CS 170 HW 11

Due **2020-04-13, at 9:59 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Opting for releasing your solutions

We are considering releasing a subset of homework submissions written by students for students to see what a full score submission looks like. If your homework solutions are well written, we may consider releasing your solution. If you wish that your solutions not be released, please respond to this question with a "No, do not release any submission to any problems". Otherwise, say "Yes, you may release any of my submissions to any problems".

3 Project Pregame

Please refer to the project spec to answer these questions.

(a) Respond to the following logistics questions.

(i) How many members from each group should fill out the Phase 0 group sign-up form?

(ii) Under what circumstances are you allowed to make changes to your group after April 12th?

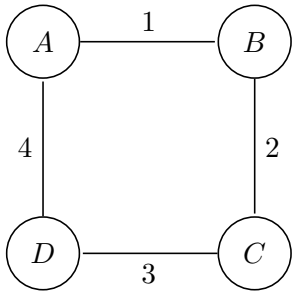
(iii) When are the Phase 1 and Phase 2 due dates?

(iv) What are the deliverables in Phase 1?

(v) What are the deliverables in Phase 2?

(b) If the input graph is **complete**, what is the cost (average pairwise distance between all pairs of vertices) of the optimal cell tower network?

(c) What is the cost of the optimal network for the following graph?



(d) Give an example of a graph for which the **optimal** network T has vertex $v \in T$ whose removal keeps the resulting sub-graph a **valid** network.

(e) Let T be a line graph on three vertices with both edge weights equal to 1. What is the cost of T?

(f) Let T be a line graph on $n \geq 2$ vertices with all edge weights equal to 1. Prove that the cost of T is $\frac{n+1}{3}$.
(Hint: induction!)

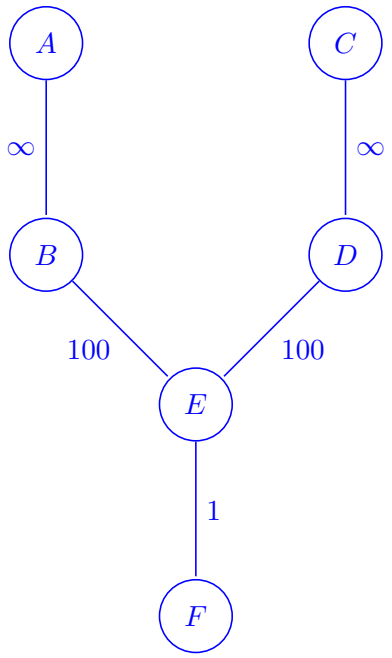
Solution:

(a) (i) Only one member should fill out the Phase 0 group sign-up form.
(ii) Under no circumstances can you change your group after April 12th.
(iii) Phase 1 is due April 19th. Phase 2 is due May 3rd.
(iv) 3 graphs, of up to 25, 50, and 100 vertices, as 3 different files to the Gradescope autograder.
(v) Outputs for the pool of inputs generated from the class to the Gradescope autograder, and the project reflection.

(b) 0. Any vertex suffices as the optimal tree.

(c) 1 (T is just the vertices A and B)

(d) The solution is illustrated in the following graph. T has a lower cost if it includes $\{B, E, D, F\}$, instead of just $\{B, E, D\}$.



(e) The outer nodes are 2 distance away from each other, all other pairs are 1 distance away, so the answer is $\frac{4}{3}$. Computing all of the pairs of distances gives $\text{AVG}(2,1,1) = \frac{4}{3}$.

(f) Let $f(n)$ be the average pairwise distance of a line graph on $n \geq 2$ vertices. We will prove that $f(n) = \frac{n+1}{3}$ by induction.

Base case: It is clear that $f(2) = 1$. We may also verify $f(3)$ by the previous question.

