CS 141 homework7

Celyna Su

TOTAL POINTS

97 / 100

QUESTION 1

1 Problem1 25 / 25

✓ **- 0 pts Correct**

- 4 pts No relation between n and m specified

- 5 pts No/Incorrect time complexity of array based implementation

- 2 pts Provides another reason for array based being faster and not specifying the relation between n and m.

- 3 pts Incorrect relation between n and m specified

- 25 pts No solution

QUESTION 2

2 Problem2 22 / 25

✓ **+ 10 pts Gives O(n + m) algorithm to find route from s to t.**

+ 8 pts Insufficient or incorrect explanation of algorithm to find route from s to t.

+ 5 pts Use DFS/BFS without modification

+ 0 pts Incorrect solution for (1)

+ 15 pts Gives O((n + m) log n) algorithm to find minimum tank capacity from s to t.

✓ **+ 12 pts Insufficient or incorrect explanation of algorithm to find minimum tank capacity from s to t.**

+ 5 pts Use Dijkstra without any modification or didn't indicate how to realize the modification

+ 0 pts Incorrect/Missing solution

QUESTION 3

3 Problem3 25 / 25

✓ **- 0 pts Correct**

- 15 pts Incorrect Proof

- 8 pts Did not prove using contradiction

- 25 pts No Attempt

QUESTION 4

4 Problem4 25 / 25

✓ **- 0 pts Correct**

- 2.5 pts 1 is incorrect

- 2.5 pts 2 is incorrect

- 2.5 pts 3 is incorrect

- 5 pts 4 is incorrect

- 2.5 pts 5 is incorrect

- 5 pts 6 is incorrect

- 20 pts Incorrect solution

- 25 pts No solution

CS 141, Spring 2019 Homework 7

Posted: May 30th, 2019 Due: June 6th, 2019

Name: Celyna Su

Student ID #: 862037643

- You are expected to work on this assignment on your own
- Use pseudocode, Python-like or English to describe your algorithms. Absolutely no C++/C/Java
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your algorithms
- Homework has to be submitted electronically on Gradescope by the deadline. No late assignments will be accepted
- Unless otherwise noted, for all questions about graphs, $n = |V|$ is the number of vertices/nodes, and $m = |E|$ is the number of edges/links

Problem 1. (25 points)

You are given an implementation of Dijkstra that uses an unsorted array for the priority queue. What is the time complexity of Dijkstra as a function of n and m? Under which conditions on n and m is the array-based implementation faster than the binary-heap-based implementation?

Answer:
With an unsorted array, the time it takes to insert all the vertices into the queue is $O(n)$. Then, we have to look through the entire array to find the minimum, which also takes $O(n)$ time (because it is done n times). Checking each edge relaxation takes $O(1)$ each, so $O(m)$ overall.
Therefore, the running time is $O(n^2 + m)$, simplified to $O(n^2)$ because $m \in O(n^2)$.

Binary-heap Based Implementation: $O((n+m)\log n)$, simplified to $O(m \log n)$ assuming that $m \in O(n^2)$

Thus, the array-based implementation will be faster if $m > \frac{n^2}{\log n}$.

Problem 2. (25 points)

You are given a set of cities, along with the pattern of highways between them in the form of an undirected graph $G = (V, E)$. Each stretch of highway $e \in E$ connects two of the cities, and you know its length in miles, l_e. You want to get from city s to city t. There is one problem: your car can only hold enough gas to cover L miles. There are gas stations in each city, but not between cities. Therefore, you can only take a route (path) if every one of its edges has length $l_e \leq L$.

1. Given the limitation of your car's fuel tank capacity, show how to determine in $O(n+m)$ time whether there is a feasible route from s to t.

2. You are planning to buy a new car, and you want to know the minimum tank capacity that is needed to travel from s to t. Give a $O((n+m)\log n)$ algorithm to determine this capacity. **Hint:** Modify Dijkstra's algorithm to find paths that minimize the maximum weight of any edge on the path (instead of the path length).

Answer:

1. DFS from s, ignore edges of weight (W) larger than L.

2. Modification of Dijkstra's: change the distance from s and t to be the minimum over all paths p from s to u of the maximum length edge over all edges of p. Then, compare with the minimum over all p of the sum of lengths of edges in p.

Algorithm modification of final loop in Dijkstra's:

1. while priority queue (Q) is not empty
   a. u = deletemin(Q)
   b. for all edges$(u, v) \in E$:
      if distance(v) ¿ max(distance(u, l(u, v))
         distance(v) = max(distance(v), l(u, v))
         decreasekey(Q, v)

Problem 3. (25 points)

Let $G = (V, E)$ be an undirected weighted graph. Prove that if all its edge weights in G are distinct, then G has a unique minimum spanning tree.

Answer: (Answer from lecture notes)

Proof: WTS if all edge weights in G are distinct, then G has a unique minimum spanning tree (MST).

- Suppose an undirecte graph contains 2 different MST, denoted T_a and T_b.
- Let e_1 be the smallest edge weight in which $e_1 \in T_a$.
- Add e_1 into T_b to form a cycle. Thus, the cycle contains the edge e_2 in T_b, which is larger than e_1.
- Adding e_1 into T_b and deleting e_2 from T_b creates a better MST than T_b.

This is a contradiction because it states that the undirected graph contains 2 different MST.

Problem 4. (25 points)

You are given an undirected graph $G = (V, E)$ with positive weights, and a minimum spanning tree $T = (V, E')$ for G; you might assume that G and T are given to you as adjacency lists. Now suppose that edges weights in G are modified from $w(e)$ to $w'(e)$. You wish to quickly update the minimum spanning tree T to reflect these changes, without recomputing the tree from scratch. Consider the following six distinct scenarios for updates: for each give a $O(n + m)$-time algorithm to update the MST.

1. $\forall e \in E$, update $w'(e) = w(e) + C$ where C is a positive constant
2. $\forall e \in E$, update $w'(e) = w(e) - C$ where C is a positive constant
3. for a single edge $e \notin E'$, update $w'(e) = w(e) + C$ where C is a positive constant
4. for a single edge $e \notin E'$, update $w'(e) = w(e) - C$ where C is a positive constant
5. for a single edge $e \in E'$, update $w'(e) = w(e) - C$ where C is a positive constant
6. for a single edge $e \in E'$, update $w'(e) = w(e) + C$ where C is a positive constant

Answer: (Credit to M.I.T for help)

1. $T' = T$; we are scaling the graph by addition.
2. $T' = T$; we are scaling the graph by subtraction.
3. $T' = T$; we are scaling the non-MS graph by addition.
4. a. IF e not in MST decreases, compare it to all the other edges in the MST.
   b. IF $e <$ some edge (denoted n) in the MST AND doesn't create cycle, update MST to include n.
   c. IF e is NOT smaller than some edge in the MST, we do not change the original MST.
5. $T' = T$
6. a. IF e not in MST increases, compare it to all the other edges in AND outside MST.
   b. IF $e <$ some edge (denoted n) not in the MST AND doesn't create a cycle, update MST to include n.
   c. IF e is NOT $>$ another edge not in the MST, we do not change the original MST.