CS180 Exam 2

Justin Ma

TOTAL POINTS

23 / 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 **1 / 1**

✓ **- 0 pts** Correct answer and correct explanation

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 **2 / 2**

✓ **- 0 pts** Correct DFS with reverse order

1.7 Knapsack broken item **0.5 / 1**

✓ **- 0.5 pts** You can do much better.

1.8 Cycle property **0.5 / 2**

✓ **- 1.5 pts** Not a proof or incomplete

QUESTION 2

Dijkstra 4 pts

2.1 Algorithm **1.75 / 2**

✓ **- 0.25 pts** No path finding

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 **0.5 / 1**

✓ **- 0.5 pts** the proof is not complete or fully rigorous

QUESTION 4

4 Counting paths **4 / 4**

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

QUESTION 5

5 Weighted interval knapsack **3.75 / 4**

✓ **- 0.25 pts** Not mentioned how to find set of jobs selected

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!

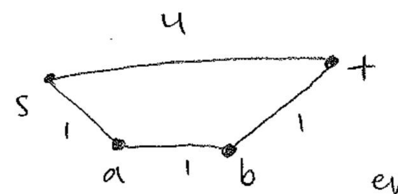
Problem	Points	Maximum
1		10
2		4
3		4
4		4
5		4
Total		26

Name	Justin Ma
UID	604805167
Section	1E

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 ; Counterexample:



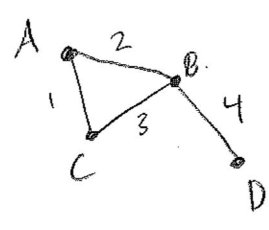
Initially, the shortest path is $\langle s, a, b, t \rangle$ with weight $= 1+1+1 = 3$. However, when 1 is added to every edge, we get $P = 2+2+2 = 6$ while path $\langle s, t \rangle$ is $4+1 = 5 < 6$

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 ; kruskal's algorithm finds a MST by continuously selecting the shortest edge that doesn't create a cycle.
If 1 is added to every edge, the different between each edge remains the same, and kruskal's algorithm is unaffected.

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 ; Counterexample:



$E > |V| - 1$
$4 > |4| - 1$
$4 > 3$ ✓

The heaviest edge $\langle B, D \rangle$ must be part of the MST because it is the only edge connected to D, a leaf. All edges connected to leaves are part of the MST.

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]

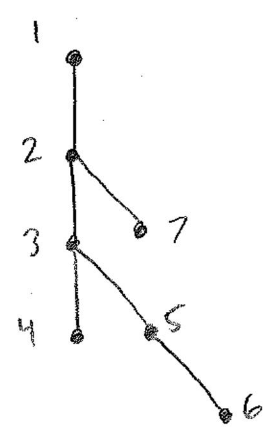
False

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.)



$OPT(j,w) = \max(v_j + OPT(j-1, w-w_j), OPT(j-1, w))$

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 $1 \le j \le n$ and $1 \le w \le W$. 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]

$O(W)$

- Let the $(n-1)$'th item be the new $(n-2)$'th item
- Let the nth item be the new $(n-1)$th item
- Recompute $OPT(j,w)$ for $n-2 \le j \le n-1$ & $1 \le w \le W$

2W steps →

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.]

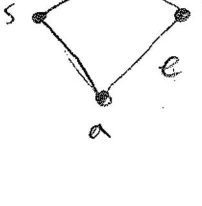
Reverse - Delete algorithm proves this. It creates a MST by deleting the largest weight edge that doesn't disconnect the graph.

The cycle C must have atleast 1 edge removed which won't break the graph, and is a necessary step since the MST cannot have cycles.

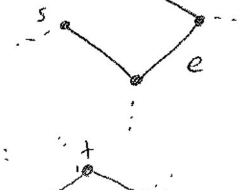
Using reverse-delete, edge e must be removed since it is the heaviest edge in the cycle and cannot be a part of the MST.

Another explanation: if e was in MST in graph A. Path ⟨s, t⟩ includes all vertices in cycle C.

cycle C:



A:



B:

7 To be in MST the length of ⟨s, t⟩ ≤ ⟨a, b⟩ in graph B.

However, since e is the largest edge, this is a contradiction.

