PSET 2

Due on **Tuesday October 08, 2024 at 11:59 pm ET**

CS 601: Algorithms and Complexity, Fall 2024 (Akash Kumar) Version: **1.0**

Submitted by <**name**>

Grading Policies:

(a) **Total points: 100**

(b) There are **five** problems in this pset (each of which have one or more subparts). Please solve/attempt them all.

(c) You are allowed –indeed you are encouraged– to, collaborate with other students in the class. Please list all the people you collaborated with.

(d) **Quite a few problems should be available online.** You should feel free to consult online resources, but please use those only after you spend some serious time thinking about the problems.

(e) I cannot emphasize this point enough. **With all the freedom given in the points above, it is imperative that you write your solutions yourself.** Please do not copy/paste what your friends wrote or what electronic material you find online. This defeats the purpose of solving a problem set. Indeed, **if you are caught engaging in unseemly practices, severe action will be taken.**

(f) Please write your pseudocode as you see those written in textbooks. Confusing/inscrutable psuedocodes will be penalized with no room for a regrade.

(g) Please write your mathematical claims rigorously and unambiguously. Again, follow the style laid out in textbook or in the class if you are unsure. Ambiguous or unclear claims/proofs will be penalized. Your TAs are not here to debug your proofs/algorithms. If your writeup does not convince them of the correctness, you will rightfully lose points.

1 (20 PTS.) VERTEX CUTS

Let $G = (V, E)$ be an undirected, unweighted graph with $n = |V|$ vertices. The distance between two vertices $u, v \in V$ is the length of the shortest path between them. A vertex cut of G is a subset $S \subseteq V$ such that removing the vertices in S (as well as incident edges) disconnects G. Show that if there exist $u, v \in V$ of distance $d > 1$ from each other, then there exists a vertex cut of size at most $\frac{n-2}{d-1}$. Assume G is connected.

Solution:

The key here is to think of the graph as being organized into "layers". The zeroth layer contains just the source vertex, u and the last layer – the d-th layer contains just the target vertex, v. To get some intuition, let us first take the case where $d = 2$. In this case, note that all the other vertices belong to the first layer. You can remove all the vertices in layer 1 and this constitutes a vertex cut of size $n - 2$ as desired. Let's try to extend this idea for bigger values of the parameter d.

You note that this extension is immediate. After all, this time around, the remaining $n-2$ vertices are sitting in $d-1$ intermediate layers between layer 0 and layer d. By an averaging argument, one notices that the smallest layer cannot contain any more than $(n - 2)/(d - 1)$ vertices. You can remove them all to obtain a vertex cut of size $(n - 2)/(d - 1)$.

2 (30 PTS.) MINIMUM SPANNING k-FOREST

Given a graph $G = (V, E)$ with non-negative weights, a spanning k-forest is a cycle-free collection of edges $F \subseteq E$ such that the graph with the same vertices as G but only the edges in F has k connected components. For example, consider the graph $G = (V, E)$ with vertices $V = \{A, B, C, D, E\}$ and all possible edges. One spanning 2-forest of this graph is $F = \{(A, C), (B, D), (D, E)\}$, because the graph with vertices V and edges F has components $\{A, C\}, \{B, D, E\}$.

The minimum spanning k-forest is defined as the spanning k-forest with the minimum total edge weight. (Note that when $k = 1$, this is equivalent to the minimum spanning tree). In this problem, you will design an algorithm to find the minimum spanning k-forest. For simplicity, you may assume that all edges in G have distinct weights.

2.A. (10 PTS.) Define a j-partition of a graph G to be a partition of the vertices V into j (non-empty) sets. That is, a j-partition is a list of j sets of vertices $\mathcal{S} = \{S_1, S_2, \cdots, S_j\}$ such that every S_i includes at least one vertex, and every vertex in G appears in exactly one S_i. For example, if the vertices of the graph are $\{A, B, C, D, E\}$, one 3-partition is to split the vertices into the sets $\mathcal{S} = \{\{A, B\}, \{C\}, \{D, E\}\}$. Define an edge (u, v) to be crossing a j-partition $\mathcal{S} = \{S_1, S_2, \cdots, S_j\}$ if the set in $\mathcal{S}$ containing u and the set in $\mathcal{S}$ containing v are different sets. For example, for the 3-partition $\mathcal{S} = \{\{A, B\}, \{C\}, \{D, E\}\}$, an edge from A to C would cross $\mathcal{S}$.

