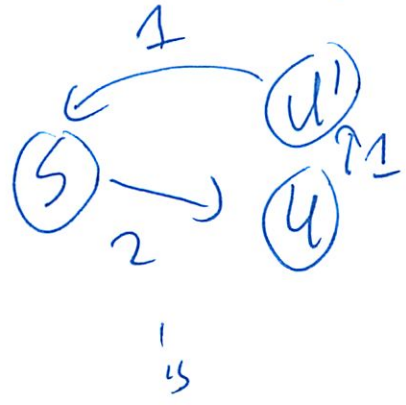
So min Flow

min cut = n

of edges across

(never looked at this in depth)

For no self loops can convert



(34)

(actually reading my notes
↳ get those now
its mostly the stuff I have not studied...)

Network Flow + Applications

$|f|$ = flow value

like (e, c) stable matching

(yeah here is the matching picture ...)

Baseball ...

to prove is eliminate

(Should at some pt actually try...)

eliminate if max flow does not saturate
all edges leaving source

saturation = playing all remaining games

Finding if subset of teams that will
play each other x times

(while should think about it...)

NP - Completeness

Halting \rightarrow don't know if terminates / crashes

tractable = poly time
 n^{2000} ok

$n^{c \log n}$ ok

n^k k some fixed constant

$n^{\log n}$ ok \leftarrow don't know why ... ?

3 types of problems

1. Decision \rightarrow Yes/No solvable
2. Search \rightarrow find an object
3. Optimization \rightarrow find best

(36)

P = poly time
Yes iff True

NP = non-deterministic poly time

~~non~~ deterministic Turing machine

always only 1 rule for each situation

non-deterministic Turing machine

might have 2 conflicting rules
in rule set

iso how does it decide?

branches into several states

if any halt w/ "accept" then

what we have done works!

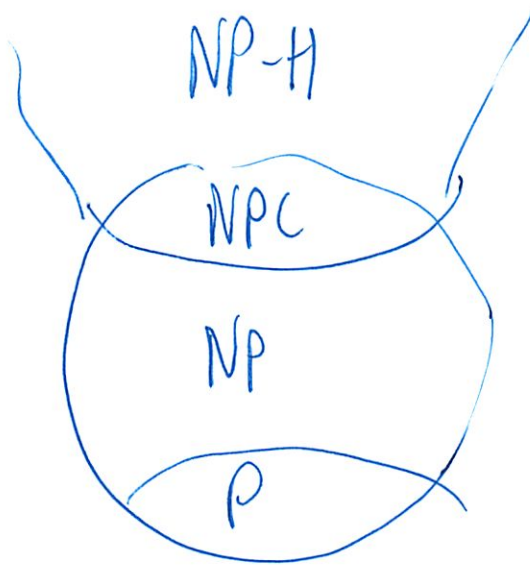
NP can be solved in poly time w/
non-deterministic Turing machine

NP can be verified in poly time

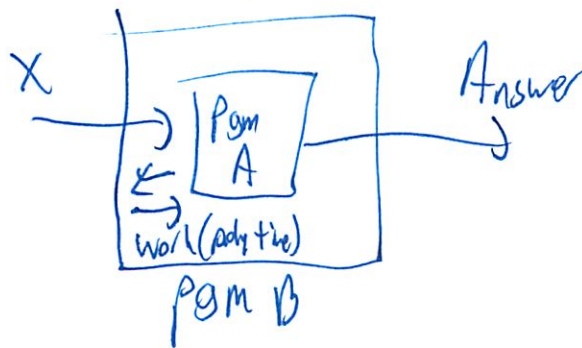
(37)

NP-Hard If all problems in NP can be poly time reduced to it

NP-Complete if in NP + NP-Hard



Cook reduction



have $A \in P$ want to show $B \in P$

38

(can't tell the 2 apart...)

karp

We always used karp - right?

(write on cheat sheet)

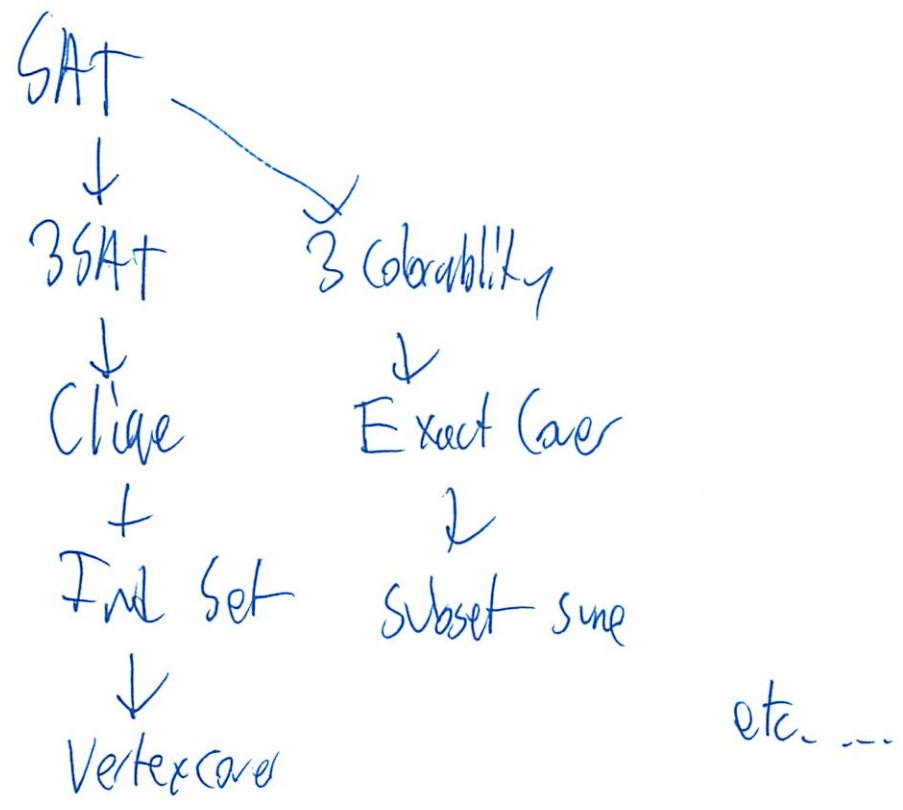
General Technique

Cheat sheet

(totally forget doing the quiz de/brief...)

NPC Lecture 2

Verifier is the judge, not the detative....



(39)

Some problems not even in NP!

Linear Programming

ex: Politician trying to buy an election through advertising ...
minimize amt of \$ needed

$$-2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50 \quad \leftarrow \text{at least 50k votes}$$

etc

$$\text{minimize } x_1 + x_2 + x_3 + x_4$$

but no neg cost advertising

$$x_1 \geq 0 \quad x_3 \geq 0$$

$$x_2 \geq 0 \quad x_4 \geq 0$$

General form

LP vs ILP

↑
poly time

$$O(n^3)$$

↑
np-hard

(40)

Std Form

$$\text{Maximize } \sum_{j=1}^n c_j x_j = \text{Objective Eq } c^T x$$

$$\sum_{j=1}^n a_{ij} x_j \leq b_i \text{ for } i = 1, 2, \dots, m$$

$$x_j \geq 0 \text{ for } j = 1, \dots, n$$

1. If minimize
negate coeffs to maximize

2. Add non negativity constraints if none

$$x_j \text{ replaced by } x_j' - x_j'' \quad \begin{array}{l} x_j' \geq 0 \\ x_j'' \geq 0 \end{array}$$

3. Translate ~~\geq~~ to $< >$

$$\text{like } x_1 + x_2 = 7$$

4. (see cheat sheet)

Book doesn't say $>$ to \geq hmmm

(11)
Did we study slack form?
Need to review dead

Totally Unimodular

(see cheat sheet)

long proof I am skipping...

Min Cost Network Flow

E-k can't handle

Can have sink/source within a node

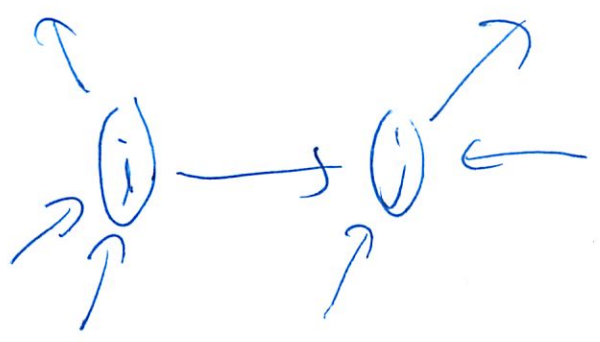
Want min cost of 1 unit of flow

min cost flow matrix if it is TU

$$A' \begin{pmatrix} A \\ -A \\ I \\ -I \end{pmatrix} x \leq \begin{pmatrix} d' \\ \cancel{d'} \\ e' \\ e \end{pmatrix}$$

(I don't get this section!)

42



$$\begin{bmatrix} 1 & 1 & 1 & -1 \\ -1 & -1 & & \\ -1 & & & 1 \end{bmatrix} \text{ so is TV}$$

* Show that the A matrix in the $Ax \leq b$ constraints of a LP is totally unimodular
 (trying to show int sol?)

