

Final Exam

You will have 3 hours to complete this exam. No notes or other resources are allowed. Unless otherwise specified, full credit will only be given to a correct answer which is described clearly and concisely.

Do not discuss this exam with anyone who has not yet taken it.

Problem	Points	Grade	Initials
Name	2	2	DI
1	24	8	\$
2	20	20	\$
3	24	12	DI
4	20	7	ATD
5	15	1	CD
6	25	6	SK
Total	130	56	HP

Name: [1 point]

Michael Plasmator

R01
WF10
Shaunak

R02
WF11
Shaunak

R03
WF12
Alan

R04
WF1
Jeff

R05
WF2
Rafael

R06
WF3
Henrique

R07
WF3
Dragos

Problem 1. True/False [24 points]

Answer True or False. You do not need to explain your answer.

Correct answers are worth 2 points. Blanks and incorrect answers are worth 0 points.

- (a) The asymptotic solution to the recurrence $T(n) = 3T(n/3) + \Theta(n^2)$ is $T(n) = \Theta(n^2)$.
no guessing penalty

Circle one: True or False

- (b) Given a binary search tree, it is possible to build a balanced binary search tree containing the same elements in linear time. *no*

Circle one: True or False

- (c) For any node in an AVL tree, the size of the subtree rooted at the node's left child and the size of the subtree rooted at the node's right child differ by at most 1. (The size of a BST is the number of nodes in the tree.)

Circle one: True or False

- (d) Suppose that we are using linear probing to maintain a hash table that already contains some elements. Under the SUHA assumption for our hash function, the next element is equally likely to end up in any of the empty slots of the table.¹

Circle one: True or False

- (e) If you use chaining to resolve collisions in a hash table under the SUHA assumption, you can perform inserts, deletions, and lookups in $O(1 + \alpha)$ time in the average case, but not the worst case. (If there are n elements stored in a hash table of size m , then the load factor α is $\frac{n}{m}$.)

Circle one: True or False

- (f) Performing the extract-min operation k times in a row on a min-heap can be done in time $O(k)$.

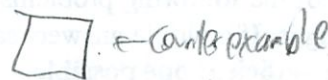
Circle one: True or False

¹If we are using $h_i(x) = h(x) + i \bmod m$, we are assuming SUHA for the function $h(x)$.

- (g) Given any undirected graph, it is always possible to assign a direction to each edge so that the resulting graph is a directed acyclic graph (DAG).

Circle one: True or False

cycles



- (h) Suppose that in an unweighted, undirected graph G , an edge e is on a shortest path from a vertex s to a vertex t . If we perform BFS on G starting at s , then e must appear in the BFS tree.

Circle one: True or False



BFS finds shortest paths

- (i) If you compute the n th Fibonacci number recursively using the Fibonacci recurrence, and you do not memoize any values, then the runtime of this computation will be exponential in n .

stupid

Circle one: True or False

- (j) There exists an algorithm that, given positive integers a and b , returns a^b in time polynomial in $(\log a + \log b)$.

rewards from recursion

Circle one: True or False

binary fast exp

- (k) There exists an algorithm that, given positive integers a , b , and c , returns $a^b \pmod{c}$ in time polynomial in $(\log a + \log b + \log c)$.

Circle one: True or False

divide by c not $\log c$

- (l) Suppose that being given a black box algorithm for the problem L_1 would let us solve problem L_2 in polynomial time. Then, if L_2 is solvable in polynomial time, so is L_1 .

Circle one: True or False

NP-team

wording weird

unless linguistic trick

Problem 2. Algorithmic Techniques [20 points]

For each of the following problems, indicate which algorithmic technique would be most useful for solving it. If multiple answers satisfy the requirements of the problem, you should choose the most time-efficient one possible.

- (a) Given a string s of length n and a string p of length k , you want to determine if p is a substring of s in $O(n + k)$ expected time.

Circle one: **Sorting** or **Hashing** or **Binary Search Trees**

- (b) Yelp maintains a database of information about restaurants: it assigns a quality rating and a cost to each restaurant. Users should be able to search the database for the highest rated restaurant they can dine at without exceeding a given cost. Yelp should also be able to add new restaurants and to delete old ones which go out of business.

Circle one: **Sorting** or **Hashing** or **Binary Search Trees**

- (c) In a rock-paper-scissors tournament of n players, each player plays one game against each other player. A player's record is the triple (w, t, l) which records the number of games they won, tied, and lost, respectively.

One player's record is better than another's if the first player won more games than the second did, or if they won the same number of games and the first player tied more. We want to build a data structure, as efficiently as possible, which can answer queries "Who had the i th best record?" in $O(1)$ time.

Circle one: **Sorting** or **Hashing** or **Binary Search Trees**

- (d) The TAs are compiling a collection of algorithms problems for a new algorithms class. Each of these problems has a (distinct) difficulty rating, which is a real number. When building a test, they want to be able to efficiently find the problem with difficulty closest to a given value d . In addition, they want to be able to add new problems to the collection and to delete problems when they are used on the test.

Circle one: **Sorting** or **Hashing** or **Binary Search Trees**

- (e) Alan has a large file that he wants to distribute to many other people over the Internet. He wants to send them a small amount of additional information that they can use to verify that the downloaded file is correct.

Circle one: **Sorting** or **Hashing** or **Binary Search Trees**

TA: Integer
b/w 0, 300
Not \$, #, or %

seems BST
but how
hashing seems better

for each cost \rightarrow the highest rated rest

"majority"

quickly find d

- 60
- (f) The dating website AlrightCupid wants to use a hash table to store the information about its users. Because of the architecture of their system, they cannot afford to ever resize the hash table, yet they want to be able to insert arbitrarily many users into the database.

Circle one: Chaining or Open addressing

- (g) Dragos wants to find a cycle in a directed graph.

TA: Unweighted

Circle one: BFS or DFS or Dijkstra's algorithm or Bellman-Ford

- (h) Rafael wants to determine if there is a positive-weight cycle in an graph with arbitrary integers as edge weights.

Circle one: BFS or DFS or Dijkstra's algorithm or Bellman-Ford

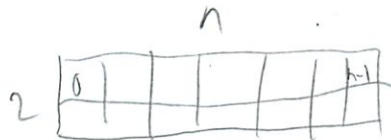
- (i) Jeff wants to determine the set of vertices that can be reached in a graph by traversing at most k edges starting at a node s .

Circle one: BFS or DFS or Dijkstra's algorithm or Bellman-Ford

- (j) FaceLook stores a social network graph as a set of adjacency lists: for each user, it stores a list of that user's friends. Because friendship is mutual, this graph is undirected. The social network graph has diameter 6, and for k up to 6, there are roughly 100^k users at distance k from any given user. FaceLook wants to implement an algorithm to compute the shortest path between two users in this graph.

Circle one: Bi-directional BFS or A* search

What is the heuristic
- is none

**Problem 3. Short Answers [24 points]****(a) Peak finding in a $2 \times n$ array [4 points]**

A $2 \times n$ array A of integers has elements indexed by the notation $A[0][i]$ and $A[1][i]$ for $i = 0, \dots, n-1$. Two elements are adjacent if they differ by 1 in one coordinate and are equal in the other coordinate. For example, $A[0][5]$ is adjacent to $A[1][5]$ and $A[0][4]$, but these two elements are not adjacent to each other. A *peak* is an element with a value at least as large as all its adjacent elements' values.