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) Let S = NULL, Let $d'(s) = 0$, let $d'(x) = \infty$ for all verties $\neq s$

While $t \notin S$ {

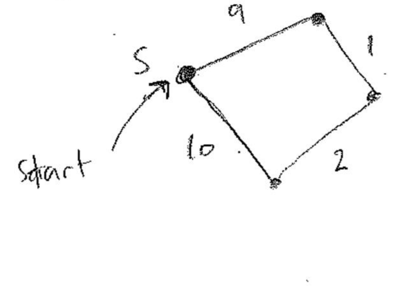
Iterate through all vertices not in S and find smallest $d'(u)$
Set $d(u) = d'(u)$
Add u to S
For all vertiles $v \notin S$
$d'(v) = \min(d'(v), \langle u,v \rangle + d(u))$

}

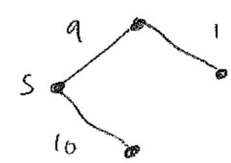
2) False; both algorithms run on the principle of adding the vertex with the least cost, but they calculate cost differently

Counterexample:

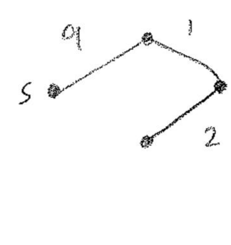
G:



Dijkstra's:



Prim's:



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.

1) - Sort X from least to greatest
- While X not empty
  - Set p = smallest position + 1
  - Add p to G
  - Remove all paintings in X with position ≤ p + 1
- return G

2) Sorting will order the positions from least to greatest. This algorithm stays ahead by determining the furthest guard possible that still guards the lowest unguarded painting (smallest position + 1). After this guard is placed, all paintings within range of this guard are discarded since we want to avoid overlapping ranges producing redundancy. Then from the group of unguarded paintings we find the next guard position that is the furthest possible while still guarding the lowest unguarded painting.

- This algorithm will never leave a painting unguarded since it keeps going until all paintings are protected.

More formal proof on back

$x_1 < x_2 < \ldots < x_n$

A has guards $(x_1, x_2, \ldots x_n)$ ← my algorithm

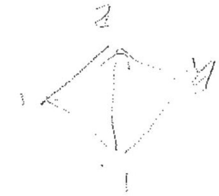
O has guards $(y_1, y_2, \ldots y_m)$ ← optimal solution

$y_1 < y_2 < \ldots < y_m$

$x_1 \geq y_1$ since x_1 is the furthest guard that can be picked while still guarding the closest painting

$x_j \geq y_j > y_{j-1}$ A stays ahead since the guard after its previous guard will again be the furthest possible position while still guarding the closest unguarded painting.

As stated earlier, my algorithm will finish since it keeps going until all paintings are protected. If there exists an unprotected painting, my algorithm still has a step to perform.



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]

Let $p(1) = 1$; let $p(i) = 0$ for $i = 2, 3, \ldots, n$ $O(V)$

For $a = 1, 2, 3, \ldots, n-1$
For every vertex b in list of vertex a } $O(E)$
$p(b) \mathrel{+}= p(a)$ $O(1)$

return $p(n)$

total run time $= O(V + E)$

The total number of elements in the adjacency list is E (since it is directed). Iterating through all items will take $O(E)$ but also must initialize all vertices i to have $p(i) = 0$ except for $p(1)$. This takes $O(V)$ since there are V vertices.

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]

- Set $P[j][0] = 0$ for $j = 0, 1, \ldots, n$
- Set $P[0][\ell] = 0$ for $\ell = 0, 1, \ldots, k$
- Sort jobs from least finish time to greatest finish time
- For $j = 1, 2, \ldots, n$
  - Set $p(j) =$ # jobs q where $f(q) < s(j)$
- For $j = 1, 2, \ldots, n$
  - For $\ell = 1, 2, \ldots, k$
    - $P[j][\ell] = \max(v_j + P[p(j)][\ell-1], P[j-1][\ell])$
- return $P[n][k]$