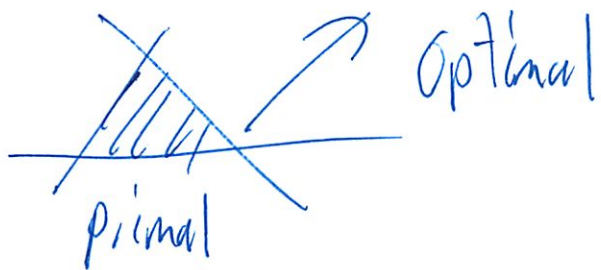
feasible vs optimal

Recall

(cheat sheet)

I remember I had a great deal of problems here
 Found on some p-set

(43)



dual looks for lin combo of constraints

(I don't get)

want Θ of cost fn to be a lin combo
(more on)

Simplex Algorithm

A way to solve LP

Exponential in theory

But works well in practice

Std form

Wants slack form

(44)

add y_i to each
and $y_i \geq 0$

basic sol set not basic to 0
basic according to eqn

but might (probably) not optimal

pivot ~~swap~~ work

Swap role of x_1 and x_6

Solve for x_1 in eqn w/ x_6
↑ limiting constraint

$$x_1 = \frac{36 - x_2 - 2x_3 - x_6}{4}$$

rewrite x_1 as fn x_2, x_3, x_6

x_1 now basic

x_6 now non basic

45

More pivoting

remove x_4 and replace

$$x_4 = 30 - \left(9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \right) - x_2 - 3x_3$$

etc

Optimal is better now

Can't make x_6 neg \Rightarrow can't pivot that

x_2, x_3 can be bigger, so can pivot

don't fully get

will try ask about

prob not

gets messy fast

(4/6)

Hashing

(the one I forgot in that interview...)

Never did fully study for tech interviews...

Direct address table

$O(1)$ time
huge

Balanced BST

$O(\log n)$ search

$O(\text{keys})$ space

$n = \# \text{ keys}$

$m = \# \text{ slots in table}$

Pick f_n at random

(all this formal notation!)

Universal hash if $\forall \text{ keys}$
very low prob of collision

$x_1 \neq x_2$

(47)

E # of collisions

$$X \leq 1 + \frac{S}{m}$$

Proof w/ indicator variables

Actually kinda handy for my work ...

but their proof seems silly

Oh well I guess that's how you do it ...

Universal Hash F_n



perfect maps $\downarrow \uparrow$

$O(1)$ actions for every student

(read this too fast ...)

Geometric LP

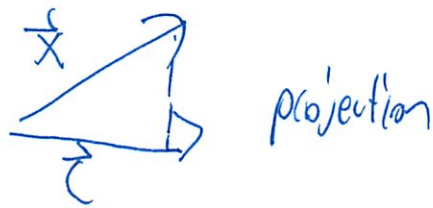
I guess another way to solve

n -dim space

\vec{c} vector in \mathbb{R}^n

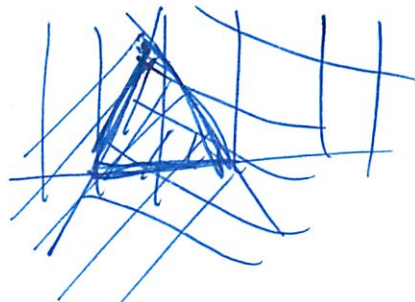
\vec{x} vector in \mathbb{R}^n

Dot product



Can multiply - same thing

Overlapping regions



Corners = basic feasible sols

(49)

Simplex Intuition

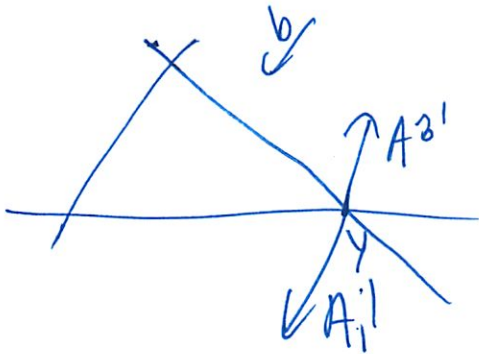
Start w/ basic feasible

jump to vertex in direc want to go

max constraint until you have a soln.

Can always max Δx and find at least 1 pt

Geometric Duality



Some lin combo A_1' A_3' that is $=$ to b

So basic feasible sol y is optimal sol of dual

(I need to practice this stuff at some point!)

(5)

12/17
5:50P

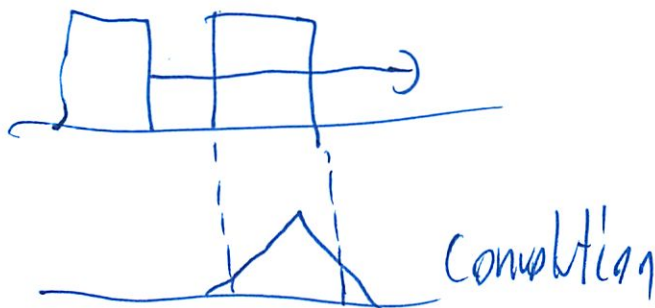
FFT more notes

~~like FOIL~~
~~rep~~ (show)

like FOIL

represent dynamics
What is convolution?

area that overlaps in 2 signals
as 2nd shape moves



(51)
point-value representation

faster way to multiply

2. Eval


3. Pointwise multiply

4. Interpolate

Root of unity

= 1 when raised

like

 $\pm \frac{1}{2} + \frac{i\sqrt{3}}{2}$

$$e^{\pm \frac{2i\pi}{3}}, e^{\pm \frac{4i\pi}{3}}$$

$$-5 \pm 8ki$$

DFT

FFT uses special property
(skip, I will never understand...)

(52)

Binary #s used in computer families

Assume # in binary for FFT

Dual \rightarrow some optimal value

Approx Alg

Get some guarantee about output

No more than 2 worse

So if ans = 50

Then get 25 \rightarrow 100 is ok @

$P(n)$

$$\text{s.t. } \max \left(\frac{C}{C_{\text{opt}}}, \frac{C_{\text{opt}}}{C} \right) \leq P(n)$$

C = our approx response

$C^* = C_{\text{opt}}$ = mythical perfect result

(53)

So before

$$\frac{25}{50} \quad \frac{50}{100}$$

$$\max \left(\frac{25}{50}, \frac{50}{25} \right) \leq 2$$

\uparrow \uparrow
1.5 2

$$\max \left(\frac{100}{50}, \frac{50}{100} \right) \leq 2$$

\uparrow \uparrow
2 1.5

Approx vertex cover

use # of edges connected to node

Vertex cover every ~~vertex~~ edge incident to at least
1 vertex

54

We are concerned w/ worst case
↳ not any case

To prove

1. Correct
2. Poly time
3. Ans within 2 of optimal
↳ how do we do it?

Notice ~~KAM~~ no 2 edges in A share endpoint

So $|A|$ is a lower bound on C_{opt}

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

Since # vertices = $2|A| = C$

$$|C| \leq 2C_{opt}$$

Wait how are these 2 connected
but I do get the right answer

(55)

$k!$, $\log k + k$ edges in other example

↳ how do we know this?

typical counting stuff I'm bad at!

(cant find in notes)

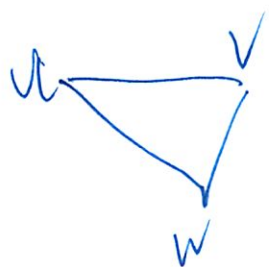
Basically ways to show set cover is under a certain size...

Traveling Salesman

△ inequality

Hamiltonian Cycle visit each vertex once
w/ min cost

$$c(u, w) \leq c(u, v) + c(v, w)$$



manhattan distance

(56)

Claim MST has weight that is lower bound

MST is cycle - edge

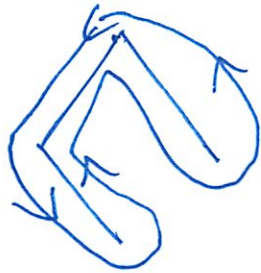
$$\text{Weight}(T_{\text{ar}}) \leq \text{Weight MST}$$

But if $\frac{\text{MST}}{\text{opt}} \geq \frac{T_{\text{ar}}}{\text{opt}}$ Contradiction!

So if we can find lower bound

So MST is

Then walk around L



preorder walk - drop vertices ya saw before

(5)

$$C(T) \leq C(H^*)_{\text{optimal}}$$

Full walk ~~never~~ cross each edge twice

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

W visits more than once

Think for pre-order walk

Triangle inequality is invariant which can bound our cost

$$C(H) \leq C(W) \leq 2(C(H^*))_{\text{optimal tour}}$$

* Since by deleting node, our cost will not ↑ *

(58)

Weighted Vertex Cover

need diff approach

greedy won't work

ILP

Minimize sum of weights

TV so integer

Or not?

Can use LP for approx

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

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

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

(continuous)

So round to 0 or 1

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

Rounding up gives a cover

didn't get

$w()$ is weights

V is vertices

$x(w)$ is 0 or 1 is in cover or not

$x(u) + x(v) \geq 1$ for each pair

↳ so 1 or both ends must be included

So x is for the ~~vertices~~
well for the line

Somehow know optimal sol for ~~ILP~~ ILP is lower

bound on optimum for ILP

So came up values

And know that is within 2 of optimal

(20)

Amortized Analysis

Time per op worst case
instead ~~over~~ avg over seq of ops

Like table doubling
(this silly potential methods)

Aggregate

$$\sum \text{costs}$$

Accounting

accumulate credit

When amortized credit $>$ actual credit
pay for later operations

Amortized cost must be an upper bound

$$\sum_{i=1}^n \hat{c}_i \geq \sum_{i=1}^n c_i$$

(61)

Potential method

represent the prepaid as the potential

assign to whole data structure

$$\begin{aligned}\sum_{i=1}^n \hat{c}_i &= \sum_{i=1}^n (c_i + \phi(D_i) - \phi(D_{i-1})) \\ &= \sum_{i=1}^n c_i + \phi(D_n) - \phi(D_0)\end{aligned}$$

$\phi(D_i)$ terms telescope

telescoping sums

Subsequent terms cancel leaving only
1st + final term

$$\sum_{i=1}^{n-1} (a_i - a_{i-1})$$

only a_1, a_n left
($a_1 - a_n$)

(62)
 \hat{C}_i gives upper bound on actual total cost

So for each push potential diff is

$$\phi(D_i) - \phi(D_{i-1}) = (s+1) - s \\ = 1$$

So amortized cost push

$$\hat{C}_i = c_i + \phi(D_i) - \phi(D_{i-1}) \\ = 1 + 1 \\ = 2$$

ϕ is "bank balance" = amt of prepaid work

$\Delta \phi_i \geq 0$ prepay

$\Delta \phi_i \leq 0$ use/withdraw

So \hat{C}_i is max cost

Can see actual cost

Charge 2 each time

↳ actual cost and potential diff

(63)

Never realized the pay table!