Consider the following outlines of algorithms for finding a peak in a $2 \times n$ array.

- i. Use any $O(\log n)$ 1-D peak finding algorithm on the arrays B_0 and B_1 , where $B_0[i] = A[0][i]$ and $B_1[i] = A[1][i]$. (The B_i are the bottom and top halves of A .) Return the maximum of the two values found.

Circle: **Correct** or **Incorrect**

B_0
 B_1

- ii. Use any $O(\log n)$ 1-D peak finding algorithm on the array of column maxes $B[i] = \max(A[0][i], A[1][i])$. Do not explicitly store B ; only compute values of B on demand (that is, only when they are needed).

Circle: **Correct** or **Incorrect**

(b) Locating the max [8 points]

Suppose you are given a perfectly balanced binary tree. There are n nodes in this tree. Each non-leaf node has a left and right child, and all the leaves are at the same distance from the root. We want to store the elements $1, 2, \dots, n$ in the nodes of this tree. How many different nodes could contain the maximum element n , if:

- i. The tree satisfies the binary search tree property? (For each non-leaf node, all of the nodes in its left subtree have smaller values than it, and all of the nodes in its right subtree have larger values than it.)

Circle one: **$\Theta(1)$** or $\Theta(\log n)$ or $\Theta(n)$

- ii. The tree satisfies the max-heap property? (For each non-leaf node, all of its children have smaller values than it.)

Circle one: **$\Theta(1)$** or $\Theta(\log n)$ or $\Theta(n)$

- iii. The tree satisfies the min-heap property? (For each non-leaf node, all of its children have larger values than it.)

Circle one: $\Theta(1)$ or $\Theta(\log n)$ or **$\Theta(n)$**

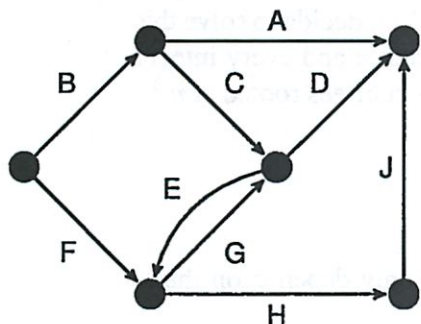
- iv. The tree satisfies the property that for each non-leaf node, its left child has a smaller value than it and its right child has a larger value than it?

Circle one: **$\Theta(1)$** or $\Theta(\log n)$ or **$\Theta(n)$**

6.006 Final Exam

(c) BFS and DFS trees [12 points]

Consider the graph below. Each part of this problem contains a tree, for which you must:

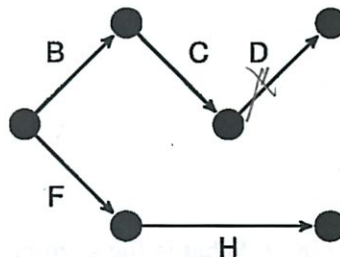


- Write "Valid BFS" if the tree is a valid BFS tree of the graph. Otherwise, write the label of a single edge (not in the tree) which can be deleted from the original graph so that the tree is a valid BFS tree on the graph (with the edge removed).
- Write "Valid DFS" if the tree is a valid DFS tree of the graph. Otherwise, write the label of a single edge (not in the tree) which can be deleted from the original graph so that the tree is a valid DFS tree on the graph (with the edge removed).

1. For the tree on the right,

i. Write "Valid BFS" or label of edge to delete:

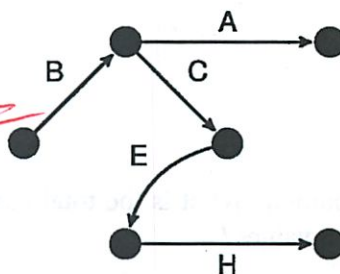
ii. Write "Valid DFS" or label of edge to delete:



2. For the tree on the right,

i. Write "Valid BFS" or label of edge to delete:

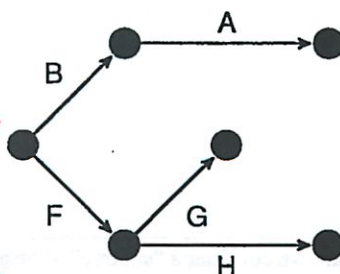
ii. Write "Valid DFS" or label of edge to delete:



3. For the tree on the right,

i. Write "Valid BFS" or label of edge to delete:

ii. Write "Valid DFS" or label of edge to delete:



TA pulls arbitrarily

TA can't kill F, leave H floating (in 1)

(A instead) ✓

~~Valid~~ E

~~Valid~~ - (D instead) ✓

Valid ✓

Valid - C next ✓

~~H (D instead)~~

Should go to E or A, but can't kill F

6/12

I like this problem

Problem 4. Dynamic Programming [20 points]

You do not need to show your work for this problem.

(a) Counting subtrees [10 points]

You are given a binary tree T with edges directed downwards from a root. Your goal is to find the number of subtrees of T with exactly k vertices.² You decide to solve this problem by dynamic programming, and define, for every vertex v and every integer $i = 1, \dots, k$, the subproblem $S[v, i]$ to be the number of size- i subtrees rooted at v .³

i. [1 points] If v is a leaf, what is $S[v, i]$ as a function of i ?

$$S(\text{Leaf}, i) = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{if } i \neq 1 \end{cases}$$

ii. [5 points] Write a recurrence relation for $S[v, i]$ which only depends on the S values for v 's children, u and w .

$$S[v, i] = S[u, i] + S[w, i]$$

left

iii. [3 points] What is the asymptotic running time to compute all the $S[v, i]$ values (for all v and all i) using the above recurrence with memoization?

$$O(n)$$

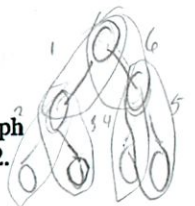
iv. [1 points] What is the total number of subtrees of T of size k , as a function of the S values?

$$S[T, k]$$

²In this problem, we consider a "subtree" to be any rooted subgraph of the tree such that all vertices in the subgraph are reachable from the root. For example, the depth-3 complete binary tree (7 total vertices) has 6 subtrees of size 2.

³By "rooted at v ", we mean that all vertices in the subtree are descendants of v .

Correction means v is in the subtree and all other vertices are descendants of v .



not trying to minimize cost \rightarrow just don't exceed it



(b) Shortest Paths on a Budget [10 points]

Suppose that G is a directed acyclic graph with a single source s and a single sink t . Each edge (i, j) of this graph is labeled with two values: a length $w_{(i,j)}$ and a cost $c_{(i,j)}$. The edge lengths are all non-negative real numbers and the edge costs are all non-negative integers. You have a budget C , and you cannot traverse any path with total cost greater than C . *like price*

Your goal is to find a path from s and t that meets this constraint and that has the shortest total length. You decide to solve this problem with dynamic programming.

