28 (100 PTS.) Network deployment.

Consider an undirected connected graph G that represents a map. Every vertex is a location, and every edge represents a road that connects the two locations, with the positive weight of the edge being the distance between the two locations. Let $d(x, y)$ denote the shortest path distance between any two vertices $x, y \in \mathsf{V}(\mathsf{G})$. We have a set $X \subseteq \mathsf{V}(\mathsf{G})$ of locations we would like to connect together via a common *connected* network. Computing the optimal such network is surprisingly difficult, but one can build a reasonably good network.

A natural strategy for deploying/building such a network is as follows – order the vertices of X in some order $v_1, v_2, \ldots, v_t$, where $t = |X|$. In the ith step of the deployment, we install a server at v_i. Let u_i be the closest server among $v_1, \ldots, v_{i-1}$ to v_i. We connect v_i to u_i (paying $d(v_i, u_i)$), and continue to the next location. The ***cost*** of the deployment is $\sum_{i=2}^{n} d(v_i, u_i)$.

28.A. (20 PTS.) Consider the graph $\mathsf{G} = (\{1, 2, \ldots, n\}, \{12, 23, \ldots (n-1)n\})$, where the weight of all edges is 1. Show, that there is an ordering over the vertices of G such that the deployment cost of $X = \mathsf{V}(\mathsf{G})$ is $\Omega(n \log n)$.

Solution:

Always pick the number furthest away from all the numbers inserted so far, starting with 1. As such, the second number inserted would be n, and third one would be $n/2$, and so on. If there are several candidates, pick one of them arbitrarily. Formally, if we had picked the set $X_{i-1} = \{x_1, \ldots, x_{i-1}\}$ so far, then the ith number is

$$x_i = \arg \max_{y \in [\![n]\!] \setminus X_{i-1}} \min_{z \in X_{i-1}} |y - z|,$$

where $[\![n]\!] = \{1, 2, \ldots, n\}$ and $x_1 = 1$.

Observe, that $d(x_i, X_{i-1}) = \min_{z \in X_{i-1}} |x_i - z| \geq (n-i)/2i$. Indeed, inserting $i-1$ numbers, breaks the set $[\![n]\!]$ into i intervals, the longest of them must be of length $\geq (n-i)/i$. Putting the next point in the middle of this interval results in the stated bound (this analysis is somewhat conservative). As such, we have that

$$\sum_{i=2}^{n} d(x_i, X_{i-1}) \geq \sum_{i=2}^{n} \frac{n-i}{2i} \geq \frac{n}{2} \sum_{i=2}^{\lfloor n/2 \rfloor} \frac{1}{i} \geq \frac{n}{2}(\ln n - 2) = \Omega(n \log n).$$

28.B. (20 PTS.) Let T be the cheapest tree, such that $X \subseteq \mathsf{V}(\mathsf{T})$. Verify that the cost of the tree $w(\mathsf{T}) = \sum_{e \in \mathsf{T}} w(e)$ is a lower bound on the deployment cost of X for any ordering.

Prove that there exists a closed walk that visits all the vertices of X, and the total weight of the edges of the walk is at most $2w(\mathsf{T})$.

Solution:

Let $v_1, \ldots, v_t$ be the minimum cost permutation, and let π_i be the shortest path between v_i and any vertex in $\{v_1, \ldots, v_{i-1}\}$. Let ℓ_i be the length of π_i. Clearly, the union $\bigcup_i \pi_i$ is a connected subgraph H and connects all the vertices of X. Clearly T is the minimum cost graph that has this property. We thus have that

$$\mathrm{cost}(\mathsf{T}) \leq \mathrm{cost}(\mathsf{J}) \leq \sum_i \ell_i,$$

thus completing the verification part.

Do a **DFS** of T, and consider how the **DFS** traverses it. It uses an edge to go down to traverse a subtree, and it (conceptually) uses the same edge on the way back when returning from the recursive call. As such, the **DFS** yields a walk C around T which can be interpreted as a closed walk. Every edge of T is used twice by this walk. Thus, we have that $w(C) \leq 2w(\mathsf{T})$.