Kinda see how we only need to pay ~~table 2~~ 2

but some we move more than twice

(but ten new ones come in)

$$\begin{aligned} \uparrow C(s) &\leq 3m \\ &\text{so } \Theta(m) \end{aligned}$$

Or we pay 3 on each move ...

On each time we insert we pay for making current one
and move them before

I see ...

And have more of those cards as get bigger

And pay current cost ...

(64)

Move to Front

(did I ever look at this one)

Online = ^{doesn't} know future

offline = knows future

How much does it help to know the future?

$$C_{MTF}(s) \leq 2 C_{opt}(s)$$

↑ never worse than twice as bad

Use Φ to compare

$\Phi = \#$ of inversions in list

(I don't get what we are trying to do here)

Showing knowing future helps by factor of ≤ 2 ...

So goal was search fine
MTF = move to front

(65)

Cost is what?

Cost of an optimal search alg?

Moving item to front makes search go faster

But what is the optimal?

Distributed Alg

Can send messages

Sync w/ common clock

like leader election

Could be biggest VID seen so far

if get own VID back, you are the leader!

$O(n)$ rounds

$O(n^2)$ messages

Can also divide + conquer
groups that elect a leader

(6)

Send to 2^i people

Send distance 2^{i-1}

~~All~~ Then winner goes to next round

I'm bad at the combinatorics reduction

If you know n

$2^{i-1} \geq n/2$ is sufficient

but if don't know $n \rightarrow$ must wait for message to return to you

$$\# \text{ messages} \leq \frac{n}{2^{i-2}} \times 2^{i+1} \leq 8n$$

$$\# \text{ rounds} = \log n$$

$$\# \text{ participating processes} \leq \frac{n}{2^{i-2}}$$

why are all of these?

(67)

Oh that's how many (2^{i-2}) non participating processes

nodes	n	100		
participating per node	n	100	50	per $\frac{50}{4} = 12.5$
per round	2^i	2	4	8

participating & each round
Size of each round \uparrow) 2 effects

So round 3

$$\frac{100}{1} \quad \frac{n}{2^{3-2} + 1} \quad = \quad \frac{100}{3} \quad \frac{100}{5} \quad \frac{100}{9}$$

participating per round

Maximal Ind Set

Could be matching/coloring problem under reduction

Ind set set of vertices of which no 2
are adjacent
maximal is NP hard

Luby's Alg

- local
1. Add all nodes
 2. Each live node marks it w/ prob $\frac{1}{2}$
 3. If neighbors marked, unmarks itself
 4. Each remaining marked node adds it
 5. When live ~~set~~ terminates

So prob that $\#$ nodes $> 8d \ln(n)$
is at most $\frac{1}{n}$
How would one find it

$$\mathbb{E}[\# \text{ of nodes}] = O(d \ln n)$$

(69)

$$P[\text{live } v \text{ adds self to MTS in } l \text{ rand}] \geq \frac{1}{4d}$$

Since $P[v \text{ marks}] = \frac{1}{2d}$

$$P[\text{neighbors mark}] \leq \sum \frac{1}{2d}$$

$$\leq \frac{d}{2d} = \frac{1}{2}$$

Boole / union bound

Prob that at least 1 of events happens
is no greater than sum of prob of
indv events

$$P\left(\bigcup_i A_i\right) \leq \sum P(A_i)$$

$P[v \text{ marks itself and stays marked}]$

$$\geq \frac{1}{2d} \cdot \frac{1}{2} = \frac{1}{4d}$$

(76)

$P(v \text{ stays live after } 64d \text{ in rounds})$

$$\leq \left(1 - \frac{1}{4d}\right)^{4d \cdot c \ln c}$$

$$\leq e^{-c \ln c}$$

$$= e^{-c}$$

$O(d \ln c)$ rounds

'i yeah how many rounds'

$\hookrightarrow O(d \ln c)$ but how got that'

Crypto

hash fn

input ~~fixed~~ arbitrary
output deterministic
random

∞ domain
fixed range

$\{0, 1\}^d$ len of hash

~~70~~ (71)

Like a random Oracle who assigns ya a random #
and writes it down
So its the same each time

Properties

One way
Collision Resistant (Any)
" " (Targeted)

L header

Pseudo-Randomness
Non Malleability

Store PW on disk

Finding collisions \rightarrow birthday problem
any pair can collide

12

OW \nrightarrow TCR

TCR \nrightarrow OW

Digital Signatures

$$\sigma = \text{Sign}(sk_A, M)$$

$$\text{Verify}(M, \sigma, pk_A) = \{T, F\}$$

Actions + Commitments

Sealed bids

$$c(x) = h(r \parallel x)$$

Random

(why do we need r ?)

Does she publish r when made bid?

~~well~~ well don't want someone else to guess...

73

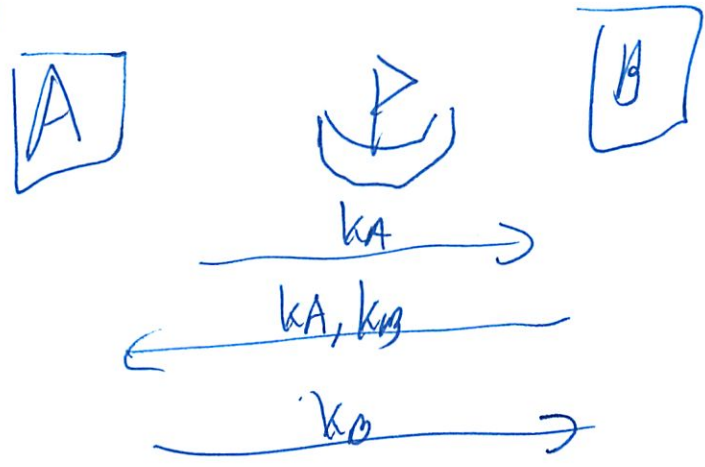
Symmetric key Encryption

$$C = \underbrace{e_k}_{\text{encryption key}}(m)$$

$$m = \underbrace{d_k}_{\text{decryption key}}(c)$$

key Exchange

put keys on message



'need commutativity of keys

74

$G = F_p^*$ finite field

a selected

b

$g^a \pmod p$

$g^b \pmod p$

$$k = (g^b)^a \pmod p = (g^a)^b \pmod p = k$$

↑
all mat which works

- eavesdropper = passive

- m in the $m =$ active

RSA

(seen in 3x now

never memorize it!)

(75)

public key (N, e)

Decryption exponent d found by extended Euclid's
(on cheat sheet)

(skipping the exact details...)

People tried to use NP-complete problems to make
~~XP~~ Crypto systems

↳ but ended badly

Why exactly?

Lattice-based technique

Some sort of data flow technique ...

Merkle tree

verify n elements at once

Shor's algorithm - Rivest

↳ silly and crazy complicated

26

Sub linear alg

testing for sortedness of list

Can do binary search for value

if don't end up at i , output Fail

↑ pick an i , look at its value

if $\geq \epsilon n$ we need

then in $\frac{c}{\epsilon}$ loops we will choose

one w/ prob $\geq 1 - (1-\epsilon)^{\frac{c}{\epsilon}}$

$\geq 1 - e^{-c}$

$\geq \frac{3}{4}$

then test fails

* If ~~prob~~ correct, must pass

If incorrect, must fail

If ϵ -close can pass ($\geq \frac{3}{4}$ time) ^{correct} or fail (incorrect)

Sampling

6.0342
(need to resty!)

Classical approx
not enough time to construct proper sol
output is close to proper output
for given input

Property testing

output is correct ans for give input
or at least ~~all~~ other inputs are close

what is the diff?
LI see, subtle

Classical

(see cheat sheet)

78

A list is ϵ -Close sorted if we can delete at most ϵn values to make it close to sorted.

So $\epsilon = \frac{1}{10}$ means 90% sorted

(this is what I wrote earlier)

Compression

lossless

lossy

lossless

run length

huffman

lempel-ziv

web graphs

lossy

lose info that is not core / undetectable

bloom filters

support membership queries

save space

Some false pos

~~xxxxxx~~

$h_1(x_1)$ $h_2(x_1)$ $h_3(x_1)$

turn to 1

look up by checking if each is a 1

(can have false pos)

- not in set

- but reports that it is

$$\left(1 - \frac{1}{m}\right)^k$$

arrive for a fixed x for all k hash fns

(80)

(some prob stuff I should prob review...)

Rec 11

Missed, never read

Graph Diameter testing

diameter max distance b/w vertices

Check if diameter at most D w/ boundry

$$\beta(D) = 4D + 2$$

accept all graphs that have diameter D

reject graphs that are ϵ far from having

diameter $\beta(D)$ w/ prob $\geq \frac{2}{3}$

graphs that are near can be accepted or rejected

91

1. Select S vertex

2. For each vertex perform BFS. Terminate when

a) k vertices have been reached

b) all vertices in D neighborhood have been reached

3. Accept if all vertices have D -neighborhood $\geq k$

What is a D -neighborhood?