You first find a topological sort of the vertices of G . Then, for each vertex v and for each integer $i = 0, 1, \dots, C$, you define the subproblem $S[v, i]$ to be the shortest length of any path from s to v with a total cost at most i . For ease of notation, you define $S[v, i]$ to be ∞ if $i < 0$. *Floyd Warshall*

- i. [1 points] If s is the source vertex, what is $S[s, i]$ for $i \geq 0$? *Correction: answer can depend on C.*

$$S[\text{source}, i] = 0 \text{ for all } i \geq 0$$

- ii. [5 points] Write a recurrence relation for $S[v, i]$ which depends only on S values for vertices before v in the topological order.

$$S[v, i] = \min_j [S[j, i - w_{(j,v)}]]$$

*most track cost sep
Las San on old exam*

- iii. [3 points] What is the asymptotic running time to compute all the S values for all v and all i using the above recurrence with memoization? Express your answer in terms of the number of vertices V and the number of edges E .

Only run once since DAG

Check each vertex once and all edges
 $O(V+E)$

- iv. [1 points] What is the shortest path from s to t with total cost at most C , as a function of the S values?

$S[t, C]$ and (un)recurrence

*What does this for edges have cost
- only traverse once*

*i is both current cost and an index
Say C_i for cost up to i*

Problem 5. Directed (Almost) Acyclic Graphs [15 points]

You are given a directed acyclic graph G with vertex set V and edge set E and a topological ordering of the vertices. Additionally, you are given a set K of k new edges which are inserted into this DAG to form a new graph G' , with edge set $E \cup K$. (The edges in K do not necessarily conform to the topological ordering, so they may create cycles.) Lastly, the edges in G' are weighted with arbitrary edge weights.

Give an $O(k \cdot (V + E + k))$ algorithm for finding the shortest path from a vertex s to a vertex t in G' . Be sure to argue your algorithm's correctness and analyze its running time.

Note: You will receive 3 points for a blank answer to this question. You will get more than 3 points for progress towards a correct solution, but any other text that is not crossed out will count against you.

So in the old topo sort we knew the shortest path in $O(V+E)$ since that was what it took to make a topo sort w/ DFS.

Now our new vertices k are screwing things up.

But we can check each of our k new nodes to see if they create a shorter path. This is $O((V+E+k) \log(V+k))$ for each of the k nodes. Thus $O(k \cdot (V+E+k))$

They may also create a cycle - so we must be checking for that - as we do in Dijkstra's.

?? This is correct because we check each of our k new nodes if they create a shorter path.

Checking is exploring out from the node in $O((V+E+k) \log(V+k))$ all dir. Dig-style

Problem 6. Passing an Evil Exam [25 points]

Shaunak has designed the grading system for a new algorithms course, 6.666. Your task is to design an algorithm which computes the optimal test-taking strategy for Shaunak's grading system.

In Shaunak's proposal, an exam has n problems. Each problem i has a point value, v_i , which is a positive number. All problems are optional. A student selects a subset S of the problems to attempt, and leaves all the other problems blank. If the student correctly answers **all** of the attempted problems, the student's score is $\sum_{i \in S} v_i$, the total point value of all attempted problems. However, if **any** of the answers are incorrect, the student's score on the **entire exam** is 0. *diabolical*

Suppose that for each problem i , you know the probability p_i of your answer being correct if you were to attempt that problem. The probabilities of successfully answering the attempted questions are independent: If you attempt a set S of problems, the probability of successfully solving all of those problems is $\prod_{i \in S} p_i$. Your goal is to determine a set of problems S to attempt to maximize your expected score. Thus, you want to find a set S which maximizes:

$$E[V] = \left(\prod_{i \in S} p_i \right) \cdot \left(\sum_{i \in S} v_i \right)$$

Only the ones you are very sure of!

Throughout this problem, you may assume that arithmetic operations (+, -, *, /) and comparisons on real numbers can be performed in constant time.

Joke:

*Seeing as there are usually few n on an exam
and p_i is very low - it shouldn't be*

Continue to the next page.

too bad to do the nice way in real life ;)

- (a) [10 points] Suppose that $v_i = 1$ for all i . Give an $O(n \log n)$ algorithm to find an optimal subset S of problems. Briefly argue your algorithm's correctness and analyze its running time.

Note that for full credit, you should be able to determine the actual subset S , not just its expected value.

Note: You will receive 2 points for a blank answer to this question. You will get more than 2 points for progress towards a correct solution, but any other text that is not crossed out will count against you.

Order the probabilities from highest (≈ 1) to lowest (≈ 0)
 Using your favorite $O(n \log n)$ comparison sort,
 - merge sort
 - randomized quicksort
 note track (via annotation which p_i belongs to which problem)

Then for each one, either include it if EV goes up
 or return EV without it (previous EV) if EV goes down

Adding a value can be done in $O(1)$ (track iteration value)

Since $p_1 \cdot p_2 \cdot (1+1)$

Store separately

multiply the p_i s by p

multiply by the # of problems you are at

Explain why greedy works.

This is $O(n \log n)$ since the sort.

It is correct since because we sorted, we are doing most likely to be correct problems 1st. We do these until the extra points no longer offset the overall utility and growing

that we will screw up

We can return which problems - since we kept track of those in sorting, just return the 1st however many of them

- (b) [15 points] Now suppose that each v_i is a integer between 1 and C . Give an $O(Cn^2)$ algorithm to find an optimal subset S of problems. Briefly argue your algorithm's correctness and analyze its running time.

Again, note that for full credit, you should be able to determine the actual subset S , not just its expected value.

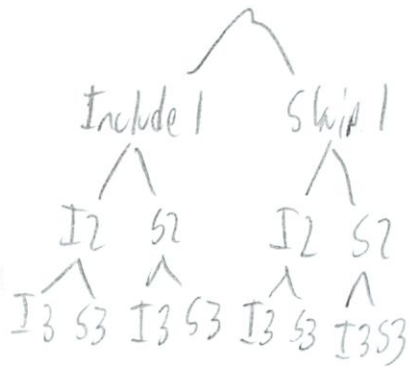
correction 3 pts
 Note: You will receive 2 points for a blank answer to this question. You will get more than 3 points for progress towards a correct solution, but any other text that is not crossed out will count against you.

Nieve $O(2^n)$

Try every thing in a tree

-2
 exponential.

We try every possible combo
 and then take the max



~~Start w/ max points~~

~~Go through each pts level v_i — since must multiply/add each term t_i
 try each — if to include $O(n)$ to try $O(n)$ max
 possible to include so $O(n^2)$~~

~~Again include if EV increases~~

Do not write on this page.

Final Exam

You will have 3 hours to complete this exam. No notes or other resources are allowed. Unless otherwise specified, full credit will only be given to a correct answer which is described clearly and concisely.

Do not discuss this exam with anyone who has not yet taken it.

Problem	Points	Grade	Initials
Name	2		
1	24		
2	20		
3	24		
4	20		
5	15		
6	25		
Total	130		

Name: [2 point] _____

R01	R02	R03	R04	R05	R06	R07
WF10	WF11	WF12	WF1	WF2	WF3	WF3
Shaunak	Shaunak	Alan	Jeff	Rafael	Henrique	Dragos

Problem 1. True/False [24 points]

Answer True or False. You do not need to explain your answer.

Correct answers are worth 2 points. Blanks and incorrect answers are worth 0 points.

- (a) The asymptotic solution to the recurrence $T(n) = 3T(n/3) + \Theta(n^2)$ is $T(n) = \Theta(n^2)$.

True – The Master Theorem implies this asymptotic bound.

- (b) Given a binary search tree, it is possible to build a balanced binary search tree containing the same elements in linear time.

