CS180 Exam 2

[redacted]

TOTAL POINTS

25 / 26

QUESTION 1

Problem 1 10 pts

1.1 Shortest path **1 / 1**

✓ **- 0 pts correct answer and correct counter example**

1.2 MST: Adding weight **0.75 / 1**

✓ **- 0.25 pts Correct answer but saying minimum edge does not change relatively but not mentioning the exact algorithm and why the algorithm flow does not change.**

1.3 MST: Heaviest edge **1 / 1**

✓ **- 0 pts Correct answer and correct counter example**

1.4 Prim update **1 / 1**

✓ **- 0 pts Correct**

1.5 Dynamic programming: recursion vs memoization **1 / 1**

✓ **- 0 pts Correct**

1.6 DFS Tree **1.25 / 2**

✓ **- 0.75 pts Wrong in one or two places and wrong order**

1.7 Knapsack broken item **1 / 1**

✓ **- 0 pts Correct. You can compute the new value in O(1) time.**

1.8 Cycle property **2 / 2**

✓ **- 0 pts Correct**

QUESTION 2

Dijkstra 4 pts

2.1 Algorithm **2 / 2**

✓ **- 0 pts Correct**

2.2 Dijkstra vs Prim **2 / 2**

✓ **- 0 pts Correct**

QUESTION 3

Art gallery guards 4 pts

3.1 Algorithm **3 / 3**

✓ **- 0 pts Correct**

3.2 Proof of correctness **1 / 1**

✓ **- 0 pts Correct**

QUESTION 4

4 Counting paths **4 / 4**

✓ **- 0 pts correct algorithm with run-time analysis**

QUESTION 5

5 Weighted interval knapsack **4 / 4**

✓ **- 0 pts Correct**

Exam 2. May 16, 2018

CS180: Algorithms and Complexity
Spring 2018

Guidelines:

- The exam is closed book and closed notes. Do not open the exam until instructed to do so. You have **one hour and fifty minutes for the exam.**
- Write your solutions clearly and when asked to do so, provide complete proofs. You may use results and algorithms from class without proofs or details as long as you specifically state what you are using.
- I recommend taking a quick look at all the questions first and then deciding what order to tackle to them in. Even if you don't solve the problems fully, attempts that show some understanding of the questions and relevant topics will get reasonable partial credit. In particular, even for true or false questions asking for justification, correct answers will get reasonable partial credit.
- You can use extra sheets for scratch work, but you can **only use the white space** (it should be more than enough) on the exam sheets for your final solutions.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly and any cheating reported with the score automatically becoming zero.
- Write clearly and legibly. All the best!

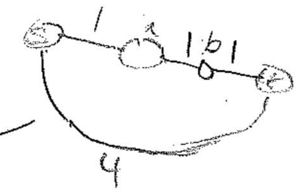
Problem	Points	Maximum
1		10
2		4
3		4
4		4
5		4
Total		26

Name	[redacted]
UID	
Section	

1 Problem

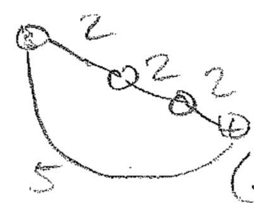
1. True or False: Let P be a shortest path from some vertex s to some other vertex t in a weighted undirected graph. If the weight of each edge in the graph is increased by one, P will still be a shortest path from s to t (with the new weights). If true, provide an explanation of why this is true and if false, provide a counterexample. [1 point]

False. before:



(s, a, b, t) : shortest path

after



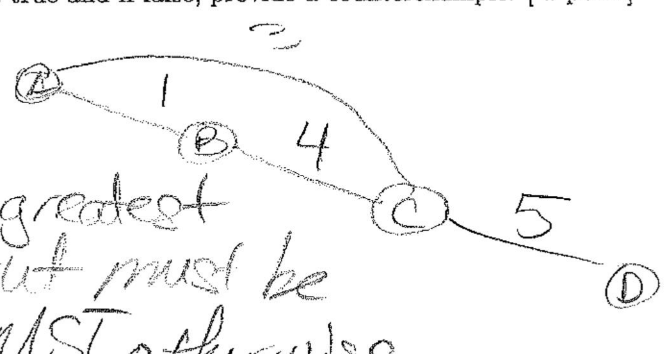
(s, t): shortest path

2. True or False: Let T be a MST in G. If the weights of all edges in the graph are changed by adding 1 to the weights, then T is still a MST in the graph (with the new weights). If true, provide an explanation of why this is true and if false, provide a counterexample. [1 point]

True, MSTs are constructed by comparing edges. If an edge e satisfied the cut property before & hence should be in MST, It should be in MST after b/c relative differences don't change.