C -neighborhood - can be reached from v in
at most C hops

call it a ball

ones on edge ~~are~~ are bandy

(don't fully see - wish I was there that day...)

(82)

Bloom Filter Distributed Cache

Cost Δ to send packets

Can use as dictionary

(don't have the time now to focus on this
- should have read earlier...)

Interactive + O knowledge Proofs

BPP Bounded error probabilistic polynomial time

Solved by probabilistic Turing machine in
poly time w/ error of at most $\frac{1}{3}$
For all instances

One of the largest practical class of problems
Contains P

p can be b/w $0 \rightarrow \frac{1}{2}$
 $\uparrow \frac{1}{3}$ is arbitrary

83

Graph Isomorphism

Are the 2 graphs isomorphic?

↳ preserve edges & non edges

Pepsi Challenge

(see cheat sheet)

Graph

(don't get)

(are

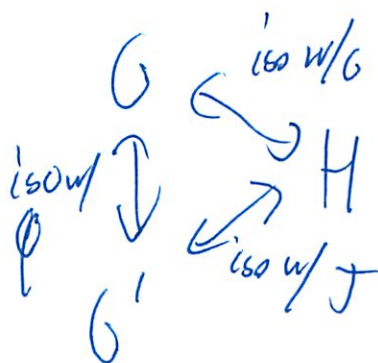
↳ I got the ~~code~~^{code} example ---

Oh right 3rd graph which is isomorphic to both

Since we want \mathcal{O} knowledge

$$\begin{aligned}
 H &= G \circ G = G \circ \varphi \circ G \\
 &= T \circ G'
 \end{aligned}$$

$$G = \varphi \circ G'$$



(84)

Bob
(veitler)

Flip the coin send either G or J

Alice if $b=1$ sends G

2

$$\rho = G \circ \phi$$

Bob checks $H = G \circ G$

$$H = \rho \circ G'$$

(letters confusing!)

85

Real

Goal write up in book!

I never read up to now

Used to prove 'ideal sol' is 'indeed optimal'

Saw w/ max flow min cut theory
turned into a minimization problem

Minimize

but end value is the same

MultiThreaded Alg

is in the book!

I need to think more about multithreaded programming
at some pt...

Review Exam

12/18

12:30A

I still don't get O vs Θ fully
well I know what it is
but I don't do anything diff...

Paranoid

keep picking until within bounds

F was more is MST

I suck at bounds stuff...

* change of variable

$$T(n) = T(n^{1/3}) + T(n^{2/3}) + \log n$$

$$m = \log n$$

$$T(2^m) = T(2^{m/3}) + T(2^{2m/3}) + m$$

$$S(m) = T(2^m)$$

②

$$S(m) = S(m/3) + S(2m/3) + m$$
$$= \Theta(m \lg m)$$

$$T(n) = \Theta(\log n \log \log n)$$

I would never get...

* Think carefully about what it's trying to say *

↳ The network (can) have stupid ones to get wrong...

Rank of set - smallest ordinal $\#$

greater than the ranks of all members
of the set

Conditional prob

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A \cap B) = P(A|B) P(B)$$

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Value req

Vandermonde matrix M (coefficients)

Master Theorem

$T(n) = aT(\frac{n}{b}) + f(n)$

$\frac{n}{b}$ = # of recursive calls

Look at $n^{\lg_b a}$ vs $f(n)$

For each $n^c \cdot (\lg n)^d$

- Compare c s
 - bigger c
- Compare d s
 - bigger d or $\lg n$

$n^{\lg_b a} \cdot \lg n$

Eval multiply by M

interp multiply by M^{-1}

Only at pts of unity

Polynomial identities

deterministic $O(n)$

random $O(1)$

Given inputs that are distinct for each possible y , exactly one degree $\leq n$ polynomial

$f(x_0) = y_0$

$f(x_1) = y_1$

$f(x_n) = y_n$

eg 2 pts determine a line

3 pts quadratic polynomial

So randomly try evaluating

if wrong, may get true + false

Single Source Shortest Path

unweighted BFS $O(n+m)$

non-weights D_{ij} $O(m \cdot \lg n)$

general Bellman-Ford $O(nm)$

acyclic Topo sort + 1 pass B.F. $O(n+m)$

topo sort DFS, sort by finishing time $O(V+E)$

strongly connected from one point to any other

MST-Prim (G, w, r) $O(E \lg V)$

for each $v \in G, v \neq r$ $O(E + V \lg V)$

$U, \text{key} = \infty$

$U, \pi = \text{nil}$ finds min edge

$r, \text{key} = 0$ min priority queue

$Q = G \setminus U$

while $Q \neq \emptyset$

$u = \text{Extract-Min}(Q)$ biased on key

for each $v \in G, v \neq u$

if $v \in Q$ and $w(u,v) < \text{key}_v$

$v, \text{key} = w(u,v)$

DP

1. Characterize optimal sol'n
2. Recursively define the value of an optimal sol from optimal subproblems
3. Compute value of optimal in bottom up fashion

top down: recurse + memorize

bottom up: iterative

What is subproblem?

Memoize!

Min Spanning tree A is min spanning tree

greedy find an edge so A still \uparrow

repeat, return A

MST-Kruskal (G, w) $O(E \lg V)$

$A = \emptyset$ best edge of any set

for each vertex $v \in G, v \neq r$

Make-Set (v)

Sort the edges G, E in nondec order by weight

for each edge $(u,v) \in G, E$

if $\text{FindSet}(u) \neq \text{FindSet}(v)$

$A = A \cup \{(u,v)\}$

Union (u,v)

return A

Floyd Warshall (w) $O(n^3)$

$n = W, \text{rows}$

$D^{(0)} = W$

for $k=1$ to n specific points

let $D^{(k)} = d_{ij}^{(k)}$ be new non matrix

for i to n

for j to n

$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

Johnson (reweight) $O(V^2 \lg V + VE)$

1. Find h s.t. $w_n(u,v) \geq 0$
2. Reweight all edges via $w_n(u,v) = w(u,v) + h(u) - h(v)$
3. Run D_{ij} for all source nodes $u \in V$ using w_n
4. Reweight all edges $w(u,v) = w_n(u,v) - h(u) + h(v)$

Network flow

Residual $f(u,v) = c(u,v) - f(u,v) > 0$

Max Flow! Ford Fulkerson $O(|E|f^2)$

initialize flow f to 0
 while there exists aug pt p in G_f
 augment path $f \rightarrow p$
 return f (picks random-silly!)

Resid G_f

$$r_f(u,v) = \begin{cases} c(u,v) - f(u,v) \\ f(v,u) \\ 0 \end{cases}$$

cut $|E|$ always the same

Max Flow = min cut

f is max flow if G_f has no aug paths

Edmond Karp $O(|V|^2)$

Smart pick shortest path w/ BFS $s \rightarrow t$ in G_f

Upward critical when P capacity, R max flow

U = set nodes in G_f reachable s
 V = A

edge in both U, V

Downward critical \downarrow max flow

no path u to v in G_f

P/NP / P^k = poly time

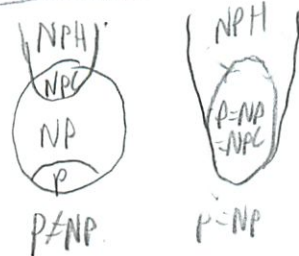
1. Decision \rightarrow Yes/No solvable
2. Search \rightarrow Find an object
3. Optimization \rightarrow Find best

NP nondeterministic poly time verifiable

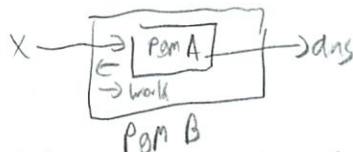
NP-hard if all problems in NP are poly reducible to it (can solve any problem in NP)

NP-complete if it is in NP and is NP-hard

Final Exam



Look have A ∈ P, want to show B ∈ P



Showed that every NP problem reduces to SAT

Karp Know A ∈ NPC, show B ∈ NPC

$A \leq B$ Know \downarrow want to prove



If A is NP-H, then B is NP-H
 A is impossible, if we can solve A by reducing it to B (black box) then done magic
 Since no magic, no B black box, so B is NP-H
 Given input to A, create an input for B
 if A yes \rightarrow B yes
 A no \rightarrow B no

General Technique know A is NPC, show B is NPC

1. Prove verifier to show B ∈ NP
2. Reduce A to B
 - a) give poly-time alg R s.t. if x is input for A, then $R(x)$ input for B
 - b) Show that if $x \in A$, then $R(x) \in B$
 - c) Show that if $R(x) \in B$, then $x \in A$
3. Conclude B is NPC.

