CS180 Exam 2

Shrey Kakkar

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 **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** Replace with an edge may not exist /didn't specify which edge

QUESTION 2

Dijkstra 4 pts

2.1 Algorithm **2 / 2**

✓ **- 0 pts** Correct

2.2 Dijkstra vs Prim **0.5 / 2**

✓ **- 1.5 pts** True

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!

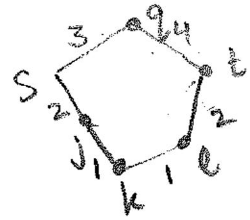
Problem	Points	Maximum
1		10
2		4
3		4
4		4
5		4
Total		26

Name	Shrey Kakkar
UID	204792652
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



Let P = { s → j → k → l → t } = 6 → shortest

Let T = { s → q → t } = 7

If we add 1 to each weight.
P = 3 + 2 + 2 + 3 = 10
T = 4 + 5 = 9 → shortest

→ Hence P is not shortest anymore

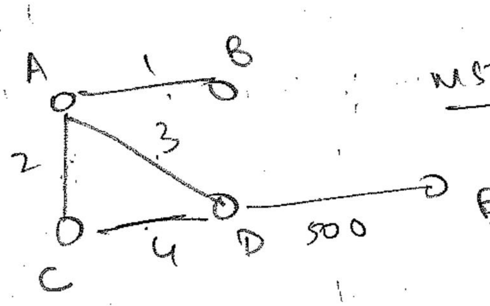
2. True or False: Let T be a MST in G. If the weights of all edges in the graph are changed by adding 1 to the weights, then T is still a MST in the graph (with the new weights). If true, provide an explanation of why this is true and if false, provide a counterexample. [1 point]

True → from cut property.

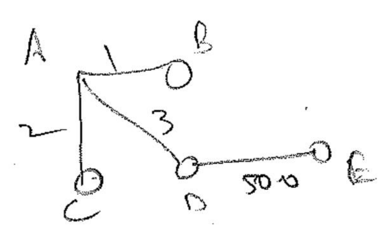
For any cut, the least edge crossing the cut still stays the same edge, as all edges had weight added by 1.

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



MST →



→ Here, we have more than $|V| - 1$ edges, but still heaviest edge (edge DE) has to be part of 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

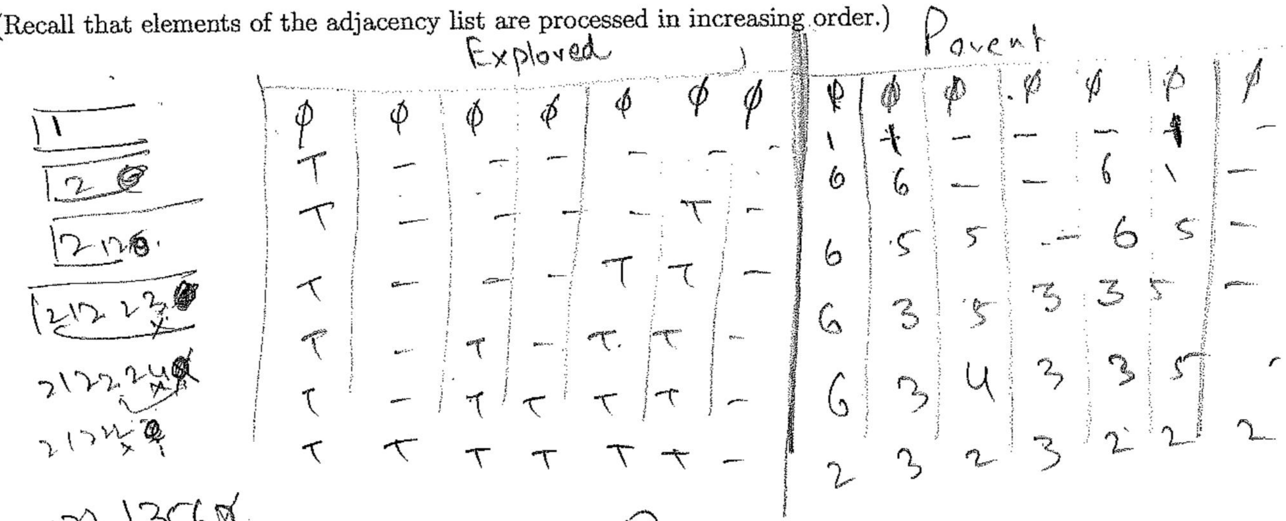
1 → 2, 6
2 → 1, 3, 5, 6, 7
3 → 2, 4, 5, 6
4 → 3
5 → 2, 3, 6
6 → 1, 2, 5
7 → 2

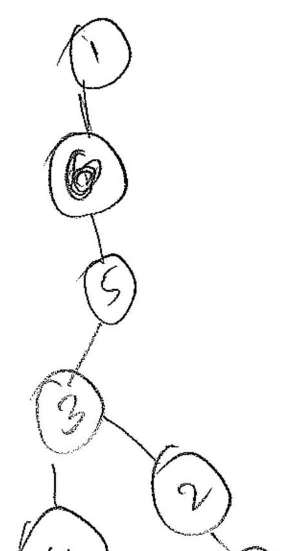
6. Let $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6, 7\}$ and

