CS180 Exam 2

TOTAL POINTS

24.5 / 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 1.25 / 2

✓ - 0.75 pts Replacing with an edge that may not create a tree

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 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!

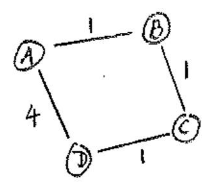
Problem	Points	Maximum
1		10
2		4
3		4
4		4
5		4
Total		26

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. Consider



$P(A, D)$ would be $A-B-C-D$
After increasing weight by 1,
$P(A, D)$ would be $A-D$.

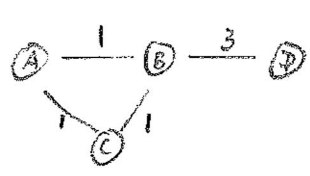
2. True or False: Let T be a MST in G. If the weights of all edges in the graph are changed by adding 1 to the weights, then T is still a MST in the graph (with the new weights). If true, provide an explanation of why this is true and if false, provide a counterexample. [1 point]

True. Observe MST has $|V|-1$ edges always.
If increase weight by 1, $w'(MST) = w(MST) + |V| - 1$ for the original MST. Suppose $\exists$ MST^* now with lower total cost.
$w'(MST^*) = w(MST^*) + |V| - 1$. By our assumption,
$w'(MST^*) < w'(MST) \Rightarrow w(MST^*) < w(MST)$,
contradiction, because MST is the minimum spanning tree with original weights. So such MST^* does NOT exist.

"w'" calculated under new weights.
"w" is for old weights.

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. Consider



Then $e(B, D)$ is the only way to get to D, so it will be in every possible MST, although it's the unique 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.

(Written in algorithm)

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.

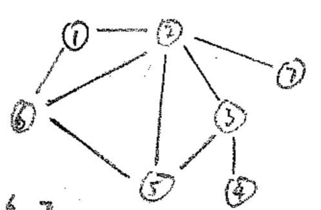
(the two are the same idea).

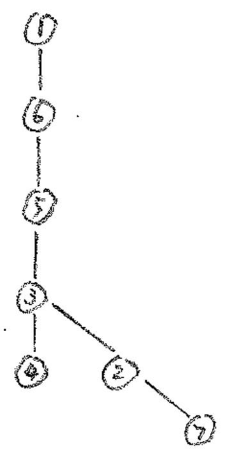
6. Let $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6, 7\}$ and

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

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.)





par	1	2	3	4	5	6	7
	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
	ϕ	1	ϕ	ϕ	ϕ	1	ϕ
	6	6	ϕ	ϕ	6	1	ϕ
	6	5	5	ϕ	6	5	ϕ
	6	3	5	3	3	5	ϕ
	1	3	4	3	3	5	ϕ
	2	3	2	3	2	2	2
	2	7	2	3	2	3	2

parent array

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]

If #n-2 ∉ optimal choice, then no affection, ⇒ $O(1)$.
If #n-2 ∈ optimal choice, then we need to redo part of DP.

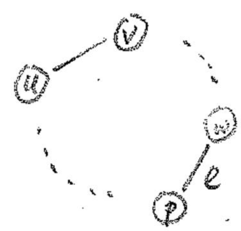
```
for ( i = n-2 ; i <= n ; i++)
    for ( j=1 ; j<=W ; j++)
        ...       // with v(n-2) update to 0.
```

So need $O(3W) = O(W)$.

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.]

Suppose the structure of the cycle is:



, where $e(w,p)$ has the most weight in this cycle.

Assume towards contradiction.
MST contains $e(w,p)$.
By tree structure, MST does ~~NOT~~ have cycles, so among all the edges in this cycle, there is at least one edge not in MST.
Without loss of generality, assume $e(u,v) \notin$ MST.
Consider now we exchange $e(w,p)$ by $e(u,v)$, observe

$T' = T + \{e(u,v)\} - \{e(p,w)\}$

is still connected, because any path requires $e(w,p)$ can be replaced by a path $p \to \cdots \to u \to v \to \cdots \to w$.
But since Weight $(e(u,v))$ < weight $(e(w,p))$, implies T' has a lower total cost, contradicts to T being the minimum spanning tree. Therefore, $e(w,p) \notin T$.

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]

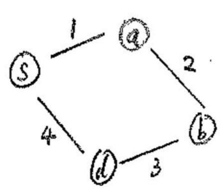
1. Starting from s, find the neighboring node of s with shortest distance, call it u_1. Add u_1 to set S (set S is the usual definition in class). Then relax other nodes by the newly added u_1,

We just need to iterate the list for node u_1 to find all its neighbors not yet in set S.

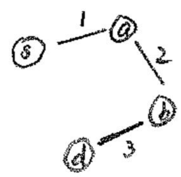
$$d[v] = \min(d[v],\ d[u_1] + weight(e(u_1, v))) \quad \text{if } \exists\, e(u_1, v).$$

Here, $d[v]$ means the current shortest distance possible by having nodes on path $s - v$ being all from set S, except v itself. (discussed in lecture)
Then find $\min_{v \notin S} \{d[v]\}$, and add that v into set S.
Repeat until reach designated node t.