Show any NPC problem reduces to it

Bellman Ford for each vertex, for each edge, relax

D_i while queue $\neq \emptyset$, extract min, for each edge, relax

Linear Programming

12/14/12

$$\min x_1 + x_2 + x_3 + x_4$$

$$\text{s.t. } -2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50$$

etc

$$x_1, x_2, x_3, x_4 \geq 0$$

Std Form

$$\max \sum_{j=1}^n c_j x_j$$

$$\text{s.t. } \sum_{j=1}^n a_{ij} x_j \leq b_i \text{ for } i=1, \dots, m$$

$$x_j \geq 0 \text{ for } j=1, \dots, n$$

- max $c^T x$
s.t. $Ax \leq b$
 $x \geq 0$
1. Negate coeff in objective
 2. Non neg $x_j \Rightarrow x_j' - x_j''$
 3. Variables non neg
 $-3x \Rightarrow -3x' + 3x''$
4. Convert equality to 2 inequalities
 5. Negate constraints

Totally Unimodular if det of each sq submatrix of A is 0, -1, or 1
 Then A will have integral optimum

Slack form Minimize $c^T x$
 s.t. $Ax = b$

Simplex alg $x_i \geq 0 \forall x_i \in X$

<u>Dual</u>	<u>Primal</u>	<u>Dual</u>
$\max c^T x$	$\max c^T x$	$\min b^T y$
s.t. $Ax \leq b$	s.t. $Ax = b$	s.t. $A^T y \geq c$
$x \geq 0$		$y \geq 0$

Since $b^T y = c^T x$

Slack $a_{11}x_1 + \dots + a_{1n}x_n + y_1 = b_1$ } m new constraints
 $y_i \geq 0$ } n new variables

Std $x_1 + x_2 + x_3 \leq 30$

Slack $x_4 = 30 - x_1 - x_2 - x_3$
 base var non-basic variable

Pivot swap x_1, x_4
 $x_1 = 30 - x_2 - 2x_3 - x_4$

$x_1 = 30 - (9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_4}{4}) - x_2 - 2x_3$
 ? leaving $x_1 =$ entering
 x_2, x_3, x_4 non basic
 x_1, x_4, x_1 basic

Approx Alg

$$\max \left(\frac{c}{c_{opt}}, \frac{c_{opt}}{c} \right) \leq p(n)$$

To prove:

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

Hamiltonian cycle - visit each vertex once

$$\text{find edge } c(u,v) \leq c(u,x) + c(v,w)$$

vertex cover covers every edge

at least once
maximal ind set = r

Amortized Analysis

Aggregate $\sum c_i$

Accounting $\sum \hat{c}_i \geq \sum c_i$

Potential $\sum \hat{c}_i = \sum c_i + \phi(D_n) - \phi(D_0)$

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

ϕ = bank bal ≥ 0 prepay
 ≤ 0 withdraw

Distributed Algorithms

happen concurrently

Hashing

1. One way
2. Collision Resistant (Any)
3. " " (Targeted)
4. Pseudo randomness
5. Non-malleability

Digital Sig

$$\sigma = \text{sign}(sk_A, M)$$

$$\text{Verify} \rightarrow (M, \sigma, pk_A) = (T, F)$$

$$\text{Sealed bid } c(x) = h(r \| x)$$

Encryption

$$c = E_k(m)$$

$$m = D_k(c)$$

even drops = passive
 m is m = active

key back + forth \xrightarrow{kg}
 $k = (g^b)^a \pmod p \rightarrow k_{ab}$

RSA 1. select 2 large primes p, q

$$2. n = pq$$

$$3. Pick e rel prime to $\phi(n) = (p-1)(q-1)$$$

$$4. Compute d as $e^{-1} \pmod{\phi(n)}$$$

$$5. Publish P = (e, n)$$

$$6. Keep secret S = (d, n)$$

$$P(m) = m^e \pmod n = C$$

$$S(c) = c^d \pmod n = m$$

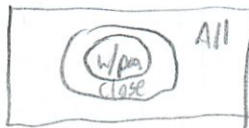
Sub-linear algorithm

Classical

- output is the # that is close to value of optimal sol for given input
- not enough time to build proper sol
- deterministic time alg
- approx ans
- sub-linear time

Property testing

output is correct ans for given input or at least other inputs that are close



- If correct \rightarrow must pass
- If incorrect \rightarrow must fail
- If ϵ -close can pass $(\frac{3}{4})$ correct or fail (incorrect)

Compression

- lossless
- lossy
- Bloom filter

$$h_1(x_1) \quad h_2(x_1) \quad h_3(x_1)$$

Zero Knowledge Proofs

If can tell diff \rightarrow get right each time

If can't prob get right k times in row is $(\frac{1}{2})^k$

Interactive

Prover - unbounded time

Verifier - poly time

$$\max 3x_1 + x_2 + 2x_3$$

$$\text{S.t. } x_1 + x_2 + 3x_3 \leq 30$$

$$2x_1 + 2x_2 + 5x_3 \leq 24$$

$$4x_1 + x_2 + 2x_3 \leq 36$$

$$x_1, x_2, x_3 \geq 0$$

$$\text{dual } \min 30y_1 + 24y_2 + 36y_3$$

$$\text{S.t. } y_1 + 2y_2 + 4y_3 \geq 3$$

$$y_1 + 2y_2 + y_3 \geq 1$$

$$3y_1 + 5y_2 + 2y_3 \geq 2$$

$$y_1, y_2, y_3 \geq 0$$

$$\text{Simplex } z = 3x_1 + x_2 + 2x_3$$

$$x_4 = 30 - x_1 - x_2 - 3x_3$$

$$x_5 = 24 - 2x_1 - 2x_2 - 5x_3$$

$$x_6 = 36 - 4x_1 - x_2 - 2x_3$$

$$x_4 = 30 - \left(9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \right) - x_2 - x_3$$

$$z = 27 + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4}$$

$$x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}$$

$$x_5 = 6 - \frac{3x_2}{2} - 4x_3 + \frac{x_6}{2}$$

Notes

How does mit cut runtime work
Well looks for augmenting paths

Class leader went around and found highest id
save w/ 2 ids

Don't know on time
 $O(n^2)$ everyone send message to everyone
that will work

12: Bloom filter

Didn't put on time on time!
a must be deterministic...

finding det!

⑦

Duplicate cost

- Compression

- lossless

- should have written down the schemes

(~~is~~ so much to need to write down!)

$\log |D|$ bits

L is binary representation of

~~the~~ 16 #s

L is 4 bits

No fancy encoding

Must sort by length

③

$$z_1 + 2z_2 \geq 5 \quad = 5$$

$$-z_1 + z_2 \geq -3 \quad = -3$$

$$\min z_1 + 2z_2$$

$$z_1 = 5$$

$$-5 + 2$$

$$z_2 = 2$$

$$z_1 = 1 \quad -z_1 + z_2 = -3$$

$$z_2 = 4$$

$$z_2 = -3 + z_1$$

$$z_1 + (-3 + z_1) = 5$$

$$2z_1 = 8$$

$$z_1 = 4$$

$$z_2 = 1$$

$$z_1 = 3$$

Practice

Final Exam

- Do not open this exam booklet until you are directed to do so. Read all the instructions first.
- The quiz contains **6** multi-part problems. You have 180 minutes to earn 120 points.
- This quiz booklet contains **11 double-sided** pages, including this one and a double-sided sheet of scratch paper; there should be 18 (numbered) pages of problems.
- This quiz is closed book. You may use **three** double sided Letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted. Cell phones must be put away.
- Write your solutions in the space provided. Extra scratch paper may be provided if you need more room, although your answer should fit in the given space.
- Do not waste time re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Points	Grade	Initials
1	42		
2	42		
3	9		
4	9		
5	8		
6	10		
Total	120		

Name: _____

Circle your recitation:

R01	R02	R03	R04	R05	R06
F10	F11	F12	F1	F2	F3
Joe	Joe	Khanh	Khanh	Emily	Emily
R07	R08	R09	R10		
F11	F12	F1	F2		
Matt	Matt	Geoff	Geoff		

Problem 1. True or False, and Justify [42 points] (14 parts)

Circle **T** or **F** for each of the following statements, and briefly explain why. Your justification is worth more points than your true-or-false designation. If you need to make a reasonable assumption in order to answer a question (for example, if you need to assume that $P \neq NP$), please state that assumption explicitly.

(a) **T F** [3 points] If problem A can be reduced to 3SAT via a deterministic polynomial-time reduction, and $A \in NP$, then A is NP-complete.

(b) **T F** [3 points] Let $G = (V, E)$ be a flow network, i.e., a weighted directed graph with a distinguished source vertex s , a sink vertex t , and non-negative capacity $c(u, v)$ for every edge (u, v) in E . Suppose you find an s - t cut C which has edges e_1, e_2, \dots, e_k and a capacity f . Suppose the value of the maximum s - t flow in G is f .

Now let H be the flow network obtained by adding 1 to the capacity of each edge in C . Then the value of the maximum s - t flow in H is $f + k$.

(c) **T F** [3 points] Let A and B be optimization problems where it is known that A reduces to B in polynomial time. Additionally, it is known that there exists a polynomial-time 2-approximation for B . Then there must exist a polynomial-time 2-approximation for A .

(d) **T F** [3 points] There exists a polynomial-time 2-approximation algorithm for the Traveling Salesman Problem.

(e) **T F** [3 points] A dynamic programming algorithm that solves $\Theta(n^2)$ subproblems could run in $\omega(n^2)$ time.

(f) **T F** [3 points] If A is a Monte Carlo program computing a predicate $f(x)$, and B is a Las Vegas program computing a predicate $g(x)$, then

```
if A(x) then
  return B(x)
else
  return False
```

is a Monte Carlo program computing $(f(x) \wedge g(x))$.

(g) **T F** [3 points] Dynamic programming programs require space at least proportional to the number of subproblems generated (in order to “memoize” the solution to each subproblem).

(h) **T F** [3 points] Let $H = \{h_i : \{1, 2, 3\} \rightarrow \{0, 1\}\}$ be a hash family defined as follows.

	1	2	3
h_1	0	1	0
h_2	1	0	1
h_3	1	1	0

(For example, $h_1(3) = 0$.)

Then H is a universal hash family.

- (i) **T F** [3 points] If we use a max-queue instead of a min-queue in Kruskal's MST algorithm, it will return the spanning tree of maximum total cost (instead of returning the spanning tree of minimum total cost). (Assume the input is a weighted connected undirected graph.)
- (j) **T F** [3 points] Define a graph as being *tripartite* if its vertices can be partitioned into three sets X_1, X_2, X_3 such that no edge in the graph has both vertices in the same set. (That is, all edges are between vertices in different sets.) Then deciding whether a graph is tripartite can be done in polynomial time.

