CMSC 27200 - Winter 2021 — Andrew Drucker, Lorenzo Orecchia

due tuesday, february 2 at 12 noon via gradescope

H3.Hamlet

- The assignment is due at Gradescope on Tuesday, February 2 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 whether such a tie can or does occur.*

Solution: Collaborated with: Yael Sulkin

Let $K = \emptyset$, Let S be a single pair of $(P_i, FindMin())$ where P_i is a partition and FindMin() searches for the difference of a given edge and the second greatest edge in the same partition.
Assume $w(e_1) < w(e_2) < \ldots < w(e_m)$
Assume $i \geq 2$.
if $K \cup e_i$ is cycle-free **then**
    $K \leftarrow K \cup e_i$
else
    if $FindMin(e_i) < S.FindMin()$ **then**
        $S \leftarrow (P_{e_i}, FindMin(e_i))$ = the minimum difference of edge e_i and some edge e_j in the same P_i.
    end if
end if
swap edge e_j with e_i in K
return $T \leftarrow K$

Proof of polynomial time The algorithm for this proof is Kruskal's algorithm which we know runs on polynomial time. Added to Kruskal's algorithm is another query for the usual Union-Find abstract data type where we search for the minimum difference between a given edge and the edges in the current partition of the algorithm. However, looping over edges of Partitions of edges of size ¡ —E— keeps our algorithm on polynomial time.

Proof of correctness We want to show that our algorithm produces a spanning tree of second smallest weight. We will do this by proving that our algorithm produces a spanning tree of minimal weight and that with for any possible edge swaps of edges e, e' where e is in K and e' is not, any reduction in weight greater than o results in the MST. Since we know all edge weights are unique, the Minimum Spanning Tree is also unique.

Feasibility For our proof of feasibility we want to show that our algorithm produces a spanning tree. Since Kruskal's algorithm produces a spanning tree of no cycles by design, we can assume that after our run we receive a spanning tree. However, we want to show that a swap of edges afterwards also results in a spanning tree. We will show this using the contradpositive: if after the swap there was no spanning tree then before the swap there was no spanning tree. Let our algorithm produce tree K. Assume after some swap of edges e connecting vertices w, x and e' connecting vertices y, z left vertex x without a connecting edge. Let e be in some partition P. Then if we were to add e' to P, we know that it would not create a cycle in containing vertex x because if removing edge e disconnecting x, then edge e was the only edge containing x. However, P no longer contains vertex x and so adding e' to P does not help merge x into P. Thus, our algorithm always results in a spanning tree.

Optimality For our proof of optimality we want to show that our algorithm produces a spanning tree of second smallest weight where any possible swap of edges that reduces the weight our tree automatically results in the distinct Minimum Spanning Tree. We will prove this using an exchange argument. First we want to define our solutions. Let the spanning tree our algorithm produces be tree K and the spanning tree of some other optimal algorithm be tree O. Then we want to show that for some edge in O, e_o with weight w_{e_o} that is greater than some edge e_k, in K such that : $w_{e_o} > w_{e_k}$ that once swapped, the total weight of O becomes smaller. However, we want to show that in doing so it does not result in the Minimum Spanning Tree. Suppose there existed some edge e_m in the distinct MST where if swapped with e_o, O becomes the minimum spanning tree. Then we know that $w_{e_o} > w_{e_m}$. However, we also know that the weight of edge e_k is smaller than the weight of e_o, so we can swap the edges to get closer to an optimal solution. However, since the Minimum Spanning Tree is distinct $e_k \neq e_m$. Thus, we know $w_{e_o} > w_{e_k} > w_{e_m}$ where e_k is squeezed in between any edge in O and any edge in the MST. This shows that there does not exist an edge where a swap could be possible, reduce the weight of the tree produced by algorithm and still not be the MST.

Extra Space for your solution

Extra Space for your solution

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

Solution: Collaborated with: Yael Sulkin

S is the set of explored vertices in V
Assume P is the set of edges in our working Spanning Tree
choose a source node, s_0, $S \leftarrow s_o$
Let function bandwidth be the bandwidth b_e of a given edge (v, u)
while do$S \neq V$
    choose $u \in V - S$ of maximum AttachmentCost(u)
    let $e = (v, u)$ be maximizing edges
    $P \leftarrow P \cup e, S \leftarrow S \cup u$
end while
return $T \leftarrow P$

Proof of Runtime Our algorithm works off of Prim's algorithm and the priority queue abstract data structure. Since we only changed the measurement we iterate over and nothing else of the design of Prim's algorithm we know that our runtime is equal to $O(|V|^2)$ according to Prim's Worst Time Complexity in the Big-O complexity Time chart cheat sheet.

Proof of correctness We want to show that for our algorithm it returns 1) a spanning tree and 2) a spanning tree with the best-acheivable bottle neck rate for any pair of vertices (v,u) in our result. We will show this inductively using a greedy stays ahead argument, considering Prim's algorithm is a greedy algorithm.

feasibility

correctness With greedy stays ahead arguments we first want to define our solutions. Let the solution our algorithm produces be K and the result of some other solution be O. For some edge in K we will note e_k and for some edge in O we will note e_o. Next, we want to define our measurement where we will be inductively iterating upon. Since our goal is to produce the best achievable bottleneck rate between any two vertices (u, v) we want to make sure that at every iteration our algorithm is producing a best-achievable bottle neck rate as great as the optimal solution. That is, $b(P)_k \geq b(P)_o$.

Your solution goes here.

Extra Space for your solution

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: Collaborated with: Yael Sulkin

S is the set of explored vertices in V
Assume P is the set of edges in our working Spanning Tree
choose a source node, s_0, $S \leftarrow s_o$
Let $K = T_1, T_2, T_3, \ldots T_n - 1$
Let function $SharedTrees(K, v, u)$ count number of Trees in K that contain edge (v, u)

while K is not empty **do**
    choose $u \in V - S$ of maximum $SharedTrees(K, v, u)$
    let $e = (v, u)$ be maximal shared tree count
    Delete all trees containing edge e in K
    $P \leftarrow P \cup e, S \leftarrow S \cup u$
end while
return $T \leftarrow P$

Proof of Runtime

Proof of correctness

feasibility

Extra Space for your solution

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: Collaborated with: Yael Sulkin

Dijkstra's algorithm (G, l)
S be the set of explored nodes
for each $u \in S$, we store distance $d(u)$ **do**
Inititally $S = i$ and $d(s) = 0$
while $S \neq V$ **do**
Select a node $v \notin S$ with at least one edge from S for which $d'(v) = min_{e=(u,v):u \in S} d(u) + l_e$ is as small as possible Add v to S and define $d(v) = d'(v)$
end while
end for

Proof of Runtime

Proof of Correctness

Extra Space for your solution