# CS180 Exam 2

TOTAL POINTS

**20.9 / 26**

QUESTION 1

## Problem 1 10 pts

### 1.1 Shortest path **0.4 / 1**

✓ **- 0.6 pts** wrong answer with reasonable attempt

### 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 **0 / 4**

✓ **- 4 pts** No solution provided

# 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 | |
| UID | |
| 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]

True.

Suppose we used Dijkstra's Algorithm to find the path P for the old graph.

For the new graph with incremented weights, every iteration of the algorithm would still choose the same edge with minimum $d(v)$ because the relative weights did not change if all weights are incremented

So the shortest path built from the algorithm will not change.

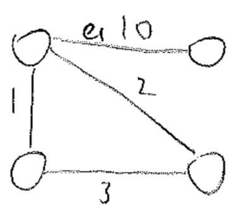
2. True or False: Let T be a MST in G. If the weights of all edges in the graph are changed by adding 1 to the weights, then T is still a MST in the graph (with the new weights). If true, provide an explanation of why this is true and if false, provide a counterexample. [1 point]

True.

Suppose we used Kruskal's algorithm to find T. The order of edges in the sorted array of edges by their weights does not change because all weights are incremented. The following steps of the algorithm only considers whether adding an edge will create a cycle, and not the weights of the edges. Therefore, T is still a MST (that will be found by the algorithm)

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



G

G has 4 vertices and 4 edges

$|E| > |V| - 1$

MST contains edge $e_1$.

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

3 : 2 4, 5
4 : 3
5 : 2 3 6
6 : 1 2 5

vis: 1 6

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

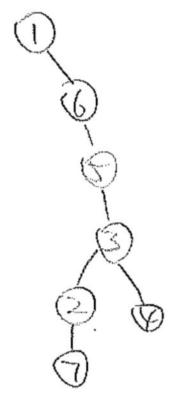
1 2 6 2 5 2 3
2 4

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]7 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.)



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]

$$W \times (n-(n-2)+1) = 3W = O(W)$$

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.]

Assume the statement is true

$e$ is the edge with the most weight of a cycle $C$ and in a MST $T$.

~~Consider the two disjo~~

there have no edge between them

Remove edge $e$ from $T$, there will be two connected components.

Let $S_1, S_2$ be the vertices of the two components.

Because $T$ is a tree, all vertices in $S_1$ are connected.
all vertices in $S_2$ are connected.

Since $e$ is in a cycle $C$, there must be another edge $e' = (u, v)$ such that $u \in S_1$ and $v \in S_2$. Because $e$ has the most weight, $W_{e'} < W_e$. $e'$ connects the two components $S_1$ and $S_2$.

Therefore, we can form a new spanning tree by removing $e$ and add $e'$. The ~~total wei~~ new total weight is $|W_T - W_e + W_{e'}|$

because $W_{e'} < W_e$, the new weight is smaller than $W_T$



$\Rightarrow$ $T$ is not a MST, a contradiction

Therefore, the statement is true.


## 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 $d[1 \text{ to } |V|] = \infty$ | Initialize $parent[1 \text{ to } |V|]$: $parent[i] = i$

$d[s] = 0$

$S = \{s\}$

while $t \notin S$:

iterate through $d$. Find the value $i$ such that vertex $i \notin S$ and $d[i]$ has the minimum value among vertices not in $S$

$S = \{i\} \cup S$

iterate through the adjacency list of $i$

for every vertex $j$ connected to $i$:

if $j \notin S$:

$$d[j] = \min \begin{cases} d[j] \\ d[i] + e(i,j) \end{cases}$$

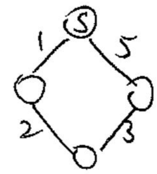
($e(i,j)$ is the weight of edge $\{i,j\}$)

$parent[j] = i$

parent is the shortest path tree.

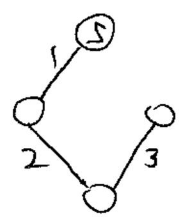
2. False.