- (k) **T F** [3 points] A randomized algorithm for a decision problem with one-sided-error and correctness probability $1/3$ (that is, if the answer is YES, it will always output YES, while if the answer is NO, it will output NO with probability $1/3$) can always be amplified to a correctness probability of 99%.

- (l) **T F** [3 points] Let B_0, B_1, B_2, \dots be an infinite sequence of decision problems, where B_0 is known to be NP-hard and

$$B_i \leq_P B_{i+1} \text{ for all } i \geq 0.$$

Then it must be the case that B_i is NP-hard for all $i \geq 0$.

(m) **T F** [3 points] Let L be a decision problem. If there exists an interactive proof for L where the verifier is deterministic, then $L \in NP$.

(n) **T F** [3 points] Let L be a decision problem. If there exists an interactive proof for L where the prover runs in polynomial time, then $L \in P$.

Problem 2. Short Answer [41 points] (9 parts)

Give *brief*, but complete, answers to the following questions.

- (a) [4 points] Let $F_1, F_2, \dots = 1, 1, 2, 3, 5, 8, \dots$ denote the usual sequence of Fibonacci numbers (defined by $F_1 = 1$, $F_2 = 1$, and $F_i = F_{i-1} + F_{i-2}$ for $i > 2$).

Suppose that a file to be compressed contains k different symbols a_1, a_2, \dots, a_k and that it contains F_i occurrences of a_i for each i . Thus, if $k = 4$, the string has length 7 and contains 2 occurrences of a_3 .

Assume the file is encoded with Huffman encoding. How many bits will be used to encode a_i , as a function of i and/or k ? State your answer concisely. You do not need to provide a proof.

- (b) [4 points] As a final project for one of your other Course 6 classes, you have a massive program to run. After much effort, you are able to parallelize 90% of your code. The computer lab has two systems on which you could run your program:

- a cluster of 90 single-core computers each running at 1GHz, and
- a computer with 9 cores each running at 2GHz.

Which one should you choose to complete your project as quickly as possible?

(c) [5 points] Recall the clique problem from lecture: Given an undirected graph $G = (V, E)$ and a positive integer k , is there a subset C of V of size at least k such that every pair of vertices in C has an edge between them?

Ben Bitdiddle thinks he can solve the clique problem in polynomial time using linear programming.

- Let each variable in the linear program represent whether or not each vertex is a part of our clique. Add constraints stating that each of these variables must be nonnegative and at most one.
- We go through the graph G and consider each pair of vertices. For every pair of vertices where there is *not* an edge in G , add a constraint stating that the sum of the variables corresponding to the endpoint vertices must be at most one. This ensures that both of them cannot be part of a clique if there is no edge between them.
- The objective function is the sum of the variables corresponding to the vertices. We wish to maximize this function.

Ben argues that the value of the optimum must be the size of the maximum-size clique in G , and we can then simply compare this value to k . Explain the flaw in Ben's logic.

- (d) [4 points] In a weighted connected undirected graph that might have negative-weight edges but no negative-weight cycles, how would you find a triple of distinct vertices x, y, z that minimizes $f(x, y, z) = d(x, y) + d(y, z) + d(z, x)$ where $d(u, v)$ is the length of the shortest path from u to v ?

The running time of your algorithm should be $O(n^3)$, where n is the number of vertices in the graph.

- (e) [4 points] Suppose you are using RSA and you change your public key (e, N) every so often, where $N = pq$ is the product of your two large secret primes.

Why is it *not* a good idea to leave p the same and just replace q with a different secret prime q' (so your new N' is just pq')?

- (f) [7 points] You are working at a hospital trying to diagnose patients; you may assume that each patient has exactly one disease. You know of m different diseases d_1, d_2, \dots, d_m . You have n different tests you can run (labeled T_1, T_2, \dots, T_n), each of which comes up positive for some set of diseases and negative for other diseases. You would like to correctly diagnose all patients while giving them the minimum necessary number of tests—or, at least, close to the minimum number. Since you must send the tests to the lab for processing, all tests must be performed in parallel.

We say that a set of tests $T \subseteq \{T_1, T_2, \dots, T_n\}$ is *comprehensive* if, for every pair of diseases (d_i, d_j) , there is some test $T_k \in T$ that distinguishes them—that is, it returns positive for one and negative for the other. The minimum-comprehensive-set problem (MCS) is the problem of finding a comprehensive set of tests of minimum cardinality. MCS is known to be NP-hard.

Describe a polynomial-time α -approximation algorithm for the MCS problem, where $\alpha = \ln(m(m-1)/2)$.

- (g) [3 points] State the three properties a trapdoor function should have.

- (h) [4 points] Suppose you are given a polynomial time algorithm DECISION-FACTOR that, given two integers k and n , returns YES if n has a prime factor less than k , and NO if n does not. Give a polynomial time algorithm for computing a single prime factor of n .

Problem 3. More Spy Games [9 points]

An enemy country, Elbonia, has n transmitter/receiver pairs (t_i, r_i) . You can model the position of each t_i and each r_i as a point in the plane. Enemy communications travel along the straight-line segment from t_i to r_i . You can place eavesdrop units at any point in the plane, but a unit must be on the line segment from t_i to r_i in order to eavesdrop successfully. If you put a unit at the intersection of two such segments, that unit can eavesdrop on both transmitter/receiver pairs. Assume no three such segments intersect at a point.

Your intelligence agency has given you a list of the coordinates of all n enemy transmitter/receiver pairs. Briefly describe a polynomial time algorithm for finding the minimum number of eavesdrop units required to eavesdrop on all n transmitter/receiver pairs. (No proof needed.)

Problem 4. Almost Sorted [9 points]

A sequence x_1, x_2, \dots, x_n of real numbers is said to be **sorted** if

$$x_1 \leq x_2 \leq \dots \leq x_n.$$

We say that x_1, x_2, \dots, x_n is **D-almost-sorted** for a non-negative real number D if there exists another sequence y_1, y_2, \dots, y_n of real numbers such that y_1, y_2, \dots, y_n is sorted, and $\sum_i |x_i - y_i| \leq D$. (That is, by “shifting” values x_i to new values y_i , such that the total amount of shifting is at most D , the new set of numbers is sorted.)

Describe concisely a polynomial-time algorithm which, given an input sequence x_1, x_2, \dots, x_n and a non-negative real number D , determines whether x is D -almost-sorted.

Problem 5. Randomized 3-Coloring [8 points] (3 parts)

In an undirected graph $G = (V, E)$, a *coloring* is a mapping c which assigns colors to vertices. We denote the color of vertex v by $c(v)$.

We say a coloring c *satisfies* an edge $e = (u, v)$ if $c(u) \neq c(v)$ (that is, the endpoints of the edges are assigned different colors). Let the function $s(c)$ count the number of satisfied edges under a coloring c .

Define the *3-coloring optimization problem* as follows: Given an undirected graph $G = (V, E)$, output a coloring c such that $c(v) \in \{R, W, B\}$ for all $v \in V$, such that $s(c)$ is maximized.

Here is one very simple randomized algorithm:

RANDOMIZED-COLOR(G)

```
1 for each  $v \in V$ 
2   Pick a color uniformly at random in  $\{R, W, B\}$ 
3   Let  $c(v) =$  color picked
4 return  $c$ 
```

- (a) [2 points] Let e be any edge. What is the probability that the coloring picked satisfies e ?

(b) [2 points] What is the expected number of edges satisfied by the coloring produced by c ? Justify.

(c) [4 points] Show that RANDOMIZED-COLOR is a polynomial-time randomized $(3/2)$ -approximation algorithm for the 3-coloring optimization problem. That is, show that $E(s(c)) \geq (2/3)s(c^*)$ where c^* is the optimal coloring.

Problem 6. Sublinear-Time Unimodal Testing [10 points]

We say that an array $A[1..n]$ of real numbers is **unimodal** if there exists an integer k such that $1 \leq k \leq n$, $A[1..k]$ is monotonically non-decreasing, and $A[k..n]$ is monotonically non-increasing.

We say that A is ϵ -far from being unimodal if you have to remove more than ϵn elements from A in order for the remaining sequence to be unimodal.

Give a sublinear-time property tester that, given an array $A[1..n]$ of *distinct* real numbers:

- if A is unimodal, outputs YES with probability 1, and
- if A is ϵ -far from being unimodal, outputs NO with probability at least $2/3$.

SCRATCH PAPER

SCRATCH PAPER

Final Exam

- Do not open this exam booklet until you are directed to do so. Read all the instructions first.
- The quiz contains 6 multi-part problems. You have 180 minutes to earn 120 points.
- This quiz booklet contains **11 double-sided** pages, including this one and a double-sided sheet of scratch paper; there should be 18 (numbered) pages of problems.
- This quiz is closed book. You may use **three** double sided Letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted. Cell phones must be put away.
- Write your solutions in the space provided. Extra scratch paper may be provided if you need more room, although your answer should fit in the given space.
- Do not waste time re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Points	Grade	Initials
1	42		
2	42		
3	9		
4	9		
5	8		
6	10		
Total	120		

Name: _____

Circle your recitation:

R01	R02	R03	R04	R05	R06
F10	F11	F12	F1	F2	F3
Joe	Joe	Khanh	Khanh	Emily	Emily
R07	R08	R09	R10		
F11	F12	F1	F2		
Matt	Matt	Geoff	Geoff		

Problem 1. True or False, and Justify [42 points] (14 parts)

Circle **T** or **F** for each of the following statements, and briefly explain why. Your justification is worth more points than your true-or-false designation. If you need to make a reasonable assumption in order to answer a question (for example, if you need to assume that $P \neq NP$), please state that assumption explicitly.