28.C. (20 PTS.) Let x, y be the closest pair of vertices in X. Formally, it is one of the pairs realizing $\min_{u\in X} \min_{v\in X\setminus\{u\}} d(u, v)$. Prove, using the above, that $d(x, y) \leq 2w(\mathsf{T})/|X|$,

Solution:

Let C be the above cycle, and observe that the vertices of X break this cycle into at least i intervals (in fact, it might be more, since a vertex of degree d in T is going to be visited d times by C). An interval is boring if its two endpoints are the same. An interval is interesting if its two endpoints are two different elements of X. Clearly, there are at least k interesting intervals, and let $I_1 = [x_1, y_1], \ldots, I_k = [x_k, y_k]$ be k of these intervals, where $k = |X|$. We have that $d(x_i, y_i) \leq w(I_i)$. As such, we have that

$$d(x, y) = \min_{u\in X} \min_{v\in X\setminus\{u\}} d(u, v) \leq \min_i d(x_i, y_i) \leq \min w(I_i) \leq \frac{w(C)}{k} \leq \frac{2w(\mathsf{T})}{k}.$$

28.D. (40 PTS.) Consider the greedy algorithm that computes the closest pair of vertices $x, y \in X$, sets $v_t = x$, and then recursively computes the ordering for $X \setminus \{x\}$. Let Π be the resulting ordering of X. Prove that the deployment cost of X is bounded by $O(w(\mathsf{T}) \log n)$.

(Clearly, this algorithm can be implemented in polynomial time – but the exact running time here is unimportant.)

Solution:

Let $v_1, \ldots, v_t$ be the computed permutation Π, and let π_i be the shortest path between v_i and any vertex in $\{v_1, \ldots, v_{i-1}\}$. Let ℓ_i be the length of π_i, and let $\mathsf{H} = \bigcup_i \pi_i$. Using the above, we have

$$\mathrm{cost}(\mathsf{T}) \leq \mathrm{cost}(\mathsf{H}) \leq \sum_{i=2}^{t} \mathrm{cost}(\pi_i) \leq \sum_{i=2}^{t} \ell_i \leq \sum_{i=2}^{t} \frac{2w(\mathsf{T})}{i} \leq 2w(\mathsf{T}) \sum_{i=2}^{t} \frac{1}{i} \leq 2(\ln t + 1) w(\mathsf{T}).$$

29 (100 PTS.) Few spies.

Let $\mathsf{G} = (C \cup S, \mathsf{E})$ be a graph with n vertices and m edges. The vertices of $C \cup S$ represents citizens in the glorious democratic republic of north Narnia (GDRNN). An edge between two people indicates that they are friends. The vertices of S represents people that are willing to spy on their friends to see if they do any illegal activities (like enjoying themselves, etc). The government of GDRNN would like to choose a set of spies $S' \subseteq S$, of minimum size, such that every citizen in C is connected to some vertex of S' (the members of S are trusted by the government – so no need to spy on them). Computing the smallest such set is surprisingly difficult. Again, we are going to be happy with a simple greedy strategy[1].

Let $C_0 = C$ and $S_0 = \emptyset$. In the ith iteration, for $i > 0$, we pick the vertex s_i in S that is connected to the largest number of citizens in C_{i-1} (resolving ties arbitrarily). We then set $S_i = S_{i-1} \cup \{s_i\}$, and $C_i = C_{i-1} \setminus \Gamma(s_i)$, where $\Gamma(s_i)$ is the set of citizens connected to s_i. We stop as soon as C_i is empty, and output S_i as the desired set of spies.

29.A. (20 pts.) Assume that the optimal solution is of size k. Prove, that for any i, s_i is connected to at least $|C_{i-1}|/k$ citizens in C_{i-1} (hint: think about the k optimal spies $o_1, \ldots, o_k$).

Solution:

Observe that all the vertices of C_{i-1} are connected to the optimal k spies $o_1, \ldots, o_k$. In particular, at least one of these optimal spies is connected to at least $|C_{i-1}|/k$ candidates. Otherwise, these k optimal spies are connected to $< (|C_{i-1}|/k)k = |C_{i-1}|$, which would imply that there are citizens in C_{i-1} that are not spied on by the optimal solution, which is a contradiction.

29.B. (20 pts.) Prove that $|C_i| \leq (1 - 1/k)|C_{i-1}|$.

Solution:

Let D_i be the elements of C_{i-1} connected to the ith spy s_i. We have that

$$|C_i| = |C_{i-1} \setminus D_i| = |C_{i-1}| - |D_i| \leq |C_{i-1}| - |C_{i-1}|/k = (1 - 1/k)|C_{i-1}|,$$

using the above, which implies that $|D_i| \geq |C_{i-1}|/k$.

