CS 141, Spring 2019 Homework 7

Problem 1.

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: If one uses an unsorted array, the time to insert all the vertices in the queue is $O(n)$. In order to find the minimum, we have to scan the array which takes $O(n)$. This is done n times. Each edge relaxation may imply a key update which is fast, just $O(1)$ time each for an overall $O(m)$. The running time is therefore $O(n^2 + m)$, which can be simplified to $O(n^2)$ because $m \in O(n^2)$.

The binary-heap based implementation takes $O((n+m)\log n)$. We can simplify the complexity to $O(m \log n)$ assuming that $m \in O(n^2)$.

Therefore the array-based implementation is faster if $m > n^2/\log n$.

Problem 2.

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:

a) This can be done by performing DFS from s ignoring edges of weight larger than L.

b) This can be achieved by a simple modification of Dijkstra's algorithm. We redefine the distance from s to t to be the minimum over all paths p from s to u of the maximum length edge over all edges of p. Compare this with the original definition of distance, i.e. the minimum over all p of the sum of lengths of edges in p. This comparison suggests that by modifying the way distances are updated in Dijkstra we can produce a new version of the algorithm for the modified problem. It is sufficient to change the final loop to:

while priority queue Q is not empty:
  u = deletemin(Q)
  for all edges $(u, v) \in E$:
    if dist(v) > max(dist(u), l(u, v))
      dist(v) = max(dist(u), l(u, v))
      decreasekey(Q,v)

When implemented with a binary heap, this algorithm achieves the required running time because its structure is identical to Dijkstra's.

Problem 3.

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: Proof by contradiction. For the sake of contradiction, let's suppose the graph has two different MSTs T_1 and T_2. Let e be the lightest edge which is present in exactly one of the trees (there must be some such edge since the trees must differ in at least one edge). Without loss of generality, say $e \in T_1$. Then adding e to T_2 gives a cycle. Moreover, this cycle must contain an edge e' which is (strictly) heavier that e, since all lighter edges are also present in T_1, where e does not induce a cycle. Then adding e to T_2 and removing e' gives a (strictly) better spanning tree than T_2 which is a contradiction.

Problem 4.

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: Let's call T' the MST on the new graph G' with the updated weights.

1. $T' = T$
2. $T' = T$
3. $T' = T$
4. We include e in the tree, thus creating a cycle. We then remove the heaviest edge e' in the cycle, which can be found in linear time, to get a new tree T'. It is intuitively clear that this algorithm should work, but a rigorous proof seems surprisingly tricky. To prove this, we argue that T' contains a least weight edge across every cut of G and is hence an MST. Note that since the only changed edge is e, $T \cup \{e\}$ already includes a least weight edge across every cut. We only removed e' from this. However, any cut crossed by e', must also be crossed by at least one more edge of the cycle, which must have weight less than or equal to e'. Since this edge is still present in T', it contains a least weight edge across every cut.
5. $T' = T$

6. We remove e from the tree to obtain two components, and hence a cut. We then include the lightest edge across the cut to get a new tree T'. We can now "build up" T' using the cut property to show that it is an MST. Let $X \subseteq T'$ be a set of edges that is part of some MST, and let $e_1 \in T' - X$. Then, $T' - \{e_1\}$ gives a cut which is not crossed by any edge of X and across which e_1 is the lightest edge. Hence, $X \cup \{e_1\}$ is also a part of some MST. Continuing this, we can grow X to $X = T'$, which must be then an MST.