CS180 Exam 2

Alexander Xu

TOTAL POINTS

23.25 / 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 **1.5 / 2**

✓ **- 0.5 pts** Wrong in one or two places

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 **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 **4 / 4**

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

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

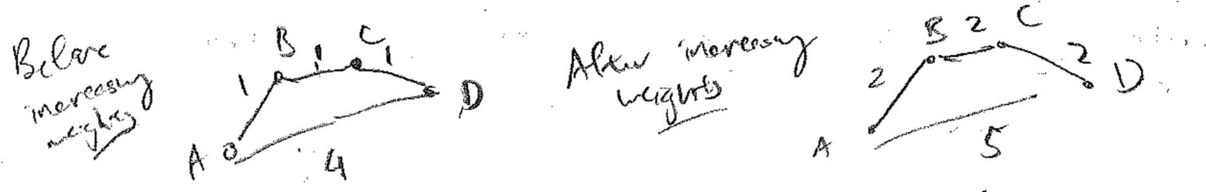
Name	Alexander Xu
UID	504966392
Section	1F

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

Counter example. SP from A to D



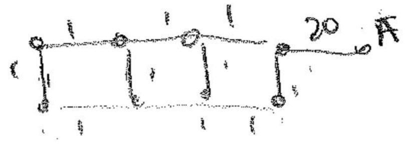
P: { A, B, C, D }   P = { A, D }

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 - using Kruskal's algorithm, we can show that T is still an MST. Kruskal's picks edges in non-descending order - thus, since the structure of the graph doesn't change, we only care about the relative weights of the edges. e_i, before and after addition, will still be the ith edge evaluated because relative weights do not change

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 - counter-example below



The edge of length 20 is uniquely heaviest, and must be in the MST due to the cut property on the cut { A }.

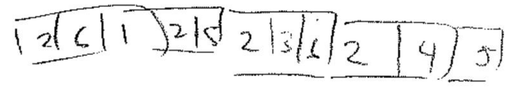
4. True or False: When running Prim's algorithm, after updating the set S, we only need to recompute the attachment costs for the neighbors of the newly added vertex. No justification necessary. [1 point]

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

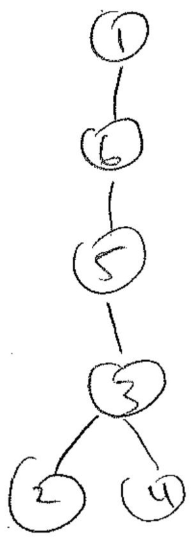
6. Let $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6, 7\}$ and



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

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.)



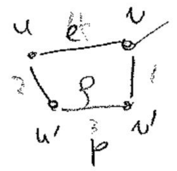
i from 1 to n
w from 0

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]

You can recompute it in $O(|W|)$, i.e. 2 iterations of the inner loop.

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 an MST which contains the edge $e = \{u, v\}$. Since e is part of a cycle C, there exists some other path P which connects u and v.

Let T' be $T / \{e\}$. We can add Path P to T' so that T' still spans the entire graph G.
Let f be some edge in P that connects $\{u', v'\}$.
Since $w(f) < w(e)$ by definition, and f is the only new edge, we find that T' has less total weight than T.
Since T' also spans the entire graph, T cannot be an MST, and e cannot be part of any MST in G.

2 Problem

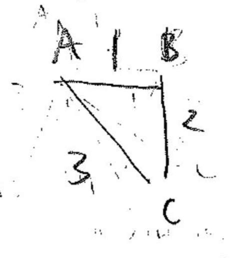
1. Write down Dijkstra's algorithm for computing a shortest path between two vertices s and t in a weighted undirected graph $G = (V, E)$ given in adjacency-list representation. [2 points]

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]

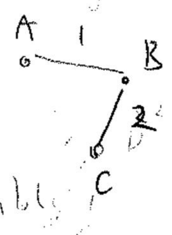
1.
```
dijkstra(G, s, t):
    Let A = {s}, (V, E) = G  → Let Parent[s] = s
    Let d(0) = 0
    For all v ≠ s:
        Let d(v) = ∞
    While A ≠ V:
        For all v ∈ V not in A:
            Let d'(v) = min(d(v), minimum d(u) + w({u, v}))
                        (u: some vertex u in A; {u,v}: some edge in E)
        For the v with minimum d'(v):
            d(v) = d'(v)
            add v to A
            Parent[v] = u (the vertex that obtained the shortest path to v)
    return d(t)
```

2. False : counter example:

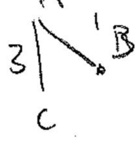
G:



Prim's :



Dijkstra's



Depending on how we write it, Dijkstra's only updates a parent if a new minimum is found

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.

0.5 0.9 1 2 5.6 5.9 6.6 7.5
0 1 2 3 4

|0|2|5|

1.
```
PlaceGuards(X, L):
    Let SOL = ∅
    Sort X in ascending order
    Let S = 0
    For i from 1 to |X|:
        if X[i] >= X[S] + 2:
            Add (X[i] + X[S])/2 to SOL
            S = i
    Return SOL
```

2. This algorithm utilizes a greedy approach to optimize guard placement
It always makes "safe choices". X[0] should always be on the lower left bound of the first guard's radius. Then, the next guard will ideally be staggered such that its radius begins after where the previous one ends. Essentially we separate locations into distance-2 ranges, and only place guards in the centers of those ranges. In this way, our algorithm stays ahead, by making safe choices and only placing guards when absolutely necessary

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 $j > i$.

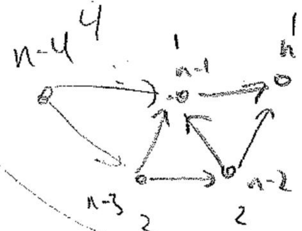
Given an ordered graph $G = (V, E)$ in adjacency-list representation with the adjacency-lists specifying vertices in increasing order, give an algorithm to compute the number of paths that begin at 1 and end at n.

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]

Recurrence: Path[n-1] = 1

Path[i] ⇒ number of paths from i to n, going backwards

Path[i] = Path[i] + Path[j] ← {i, j} being an edge



```
Num Paths (G):
    Let E = edges of G
    Let Path[n-1], Path[n] = 1
    For i from n-2 to 1:
        For every edge {i, j} in E:
            Path[i] += Path[j]

    Return Path[1]
```

This runs in O(|V| + |E|), since the inner loop will eventually hit every edge once due to the outer loop, giving us O(|E|). Additionally, The outer loop causes us to consider each vertex once, adding an additional O(|V|) 13 to our solution's runtime complexity

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]

$$\begin{tikzcd}
 & n \arrow[dl, "\in O"'] \arrow[dr, "\notin O"] & \\
v_n + DP[P[n], k-1] & & DP[n-1, k]
\end{tikzcd}$$

```
Scheduling (J, k):
    let P[i] be the index of the last job that finishes
    before job i starts
    let DP[i, K] = 0 for i from 0 to n
    let SOL[i, k] = ∅ for k from 0 to K
    For i from 1 to n:
        For k from 1 to K:
            DP[i, k] = max( DP[P[i], k-1] + v_i ,
                            DP[n-1, k] )
            SOL[i, k] = SOL[P[i], k-1] ∪ {v_i} if the first choice was grown,
                        SOL[n-1, k] otherwise
    let MAX = 0
    For k from 1 to K
        MAX = max(MAX, DP[n, k])
        if MAX changed, record max_k
    return SOL[i, max_k]
```

This loop is because we might use < k items.