3. True or False: If a weighted undirected graph G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree. If true, provide an explanation of why this is true and if false, provide a counterexample. [1 point]

False



CD has greatest weight but must be part of MST otherwise MST wouldn't be connected.

4. True or False: When running Prim's algorithm, after updating the set S, we only need to recompute the attachment costs for the neighbors of the newly added vertex. No justification necessary. [1 point]

True

5. True or False: For a dynamic programming algorithm, computing all values in a bottom-up fashion (using for/while loops) is asymptotically faster than using recursion and memoization. No justification necessary. [1 point]

False

6. Let $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6, 7\}$ and

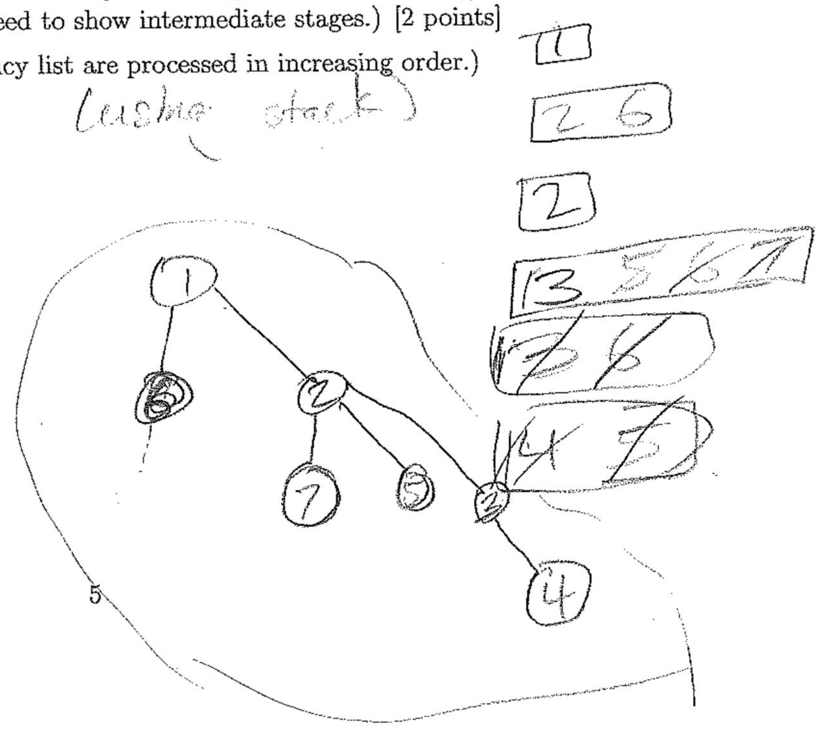
$$E = \{\{1,2\}, \{1,6\}, \{2,3\}, \{2,5\}, \{2,6\}, \{2,7\}, \{3,4\}, \{3,5\}, \{5,6\}\}.$$

Suppose that G was given to you in adjacency list representation where the elements in the adjacency list are ordered in increasing order. For example, the adjacency list of vertex 2 would be $[1, 3, 5, 6]$. Draw the DFS tree that you would get when doing DFS starting from 1. (Just the final tree is enough. No need to show intermediate stages.) [2 points]

(Recall that elements of the adjacency list are processed in increasing order.)

adjacency list

1	2, 6
2	1, 3, 5, 6, 7
3	4, 5
5	6



7. Consider an instance of the knapsack problem with n items having values and weights $(v_1, w_1), \ldots, (v_n, w_n)$ and knapsack having total weight capacity W. Suppose you have computed the values $OPT(j, w)$ for $1 \leq j \leq n$ and $1 \leq w \leq W$. However, in your excitement you broke the $(n-2)$'th item and it has no value anymore. How fast can you compute the new best value? No justification necessary. [1 point]

~~$\Theta(W)$ ← just have to use existing matrix, but remove n-2nd item and re-calculate the last 2 rows~~

you're redoing most of matrix

O(1): just have to consider whether addition of n-1st and nth items improve on values calculated for n-3rd item

4 options to consider

8. Suppose you have a weighted undirected graph $G = (V, E)$ where all the weights are distinct. Prove that if an edge e is part of a cycle C and has weight more than every other edge in the cycle, then e cannot be part of the minimum spanning tree in G. [2 points]

[Hint: Assume that the statement is false for the sake of contradiction and let T be a MST that contains the edge e. Arrive at a contradiction by a swapping argument as we did in class for proving the cut property.]

Consider an MST T containing e (u,v)

Disconnect T by removing e

Consider the cut represented by the connected component of u.