Show that for any j-partition $\mathcal{S}$ of a graph G, if $j > k$ then the lightest edge crossing $\mathcal{S}$ must be in the minimum spanning k-forest of G.

2.B. (20 PTS.) Give an efficient algorithm for finding the minimum spanning k-forest. Give the algorithm, the proof of correctness and the runtime analysis.

Solution:

(a) To get some intuition, consider the case $k = 1$ and $j = 2$. Note that this situation can be handled by using the cut property. We will use a similar argument to handle the situation where $j > k > 1$.

Towards getting a contradiction, suppose e is the lightest edge which crosses the j-partition $\mathcal{S}$. Suppose e is not in the minimum spanning k-forest (which we call F). Now, consider adding e to F. Two cases occur.

Either, you get a cycle. In this case, some other edge e' also crosses $\mathcal{S}$. This means $F + e - e'$ is also a spanning k-forest which has weight smaller than F. Contradiction.

Otherwise, you have a case where $F + e$ does not have a cycle. In this case, the endpoints of e lie in two different components of F and so $F + e$ has $k - 1$ components. Since $\mathcal{S}$ is a j-partition with $j > k$, there is some edge $e' \in F$ which crosses $\mathcal{S}$. Again, note that $F + e - e'$ is a k-forest and since e is the lightest edge crossing $\mathcal{S}$, it has weight smaller than F. Again a contradiction.

(b) Just run Kruskal's algorithm and stop when you have included $n - k$ edges.

There are multiple ways to see this is correct. One way I like is to use an exchange argument. Let us sort the edges used by Kruskal in increasing order of weights and let us write the edges in this order as: $e_1, e_2, \ldots, e_{n-k}$. Suppose there is some other optimal sequence which uses $n - k$ edges which we write in increasing order of weights as $f_1, f_2, \ldots f_{n-k}$. We will like to show that these two sequence are the same. To this end, let us consider the first place where the two sequences differ. Say, the first difference occurs at index i. Note that this means $f_i > e_i$ (we have a strict inequality since all weights are distinct). Let $S_1, S_2, \ldots, S_j$ be the components defined by the solution Kruskal's arrived at prior to adding e_i. $S_1, S_2, \ldots S_j$ form a j-partition and by definition of the algorithm, $j > k$. By the preceding part, note that it better hold that $f_i = e_i$. Thus, the two sequence must be identical and therefore, our final solution must be the minimum spanning k-forest.

The runtime is the same as Kruskal's: $O(m \log n)$.

3 (20 PTS.) MSTS – SOME OF THIS WAS COVERED IN CLASS. JUST CHECKING IF YOU REMEMBER IT.

For each of the following statements, either prove or give a counterexample. Always assume $G = (V, E)$ is undirected and connected. Do not assume the edge weights are distinct unless specifically stated.

3.A. (5 PTS.) Let e be any edge of minimum weight in G. Then e must be part of some MST.

3.B. (5 PTS.) If e is part of some MST of G, then it must be a lightest edge across some cut of G.

3.C. (5 PTS.) If G has a cycle with a unique lightest edge e, then e must be part of every MST.

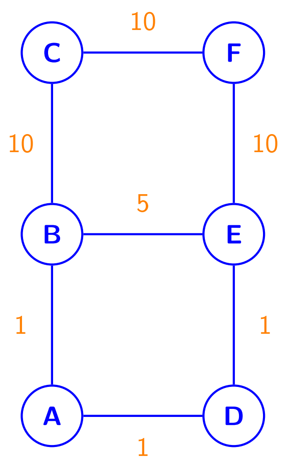
3.D. (5 PTS.) For any $r > 0$, define an r-path to be a path whose edges all have weight less than r. If G contains an r-path from s to t, then every MST of G must also contain an r-path from s to t.