True — You can do an in-order traversal to recover the entire sorted list in $O(n)$ time, then build the BST manually in $O(n)$ time: make the median the root, recursively build balanced BSTs that store the left and right halves, and then set the root's children.

- (c) For any node in an AVL tree, the size of the subtree rooted at the node's left child and the size of the subtree rooted at the node's right child differ by at most 1. (The size of a BST is the number of nodes in the tree.)

False — The heights of the subtrees are at most 1 apart, but the sizes may not be.

- (d) Suppose that we are using linear probing to maintain a hash table that already contains some elements. Under the SUHA assumption for our hash function, the next element is equally likely to end up in any of the empty slots of the table.¹

False — For example, it is twice as likely for x to end up in a cell preceded by an occupied cell as a cell preceded by an empty cell.

- (e) If you use chaining to resolve collisions in a hash table under the SUHA assumption, you can perform inserts, deletions, and lookups in $O(1 + \alpha)$ time in the average case, but not the worst case. (If there are n elements stored in a hash table of size m , then the load factor α is $\frac{n}{m}$.)

True

- (f) Performing the extract-min operation k times in a row on a min-heap can be done in time $O(k)$.

False – If this was possible, then it would be possible to comparison sort a list of size n in $O(n)$ time by building a heap and calling extract-min n times.

¹If we are using $h_i(x) = h(x) + i \bmod m$, we are assuming SUHA for the function $h(x)$.

- (g) Given any undirected graph, it is always possible to assign a direction to each edge so that the resulting graph is a directed acyclic graph (DAG).

True – Choose an arbitrary order for the vertices and direct all edges from left to right.

- (h) Suppose that in an unweighted, undirected graph G , an edge e is on a shortest path from a vertex s to a vertex t . If we perform BFS on G starting at s , then e must appear in the BFS tree.

False – If some other shortest path does not use e , then e might not be in the tree.

- (i) If you compute the n th Fibonacci number recursively using the Fibonacci recurrence, and you do not memoize any values, then the runtime of this computation will be exponential in n .

True – It will run in time proportional to the n th Fibonacci number, F_n , which is exponential.

- (j) There exists an algorithm that, given positive integers a and b , returns a^b in time polynomial in $(\log a + \log b)$.

False — a^b has $b \log a$ bits, so writing down the answer could take exponential time.

- (k) There exists an algorithm that, given positive integers a , b , and c , returns $a^b \pmod{c}$ in time polynomial in $(\log a + \log b + \log c)$.

True — Use repeated squaring and take the value mod c at intermediate steps.

- (l) Suppose that being given a black box algorithm for the problem L_1 would let us solve problem L_2 in polynomial time. Then, if L_2 is solvable in polynomial time, so is L_1 .

False — If L_1 is solvable in polynomial time, so is L_2 .

Problem 2. Algorithmic Techniques [20 points]

For each of the following problems, indicate which algorithmic technique would be **most** useful for solving it. If multiple answers satisfy the requirements of the problem, you should choose **the most time-efficient one possible**.

- (a) Given a string s of length n and a string p of length k , you want to determine if p is a substring of s in $O(n + k)$ expected time.

Circle one: Sorting or **Hashing** or Binary Search Trees

Explanation: Use a rolling hash to compare substrings of s to p .

- (b) Yelp maintains a database of information about restaurants: it assigns a quality rating and a cost to each restaurant. Users should be able to search the database for the highest rated restaurant they can dine at without exceeding a given cost. Yelp should also be able to add new restaurants and to delete old ones which go out of business.

Circle one: Sorting or Hashing or **Binary Search Trees**

Explanation: Use a BST keyed by costs in which each node is augmented with the highest quality restaurant in its subtree.

- (c) In a rock-paper-scissors tournament of n players, each player plays one game against each other player. A player's record is the triple (w, t, l) which records the number of games they won, tied, and lost, respectively.

One player's record is better than another's if the first player won more games than the second did, or if they won the same number of games and the first player tied more. We want to build a data structure, as efficiently as possible, which can answer queries "Who had the i th best record?" in $O(1)$ time.

Circle one: **Sorting** or Hashing or Binary Search Trees

Explanation: Use radix sort on the values. Each coordinate is bounded by n .

- (d) The TAs are compiling a collection of algorithms problems for a new algorithms class. Each of these problems has a (distinct) difficulty rating, which is a real number. When building a test, they want to be able to efficiently find the problem with difficulty closest to a given value d . In addition, they want to be able to add new problems to the collection and to delete problems when they are used on the test.

Circle one: Sorting or Hashing or **Binary Search Trees**

Explanation: Store the problems in an AVL tree keyed by difficulty. The problem with difficulty closest to d is either the predecessor and successor of d in the tree.

- (e) Alan has a large file that he wants to distribute to many other people over the Internet. He wants to send them a small amount of additional information that they can use to verify that the downloaded file is correct.

Circle one: Sorting or **Hashing** or Binary Search Trees

Explanation: Alan should send the hash of the file. The user computes the hash of the file he gets and checks that it is correct.

- (f) The dating website AlrightCupid wants to use a hash table to store the information about its users. Because of the architecture of their system, they cannot afford to ever resize the hash table, yet they want to be able to insert arbitrarily many users into the database.

Circle one: **Chaining** or Open addressing

Explanation: With chaining, it is possible to store n values in a hash table of size m even if $n > m$.

- (g) Dragos wants to find a cycle in a directed graph.

Circle one: BFS or **DFS** or Dijkstra's algorithm or Bellman-Ford

Explanation: BFS does not necessarily detect cycles in directed graphs. For example, consider a large cycle with all edges directed in both directions. Bellman-Ford can be used to find cycles, but DFS is faster.

- (h) Rafael wants to determine if there is a positive-weight cycle in an graph with arbitrary integers as edge weights.

Circle one: BFS or DFS or Dijkstra's algorithm or **Bellman-Ford**

Explanation: Negate the edge weights and use Bellman-Ford to find a negative-weight cycle.

- (i) Jeff wants to determine the set of vertices that can be reached in a graph by traversing at most k edges starting at a node s .

Circle one: **BFS** or DFS or Dijkstra's algorithm or Bellman-Ford

Explanation: Run BFS for k levels. This algorithm takes $O(|V| + |E|)$ time.

- (j) FaceLook stores a social network graph as a set of adjacency lists: for each user, it stores a list of that user's friends. Because friendship is mutual, this graph is undirected. The social network graph has diameter 6, and for k up to 6, there are roughly 100^k users at distance k from any given user. FaceLook wants to implement an algorithm to compute the shortest path between two users in this graph.

Circle one: **Bi-directional BFS** or A* search

Explanation: A regular BFS would expand about 100^6 nodes, while a bi-directional BFS would expand about 100^3 . A* is not useful on this graph, because there may be no meaningful notion of distance.

Problem 3. Short Answers [24 points]**(a) Peak finding in a $2 \times n$ array** [4 points]

A $2 \times n$ array A of integers has elements indexed by the notation $A[0][i]$ and $A[1][i]$ for $i = 0, \dots, n-1$. Two elements are adjacent if they differ by 1 in one coordinate and are equal in the other coordinate. For example, $A[0][5]$ is adjacent to $A[1][5]$ and $A[0][4]$, but these two elements are not adjacent to each other. A *peak* is an element with a value at least as large as all its adjacent elements' values.