Along some edge q (a,b) where at least one of a, b ≠ u, v) ~~an~~ edge in the cycle containing e leaves the cut. This edge q may replace e to form T'. B/c e had highest weight (where all weights distinct), $|T'| = |T| - e + q < |T|$.

weights

T' is connected because it provides an edge connecting the two connected components associated with u and v. No cycle is created b/c the only possible cycle in T+q was removed when e was removed. Thus T' is a spanning tree of smaller weight than T. This forms a contradiction with the assertion that T is an MST. Thus, the cycle property has been proven.

2 Problem

1. Write down Dijkstra's algorithm for computing a shortest path between two vertices s and t in a weighted undirected graph $G = (V, E)$ given in adjacency-list representation. [2 points]

2. True or False: Given a weighted undirected graph $G = (V, E)$ with distinct weights and a vertex $s \in V$, the shortest-path tree computed by Dijkstra's algorithm starting from s and the tree computed by Prim's algorithm starting from s are the same. If true, provide an explanation of why this is true and if false, provide a counterexample. [2 points]

1) $S = s$ parent[i] = ∅ for all i in V
$d(s) = 0$

while $S \neq V$:
$a, b =$ u,v that minimizes $d(u) + l(u,v)$
$u \in S$
$v \notin S$

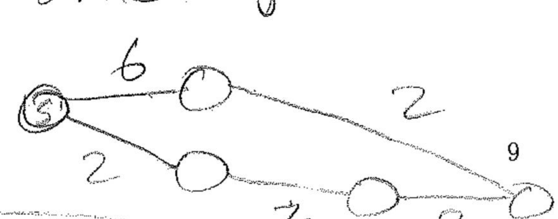
parent[b] = a
add b to S
$d(b) = d(a) + l(a,b)$

no assumption made about connectedness, so if there is no a,b, one should return

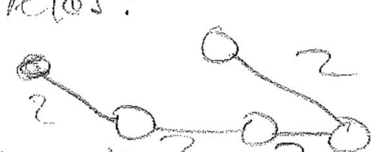
backtrack:
v = t
path = ∅
while parent[v] != ∅: ← verilog style concatenation
path = (parent[v], path)
v = parent[v]

return path

2) Let's see... Prim's creates MST based on minimal connection while Dijkstra's creates tree minimizing path length (cost)

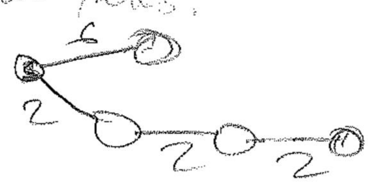


Prim's yields:



$\boxed{\text{FALSE}}$

Dijkstra's yields:



3 Problem

We are given a line L that represents a long hallway in a art gallery. We are also given a set $X = \{x_1, x_2, \ldots, x_n\}$ of distinct real numbers that specify the positions of paintings in this hallway. Suppose that a single guard can protect all the paintings within distance at most 1 of his or her position (on both sides). For instance, if $X = [0.5, 2.5, 0.8, 1, 1.5]$, then one guard placed at position 1.5 can cover all the paintings; if $X = [0.5, 7.5, 5.6, 0.9, 1, 2, 5.9, 6.6]$, then two guards (placed at, say, 1.5 and 6.5) are enough. Solve the following. [4 points]

1. Design an algorithm for finding a placement of guards that uses the minimum number of guards to guard all the paintings. For full-credit, your algorithm should run in time $O(n \log n)$. You don't have to analyze the running-time.
2. Prove the correctness of your algorithm.

greedy algo
time

1) X_sort = sort X by position (ascending) using, say, mergesort
guards = $\emptyset$ (set of positions)

while X_sort $\neq \emptyset$:
add x+1 to guards (where x is smallest pos left in X_sort)
remove all values from X_sort where $|x_i - x| \leq 1$
return guards

2) Can be proved using greedy stays ahead.

(b) Lemma: Assume guard positions are sorted.
For a given guard, its position in solution from algo is greater than or equal to ith position of ith guard in optimal solution.
(I'll prove momentarily; first, should prove more basic lemma)

(a) Lemma: Given algorithm ensures that all paintings are protected & hence is a solution.
Proof: Let's prove by contradiction. Say that it is not a solution, then some painting must be unprotected. This is immediately a contradiction because such a painting would still have to be in X_sort and thus the algorithm would not have terminated.

(Continued on back)

(b) Proof: Use Induction

Base Case: First guard is placed at smallest sorted position +1. If guard were placed any further away, the first painting would be unguarded. Thus, the base case holds and algo can't do worse than optimal here.

Inductive step: By induction hypothesis, the $i_\ell \geq j_\ell$. Must prove that $i_{\ell+1} \geq j_{\ell+1}$.

Let's prove by contradiction, namely $i_{\ell+1} < j_{\ell+1}$.

Let's call the first unguarded painting's position after i_ℓ by the name u_i and for j_ℓ, u_j.

