CS180 Exam 2

Rajiv Anisetti

TOTAL POINTS

24.25 / 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 1 / 1

✓ - 0 pts Correct. You can compute the new value in O(1) time.

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 2.5 / 4

✓ - 1.5 pts correct algorithm with exponential running-time (You are exhausting all possible paths by exploring all possible paths in BFS/DFS)

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!

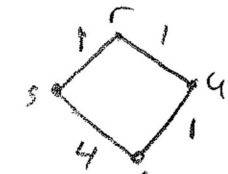
Problem	Points	Maximum
1		10
2		4
3		4
4		4
5		4
Total		26

Name	Rajiv Anisetti
UID	904804422
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; example: given the following graph: the shortest path from s to t is s->r->u->t. When each edge is increased by one, the new shortest path will be s->t as 5 < 6.

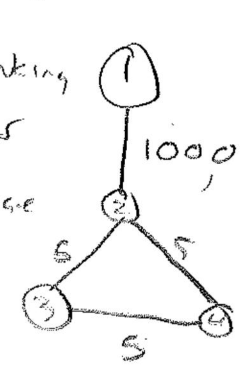


2. True or False: Let T be a MST in G. If the weights of all edges in the graph are changed by adding 1 to the weights, then T is still a MST in the graph (with the new weights). If true, provide an explanation of why this is true and if false, provide a counterexample. [1 point]

True; Say that the weights were changed (added 1) and the MST changes such that one edge e is removed and replaced with an edge k. That means originally that w(e) < w(k), but w(e)+1 > w(k)+1, which is impossible. Thus, if we chose the edge e to cross the cut, we would choose it again.

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; a counterexample is given in the following graph. Taking out the edge (1,2) disconnects ~~the~~ graph so the heaviest edge must be in the MST.



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

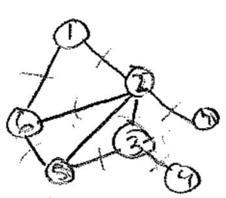
6. Let $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6, 7\}$ and

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

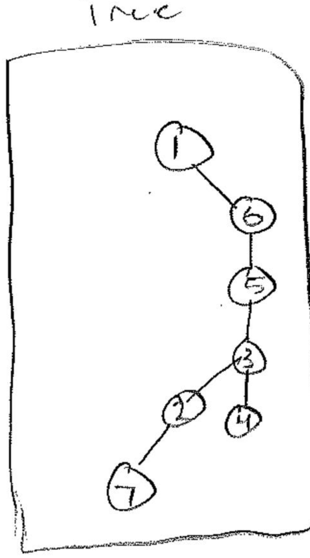
Suppose that G was given to you in adjacency list representation where the elements in the adjacency list are ordered in increasing order. For example, the adjacency list of vertex 2 would be $[1, 3, 5, 6]$. Draw the DFS tree that you would get when doing DFS starting from 1. (Just the final tree is enough. No need to show intermediate stages.) [2 points]

(Recall that elements of the adjacency list are processed in increasing order.)

6



Tree



Explored | Parent

Stack: 2, 6 — T, F, F, F, F, F, F | 1, 1, -, -, -, 1, -

Popped 6 Stack: 2, 1, 2, 5 — T, F, F, F, F, T, F | 6, 6, -, -, 6, 1, -

Popped 5 Stack: 2, 1, 2, 2, 3, 6 — T, F, F, F, T, T, F | 6, 5, 5, -, 6, 5, -

Pop 6 S: 2, 1, 2, 2, 3

Pop 3 S: 2, 1, 2, 2, 2, 4, 5 — T, F, T, F, T, T, F | 6, 3, 5, 3, 3, 5, -

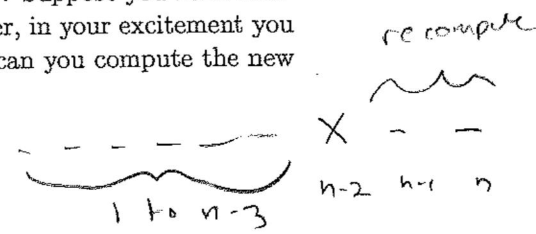
Pop 5 S: 2, 1, 2, 2, 2, 4.

Popped 4 S: 2, 1, 2, 2, 2, 3 — T, F, T, T, T, T, F | 6, 3, 4, 3, 3, 5, -

Popped 3 S: 2, 1, 2, 2, 2

Popped 2 S: 2, 1, 2, 2, 1, 3, 5, 6, 7 — T, T, T, T, T

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]



O(1), only need to recalculate last 2.

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 that e is part of the MST for G. When we take e out of the MST, we derive two connected components that must be reattached. Let's say e connected vertices u and v and S = all connected vertices to u. Because e is part of a cycle, we know there is another edge e' that crosses the cut from S. If we replace e in the MST with e', knowing that w(e') < w(e), we know that the overall tree T' has a lower cost than the original MST T. In addition, T' is still connected as we can now "take the long way around" to get to v from u. Thus T' is a spanning tree with less weight than the original supposed MST with edge e, a contradiction!

