CS180 Exam 2

[redacted]

TOTAL POINTS

20.25 / 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 **0.5 / 2**

✓ **- 1.5 pts** True

QUESTION 3

Art gallery guards 4 pts

3.1 Algorithm **2 / 3**

✓ **- 1 pts** the greedy rule is wrong

3.2 Proof of correctness **0.5 / 1**

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

QUESTION 4

4 Counting paths **2 / 4**

✓ **- 2 pts** moderate attempt (BFS/DFS doesn't return the correct number of paths)

QUESTION 5

5 Weighted interval knapsack **3.75 / 4**

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

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	[redacted]
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 . First path was A-to B and A to B to C.
Now it became A to B and A - to C

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]

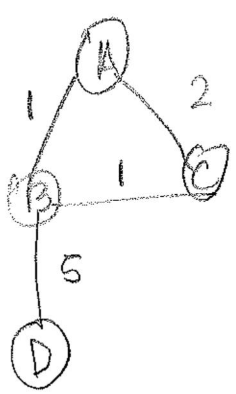
True. In MST, we care about the edge (u,v) only and not about the path.
In Prim and kruskal, we look at one edge and not about its connection to others.

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

$|N| = 4$

$|E| = 4 > 3$



(B,D) is the part of MST and it is the heaviest edge.

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

Because only the distances of the neighbors of newly added vertex changes

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

storing values in array
↓
same

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.)

1 T
2 T
3 T
4 T
5 T
6 T
7 T

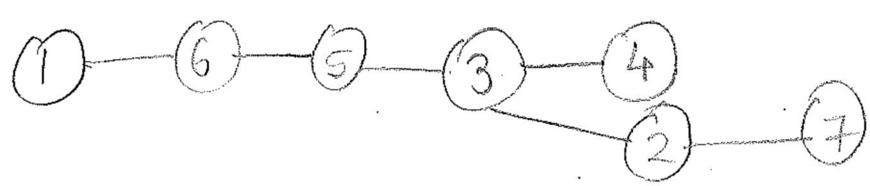
$\boxed{1} \to 2, 6$
$\boxed{2} \to 1, 3, 5, 6, 7$
$\boxed{3} \to 2, 4, 5$
$\boxed{4} \to 3,$
$\boxed{5} \to 2, 3, 6$
$\boxed{6} \to 1, 2, 5$
$\boxed{7} \to 2$

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

1 2 3 4 5 6 7
6 6 5 6 1
6 3 ~~5~~ 3 3 1
2 3 2 3 2 2 2

(1, ∅)
(1, 6)
(5, 6)
(3, 5)
(4, 3)
(2, 3)
(7, 2)

TREE:



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]

$opt(n-2, w)$, $1 < w < W$.

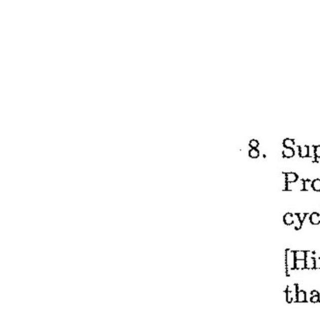
$O(W)$

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 T be a min spanning tree in G and has the edge e.

Now if we remove e, the tree T breaks into two connected components. Now to make T connected again, we need to find another edge crossing the cut (one of the connected components). As e is a part of the cycle, we can find another edge v which can connect the two connected components. So, if we swap u with v than the tree T' is still connected. Now cost of T' is



$cost(T') = cost(T) - w(e) + w(v)$

As $w(v) < w(e) \Rightarrow cost(T') < cost(T)$.

This is a contradiction. Hence, e cannot be in the minimum spanning tree of G.

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 $G = R$, $S = \{s\}$

Parent $[u] = \phi$, $u \neq s$; Parent $[s] = s$

$d'[u] = \infty$, $u \neq s$; $d[u] = \infty$, $u \neq s$, $d[s] = 0$

While $R \neq \phi$

compute $d'[v] = \left(\min_{u \in S} (d(u) + (u,v)) \right)$ for all $v \notin S$

Pick a v with least $d'[v]$

$d[v] = d'[v]$

Remove v from R

Add V to S

Find the vertex u which gave minimum distance to V from S

Parent $[v] = u$

path $= \{t\}$;

$v = t$

while $(v \neq s)$

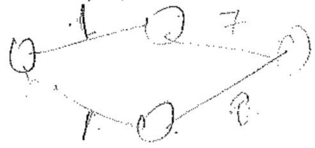
path $=$ parent $[v]$ $\cup$ path;

