CS180 Exam 2

Alex Longerbeam

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 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

1.7 Knapsack broken item 0.5 / 1

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

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 3.5 / 4

✓ **- 0.25 pts** Initially jobs not sorted by finish time

✓ **- 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

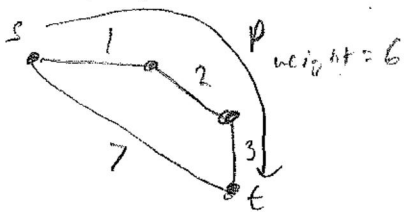
Name	Alex Longerbeam
UID	
Section	

1 Problem

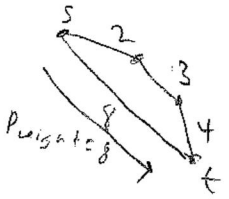
1. True or False: Let P be a shortest path from some vertex s to some other vertex t in a weighted undirected graph. If the weight of each edge in the graph is increased by one, P will still be a shortest path from s to t (with the new weights). If true, provide an explanation of why this is true and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

False

Initial graph



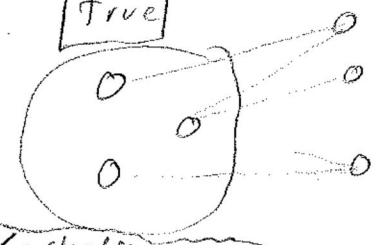
weight+1 graph



Paths change

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



For any cut, the lowest weight edge crossing the cut must be in the MST.

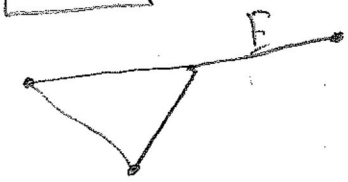
In MST $(e_1) < e_2 < e_3 < e_4$

After adding 1 to all edges $\rightarrow (e_1+1) < e_2+1 < e_3+1 < e_4+1$

Additionally, Kruskal's algorithm builds by picking lowest weight edge, all edges will still be picked in the same order

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



E is heaviest edge, it must be included in MST.

If a graph has $> |V| - 1$ edges, we know it must have a cycle, but there is no guarantee that the heavy edge E is in that cycle.

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]

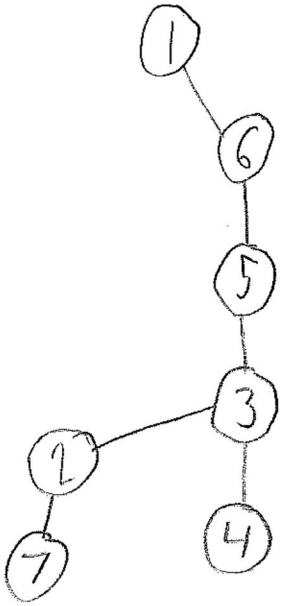
False

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.)



stack

1 2 6

1 2 1 2 5

1 2 1 2 2 3

1 2 1 2 2 4

1 2 1 2 2 2

1 2 1 2 2

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]

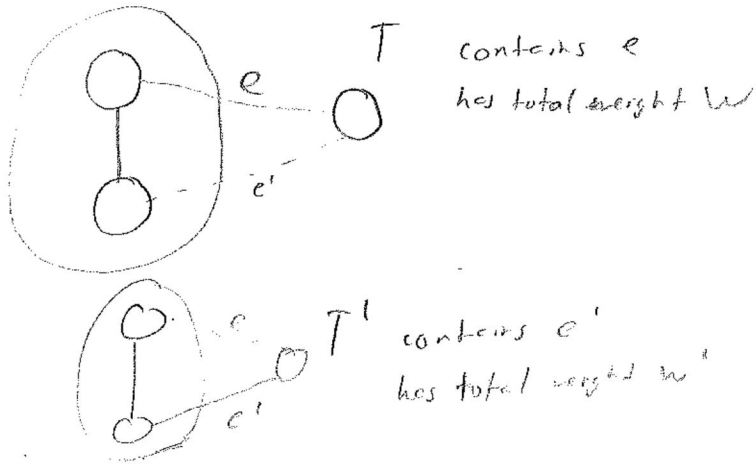
$O(3W)$

$n \quad n-1 \quad n-2$

for all weights b/c we know $n - x \quad x > 2$ so that doesn't change

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.]