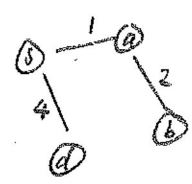
2. False. Consider



Prim's Tree :



Dijkstra's 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. Sort $\{x_i\}$ from low to high by merge-sort : $O(n\log n)$.

Greedy : place first guard at $y_1 = x_1 + 1$.
Scan through the painting until the first $x_j > y_1 + 1$ for some $1 \le j \le n$.
place second guard at $y_2 = x_j + 1$.
continuely scan through the painting until the first $x_k > y_2 + 1$ for some $1 \le k \le n$.
place third guard at $y_3 = x_k + 1$.
Repeat the same process until all paintings are covered.

Requires only one full iteration through paintings : $O(n)$.

2. Let T be the set of guard positions generated by Greedy.
O be the optimal position of guards to minimize the number of guards.
Let $P_i \in T$ be the ith guard's position in T, $g_i \in O$ be the ith guards position in O. Claim $P_i \ge g_i$.
We prove by induction. $P_1 \ge g_1$. Indeed, P_1 is the right-most position the first guard can get, because if we move further down right, we lose the cover for at least the left-most painting. Since O is a valid set, $P_1 \ge g_1$. Suppose now $P_k \ge g_k$ for $1 \le k \le r-1$. Claim $P_r \ge g_r$. Assume towards contrary. If $P_r < g_r$, since $P_{r-1} \ge g_{r-1}$, we have,

(* represents a painting)

Since $g_{r-1} \le P_{r-1}$, if guard at P_{r-1} cannot cover *, certainly cannot a guard at g_{r-1}. But $dist(*, g_r) > dist(*, P_r) = 1$, implies the next guard in O cannot cover * either. Contradict to O is a valid set. So $g_r \le P_r$. Complete induction step.

Since the jth guard in T is always standing farther to right than jth guard in O, it's obvious T requires fewer or equal number of guards than O. Since O is optimal, $|T| = |O|$, T is optimal as well.

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]

Define $f[i]$ be the total number of paths from 1 to i.

Initialize $f[1] = 1$ since it just need to stay there without moving.

First traverse the graph by BFS to build up a "parent-adjacency-list",

i.e. if $e(u, v) \in G$, in normal adjacency list for directed graph,

$v \in list\_u$, but $u \notin list\_v$,

but here we're building the parent list, meaning $u \in list\_v$, $v \notin list\_u$.

A "parent-adjacency-list" is easier to use in DP.

DP : for $(i: 2, \ldots, n)$

for (j : iterate through "parent-list" for node i)

$f[i] \mathrel{+}= f[j]$

Since $j \in list\_i$ in "parent-list", it means $i \in list\_j$ in original adjacency list. Since G is ordered, we have $i > j$. So in DP loop, we only consult the entries we've already calculated.

For DP loop, time complexity is:

$O(\sum_{i=2}^{n} |E_i|) = O(|E|)$, where $|E_i|$ is the number of edges pointing to node i (of course, by orderness of G, the nodes on the other side of the edges have index smaller than i).

Therefore, total time: $O(|E|+|V|) + O(|E|) = O(2|E| + |V|) = O(|E|+|V|)$

Then we can trace back to find the actual set.
Suppose $f[k][j^*]$ is the max, then we have $\#j^* \in \mathcal{O}$, where $\mathcal{O}$ is the optimal set.
Then among $\{f[k-1][i]\}_{1\le i<j^*}$, find the $\#i^*$ that leads to $f[k][j^*]$ we just found. $\Rightarrow$ we now know $\# i^* \in \mathcal{O}$.
Then among $\{f[k-2][p]\}_{1\le p<i^*}$, find the $\#p^*$ that leads to $f[k-1][i^*]$, so we know $\#p^* \in \mathcal{O}$. Continue repeating this process until we come up with the whole optimal set $\mathcal{O}$.

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]

Sort the tasks by their starting time from low to high. Define $f[i][j]$ to be the most value under the setting that a total number of i jobs is allowed to run on the server, and we are considering the first j jobs from the set of all tasks, AND we require to run the jth job.

We emphasize that $f[i][j]$ <u>only optimize COMPATIBLE</u> job sets with the last job "j".

$$f[i][j] = \max_{\substack{1\le k<j,\\ \text{job } \#k \\ \text{and job} \\ \#j \text{ are} \\ \text{compatible}}} \{ f[i-1][k] \} + v_j$$

Note if no $\#k$ is compatible with $\#j$, then we require $\max_k \{f[i-1][k]\}$ be 0 (for maximize an empty set).
Then $f[i][j] = v_j$ in this case, meaning we're only able to run task $\#j$, a single task, because it always has time conflict.

Since any optimal choice will end with some job, we output

$$\max_{1\le j\le n} \{ f[k][j] \}.$$

Pseudocode:

```
for ( i : 1, ..., k)
    for ( j : 1, ..., n)
        update f[i][j].

for ( j : 1, ..., n)
    find the maximum among { f[k][j] }
Output MAX.
```