Assume that $f(k) = \frac{k+1}{3}$ for some $k \geq 2$. When we extend a line graph on k vertices by one vertex, we increase the total pairwise distance by $1 + 2 + ... + k = \frac{k(k+1)}{2} = \binom{k+1}{2}$. Thus,

$$f(k+1) = \frac{\binom{k}{2} f(k) + \binom{k+1}{2}}{\binom{k+1}{2}} = \frac{\frac{k(k-1)}{2}\frac{k+1}{3} + \binom{k+1}{2}}{\binom{k+1}{2}} = \frac{\binom{k+1}{2}\frac{k-1}{3} + \binom{k+1}{2}}{\binom{k+1}{2}}$$
$$= \frac{k-1}{3} + 1 = \frac{(k+1)+1}{3}$$

4 Online Matching

Consider the maximum bipartite matching problem on a unweighted, undirected bipartite graph $G = (L, R, E)$, where L and R denote the left and right side vertices.
In class, we saw that the problem can be solved via max flow when the graph is given to you. However, consider the following harder, *online* setting. In the online version, L is known, but the vertices in R arrive one at a time. When a vertex $u \in R$ arrives, its incident edges are also revealed, and at this moment, we must make a immediate and irrevocable decision to match u to one of its available neighbors in L or not to match it at all. The decision is irrevocable in the sense that once a vertex is matched or left unmatched, it cannot be later rematched (to another vertex). The goal is to maximize the matching size.

(a) Consider the deterministic greedy algorithm, where $u \in R$ is matched to an arbitrary available neighbor. Let OPT denote the maximum possible matching size in the offline setting (*i.e.*, when the graph is fully known). Show that the greedy online algorithm obtains a matching of size at least half of OPT.
Hint: Revisit Problem 4b of Discussion 5.

(b) Show that no deterministic online algorithm can achieve strictly better than half of OPT on all possible inputs.
Hint: Construct two inputs, where if the algorithm does better than half of OPT in one of them, it must achieve at most half of OPT in another. Use the fact that the algorithm cannot see future. Keep the construction simple—four vertices are enough!

(c) *Insanely hard challenge yet worth no point*: Show that the following randomized algorithm achieves $1 - 1/e$ fraction of OPT on any input. Give a random ordering of the vertices in L. When $u \in R$ arrives, if u has an unmatched neighbor in L, then we choose the unmatched neighbor $v \in L$ that comes earliest in the random ordering. (Otherwise, leave it unmatched.)

Solution:

(a) Note that if there exists some $u \in L, v \in R$ in the end that are connected yet both unmatched, the online greedy algorithm would have matched them. Then the argument for the second part of problem 4b, discussion 5 applies: any edge in the optimal matching shares an endpoint with one of the edges in the algorithm's solution (otherwise, as we argued, the greedy algorithm would have added it). Suppose the algorithm produces a matching of size k. Each edge has two endpoints, so the optimal solution can have at most $2k$ edges.

(b) Suppose $L = \{i_1, i_2\}$ and $R = \{j_1, j_2\}$. Consider two possible inputs. In both of them, vertex j_1 arrives first and reveals that it is connected to both i_1 and i_2. Then vertex j_2 arrives and reveals that it has only one neighbor: in Input 1 this neighbor is i_1; in Input 2 it is i_2.

Notice that the maximum matching has size 2 in both of these inputs: j_2 can be matched to its only neighbor, whereas j_1 can be matched to the remaining element of L. Also notice that in both cases, this is the unique matching of size 2. Therefore, an online algorithm that seeks to select the maximum matching faces a predicament: first it must match j_1 to one of its neighbors, there is a unique choice that is consistent with picking the maximum matching, and there is no way to know which choice this is until later j_2 is revealed. Formally, on Input 1, if the algorithm makes the correct choice to match j_1 with i_2 and goes on to match j_2 with i_1, then it has to mess up on Input 2; that is, matching j_1 with i_2 necessarily leads to a matching of size 1 on Input 2. On the other hand, if the algorithm does the opposite, then it is just going to mess up on Input 1.

Hence, we conclude that for any deterministic online algorithm, we can find an input that causes the algorithm to select a matching of size at most 1, while the maximum matching has size 2.