Consider the following outlines of algorithms for finding a peak in a $2 \times n$ array.

- i. Use any $O(\log n)$ 1-D peak finding algorithm on the arrays B_0 and B_1 , where $B_0[i] = A[0][i]$ and $B_1[i] = A[1][i]$. (The B_i are the bottom and top halves of A .) Return the maximum of the two values found.

Circle: **Correct** or **Incorrect**

Explanation: If B_0 is $[5, 1, 3]$ and B_1 is $[4, 0, 8]$, then the 1-D algorithm might find the peak 3 in the first array and the peak 4 in the second. However, 4 is not a peak in A .

- ii. Use any $O(\log n)$ 1-D peak finding algorithm on the array of column maxes $B[i] = \max(A[0][i], A[1][i])$. Do not explicitly store B ; only compute values of B on demand (that is, only when they are needed).

Circle: **Correct** or **Incorrect**

Explanation: This algorithm finds a peak in $O(\log n)$ time, much like the $O(n \log n)$ algorithm for finding a peak in a 2-D array by lazy computation of maxima.

(b) Locating the max [8 points]

Suppose you are given a perfectly balanced binary tree. There are n nodes in this tree. Each non-leaf node has a left and right child, and all the leaves are at the same distance from the root. We want to store the elements $1, 2, \dots, n$ in the nodes of this tree. How many different nodes could contain the maximum element n , if:

- i. The tree satisfies the binary search tree property? (For each non-leaf node, all of the nodes in its left subtree have smaller values than it, and all of the nodes in its right subtree have larger values than it.)

Solution: $O(1)$ – The max must appear in the right-most leaf, because you can find it by starting at the root and repeatedly moving to the right.

- ii. The tree satisfies the max-heap property? (For each non-leaf node, all of its children have smaller values than it.)

Solution: $O(1)$ – The max must be at the root of the tree.

- iii. The tree satisfies the min-heap property? (For each non-leaf node, all of its children have larger values than it.)

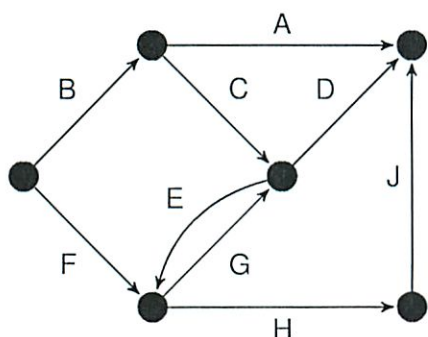
Solution: $O(n)$. Any of the leaves could be the max. About $\frac{n}{2}$ of the vertices are leaves.

- iv. The tree satisfies the property that for each non-leaf node, its left child has a smaller value than it and its right child has a larger value than it?

Solution: $O(n)$. Any leaf that is the right child of its parent could be a leaf.

(c) **BFS and DFS trees** [12 points]

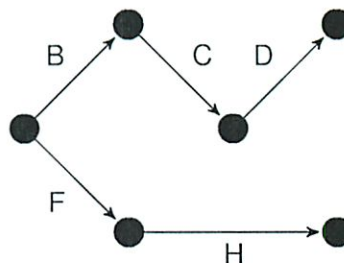
Consider the graph below. Each part of this problem contains a tree, for which you must:



- Write "Valid BFS" if the tree is a valid BFS tree of the graph. Otherwise, write the label of a single edge (not in the tree) which can be deleted **from the original graph** so that the tree is a valid BFS tree on the graph (with the edge removed).
- Write "Valid DFS" if the tree is a valid DFS tree of the graph. Otherwise, write the label of a single edge (not in the tree) which can be deleted **from the original graph** so that the tree is a valid DFS tree on the graph (with the edge removed).

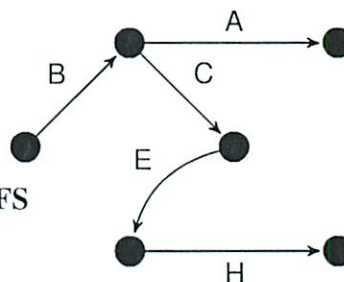
1. For the tree on the right,

- Write "Valid BFS" or label of edge to delete: **A**
- Write "Valid DFS" or label of edge to delete: **E**



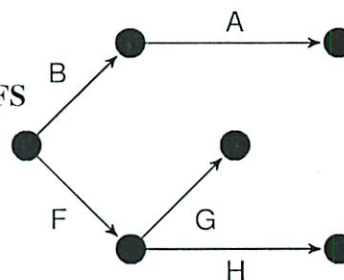
2. For the tree on the right,

- Write "Valid BFS" or label of edge to delete: **F**
- Write "Valid DFS" or label of edge to delete: **Valid DFS**



3. For the tree on the right,

- Write "Valid BFS" or label of edge to delete: **Valid BFS**
- Write "Valid DFS" or label of edge to delete: **C**



Problem 4. Dynamic Programming [20 points]

You do not need to show your work for this problem.

(a) Counting subtrees [10 points]

You are given a **binary** tree T with edges directed downwards from a root. Your goal is to find the number of subtrees of T with exactly k vertices.² You decide to solve this problem by dynamic programming, and define, for every vertex v and every integer $i = 1, \dots, k$, the subproblem $S[v, i]$ to be the number of size- i subtrees rooted at v .³

- i. [1 points] If v is a leaf, what is $S[v, i]$ as a function of i ?

Answer: 1 if $i = 1$. Otherwise 0.

- ii. [5 points] Write a recurrence relation for $S[v, i]$ which only depends on the S values for v 's children, u and w .

Answer: $S[v, i] = \sum_{j=1}^{i-2} S[u, j] \cdot S[w, i-1-j]$. To build a subtree of size i rooted at v , we have to choose a size j of the part of the subtree that lies under u . Then $i-1-j$ nodes lie under w , and v is the remaining node. (This recurrence is true for $i \geq 1$. Otherwise, $S[v, i] = 0$.)

- iii. [3 points] What is the asymptotic running time to compute **all** the $S[v, i]$ values (for all v and all i) using the above recurrence with memoization?

Answer: $O(nk^2)$. (There are $O(nk)$ subproblems. It takes $O(k)$ time to solve each one.)

- iv. [1 points] What is the total number of subtrees of T of size k , as a function of the S values?

Answer: $\sum_v S[v, k]$

²In this problem, we consider a "subtree" to be any rooted subgraph of the tree such that all vertices in the subgraph are reachable from the root. For example, the depth-3 complete binary tree (7 total vertices) has 6 subtrees of size 2.

³By "rooted at v ", we mean that the subtree contains v and that all the vertices in the subtree are descendants of v .

(b) Shortest Paths on a Budget [10 points]

Suppose that G is a directed acyclic graph with a single source s and a single sink t . Each edge (i, j) of this graph is labeled with two values: a length $w_{(i,j)}$ and a cost $c_{(i,j)}$. The edge lengths are all non-negative real numbers and the edge costs are all non-negative integers. You have a budget C , and you cannot traverse any path with total cost greater than C .

Your goal is to find a path from s and t that meets this constraint and that has the shortest total length. You decide to solve this problem with dynamic programming.

You first find a topological sort of the vertices of G . Then, for each vertex v and for each integer $i = 0, 1, \dots, C$, you define the subproblem $S[v, i]$ to be the shortest length of any path from s to v with a total cost at most i . For ease of notation, you define $S[u, i]$ to be ∞ if $i < 0$.