7

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) $T = \emptyset$ $S = \{start\}$ $d(s) = 0$ $d(u) = \infty$ for $u \in V \neq S$

parent[s] = s parent[u] = null for all $u \in V \neq S$

While $S \neq V$

For each vertex $v \notin S$

compute $d'(v) = \min(d(u) + l_{(u,v)} : u \in S)$

Find the vertex v that achieved lowest $d'(v)$

For this vertex v

add v to S

$d(v) = d'(v)$

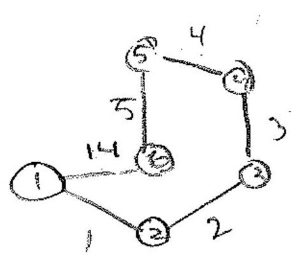
Find the vertex u that satisfied $d(u) + l_{(u,v)}$ to be the minimum

set parent[v] = u

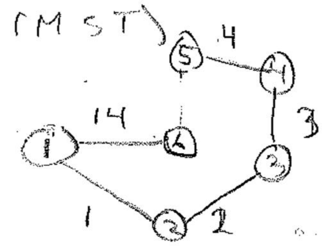
Add (parent[v], v) to T

To find shortest path from s to t, follow parent pointers from t

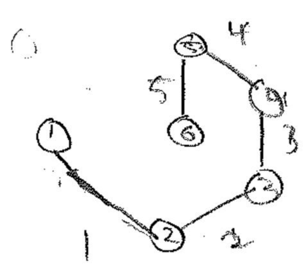
2) False: Dijkstra computes shortest path from s and Prim's computes the tree of least weight (MST)



→ Dijkstra's tree →



Prim's Tree →



3 Problem

(0.5, 0.9, 1, 2, 5.6, 5.9, 6.6, 7.5)

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) First, sort the array X into increasing distances → X'

Bodyguard placement set S = ∅

int currPos = first element in X' + 1 , int i = 2

if |X| < 2, add currPos to S and return S

while i ≤ n

  if |currPos - X'[i]| > 1

    add currPos to the set S

    currPos = X'[i] + 1

  if i equals n

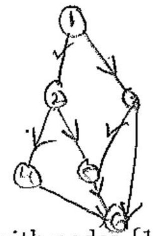
    add currPos to the set S

return S

2) – Say an optimal placement (A) has positions $i_1 \ldots i_k$ in increasing order. We want to prove that our solution $j_1 \ldots j_m$, for every $l \leq k$, $j_l \geq i_l$ so that greedy stays ahead, and we need the least space at the end to add another bodyguard if needed.

– Base case: |X| = 1 → Correct, as j_1 is the furthest distance possible from X_1, 1 away so $j_1 \geq i_1$

– Induction step: Say we have added k bodyguards to the set and are about to add the k+1th. Because we defined the position of the next bodyguard to be the maximal distance from the next painting and $j_k \geq i_k$, and $j_{k+1} > j_k$, then $i_{k+1} \geq j_{k+1}$. Thus, the algorithm has at most the optimal's space left to guard remaining at any point and greedy stays ahead.



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]

Because the graph is ordered and directed, we can use a modified DFS to achieve our goal.

int count = 0

Initialize stack R and add vertex 1 to R

while R is not empty

  u ← vertex from popping R

  if u == vertex n

    count++

  add neighbors of u (only directed ones) to stack R

return count

This algorithm clearly runs in O(V+E) time. This is a simplified approach of DFS that can only be implemented because the graph cannot have cycles. As we have less steps than DFS and we never backtrack due to the graph's ordered property, it is at most O(V+E). We are essentially just traversing the adjacency list with very few repeated edges.

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]

Let us derive a recurrence; say we have an optimal solution (O)

OPT(l, j) = optimal set of first l jobs with a job limit of j

OPT(l, j) = OPT(l-1, j).. OPT(l, j) = OPT(p(l), j-1) + v_l

Recurrence → OPT(l, j) = max(OPT(l-1, j), v_l + OPT(p(l), j-1))

Calculate p(l) = latest job before l that is non-conflicting

Set OPT(0, j) = 0 for j ≤ k

Set OPT(l, 0) = 0 for l ≤ n

for l = 1 to n
  for j = 1 to k
    Compute OPT(l, j) from recurrence above

Set Sol(0, j) = ∅ for j ≤ k

Set Sol(l, 0) = ∅ for l ≤ n

for l = 1 to n
  for j = 1 to k
    if OPT(l, j) == OPT(l-1, j)
      Sol(l, j) = Sol(l-1, j)
    else
      Sol(l, j) = {job l} ∪ Sol(p(l), j-1)

Return Sol(n, k)