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

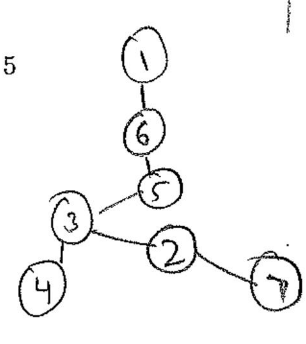
Suppose that G was given to you in adjacency list representation where the elements in the adjacency list are ordered in increasing order. For example, the adjacency list of vertex 2 would be $[1, 3, 5, 6]$. Draw the DFS tree that you would get when doing DFS starting from 1. (Just the final tree is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

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





ANS →

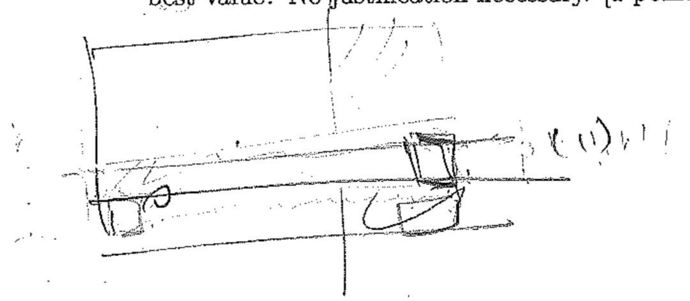


1 → 5 → 10 → 15
2 → 10 → 12
3 → 14 → 22

30

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]

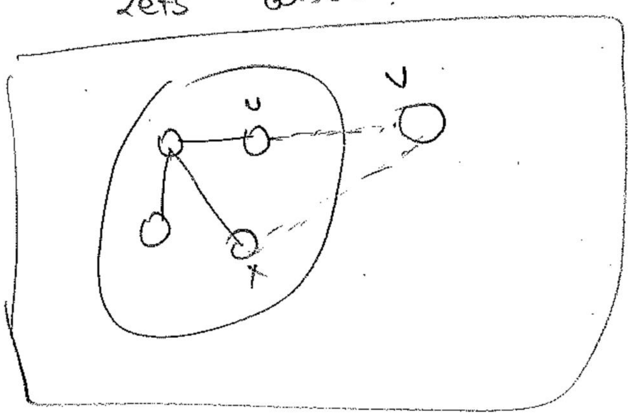
$O(w)$



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.]

Lets assume that it is part of the tree [edge (u,v)]



Lets make a cut with all vertices connected to u, except v. we assume that the edge next added to the set is {u,v}
we also know $w_{\{x,y\}} < w_{\{u,v\}}$
Hence, if we add {u,v} to the set, we are violating the cut property.
Hence, our assumption must be wrong, and {u,v} is not part of the tree.

we can also do it in the way that
$T' = T - \{u,v\} + \{x,v\}$
⇒ $T' < T$, Hence T is not an MST
Hence, heaviest edge of cycle is not part of MST

[Also, removing {u,v} & adding {x,v} doesnt disconnect the graph, as we know u and x are already connected (as they are part of cut) and we are connecting either of those vertices to v]

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's Algorithm

Let $S = \{start\}$, V = All vertices, $T = \phi$, Parent[u] = ϕ (for all $u \in G$)
$d(u) = \infty$
$d(start) = 0$

while $S \neq V$:

For all vertices $v \notin S$
compute $d'(v) = \min\{d(u) + l_{u,v}\}$ //u = vertex in S

Find v with minimum $d'(v)$
Add v to S
$d(v) = d'(v)$
Parent[v] = u (u is the vertex for which $d'(v)$ was minimum)
Add edge {Parent[v], v} to T.

To compute shortest path:
Start from t, compute {t, parent(t), parent(parent(t)) .... until you reach s.

(2) True [length is the weight of each edge]
This is because both the algorithms have the same structure and design.

Essentially, if length of edge is the weight of the edge, then prime & dijkstra are the same.

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) we will form a greedy Algorithm.

Compute_gard_pos :

(1) Arrange all paintings in ascending order of position such that for $X = \{x_1, x_2 \ldots x_n\}$ $x_1 \le x_2 \le x_3 \ldots \le x_n$

(2) $G = \phi$, $Y = X$ (ordered in ascending order)

(3) while Y is not null :

a) pick smallest distance painting from set. (say x_j)

b) Add gaurd at location $(x_j + 1)$

c) Remove all paintings from Y with location $\le x_j + 2$

(4) return G

→ Basically, we add a gaurd at the farthest position for which it can gaurd the closest painting. Then we remove all paintings from the set, that are in the added gaurd's view. we do this until all paintings are gaurded.

(2) Correctness → (1) Lemma → Our Algorithm adds gaurd at location better than or same as optimal solution

Base step → for 1 painting, both algorithms add gaurd at same location of painting.