- i. [1 points] If s is the source vertex, what is $S[s, i]$ for $i \geq 0$?

Answer: $S[s, i] = 0$ for all $i \geq 0$.

- ii. [5 points] Write a recurrence relation for $S[v, i]$ which depends only on S values for vertices before v in the topological order.

Answer: $S[v, i] = \min_u S[u, i - c_{u,v}] + e_{(u,v)}$, where u ranges over all vertices such that u, v is an edge.

- iii. [3 points] What is the asymptotic running time to compute all the S values for all v and all i using the above recurrence with memoization? Express your answer in terms of the number of vertices V , the number of edges E , and C .

Answer: $O(C(|V| + |E|))$. There are $C|V|$ subproblems. Each edge (u, v) is relaxed in C different subproblems, namely, the subproblems for v .

- iv. [1 points] What is the shortest path from s to t with total cost at most C , as a function of the S values?

Answer: $S[t, C]$

Problem 5. Directed (Almost) Acyclic Graphs [15 points]

You are given a directed acyclic graph G with vertex set V and edge set E and a topological ordering of the vertices. Additionally, you are given a set K of k new edges which are inserted into this DAG to form a new graph G' , with edge set $E \cup K$. (The edges in K do *not* necessarily conform to the topological ordering, so they may create cycles.) Lastly, the edges in G' are weighted with arbitrary edge weights.

Give an $O(k \cdot (V + E + k))$ algorithm for finding the shortest path from a vertex s to a vertex t in G' . Be sure to argue your algorithm's correctness and analyze its running time.

Note: You will receive 3 points for a blank answer to this question. You will get more than 3 points for progress towards a correct solution, but any other text that is not crossed out will count against you.

Answer: Order the edges in E by the order in which their first vertex appears in the topological sort. Append the edges in K at the end of this list. Then run Bellman-Ford on G' for $k + 1$ iterations, relaxing edges in this order. This algorithm takes $O(k(|V| + |E| + k))$ time. We need to prove that it is correct. This proof will be very similar to the normal proof of correctness for Bellman-Ford.

Suppose that there is a shortest path from s to t and that $(s, v_1), (v_1, v_2), \dots, (v_{m-1}, v_m), (v_m, t)$ are the edges on this path. We can eliminate any cycles from this path without increasing its length, so we can assume that each vertex appears at most once.

At most k of the edges on this shortest path are edges from K . If (v_i, v_{i+1}) is the first edge on the path from K , then after the first Bellman-Ford iteration, the distance from s to v_1, v_2, \dots, v_{i+1} are all correct - the edges from E are relaxed in order, and then the edge (v_i, v_{i+1}) is relaxed afterwards.

If (v_j, v_{j+1}) is the second edge from K on this path, then after the second Bellman-Ford iteration, the distance from s to $v_{i+2}, v_{i+3}, \dots, v_{j+1}$ are all correct, by similar reasoning. Continuing in this way, we find that after k Bellman-Ford iterations, the distances up to the last K edge on the path will be correct. After one more iteration, the distance to t will be correct.

Many students stated algorithms which were close to this correct algorithm, but did not prove that $O(k)$ iterations of Bellman-Ford were enough. In addition, there were some solutions which required more than $k + 1$ iterations - for example some required $2k + 1$. It is important to do an analysis similar to the above to be sure that your answer actually finds shortest paths.

Problem 6. Passing an Evil Exam [25 points]

Shaunak has designed the grading system for a new algorithms course, 6.666. Your task is to design an algorithm which computes the optimal test-taking strategy for Shaunak's grading system.

In Shaunak's proposal, an exam has n problems. Each problem i has a point value, v_i , which is a positive number. All problems are optional. A student selects a subset S of the problems to attempt, and leaves all the other problems blank. If the student correctly answers **all** of the attempted problems, the student's score is $\sum_{i \in S} v_i$, the total point value of all attempted problems. However, if **any** of the answers are incorrect, the student's score on the **entire exam** is 0.

Suppose that for each problem i , you know the probability p_i of your answer being correct if you were to attempt that problem. The probabilities of successfully answering the attempted questions are independent: If you attempt a set S of problems, the probability of successfully solving all of those problems is $\prod_{i \in S} p_i$. Your goal is to determine a set of problems S to attempt to maximize your expected score. Thus, you want to find a set S which maximizes:

$$\left(\prod_{i \in S} p_i \right) \cdot \left(\sum_{i \in S} v_i \right)$$

Throughout this problem, you may assume that arithmetic operations (+, -, *, and /) and comparisons on real numbers can be performed in constant time.

Continue to the next page.

- (a) [10 points] Suppose that $v_i = 1$ for all i . Give an $O(n \log n)$ algorithm to find an optimal subset S of problems. Briefly argue your algorithm's correctness and analyze its running time.

Note that for full credit, you should be able to determine the actual subset S , not just its expected value.

Note: You will receive 2 points for a blank answer to this question. You will get more than 2 points for progress towards a correct solution, but any other text that is not crossed out will count against you.

Answer: We can sort the probabilities p_i in decreasing order using mergesort in $O(n \log n)$ time. We store each probability in a tuple with the index of the problem that it came from; later, this will allow us to recover the set S . For the rest of this problem, we assume that $p_1 \geq p_2 \geq \dots p_n$.

The optimal subset S must be a set of the form $S_k = \{1, 2, \dots, k\}$ for some k , because the values of the problems are all the same, so it is always better to choose problems that you are more likely to solve. The expected value of doing the problems in S_k is $k \prod_i = 1^k p_i$.

We can compute an array A such that $A[k] = \prod_i = 1^k p_i$ in linear time, computing each value in $O(1)$ time using the previous one. Then we can find the k that maximizes $kA[k]$, and we can return the subset of problems S_k . Sorting the probabilities takes $O(n \log n)$ time, and computing these probabilities takes $O(n)$ time, so the total runtime is $O(n \log n)$.

Many students claimed that $kA[k]$ increased as long as k was less than the optimal value and then strictly decreased afterwards. This claim is true, but proving it is more subtle than one might expect. In addition, the greedy algorithm (sort the probabilities, then decide whether or not to include each i in the current subset) also works, but also requires proof.

As a final note, if you were to solve this problem in a real application, you would work with the sums of the logs of the probabilities of the problems you chose, instead of the actual product. This approach avoids errors from finite floating point accuracy.

- (b) [15 points] Now suppose that each v_i is an integer between 1 and C . Give an $O(Cn^2)$ algorithm to find an optimal subset S of problems. Briefly argue your algorithm's correctness and analyze its running time.

Again, note that for full credit, you should be able to determine the actual subset S , not just its expected value.

Note: You will receive 3 points for a blank answer to this question. You will get more than 3 points for progress towards a correct solution, but any other text that is not crossed out will count against you.

Answer: We will use dynamic programming. Let $dp[i, v]$ be the maximal probability of a subset of the first i problems whose total value ($\sum_S v_i$) is v . The total value of any subset of problems is at most Cn . Since there are n problems, the total number of subproblems is $O(Cn^2)$.

We claim that we can solve each subproblem in constant time. As a base case, note that $dp[i, 0]$ is 1 for all i – we can simply take an empty set of problems. For $v > 0$, we have

$$dp[i, v] = \max(dp[i-1, v], p_i \cdot dp[i-1, v-v_i])$$

because the optimal subset either includes problem i , or it is an optimal subset from the first $i-1$ problems. Therefore, we can solve all these subproblems in $O(Cn^2)$ time.