Because $i_\ell \geq j_\ell$, $u_i \geq u_j$.

If $i_{\ell+1} < j_{\ell+1}$, $u_i+1 < u_j+1$.

This, however, is a contradiction and so inductive step holds.

(c) Lemma: The algorithm produces an optimal (and hence correct) solution.

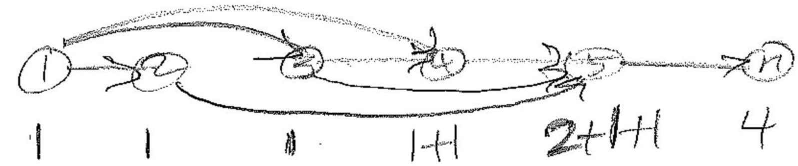
Proof: Let's again prove by contradiction:

Say $O = \{j_1, \ldots, j_m\}$ and $A = \{i_1, \ldots, i_k\}$

optimal   produced by algo

and let's say $m < k$ (so A wouldn't be optimal).

By lemma (a), we know algo produced a correct solution. By lemma (b), we can say $i_m \geq j_m$. Our algorithm terminates when there are no unprotected paintings. O would suggest that there is no unprotected painting further than j_m+1. $j_m+1 \leq i_m+1$. This, however leads to a contradiction b/c $i_{m+1}, \ldots, i_k$ should not be in A. Thus, A must produce an optimal solution.



4 Problem

Let $G = (V, E)$ be a directed graph with nodes $\{1, \ldots, n\}$. G is an *ordered graph* in that it has the following properties.

1. Each edge goes from a node with a lower index to a node with a higher index. That is, every directed edge has the form (i, j) with $i < j$.

2. Each node except v_n has at least one edge leaving it. That is, for every node i, $i = 1, 2, \ldots, n-1$, there is at least one edge of the form (i, j) with $j > i$.

Given an ordered graph $G = (V, E)$ in adjacency-list representation with the adjacency-lists specifying vertices in increasing order, give an algorithm to compute the number of paths that begin at 1 and end at n.

To get full-credit your algorithm must be correct and run in time $O(|V| + |E|)$ and you must show that your algorithm runs in $O(|V|+|E|)$ time. You don't have to prove correctness. [4 points]

O(|V|) Initialize P[i]=0 for 2...n and P[1]=1
index = 1
while index != n:
O(|E|) in total { for each vertex v_i w/ index in adj[index]:
P[i] += P[index]
O(|V|) in total index = index + 1
return P[n]

P: array on which optimal solutions are built

initialization takes O(|V|) time
contents of for loop run once for each edge in G (O(|E|)
index is incremented O(|V|) times in total
so run time is: O(|V|+|V|+|E|) = O(|V|+|E|)

basic recurrence: $opt(i) = \max \begin{cases} v_i + opt(p(i)) \\ opt(i-1) \end{cases}$

5 Problem

Consider the weighted interval scheduling setup: we have n jobs and are given as input (s_1, f_1, v_1), $(s_2, f_2, v_2), \ldots, (s_n, f_n, v_n)$ with the i'th job having start time s_i, finish time f_i, and value v_i. Now suppose that you are also given as input an integer k and are told that the server **cannot** run more than a total of k jobs. Give an algorithm that can compute the most valuable set of jobs, that is, find a set S that maximizes $\sum_{i \in S} v_i$ subject to the jobs in S not conflicting with each other and S having at most k elements.

For full-credit, your algorithm should run in polynomial-time and you don't have to analyze the running-time of the algorithm or prove correctness. You can assume that all the start and finish times are distinct. [4 points]

Looks like we'll need another dimension for constraining number of jobs to k

index of jobs — # of jobs that can be run

revised recurrence: $opt(i, j) = \max \begin{cases} v_i + opt(p(i), j-1) \\ opt(i-1, j) \end{cases}$

p(i) returns highest index of job that doesn't conflict w/ i (0 if none)

algorithm:

sort jobs by finishing time in ascending order (mergesort perhaps)

relabel jobs according to this new order (1st job has earliest finish now, nth job has latest finish)

```
opt(c,d) = 0 if c=0 or d=0
for c = 1, ..., n:
   for d = 1, ..., k:
      opt_1 = v_c + opt(p(c), d-1)
      opt_2 = opt(c-1, d)
      if opt_1 > opt_2:
         opt(c,d) = opt_1
      else:
         opt(c,d) = opt_2
```



```
backtracking: sol = ∅, c=n d=k
while c,d > 0:
   if v_c + opt(p(c), d-1) > opt(c-1, d):
      sol = c ∪ sol ; c = p(c)  d = d-1
   else:  c = c-1
return sol
```

contains answer

note: opt(a,b) = 0 if a or b is negative