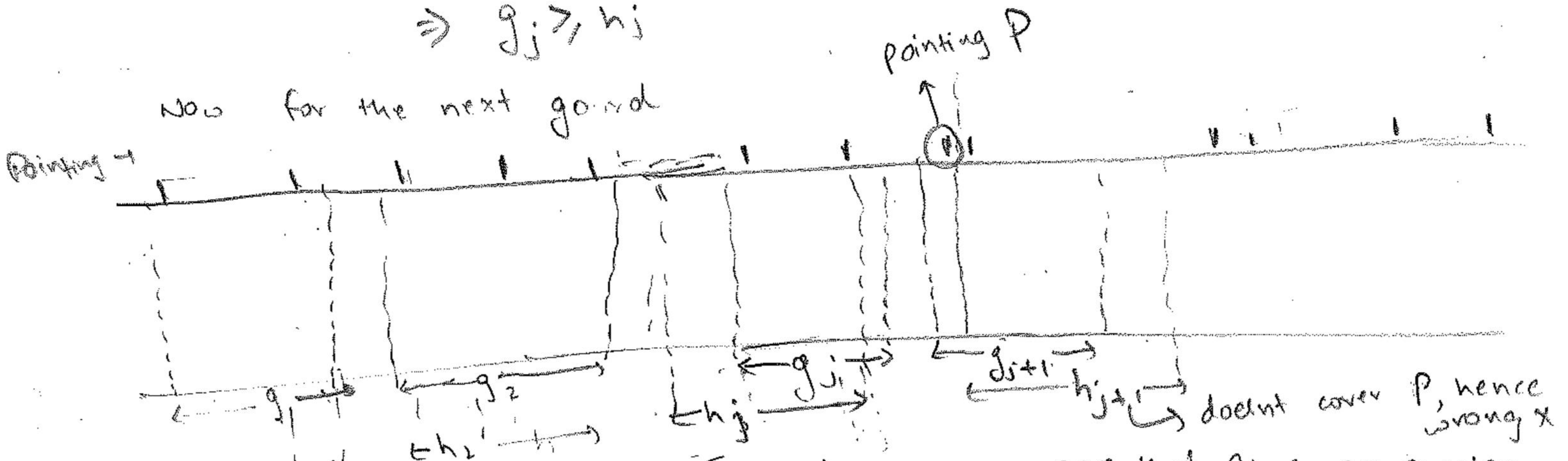
Inductive step.

↳ Assume our algorithm adds j gaurds better than optimal solution.

$$G_i = \{g_1, g_2 \cdots g_j\} \quad G_0 = \{h_1, h_2, \cdots h_j\}$$

$$\Rightarrow g_j \geq h_j$$

Now for the next gaurd



[from figure, we can see that g_j covers greater area than h_j]

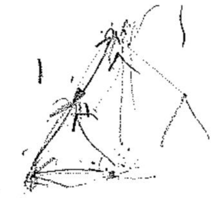
Our algorithm adds g_{j+1} at distance 1unit ahead of next pointing (P), ensuring that the next pointing is covered.

Since the gaurd's range is 2m, if h_{j+1} is located anywhere after g_{j+1}, the pointing P in the figure is left unguarded. Hence h_{j+1} has to be placed before or at some location as g_{j+1}, hence proving that our algorithm is as good as optimal one.

Lemma → our algorithm covers all pointings.

↳ Since gaurds are placed at a greater distance by our algorithm than the optimal one, it is not possible that optimal solution covers a pointing our algorithm misses.

Also, our algorithm keeps placing gaurds until all pointings are secure, hence it is correct.



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]

For any node K, number of paths from 1 to K is the sum of all paths from 1 to all vertices in adjacency list of K

Hence, we can create a dynamic programming algorithm

we first create a new adjacency list, with direction of the graph in oppisite direction. Hence 1→2, 2→3 becomes 2→1, 3→2 in the new list.

Algorithm

N[1] = 1

O(n) × O(E): for each node u in graph {1, 2 ... n} iteratively:

    for each node v in the new adjacency list of node u:

        N[u] += N[v]

return N[n]

The algorithm has run time O(V+E) just like BFS. For each node, u, the max edges it can have is |u| − 1. In total through all iterations, we are going through each edge only once, Hence runtime is O(V+E)

from algorithm = runtime = $\sum_{i=1}^{n} \text{degree}(u) = O(E) = O(V+E)$

5 Problem

we know p(j) values for all j = 1 ... n

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]

OPT [0, k] = 0 for k = {1 ... K} → k = total no. of jobs

OPT [n, 0] = 0 for n = {1 ... n} → n = no. of jobs as input.

For l = 1, ..., n

For w = 1 ... K

OPT [l, w] = max [Vn + OPT (p(n), w-1), OPT (l-1, w)]

If OPT [l, w] = OPT (l-1, w)

sol (l, w) = sol (l-1, w)

Else

sol (l, w) = [{Sn, fn, vn} ∪ sol (p(n), w-1)]

[knowing the p(j) values for each job j takes O(n²) time, and our algorithm runs in O(nk) time

~~Correctness → Induction~~

~~Base case → Algorithm works if only 1 job is available, as we only choose that job~~

~~Inductive step: → If our algorithm works till job j, it works for j+1th job as well, because for the next job, we either include it, or we~~

[p(n) = largest job i (i < n) which doesn't conflict with n]