Our final answer is the maximum of $v \cdot dp[n, v]$ over all Cn values for v . We can recover the set used to construct this answer by keeping parent pointers at each step, recording whether problem i was used to solve $dp[i, v]$ or not.

The subproblem that we used was chosen very carefully. There are many choices that did not work. For example, some students tried to use the subproblem $S[i]$ which is the maximum expected value of any subset of the first i problems, and claimed that $S[i]$ is either $S[i-1]$ or $S[i-1] \cup \{i\}$. However, it is not true that the optimal subset of the first i problems is built on earlier optimal subsets. This algorithm is basically a greedy approach; it does not produce the correct answer.

Do not write on this page.

1.

2.  [Activity](#)3.  [Homework](#)1.  [PS6](#)2.  [Quiz 2](#)3.  [PS5](#)4.  [PS4](#)5.  [PS3](#)6.  [Quiz1](#)7.  [PS2](#)8.  [PS1](#)4.  [Grades](#)5.  [Michael Plasmeier ▼](#)1. [Profile](#)2. [Change Password](#)3. [Sign out](#)

Grades

PS6

1. Total 114.0 / 130
2. Problem 4 30.0 / 40
3. Problem 1a1 2.0 / 2
4. Problem 1a2 2.0 / 2
5. Problem 1b1 2.0 / 2
6. Problem 1b2 2.0 / 2
7. Problem 1c1 2.0 / 2
8. Problem 1c2 2.0 / 2
9. Problem 1d1 0.0 / 2
10. Problem 1d2 0.0 / 2
11. Problem 1e1 2.0 / 2
12. Problem 1e2 0.0 / 2
13. Problem 2 30.0 / 30
14. Problem 3 40.0 / 40

Quiz 2

1. Total 72.0 / 100
2. Problem 0/1 22.0 / 25
3. Problem 2 15.0 / 18
4. Problem 3 10.0 / 12

5. Problem 4 11.0 / 25
6. Problem 5 14.0 / 20

PS5

1. Total 65.0 / 110
2. Problem 1a 5.0 / 5
3. Problem 1b 5.0 / 5
4. Problem 1c 5.0 / 5
5. Problem 2a 5.0 / 5
6. Problem 2b 5.0 / 5
7. Problem 2c 5.0 / 5
8. Problem 2d 0.0 / 5
9. Problem 2e 5.0 / 5
10. Problem 3 10.0 / 40
11. Problem 4 20.0 / 30

PS4

1. Total 106.0 / 120
2. Problem 1a 3.0 / 3
3. Problem 1b 3.0 / 3
4. Problem 1c 4.0 / 4
5. Problem 2 13.0 / 20
6. Problem 3 16.0 / 20
7. Problem 4a 2.0 / 2
8. Problem 4b 2.0 / 2
9. Problem 4c 2.0 / 2
10. Problem 4d 2.0 / 2
11. Problem 4e 2.0 / 2
12. Problem 5a 3.0 / 3
13. Problem 5b 3.0 / 3
14. Problem 5c 3.0 / 3
15. Problem 5d 0.0 / 3
16. Problem 5e 3.0 / 3
17. Problem 5f 3.0 / 3
18. Problem 5g 3.0 / 3
19. Problem 5h 3.0 / 3
20. Problem 5i 3.0 / 3
21. Problem 5j 3.0 / 3
22. Problem 6 30.0 / 30

PS3

1. Total 114.0 / 130
2. Problem 1a1 3.0 / 3
3. Problem 1a2 2.0 / 2
4. Problem 1b1 3.0 / 3
5. Problem 1b2 2.0 / 2

6. Problem 1c1 3.0 / 3
7. Problem 1c2 2.0 / 2
8. Problem 1d1 3.0 / 3
9. Problem 1d2 0.0 / 2
10. Problem 2a 5.0 / 5
11. Problem 2b 5.0 / 5
12. Problem 2c 10.0 / 10
13. Problem 2d 10.0 / 10
14. Problem 3 6.0 / 20
15. Problem 4 60.0 / 60

Quiz1

1. Total 50.0 / 100
2. Problem 0/1 13.0 / 15
3. Problem 2 3.0 / 10
4. Problem 3 11.0 / 20
5. Problem 4 14.0 / 15
6. Problem 5 6.0 / 10
7. Problem 6 3.0 / 30

PS2

1. Total 109.74 / 125
2. Problem 1a 5.0 / 5
3. Problem 1b 5.0 / 5
4. Problem 1c 5.0 / 5
5. Problem 1d 0.0 / 5
6. Problem 1e 5.0 / 5
7. Problem 1f 5.0 / 5
8. Problem 1g 5.0 / 5
9. Problem 2a 5.0 / 5
10. Problem 2b 5.0 / 5
11. Problem 2c 2.0 / 10
12. Problem 3 18.0 / 20
13. Problem 4 49.74 / 50

PS1

1. Total 65.0 / 75
2. Problem 1a 5.0 / 5
3. Problem 1b 5.0 / 5
4. Problem 1c 5.0 / 5
5. Problem 2a 5.0 / 5
6. Problem 2b 5.0 / 5
7. Problem 3a 5.0 / 5
8. Problem 3b 5.0 / 5
9. Problem 3c 0.0 / 10
10. Problem 4 30.0 / 30

- [6.006 TA Mailing List](#) [MIT Academic Calendar](#) [MIT EECS Department](#) [MIT Course Catalog](#)

 © Copyright 2008-2011 [Massachusetts Institute of Technology](#)

Designed by [Victor Costan](#)

note

40 views

Final Exam Statistics

Grades are up. Here are the stats:

STATISTICS (220 tests)

Mean 85.52

Median 86

Stdev 14.46

Max 124

HISTOGRAM

0 - 49	0
50 - 59	6
60 - 69	24
70 - 79	43
80 - 89	62
90 - 99	51
100 - 109	18
110 - 119	12
120 - 129	4

← Super such

✓ I was thinking more here

Final 30
Quiz 20 each
P-set 30 all

Quiz 2 stats: @572

Quiz 1 stats: @268

Pset averages:

Pset 1: 92.42% (69.31 / 75)

Pset 2: 88.85% (111.06 / 125)

Pset 3: 90.54% (117.71 / 130)

Pset 4: 88.18% (105.81 / 120)

Pset 5: 84.75% (93.22 / 110)

Pset 6: 82.53% (107.29 / 130)

essentially, avg on p-sets (except 1)
and other quizzes

#final #statistics #pin

follow 7 like 0

21 minutes ago by Shaunak Kishore 1 edit

followup discussions, for lingering questions and comments

Resolved Unresolved



Anonymous (18 minutes ago) - Can we have a rough gauge of what it takes for an A (in the class), since all these grades are up?



Jeff Wu (Instructor) (10 seconds ago) - Your grade will be based on your weighted score: (psets weighted 5% each, quizzes 20%, final 30%)

I can't tell you any cutoffs, but your grade will be on websis soon.

Write a reply...

Resolved Unresolved



Anonymous (9 minutes ago) - Is there any way we can go take a look at our test tomorrow (Friday) or sometime this weekend?