Let's assume we exchange e' for e to arrive at tree T'. Since we know e is in a cycle, we know that there exists an e' such that T' is still a tree. e' is in the same cycle as e, therefore all vertices stay connected.

as there is a path P that is part of the cycle that allows for access to vertex of e that is left out otherwise

Additionally, since e' and e are in the same cycle and e is the edge with the largest weight, we know

weight → $w(e') < w(e)$

so $w(T) - w(e) + w(e') < w(T)$

T' is an MST, → $w(T') < w(T)$

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) Dijkstra's (s, t)

① Initialize set $S = \{s\}$ $d(u) = \infty$ $\forall u \in V$,
parent[u] = null $\forall u \in V$, $T = \emptyset$, $d(s) = 0$

② while $S \neq V$:

→ compute $\min(d'(u))$ for all $u \notin S$ such that (u, v) $v \in S$ is an edge
where $d'(u) = d(v) + \ell(u, v)$

→ pick (u, v) with smallest $d'(u)$

→ $d(u) = d'(u)$

→ parent[u] = v

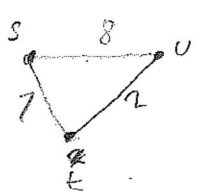
→ Add u to S

→ Add (u, v) to T

③ To find shortest path from s to t, work backwards from t following parent pointers, or find the path P in T from s to t

---

(2)



Dijkstra's:

False

Prim's

Prim's finds MST, Dijkstra's finds shortest path tree

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)

Guard placement (X)

(1) Sort X in ascending order

(2) G = ∅ is set of places of all guards

(3) For each x_i in X:

Let g = position of most recently placed guard
(highest position of any guard)
or 0 if G = ∅

if $|x_i - g| \geq 1$ OR g = 0;

place a new guard g_n at position $x_i + 1$

i.e add $g_n = x_i + 1$ to G

else

nothing, it is covered by a guard

(4) Return G

(2) Let there be some optimal solution $O = \{j_1 \cdots j_n\}$

our solution is $G = \{g_1 \cdots g_n\}$

Lemma: for all $k \leq k$ $g_k \geq j_k$ i.e. our guards after optimal guard

Induction number of guards

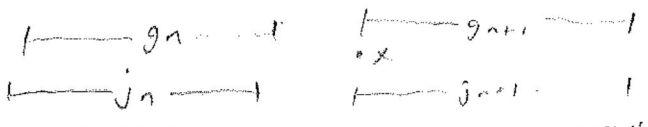
Base # = 1 our algorithm chooses the farthest guard to be able to cover the first painting so this is true

Continued on back

Inductive step: Assume true for n guards

i.e $g_n \geq j_n$

prove true for $n+1$ guards



no matter what, for each successive painting that is not covered by g_n, g_{n+1} will be the greatest position it can be to cover it.

If $j_{n+1} > g_{n+1}$,

painting x above would be left unguarded.

→ Using this lemma, prove that our solution is optimal

Assume there is an optimal solution with same guards

i.e $O = \{j_1 \dots j_n\}$ where

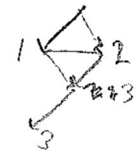
$G = \{g_1 \dots g_n\}$

we know $g_n \geq j_n$ from above lemma,

so g_n would be able to cover every painting j_n does IF NOT MORE.

IF the optimal solution could cover the last painting with j_n, then so could our solution, so our solution would not added any more guards after g_n which is a contradiction

Our solution would never have more than optimal therefore, our solution is optimal.



4 Problem

Let $G = (V, E)$ be a directed graph with nodes $\{1, \dots, n\}$. 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 $i, i = 1, 2, \dots, n-1$, 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]

we know no edge goes from something to 1, and 1 will always have something leaving it, so # paths ≥ 0

BFS w/ memoization

Motivation: go through each vertex and edge in a Breadth first way (using a queue to keep track of edges) and each time a vertex is hit, add the number of paths to the previous vertex to an array keeping track of # of paths to current vertex

Num Paths (G = (V, E))

$O(|V|)$ ① P[v]=0 for all v ∈ V , P[1]=1

~~[illegible]~~

② Q = queue, push all edges leaving 1 onto it

③ while Q is not empty:

pop e_i off Q

Let e_i go from u to v

P[v]+=P[u]

push all edges leaving v onto Q

each edge will be evaluated once, as an edge can only leave one vertex. $O(|E|)$

④ RETURN P[n]

Runtime: $O(|V| + |E|)$

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]

$P(j)$: latest job not conflicting with j

$$OPT(j,k) = \begin{cases} \max \{ v_j + OPT[P(j)][k-1] \\ \quad OPT[j-1, k] \end{cases}$$

weighted-interval:

(1) Initialize $OPT[n][k]$ array for memoization

$OPT[0][k] = 0 \quad \forall k: 1 \ldots k$

$OPT[n_i][0] = 0 \quad \forall n: 1 \ldots n$

(2) for $j = 1 \ldots n$

for $k_i = 1 \ldots k$:

$OPT[j][k_i] =$

(3) Return $OPT[n][k]$