CMSC 27200 - Winter 2021 — Andrew Drucker, Lorenzo Orecchia

due friday february 5 at 12 noon via gradescope

# H3.BRIANMCMANUS

- The assignment is due at Gradescope on Friday February 5 at 12 noon.
- You can either type your homework using LaTex or scan your handwritten work. We will provide a LaTex template for each homework. If you writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will facilitate the grading.
- You are permitted to discuss the problems with up to 2 other students in the class (per problem); however, you must write up your own solutions, in your own words. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please do list all your collaborators in your submission for each problem.
- Similarly, please list any other source you have used for each problem, including other textbooks or websites.
- Show your work. Answers without justification will be given little credit.

PROBLEM 1 (25 POINTS) *Give a polynomial-time algorithm that solves the following problem. Give a clear proof that your algorithm is correct and runs in polynomial time.* ***Input:*** *a simple (no repeated edges or self-loops), connected, undirected graph* $G = (V, E)$ *with a positive edge-weight function* $w : E \to \mathbb{N}^+$*, with all edge weights distinct;****Output:*** *a spanning tree* $T \subseteq E$ *of* $G$*, of second-smallest total weight.More precisely:* $T$ *should have minimum weight among all spanning trees of* $G$ *that are not Minimum Spanning Trees for* $(G, w)$*. If there are two or more such trees, you are free to output any one of them, and you don't need to decide whethersuch a tie can or does occur.*

**Solution:**

Steps for the Algorithm:

1. find a MST using Kruskal in $O(E^2)$
2. for each edge $e$ not already in the MST, temporarily add it to the the MST, creating a cycle
3. find the edge $k$ with maximal weight in the cycle that is not equal to $e$
4. remove $e$ temporarily, creating a new spanning tree
5. compute the weight difference $w(e) - w(k)$, and remember it together with the changed edge
6. repeat step 2 for all other edges, and return the spanning tree with the smallest weight difference the the MST

For correctness the second best minimum spanning tree differs from the MST by one edge, so we need to find an edge which is not in the MST, and replace it with an edge in the MST such that the new graph is a spanning tree and the weight difference is minimal.

The time complexity for this algorithm depend is dominated by Kruskal, and so is $O(E^2)$

PROBLEM 2 (25 POINTS) *Solve exercise 19 in Chapter 4 (bottleneck rates) in the Kleinberg-Tardos textbook.*

**Solution:**
Setting the edge cost equal to the negative of bandwidth is enough to compute a minimum spanning tree as required. Simply use Kruskal or Prim in order to generate a minimum spanning tree under these conditions.

The correctness of the claim is proven by contradiction, i.e. there is some pair of vertices $u, v$ for which the path $P$ in the minimum spanning tree does not have a bottleneck rate as high as some other $u, v$ path $P'$. Let $e = (x, y)$ be an edge of minimum bandwidth on the path $P$. $e$ has the smallest bandwidth of any edge in $P$ and $P'$ by construction. Now, using the edges in $P$ and $P'$ other than $e$ it is still possible to travel from $x$ to $y$, and so there is a cycle $C$ in which $e$ has the minimum bandwidth. This means that in out minimum spanning tree instance, $e$ has the maximum cost on the cycle $C$, but $e$ belongs to the minimum spanning tree, contradicting the cycle property.

For run time analysis Kruskal is $O(n^2)$ from class, and there are no other considerations.

PROBLEM 3 (25 POINTS) *Let $G(V, E)$ be an undirected and **unweighted** graph with $n$ nodes. Let $T_1, T_2, \ldots, T_k$ be $k = n - 1$ distinct spanning trees of $G$. Devise a polynomial-time algorithm that finds a spanning tree $T = (V, E_T)$ in $G$ that contains at least one edge from each $T_i$. (Prove correctness and polynomial runtime.)Definition: two trees (or for that matter any graphs) on a set of nodes are **distinct**, if they differ in at least one edge.*

**Solution:**

Algorithm:

- enumerate the nodes from 1 to $k$ in any order
- create a list of nodes that are in the tree $N$ that contains the starting node $n_1$
- create a list of nodes that are not in the tree $N'$
- create a tree that initially only has $n_1$
- //I will add edges to the tree one by one in the following fashion:

for each spanning tree $T_i$:
  for each edge in $T_i$:
    check if the edge has one vertex $n \in N$ and the other vertex $n' \in N'$
    we take the first such edge, add it to the tree, remove the end of the edge from $N'$ and add it to $N$

Proof of correctness:
Firstly there must be an edge $e \in T_i$ that connects some $n \in N$ to some $n' \in N'$. This is because $N$ and $N'$ form a partition of $G$, and a spanning tree must connect any two points $a \in N, b \in N'$ through some path $P$, and that path must have an edge that has a vertex in $N$ and the other in $N$. Also, by design, there can never be cycles, as this algorithm only creates edges that have at least one node that has no other nodes attached to it, which means that at every step the tree is maintained. At the end of the process we have a tree with $n - 1$ nodes which is necessarily a spanning tree, and we have a edge from each $T_k$, as for each $T_k$ we chose an edge from $T_k$ for the tree.

Proof of complexity:
The complexity of the main loop dominates to algorithm. We have $k$ iterations for the $T_k$, and then when checking edges it could take $k$ tries to match one potential edge with every member of $N$, and there could be up to $k$ edges in $T_i$ that need to be checked. This gives $O(k^3)$, which dominates the creation and maintenance of the lists and tree which is definitely less than $O(k^2)$.

PROBLEM 4 (25 POINTS) *Let $G = (V, E, \{w_e\}_{e \in E})$ be an undirected graph with positive edge weights $w_e$ indicating the lengths of the edges. Devise a polynomial-time algorithm that,given a vertex $i \in V$, computes the length of the shortest cycle containing vertex $i$ in $G$. Give a proof of correctness and a running-time analysis for your algorithm. For full credit, your algorithm should run in time $O(|V|^2)$.*

**Solution:** The idea is to run Dijkstra's algorithm except that we initialize the set of explored vertices to be empty.
For clarity I use $W$ for the weight of a vertex and $S$ to be the set of explored vertices.

Initialize $S = \{i\}$ and $W(i) = 0$. We choose $v \notin S$ which minimizes $\Omega(v) = \min_{e=(u,v):u \in S} W(u) + w(e)$. Add $v$ to $S$ and set $W(v) = \Omega(v)$. After the first iteration remove $i$ from $S$. This will allow us to find the shortest cycle. Repeat until $i \in S$, all vertices have been traversed, or the algorithm terminates, in which case there is no cycle. And just like the argument from class the proof of correctness relies on the fact that for each vertex $u \in S, W(u)$ is the weight of the lightest $s \mapsto u$ path. This algorithm is almost identical to Dijkstra's so the complexity is naturally $O(|V|^2)$ using the array implementation.