Jeff Wu (Instructor) (2 minutes ago) - Send an email to Be Blackburn (be@csail.mit.edu), and she'll have it on the 6th floor of the G tower, Stata.

Write a reply...

Michael E Plasmeier

From: kshaunak@gmail.com on behalf of Shaunak Kishore <skishore@MIT.EDU>
Sent: Tuesday, May 29, 2012 3:12 PM
To: Michael E Plasmeier
Subject: Re: Final Exam Grading

1. In 5, you don't explain what you do when you add each one. There's no indication here that you're doing iterations of Bellman-Ford - in fact, it looks like you're just relaxing the edges out of those vertices once as in Dijkstra.

2. You received only half-credit on 6a because you do a greedy solution ("include [each problem] if EV goes up or return the set without it if EV goes down") without proving that it is correct. In particular, it's not trivial to prove that the EVs are increasing up to a point and then strictly decreasing afterwards. See solution for an algorithm with a clear proof of correctness.

3. Naive solutions are slow polynomial time solutions, not exponential solutions. We consistently graded exponential solutions at less than the blank credit.

Shaunak

On Tue, May 29, 2012 at 11:09 AM, Michael E Plasmeier <theplaz@mit.edu> wrote:



Shaunak,

I was looking over my exam (which I scanned and attached before making any markings on it) and I had a few questions:

1. I think I was closer to #5 than 1/15 points. I added all the k vertices to the topo sort and processed each one – I did say Dijkstra instead of Bellman Ford. Could you take another look at that?
2. In 6a, why did I only receive half credit? I believe I had the correct solution. Did I not explain it well?
3. In 6b, I clearly designated what I had as a $O(2^n)$ naive solution. I know that it was not exponential! I didn't think you were going to take points off for that since it was clearly designated as naïve. In the beginning of the year, you or someone had said to always write a naïve solution if you don't know where to start. I can accept that you won't give me any points, but I really don't think you should take off here under the "guessing penalty" it clearly was not an attempt at an exponential solution.

Thanks! -Michael

1.

2.  [Activity](#)3.  [Homework](#)1.  [Final Exam](#)2.  [PS6](#)3.  [Quiz 2](#)4.  [PS5](#)5.  [PS4](#)6.  [PS3](#)7.  [Quiz1](#)8.  [PS2](#)9.  [PS1](#)4.  [Grades](#)5.  [Michael Plasmeier ▼](#)1. [Profile](#)2. [Change Password](#)3. [Sign out](#)

Grades

Final Exam

1. Total 56.0 / 130
2. Problem 0/1 10.0 / 26
3. Problem 2 20.0 / 20
4. Problem 3 12.0 / 24
5. Problem 4 7.0 / 20
6. Problem 5 1.0 / 15
7. Problem 6 6.0 / 25

PS6

1. Total 114.0 / 130
2. Problem 4 30.0 / 40
3. Problem 1a1 2.0 / 2
4. Problem 1a2 2.0 / 2
5. Problem 1b1 2.0 / 2
6. Problem 1b2 2.0 / 2
7. Problem 1c1 2.0 / 2
8. Problem 1c2 2.0 / 2
9. Problem 1d1 0.0 / 2
10. Problem 1d2 0.0 / 2

11. Problem 1e1 2.0 / 2
12. Problem 1e2 0.0 / 2
13. Problem 2 30.0 / 30
14. Problem 3 40.0 / 40

Quiz 2

1. Total 72.0 / 100
2. Problem 0/1 22.0 / 25
3. Problem 2 15.0 / 18
4. Problem 3 10.0 / 12
5. Problem 4 11.0 / 25
6. Problem 5 14.0 / 20

PS5

1. Total 65.0 / 110
2. Problem 1a 5.0 / 5
3. Problem 1b 5.0 / 5
4. Problem 1c 5.0 / 5
5. Problem 2a 5.0 / 5
6. Problem 2b 5.0 / 5
7. Problem 2c 5.0 / 5
8. Problem 2d 0.0 / 5
9. Problem 2e 5.0 / 5
10. Problem 3 10.0 / 40
11. Problem 4 20.0 / 30

PS4

1. Total 106.0 / 120
2. Problem 1a 3.0 / 3
3. Problem 1b 3.0 / 3
4. Problem 1c 4.0 / 4
5. Problem 2 13.0 / 20
6. Problem 3 16.0 / 20
7. Problem 4a 2.0 / 2
8. Problem 4b 2.0 / 2
9. Problem 4c 2.0 / 2
10. Problem 4d 2.0 / 2
11. Problem 4e 2.0 / 2
12. Problem 5a 3.0 / 3
13. Problem 5b 3.0 / 3
14. Problem 5c 3.0 / 3
15. Problem 5d 0.0 / 3
16. Problem 5e 3.0 / 3
17. Problem 5f 3.0 / 3
18. Problem 5g 3.0 / 3
19. Problem 5h 3.0 / 3

- 20. Problem 5i 3.0 / 3
- 21. Problem 5j 3.0 / 3
- 22. Problem 6 30.0 / 30

PS3

- 1. Total 114.0 / 130
- 2. Problem 1a1 3.0 / 3
- 3. Problem 1a2 2.0 / 2
- 4. Problem 1b1 3.0 / 3
- 5. Problem 1b2 2.0 / 2
- 6. Problem 1c1 3.0 / 3
- 7. Problem 1c2 2.0 / 2
- 8. Problem 1d1 3.0 / 3
- 9. Problem 1d2 0.0 / 2
- 10. Problem 2a 5.0 / 5
- 11. Problem 2b 5.0 / 5
- 12. Problem 2c 10.0 / 10
- 13. Problem 2d 10.0 / 10
- 14. Problem 3 6.0 / 20
- 15. Problem 4 60.0 / 60

Quiz1

- 1. Total 50.0 / 100
- 2. Problem 0/1 13.0 / 15
- 3. Problem 2 3.0 / 10
- 4. Problem 3 11.0 / 20
- 5. Problem 4 14.0 / 15
- 6. Problem 5 6.0 / 10
- 7. Problem 6 3.0 / 30

PS2

- 1. Total 109.74 / 125
- 2. Problem 1a 5.0 / 5
- 3. Problem 1b 5.0 / 5
- 4. Problem 1c 5.0 / 5
- 5. Problem 1d 0.0 / 5
- 6. Problem 1e 5.0 / 5
- 7. Problem 1f 5.0 / 5
- 8. Problem 1g 5.0 / 5
- 9. Problem 2a 5.0 / 5
- 10. Problem 2b 5.0 / 5
- 11. Problem 2c 2.0 / 10
- 12. Problem 3 18.0 / 20
- 13. Problem 4 49.74 / 50

PS1

1. Total 65.0 / 75
2. Problem 1a 5.0 / 5
3. Problem 1b 5.0 / 5
4. Problem 1c 5.0 / 5
5. Problem 2a 5.0 / 5
6. Problem 2b 5.0 / 5
7. Problem 3a 5.0 / 5
8. Problem 3b 5.0 / 5
9. Problem 3c 0.0 / 10
10. Problem 4 30.0 / 30

- [6.006 TA Mailing List](#) [MIT Academic Calendar](#) [MIT EECS Department](#) [MIT Course Catalog](#)



© Copyright 2008-2011 [Massachusetts Institute of Technology](#)

Designed by [Victor Costan](#)