(c) The algorithm is known as RANKING, first proposed by Karp, Vazirani and Vazirani in 1990; in fact, two of them were and still are Berkeley professors. The initial proof is very complicated.

Over the years, the analysis has been simplified a lot, but this is still way beyond the scope of this course. We refer you to `https://arxiv.org/abs/1804.06637` or `https://www.cs.cornell.edu/~rdk/papers/RPDFinal.pdf`.

5 Global Mincut to Min s-t Cut

In class, we showed how to use maximum s-t flow to solve the minimum s-t cut. Now we ask you to consider the *global mincut* problem. Given a weighted, undirected graph G, the problem asks you to find a minimum weight set of edges whose removal disconnects the graph. (Note that there is no notion of s and t).
Show how to use maxflow or min s-t cut algorithms to solve this problem efficiently. Prove that your reduction would lead to the optimal solution. How many times do you need to invoke the maxflow or min s-t cut subroutine?

Solution: Fix an arbitrary vertex s. Compute minimum s-t cut for all $t \neq s$. Output the minimum among all these s-t mincuts. We need to invoke the min s-t cut subroutine $n-1$ times.
The reduction works since for any choice of s, any graph cut would separate s from some other vertex $t \neq s$ (simply take any t on the other side of the cut). In particular, if the global mincut separates s from t^*, then it is the minimum s-t^* cut (otherwise, there would be a even better global mincut). Our reduction loops over all choices of $t \neq t$, which include t^*.

6 Dijsktra's Sort

Show how to use Dijsktra's algorithm to sort n real numbers (not necessarily non-negative or integral) in ascending order in $O(n \log n)$ time. You may use the intermediate outputs of Dijsktra's to do the sorting (as opposed to using it as a blackbox).
Argue that this means that improving upon the $O(m + n \log n)$ run-time of Dijsktra's algorithm (with Fibonacci heap) would lead to a faster sorting algorithm than merge and quick sort.
Hint: Given the numbers, construct a graph such that the order of vertices being extracted from priority queue by Dijsktra's algorithm corresponds to the right sorting order.

Solution: Consider the star graph with one center vertex s connected to n other vertices. Let the input numbers be $\{t_1, \cdots, t_n\}$. Let edge (s, i) have weight t_i. We consider each non-center vertex i as corresponding to the number t_i We claim that the order of vertices being extracted from the priority queue by the Dijsktra's algorithm is precisely the ascending order of the inputs. In the first iteration, s is extracted, and we can ignore that, since it doesn't correspond to any input number. Observe that then the Dijsktra's algorithm updates the vertex labels `dist` to be exactly the input numebers. Hence, the next iteration will extract exactly the vertex corresponding to the smallest number t_i. But since a non-center vertex is not adajacent to anything except the center, we do not update `dist`, so we move on. Then the algorithm proceeds to extract the vertex corresponding to the second smallest number t_i in the next iteration, and so on.
If there's any data structure that enables faster run-time for Dijsktra's, then we can always apply the reduction above to sort numbers faster. Essentially, we are just using the fact that Dijsktra's uses priority queue and priority queue can be used for sorting.

7 Path TSP and Cycle TSP

In the Traveling Salesman Problem (TSP), we are given an undirected graph with non-negative weights and asked to find a minimum weight cycle that contains each vertex exactly once.
In the s-t Traveling Salesman Problem (s-t TSP), we are given an undirected graph with non-negative weights, two vertices s and t, and are asked to find a minimum weight path that starts at s, visits all other vertices exactly once, and ends at t.
Show that if there is an algorithm to solve TSP, then you can use it to solve s-t TSP.

Solution: Let (G, s, t) be an instance of s-t TSP. Add an artificial vertex q to the vertex set. Connect it with s and t only, with edges of weight 0. Solve TSP on the new graph G'.

The optimal TSP must contain the two edges $(q, s), (q, t)$, since these are the only two edges that can visit and leave q. Observe that removing these two edges from the solution recovers the optimal s-t TSP solution on G, since any better s-t TSP solution in G would immediately correspond to a better solution of TSP on G.