$v =$ parent$[v]$;

path $= \{s\} \cup$ path;

Return path;

(2)



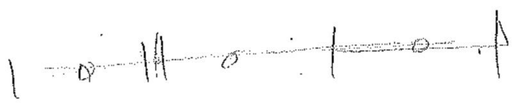
Dijkstara and Prim would give the same tree

Dijkstara looks at the total distance u whereas Prim looks at small weight edges.

small weight edges makes the path

so as we proceed prim and Dijkstara might choose different edges but in the end it gives same tree as the foundation for prim and dijsktara is same- lower weight edges.

TRUE



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 all the points in X.

distance = last element - first element.

width = $\frac{\text{distance}}{2}$; # No. of guards needed.

placement [width]; #array placement of size width;

start = first element,

previous = current = x_i (first element)

Num = 0;

while (current != x_n) last element

  if (current - start) < width

    // nothing

  else

    placement[Num] = previous;

    Start = placement[Num] + 2 ;

    Num ++

  current = previous //outside if else

  current ++ ;

2. We first sort the position of all paintings. As each bodyguard can cover a width of 2., we find no. of widths needed. This tells us the numbers of bodyguards needed. We always make a "safe choice". We choose a spot for bodyguard such that the paintings previous to it are at a distance ≤ 1. (closest to 1). Now we choose the next spot to be ≤ 2 (closest to 2). We know previous bodyguard can cover next distance of 1 and the bodyguard at newly selected place can cover 1 to its left. So, we choose a distance of 2. Thus, we always make a safe choice to ensure all paintings are guarded.

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]

$O(|V|)$

Num-path [u] = 0 for $1 \le u \le n$
Discovered [u] = False for $u \neq 1$
Discovered [1] = True
Parent [u] = ϕ $u \neq 1$
Parent [1] = 1
i = 0

$O(|E|)$

While L[i] is not empty.
 [i+1] = empty.
 For all vertices u on L[i]
  If Discovered [u] = false
   Discovered [u] = TRUE
   For all neighbours v of u
    Add v on L[i+1]
    Parent [v] = u.
  Else.
   Num path [u] ++

i ++;

Now, parent array stores the path. We get the path add the num of ways to get to each of the vertex in the path.

g(i)
If Discovered[t] = FALSE → Return 0;
v = t; path = {t}; Total = 1 # atleast 1 path

g(ii)
while (V ≠ 1)
path = parent[v] U path;
If (Num-path[v] > 0)
Total = Total + Num-path[v];
v = parent[v];

Return Total;

Total will store the numbers of paths from s to t.

Run time

initialization - O(|V|)

while loop - can have atmost |V| iterations

But let's look at how many times for loop in total can be run.

For loop = It can be run as many edges we have.
= O(|E|)

While loop for num-path = O(|V|) : atmost |V| times. a path can have atmost |V| vertices

Total run-time = O(|V|) + O(|E|) + O(|V|)

= 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]

constraint = k jobs.

$p(i)$ = the earliest job before i that does not conflict with i

Let j = No. of jobs can be done

$$OPT(i, j) = \max \begin{cases} OPT(i-1, j) & ; i \notin O \\ OPT(p(i), j-1) + v_i & ; i \in O \end{cases}$$

Algorithm

Initially $OPT(i,j) = \phi$; $0 \le i \le n$; $0 \le j \le k$

1. Calculate $p(i)$ $0 \le i \le k$;
   $p(0) = p(1) = 0$;

2. # Base case.
   $OPT(i, 0) = 0$ $\quad 0 \le i \le n$
   $OPT(0, j) = 0$ $\quad 1 \le j \le k$

3. for $i = 1, \ldots, n$
   for $j = 1, \ldots, k$
   if $OPT(i,j) \ne \phi$
   $OPT(i,j) = \max \begin{cases} OPT(i-1, j) \text{ and} \\ OPT(p(i), j-1) \end{cases}$

4. Initially $sol(i,j) = \phi$ for $0 \le i \le n$; $0 \le j \le k$

For $i = 1, \ldots, n$
    For $j = 1, \ldots, k$
        If $OPT(i-1, j) > OPT(p(i), j-1) + v_i$
            $sol(i,j) = sol(i-1, j)$

        Else
            $sol(i,j) = sol(p(i), k-1) \cup \{i\}$

5. Return $sol(n,k)$