$G =$




Tree from Dijkstra's algorithm:


Tree from Prim's algorithm:



## 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 $X$ by ascending order.
   Let $P = \{\}$ be the placement of guards
   Let last $= 1$
   while last $\leq n$
   ~~Let i = last~~
   $\boxed{\text{Let } i = x_{last} + 1}$
   ~~Find the maximum i, i ≤ n such that~~
   $P = \{i\} \cup P$
   Let $j = last + 1$, $j \leq n$
   Find the maximum $j$
   Such that $x_j - i \leq 1$
   last $= j + 1$

   $P$ is the set of guards.

2. First, during each iteration of last, we add a new guard that protects $x_{last}, x_{last+1}, \ldots, x_j$. The next iteration starts from last $= j+1$

Guards in $P$ covers all $x \in X$

Proof: The first iteration of last finds a guard that covers $x_1, x_2, \ldots, x_j$.

$\boxed{\text{Claim: In each iteration of last, } P \text{ covers } x_1 \text{ to } x_j}$

Assume after an iteration, $P$ covers $x_1$ to $x_j$, then for the next iteration, last $= j+1$, the algorithm finds a guard that covers $x_{last}$ to $x_{j'}$ ($j'$ is the $j$ in the next iteration). Therefore, this next iteration, $P$ covers $x_1, \ldots, x_j, x_{j+1}, \ldots, x_{j'}$

=> Claim is true,

For the last iteration, $j = n$, so $P$ covers the entire set $X$.

Assume there's a more optimal solution $P'$, sorted in ascending order, $\{a_1, a_2, \ldots, a_m\}$ (a)

Let the solution $P = \{b_1, b_2, \ldots, b_n\}$ (b) $m < n$

Claim: $a_i \leq b_i$ for $i = 1$ to $m$

Base case: when $i = 1$, the first iteration of the algorithm find the maximum position $j$ such that $x_1, \ldots, x_j$ are covered

Since $a_1$ must cover $x_1$, $a_1 \leq x_1 + 1$, (is the leftmost in $P'$)

In the algorithm, $b_1 = x_1 + 1$ => $a_1 \leq b_1$

Assume $a_i \leq b_i$, then in order for $P'$ to cover all arts, $a_{i+1}$ must cover the last $x$ not covered in $a_i$. Since the algorithm finds the maximum position of $b_{i+1}$ to cover all arts after the last one, and the last one has a larger $x$ because $a_i \leq b_i$, $b_{i+1}$ must be equal or bigger than $a_{i+1}$

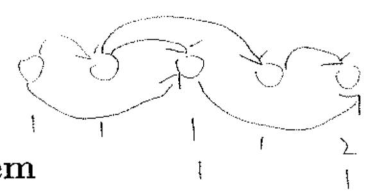
Therefore, $a_i \leq b_i$ for $i = 1$ to $m$

However, arts with $x \geq b_{m+1}$ are not covered by $b_1, \ldots, b_m$ ($P$ is sorted with ascending order)

because $a_m \leq b_m$, then arts with $x \geq b_{m+1}$ are not covered by $P'$, a contradiction

=> $P'$ does not exist,

$P$ is the solution with the minimum number of guards.



# 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[0] = 1, p[5]~~

$p[1] = 1$, $p[2 \text{ to } n] = 0$

For $i = 1, 2, \ldots, n$:

Iterate through the adjacency list of vertex $i$

Let $j$ be the vertex connected to $i$:

$P[j] = p[j] + p[i]$

$p[n]$ is the answer.

The algorithm iterates through $n$ nodes

this takes $O(n)$ time.

In each iteration, it iterates through the

edges leaving node $i$. This takes $O(deg_{out}(i))$

Total time ~~cost~~ required is:

$$O(n) + O\left(\sum_{i=1}^{n} deg_{out}(i)\right) = O(n) + O(|V|)$$

$$= O(|V|) + O(|E|)$$

Since $G$ is directed, the sum of out degrees of all vertices is the ~~total~~ number of edges, $|E|$.

## 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]