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	

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. You can take the graph with edges $\{1,2\},\{2,3\}$ with weight 1 each and edge $\{1,3\}$ with weight 2.5 and $s=1, t=3$.

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. Does not change the edges picked by any of the algorithms we did in class. For instance, say you use Kruskal's algorithm, then at every stage, the least weight edge that does not create a cycle would be the same so you would pick the same edges (in the same order) in both runs.

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. Take the graph with edges $\{1,2\},\{2,3\},\{3,1\}$ all having weights $1,2,3$ respectively and edge $\{1,4\}$ having weight 100.

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.)

   To be worked out in discussion section.

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]

   $O(1)$ **time is in fact enough** . You can use

$$\text{new best-value} = \max \begin{cases} OPT(n-3, W) & \\ v_{n-1} + OPT(n-3, W - w_{n-1}) & (\text{if } w_{n-1} \leq W) \\ v_n + OPT(n-3, W - w_n) & (\text{if } w_n \leq W) \\ v_{n-1} + v_n + OPT(n-3, W - w_{n-1} - w_n)) & (\text{if } w_{n-1} + w_n \leq W) \end{cases},$$

   where the cases correspond to the new opt not containing either $\{n-1, n\}$, containing $n-1$'th item but not n'th item, containing n'th item but not $n-1$'th item, containing both $n-1$'th item and n'item respectively.

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.]

   Solution. See Claim 4.20 in text.

   Note that there are many wrong ways of "proving" this by doing a swap (much like there were wrong ways of doing the swap in the proof of the CUT property) where if you don't use the right edge of the cycle to swap with (pretty much the only choice that works is an edge of the cycle that **crosses the cut** formed by removing the edge e), you end up with either a disconnected graph or a graph for which you don't have a proof that it is a tree. All such solutions were deducted points.

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]

To be worked out in discussion section.

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.

Solution You can follow a greedy strategy as follows:

1. Sort all items in X to get the sorted array Y.
2. Set $G = \emptyset$.
3. While Y is not empty:
   (a) Let y be first element of Y. Add $y + 1$ to G.
   (b) Starting from the left of Y, remove all elements of Y that are within distance at most 1 from y (you stop the first time you see an element at distance more than 1 from y).
4. Return G.

The run-time of the first step is $O(n \log n)$. After sorting, the algorithm takes $O(n)$ time as each element of Y is looked at most twice in (c).(ii).

Correctness (sketch). There are several ways to prove the correctness including a greedy stays ahead strategy as measured by how far out the j'th guard is.

A more direct proof is as follows. Let $y_1 < y_2 < \ldots < y_k$ be the elements chosen in G. You can use induction to prove that the difference between any two consecutive y's ($y_2 - y_1$, $y_3 - y_2$, etc.,) is more than 2. Further, by the construction of the algorithm, there must be paintings at $y_1 - 1, y_2 - 2, \ldots, y_k - 1$. As these paintings are at distance more than 2 from each other, no matter where a guard is placed, a guard can only cover at most one of these paintings. Therefore, at least k guards are needed.

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 $i, i = 1, 2, \dots, n-1$, 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]

Solution. For $j \geq 2$, let $Paths(j)$ denote the number of paths that begin at 1 and end at j. Note that $Paths(j)$ satisfy the recurrence

$$Paths(j) = \sum_{i<j:(i,j)\in E} Paths(i).$$

You can convert this recurrence into an algorithm as follows:

```
1. Initialize $Paths(j) = 0$ for $1 \leq j \leq n$.
2. For $j = 2, \dots, n$:
    (a) For each edge $(i, j)$ into vertex $j$:
        i. Set $Paths(j) \leftarrow Paths(j) + Paths(i)$. (Increment $Paths(j)$ by $Paths(i)$.)
```

The run-time of the algorithm is $O(|V|)$ for the initialization in Step (1), and for each j, the number of times the inner-most line $(a).(i)$ is triggered is exactly the number of edges into j. So the total number of times (a).(i) is executed is the sum of the total number of edges which is $|E|$. So the total run-time is $O(|V| + |E|)$.

Common mistakes: (1) Having a counter that is incremented every time you find a path from 1 to n using some variant of BFS/DFS. This takes exponential time. (2) Having a recursive formulation of the above recurrence that does not use memoization.

5 Problem

Consider the weighted interval scheduling setup: we have n jobs and are given as input (s_1, f_1, v_1), $(s_2, f_2, v_2), \dots, (s_n, f_n, v_n)$ 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]

Solution. This problem is kind of a mix of WIS and knapsack (in fact, someone asked a closely related question in class ...).

Sort the jobs according to their finish times first. In the following, for ease of notation suppose this is already done. Let $OPT(j, \ell)$ denote the best value you can get out of the first j jobs (i.e., the j jobs as ordered by the finish times) while using at most ℓ jobs in total for $1 \leq \ell \leq k$. Then, $OPT(j, \ell)$ obeys the recurrence

$$OPT(j, \ell) = \max(OPT(j-1, \ell) + v_j + OPT(p(j), \ell - 1)),$$

where $p(j)$ denotes the largest index of a job that does not conflict with the j'th job (as we defined in class as well).

You can now compute the OPT values using the above recurrence and also find the best set of jobs as we did in class by tracing the recurrence relation.

Common mistakes: (1) Having a recursive formulation of the above recurrence that does not use memoization. (2) Not sorting the jobs by finish times first. (3) Not using $p(j)$ in the recurrence. (4) Not explaining how to find the best set of jobs.