- (a) **T F** [3 points] If problem A can be reduced to 3SAT via a deterministic polynomial-time reduction, and $A \in NP$, then A is NP-complete.

Solution: False. We need to reduce in the other direction (reduce an NP-hard problem to A).

- (b) **T F** [3 points] Let $G = (V, E)$ be a flow network, i.e., a weighted directed graph with a distinguished source vertex s , a sink vertex t , and non-negative capacity $c(u, v)$ for every edge (u, v) in E . Suppose you find an s - t cut C which has edges e_1, e_2, \dots, e_k and a capacity f . Suppose the value of the maximum s - t flow in G is f .

Now let H be the flow network obtained by adding 1 to the capacity of each edge in C . Then the value of the maximum s - t flow in H is $f + k$.

Solution: False. There could be multiple min-cuts. Consider the graph s - v - t where the edges have capacity 1; either edge in itself is a min-cut, but adding capacity to that edge alone does not increase the max flow.

- (c) **T F** [3 points] Let A and B be optimization problems where it is known that A reduces to B in polynomial time. Additionally, it is known that there exists a polynomial-time 2-approximation for B . Then there must exist a polynomial-time 2-approximation for A .

Solution: False; approximation factor is not (necessarily) carried over in polynomial-time reduction. See e.g. set cover vs. vertex cover.

- (d) **T F** [3 points] There exists a polynomial-time 2-approximation algorithm for the Traveling Salesman Problem.

Solution: False, assuming $P \neq NP$. There is an approximation algorithm in the special case where the graph obeys the triangle inequality, but we don't know of one in general.

- (e) **T F** [3 points] A dynamic programming algorithm that solves $\Theta(n^2)$ subproblems could run in $\omega(n^2)$ time.

Solution: True. It could take $\omega(1)$ time per subproblem.

- (f) **T F** [3 points] If A is a Monte Carlo program computing a predicate $f(x)$, and B is a Las Vegas program computing a predicate $g(x)$, then

```

if A(x) then
    return B(x)
else
    return False

```

is a Monte Carlo program computing $(f(x) \wedge g(x))$.

Solution: False. B has a chance of taking arbitrarily long, so the algorithm is not Monte Carlo.

- (g) **T F** [3 points] Dynamic programming programs require space at least proportional to the number of subproblems generated (in order to “memoize” the solution to each subproblem).

Solution: False. Under some circumstances, we can reuse the same space for multiple subproblems; for example, if each subproblem of size k only looks at subproblems of size $k - 1$, then when calculating bottom-up we need not store subproblems of size $k - 2$ once all subproblems of size $k - 1$ have been calculated. (See the discussion of longest common subsequence in CLRS.)

- (h) **T F** [3 points] Let $H = \{h_i : \{1, 2, 3\} \rightarrow \{0, 1\}\}$ be a hash family defined as follows.

	1	2	3
h_1	0	1	0
h_2	1	0	1
h_3	1	1	0

(For example, $h_1(3) = 0$.)

Then H is a universal hash family.

Solution: False. Consider elements 1 and 3: h_1 and h_2 both cause a collision between them, so in particular a uniformly random hash function chosen from H causes a collision between 1 and 3 with probability $2/3$, greater than the $1/2$ allowed for universal hashing (since there are 2 hash buckets).

- (i) **T F** [3 points] If we use a max-queue instead of a min-queue in Kruskal's MST algorithm, it will return the spanning tree of maximum total cost (instead of returning the spanning tree of minimum total cost). (Assume the input is a weighted connected undirected graph.)

Solution: True. The proof is essentially the same as for the usual Kruskal's algorithm. Alternatively, this is equivalent to negating all the edge weights and running Kruskal's algorithm.

- (j) **T F** [3 points] Define a graph as being *tripartite* if its vertices can be partitioned into three sets X_1, X_2, X_3 such that no edge in the graph has both vertices in the same set. (That is, all edges are between vertices in different sets.) Then deciding whether a graph is tripartite can be done in polynomial time.

Solution: False, assuming $P \neq NP$. This is exactly 3-colorability; partitioning the vertices into three sets with no internal edges is the same as coloring them with three colors such that no edge has two endpoints of the same color. As seen in lecture, 3-coloring is NP-complete.

- (k) **T F** [3 points] A randomized algorithm for a decision problem with one-sided-error and correctness probability $1/3$ (that is, if the answer is YES, it will always output YES, while if the answer is NO, it will output NO with probability $1/3$) can always be amplified to a correctness probability of 99%.

Solution: True. Since the error is one-sided, it in fact suffices for the correctness probability to be any constant > 0 . We can then repeat it, say, k times, and output NO if we ever see a NO, and YES otherwise. Then, if the correct answer is YES, all k repetitions of our algorithm will output YES, so our final answer is also YES, and if the correct answer is NO, each of our k repetitions has a $1/3$ chance of returning NO, in which case our final answer is, correctly, NO, with probability $1 - (2/3)^k$, so $k = \log_{3/2} 100$ repetitions suffice.

- (l) **T F** [3 points] Let B_0, B_1, B_2, \dots be an infinite sequence of decision problems, where B_0 is known to be NP-hard and

$$B_i \leq_P B_{i+1} \text{ for all } i \geq 0.$$

Then it must be the case that B_i is NP-hard for all $i \geq 0$.

Solution: True. This can be seen by induction; if B_i is NP-hard, and there is a polynomial-time reduction from B_i to B_{i+1} , then B_{i+1} is NP-hard.

- (m) **T F** [3 points] Let L be a decision problem. If there exists an interactive proof for L where the verifier is deterministic, then $L \in NP$.

Solution: True. If the verifier is deterministic, the transcript of the interactions between the prover and verifier will always be the same. Thus, the transcript itself is a polynomial-sized certificate for L .

- (n) **T F** [3 points] Let L be a decision problem. If there exists an interactive proof for L where the prover runs in polynomial time, then $L \in P$.

Solution: False. Either the prover or the verifier could be randomized, which would allow them to prove a larger class of problems (assuming $P \neq BPP$).

Problem 2. Short Answer [41 points] (9 parts)

Give *brief*, but complete, answers to the following questions.

- (a) [4 points] Let $F_1, F_2, \dots = 1, 1, 2, 3, 5, 8, \dots$ denote the usual sequence of Fibonacci numbers (defined by $F_1 = 1, F_2 = 1$, and $F_i = F_{i-1} + F_{i-2}$ for $i > 2$).

Suppose that a file to be compressed contains k different symbols a_1, a_2, \dots, a_k and that it contains F_i occurrences of a_i for each i . Thus, if $k = 4$, the string has length 7 and contains 2 occurrences of a_3 .

Assume the file is encoded with Huffman encoding. How many bits will be used to encode a_i , as a function of i and/or k ? State your answer concisely. You do not need to provide a proof.

Solution: $(k + 1) - i$ for $i > 1$; $k - 1$ for $i = 1$

- (b) [4 points] As a final project for one of your other Course 6 classes, you have a massive program to run. After much effort, you are able to parallelize 90% of your code. The computer lab has two systems on which you could run your program:

- a cluster of 90 single-core computers each running at 1GHz, and
- a computer with 9 cores each running at 2GHz.

Which one should you choose to complete your project as quickly as possible?

Solution: Compared to a single 1GHz single-core machine, the first option offers a speedup factor of

$$\frac{1}{.1 + .9/90} = \frac{1}{.11} \approx 9,$$

while the second offers a speedup factor of

$$\frac{2}{.1 + .9/9} = \frac{2}{.2} = 10.$$

So you should go with the second.

- (c) [5 points] Recall the clique problem from lecture: Given an undirected graph $G = (V, E)$ and a positive integer k , is there a subset C of V of size at least k such that every pair of vertices in C has an edge between them?

Ben Bitdiddle thinks he can solve the clique problem in polynomial time using linear programming.

- Let each variable in the linear program represent whether or not each vertex is a part of our clique. Add constraints stating that each of these variables must be nonnegative and at most one.
- We go through the graph G and consider each pair of vertices. For every pair of vertices where there is *not* an edge in G , add a constraint stating that the sum of the variables corresponding to the endpoint vertices must be at most one. This ensures that both of them cannot be part of a clique if there is no edge between them.
- The objective function is the sum of the variables corresponding to the vertices. We wish to maximize this function.

Ben argues that the value of the optimum must be the size of the maximum-size clique in G , and we can then simply compare this value to k . Explain the flaw in Ben's logic.

Solution: This is an integer program, not a linear program, and therefore we don't know how to solve it in polynomial time. (Alternatively, if we don't add the integrality constraints, we can solve it in polynomial time but will likely get fractional values back; it's unclear what fractional values of the variables mean with regard to the clique.)

- (d) [4 points] In a weighted connected undirected graph that might have negative-weight edges but no negative-weight cycles, how would you find a triple of distinct vertices x, y, z that minimizes $f(x, y, z) = d(x, y) + d(y, z) + d(z, x)$ where $d(u, v)$ is the length of the shortest path from u to v ?

The running time of your algorithm should be $O(n^3)$, where n is the number of vertices in the graph.

Solution: Run Johnson's all-pairs shortest-paths algorithm to find all of the shortest paths $d(u, v)$. This takes $O(V^2 \log V + VE) = O(n^3)$ time, since $E = O(n^2)$. Then calculate f for all triples of vertices in the graph, and take the minimum. There are $O(n^3)$ triples, and f can be calculated in $O(1)$ time given the $d(u, v)$ values, so this step also takes $O(n^3)$ time.

(e) [4 points] Suppose you are using RSA and you change your public key (e, N) every so often, where $N = pq$ is the product of your two large secret primes.