Solution:

3.A. True. Imagine calling Prim's algorithm from one of the endpoints of the edge e.

3.B. True. Indeed, consider removing the edge $e = (u, v)$ from the MST, T. The tree breaks into two subtrees. We will call the subtree containing u as T_u and the subtree containing v as T_v. If there is an edge e' that connects the vertex set of T_u and the vertex set of T_v which has a cheaper cost than e, then the tree T could not have been a MST. In other words, e has to be the cheapest edge across $V(T_u)$ and $V(T_v)$.

3.C. False. Suppose I have a graph where the edge e is the unique smallest weight edge on some cycle and additionally, the same edge is also the heaviest edge on *some other cycle*. Consider the following picture:



3.D. True. Let $s = v_1, v_2, \ldots, v_k = t$ denote the r-path in G. Suppose the MST still does not contain an r-path from s to t. Thus, there are some edges on the r-path above which do not appear on the s-to-t path in the MST. Consider the first edge on this path that is not in the MST and write it as (v_i, v_{i+1}) where $1 \leqslant i < k$. Consider adding this edge to the MST. This produces a cycle and if there is any edge on this cycle with weight strictly greater then we can safely swap that heavy edge out and include the edge (v_i, v_{i+1}) in the MST instead and this gives a contradiction.

4 (20 PTS.) MORE MSTS

You are given a graph $G = (V, E)$ with positive edge weights, and a minimum spanning tree $T = (V, E')$ with respect to these weights; you may assume G and T are given as adjacency lists. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\widehat{w}(e)$. You wish to quickly update the minimum spanning tree T to reflect this change, without recomputing the entire tree from scratch. There are four cases. In each, give a description of an algorithm for updating T, a proof of correctness, and a runtime analysis for the algorithm. Note that for some of the cases these may be quite brief.

4.A. (5 PTS.) $e \notin E'$ and $\widehat{w}(e) > w(e)$.

4.B. (5 PTS.) $e \notin E'$ and $\widehat{w}(e) < w(e)$.

4.C. (5 PTS.) $e \in E'$ and $\widehat{w}(e) < w(e)$.

4.D. (5 PTS.) $e \in E'$ and $\widehat{w}(e) > w(e)$.

Solution:

4.A. Do nothing. The weight of T does not change. Other spanning trees either continue to have the same weight or their weights go up. Time $O(1)$.

4.B. Add e to T. Remove the heaviest edge in this cycle. Time taken $O(n)$.

4.C. Do nothing. The weight of the MST can only go down. Time $O(1)$.

4.D. Delete $e = (u, v$ from T. Call the subtrees obtained T_u and T_v. Find the lightest edge connecting the vertex sets of T_u and T_v by searching over all edges of G.Time taken: $O(n + m)$.

5 (10 pts.) Even More MSTs – A divide and conquer algorithm for MST

Is the following algorithm correct? If so, prove it. Otherwise, give a counterexample and explain why it doesn't work.

```
FindMST($G$)
  1. If $n = 1$, return the empty set.
  2. $T_1 \leftarrow$ FindMST($G_1$). Here $G_1$ is the graph induced on first $n/2$ vertices.
  3. $T_2 \leftarrow$ FindMST($G_2$). Here $G_2$ is the graph induced on last $n/2$ vertices.
  4. $e \leftarrow$ cheapest edge across the cut $\{1, 2, \cdots, n/2\}$ and $\{n/2 + 1, n/2 + 2, \cdots, n\}$.
  5. Return $T_1 \cup T_2 \cup \{e\}$.
```

Solution:

This algorithm does not work; multiple edges of the MST could cross this particular cut. Another way to see this is that the MSTs of the subgraph needn't also be part of the MST of the whole graph. As a concrete counterexample, consider a wide rectangle and the horizontal cut between the top two vertices and the bottom two. Both edges on this cut should be in the MST. It is also possible that when you divide the graph it is not possible to construct a MST of the subsection of the graph.