Why is it *not* a good idea to leave p the same and just replace q with a different secret prime q' (so your new N' is just pq')?

Solution: Anyone could compute the GCD of two of your public keys (using the Euclidean algorithm, which is polynomial-time) to find p , and thus factor N and N' .

(f) [7 points] You are working at a hospital trying to diagnose patients; you may assume that each patient has exactly one disease. You know of m different diseases d_1, d_2, \dots, d_m . You have n different tests you can run (labeled T_1, T_2, \dots, T_n), each of which comes up positive for some set of diseases and negative for other diseases. You would like to correctly diagnose all patients while giving them the minimum necessary number of tests—or, at least, close to the minimum number. Since you must send the tests to the lab for processing, all tests must be performed in parallel.

We say that a set of tests $T \subseteq \{T_1, T_2, \dots, T_n\}$ is *comprehensive* if, for every pair of diseases (d_i, d_j) , there is some test $T_k \in T$ that distinguishes them—that is, it returns positive for one and negative for the other. The minimum-comprehensive-set problem (MCS) is the problem of finding a comprehensive set of tests of minimum cardinality. MCS is known to be NP-hard.

Describe a polynomial-time α -approximation algorithm for the MCS problem, where $\alpha = \ln(m(m-1)/2)$.

Solution: For every pair of diseases, there is at least one of the tests that distinguishes them, and we want a minimum-cardinality set of the tests that between them distinguish all diseases. This is simply the Set Cover problem operating on *pairs* of diseases; we can use the standard approximation for Set Cover seen in CLRS/lecture.

(g) [3 points] State the three properties a trapdoor function should have.

Solution: Easy to compute, hard to invert without the trapdoor information, easy to invert with the trapdoor information.

- (h) [4 points] Suppose you are given a polynomial time algorithm DECISION-FACTOR that, given two integers k and n , returns YES if n has a prime factor less than k , and NO if n does not. Give a polynomial time algorithm for computing a single prime factor of n .

Solution: Use DECISION-FACTOR in a binary search to find the smallest prime factor p of n : for every $m \leq p$ we have DECISION-FACTOR(m) = NO and for every $m > p$ we have DECISION-FACTOR(m) = YES. This takes $O(\log n)$ calls to DECISION-FACTOR, which is polynomial (linear, in fact) in the length of the input n , so the overall running time is polynomial as well.

Problem 3. More Spy Games [9 points]

An enemy country, Elbonia, has n transmitter/receiver pairs (t_i, r_i) . You can model the position of each t_i and each r_i as a point in the plane. Enemy communications travel along the straight-line segment from t_i to r_i . You can place eavesdrop units at any point in the plane, but a unit must be on the line segment from t_i to r_i in order to eavesdrop successfully. If you put a unit at the intersection of two such segments, that unit can eavesdrop on both transmitter/receiver pairs. Assume no three such segments intersect at a point.

Your intelligence agency has given you a list of the coordinates of all n enemy transmitter/receiver pairs. Briefly describe a polynomial time algorithm for finding the minimum number of eavesdrop units required to eavesdrop on all n transmitter/receiver pairs. (No proof needed.)

Solution: Construct a graph G consisting of a vertex v_i for each transmitter/receiver pair (t_i, r_i) , and an edge between vertices v_i and v_j if the corresponding line segments intersect. This can be done in $O(n^2)$ time. Now, any vertex that is isolated must be eavesdropped on by its own dedicated unit, and we can remove it from consideration. The problem then reduces to finding a minimum edge cover of G , that is, the minimum number of edges such that every vertex in G is incident on at least one.

We can do this by first finding a maximum matching in G (using any of several matching algorithms covered in class, all of which run in polynomial time), and adding an edge to cover each of the remaining uncovered vertices. To see why this indeed achieves a minimum edge cover, observe that any edge cover contains a matching, each edge of which covers two vertices, together with some additional edges, each of which covers a single additional vertex. Thus the smallest edge cover we can hope to obtain comprises, in this manner, of a maximum matching of G together with an edge for each remaining unmatched vertex. But this is indeed what we construct, so it must be the minimum edge cover, and we are done.

Problem 4. Almost Sorted [9 points]

A sequence x_1, x_2, \dots, x_n of real numbers is said to be **sorted** if

$$x_1 \leq x_2 \leq \dots \leq x_n.$$

We say that x_1, x_2, \dots, x_n is **D-almost-sorted** for a non-negative real number D if there exists another sequence y_1, y_2, \dots, y_n of real numbers such that y_1, y_2, \dots, y_n is sorted, and $\sum_i |x_i - y_i| \leq D$. (That is, by "shifting" values x_i to new values y_i , such that the total amount of shifting is at most D , the new set of numbers is sorted.)

Describe concisely a polynomial-time algorithm which, given an input sequence x_1, x_2, \dots, x_n and a non-negative real number D , determines whether x is D -almost-sorted.

Solution: We solve this problem using linear programming. To determine whether a sequence x_1, x_2, \dots, x_n is D -almost-sorted, check whether the following LP is feasible:

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & \\ & x_i = y_i + c_i - d_i \quad \text{for } i = 1, \dots, n \\ & y_i \leq y_{i+1} \quad \text{for } i = 1, \dots, n-1 \\ & \sum_i (c_i + d_i) \leq D \end{array}$$

(The objective function is irrelevant.)

Alternatively, we could solve the following LP, and then check whether the optimal value of the objective function is at most D .

$$\begin{array}{ll} \text{minimize} & \sum_i (c_i + d_i) \\ \text{subject to} & \\ & x_i = y_i + c_i - d_i \quad \text{for } i = 1, \dots, n \\ & y_i \leq y_{i+1} \quad \text{for } i = 1, \dots, n-1 \end{array}$$

Linear programming can be solved in worst-case polynomial time by the ellipsoid algorithm or interior-point methods.

Problem 5. Randomized 3-Coloring [8 points] (3 parts)

In an undirected graph $G = (V, E)$, a *coloring* is a mapping c which assigns colors to vertices. We denote the color of vertex v by $c(v)$.

We say a coloring c *satisfies* an edge $e = (u, v)$ if $c(u) \neq c(v)$ (that is, the endpoints of the edges are assigned different colors). Let the function $s(c)$ count the number of satisfied edges under a coloring c .

Define the *3-coloring optimization problem* as follows: Given an undirected graph $G = (V, E)$, output a coloring c such that $c(v) \in \{R, W, B\}$ for all $v \in V$, such that $s(c)$ is maximized.

Here is one very simple randomized algorithm:

RANDOMIZED-COLOR(G)

```

1 for each  $v \in V$ 
2   Pick a color uniformly at random in  $\{R, W, B\}$ 
3   Let  $c(v)$  = color picked
4 return  $c$ 

```

(a) [2 points] Let e be any edge. What is the probability that the coloring picked satisfies e ?

Solution: 2/3

- (b) [2 points] What is the expected number of edges satisfied by the coloring produced by c ? Justify.

Solution: $2|E|/3$, due to linearity of expectation over all edges.

- (c) [4 points] Show that RANDOMIZED-COLOR is a polynomial-time randomized $(3/2)$ -approximation algorithm for the 3-coloring optimization problem. That is, show that $E(s(c)) \geq (2/3)s(c^*)$ where c^* is the optimal coloring.

Solution: The optimal coloring can satisfy at most $|E|$ edges, so $s(c^*) \leq |E|$. From (b), $E[s(c)] = 2|E|/3$. Thus, $E(s(c)) \geq (2/3)s(c^*)$.

Problem 6. Sublinear-Time Unimodal Testing [10 points]

We say that an array $A[1..n]$ of real numbers is **unimodal** if there exists an integer k such that $1 \leq k \leq n$, $A[1..k]$ is monotonically non-decreasing, and $A[k..n]$ is monotonically non-increasing.

We say that A is ϵ -far from being unimodal if you have to remove more than ϵn elements from A in order for the remaining sequence to be unimodal.

Give a sublinear-time property tester that, given an array $A[1..n]$ of *distinct* real numbers:

- if A is unimodal, outputs YES with probability 1, and
- if A is ϵ -far from being unimodal, outputs NO with probability at least $2/3$.

Solution: We first use binary search to find a candidate k : each query will be of two consecutive indices, to see if we are to the left of k or to the right. Since A consists of all distinct values, we will never get a tie. Then, once we have such a k , we run our monotonicity tester with parameter ϵ on $A[1..k]$ and $A[k..n]$. If both return YES, we return YES, otherwise we return NO.

To see that this is correct, suppose that A is unimodal. Then our search for k must return the correct k , and $A[1..k]$ and $A[k..n]$ must both be monotone, so our subroutines both return YES and we return YES as well, as required. On the other hand, if A is ϵ -far from being unimodal, meaning we need to remove ϵn elements from A to make it unimodal, then no matter what k we pick, either $A[1..k]$ or $A[k..n]$ must be ϵ -far from being monotone. Specifically, suppose, to get a contradiction, that $A[1..k]$ and $A[k..n]$ are both at most ϵ' -far from being monotone, for some $\epsilon' < \epsilon$. This means we can remove $\epsilon'k$ elements from $A[1..k]$ to make it monotone, and $\epsilon'(n-k)$ elements from $A[k..n]$ to make it monotone. But then we could remove these same $\epsilon'k + \epsilon'(n-k) = \epsilon'n < \epsilon n$ elements from $A[1..n]$ to make it unimodal, contradicting the fact that A is ϵ -far from being unimodal. It follows that at least one of our subroutines must return NO with probability at least $2/3$, so we also return NO with probability at least $2/3$.

Note that our monotonicity tester allows for a specification of the direction of monotonicity (i.e., increasing or decreasing).

SCRATCH PAPER

SCRATCH PAPER