29.C. (20 pts.) Using that $(1 - 1/k)^k \leq 1/e$, prove that for all i, we have that $|C_{i+k}| \leq |C_i|/e$.

Solution:

Using the above, we have

$$|C_{i+k}| \leq (1 - 1/k)|C_{i+k-1}| \leq (1 - 1/k)^2 |C_{i+k-2}| \leq \cdots \leq (1 - 1/k)^k |C_{i+k-k}| \leq \frac{1}{e}|C_i|.$$

29.D. (40 pts.) Prove, that the above algorithm outputs a set of at most $k(\lceil \ln n \rceil + 1)$ spies, such that all the citizens of C are spied on by these spies.

[1]Seeing the movie "The lives of others" might not help you solve this problem, but you might enjoy it anyway.

Solution:

Let $C_0 = C$, and $m = k(\lceil \ln n \rceil + 1)$. By the above, for any $i = 0, \ldots, m/k$, we have that

$$|C_m| \leq |C_{m-ik}| \leq \frac{|C_m|}{\exp(i)} \leq \frac{n}{\exp(i)}.$$

Setting $i = m/k$, we have

$$|C_m| \leq \frac{n}{\exp(m/k)} \leq \frac{n}{\exp(\lceil \ln n \rceil + 1)} < 1.$$

This implies that $|C_m| = 0$, which implies that the algorithm terminated before picking m spies, thus implying the claim.

30 (100 PTS.) Undecidable, that's what you are.

For each of the following languages, either prove that it is undecidable (by providing a detailed reduction from a known undecidable language), or describe an algorithm that decides this language – your description of the algorithm should be detailed and self contained. (Note, that you cannot use Rice Theorem in solving this problem.)

30.A. (25 PTS.) $L_1 = \{\langle M, N \rangle \mid L(M) = \overline{L(N)}$, where M is a Turing machine, and N is a NFA$\}$.

Solution:

This language is undecidable. The reduction is from HALTING. So, let $\langle M, w \rangle$ be an input to halting, and the question is whether M stops on w. We construction a new machine M_w, which simulates M on w, and if it halts, then it stops and accepts whatever the real input is. As such, the language of $L(M_w) = \Sigma^*$ if M halts on w, and $L(M_w) = \emptyset$ otherwise. Now, we are ready to the kill - let N be an NFA that accepts no string (i.e., $L(N) = \emptyset$). As such, we have $\overline{L(N)} = \Sigma^*$. Clearly, $\langle M_w, N \rangle$ is in the language $L_1 \iff M$ halts on w. As such, if L_2 was decidable then HALTING was decidable, but we know that HALTING is not decidable. A contradiction.

30.B. (25 PTS.) $L_2 = \{\langle N \rangle \mid L(N)$ is infinite, where N is an NFA$\}$.

Solution:

This is decidable. Convert N into a DFA D. We now have to decide if the language of D is infinite. To this end, compute the strong connected components of D. A strong connected component is non-trivial if it contains more than one state. From the start state, compute all the non-trivial strong connected components, using DFA. Let U be the set of all the vertices in the non-trivial SCCs that are reachable from the start state. Similarly, reversing D, compute all the non-trivial strong connected components that can reach an accepting state, and let V be the set of their states. Clearly, D has an infinite language, if and only if $U \cap V$ is not empty – indeed, in such a case, there is a walk in D start at start state, goes into a non-trivial SCC, where it can perform an arbitrarily long walk, and then it can walk to an accept state. Clearly, there are infinite number of such walks, and each such walk corresponds to accepting string for the original N.

30.C. (25 pts.) $L_3 = \{\langle R, N\rangle \mid L(R) = L(N)$, where R is a regular expression, and N is an NFA$\}$.

Solution:

This is decidable. Convert R into an NFA M. Now, convert N and M into two DFAs, say D_1 and D_2. We now need to decide if $L_1 = L_2$, where $L_1 = L(D_1)$ and $L_2 = L(D_2)$. This is equivalent to deciding if $L_1 \setminus L_2 \cup L_2 \setminus L_1$ is empty. But this can easily be done using product construction. Indeed, build the DFA D that is the product of D_1 and D_2, and set a state $(q_1.q_2)$ to be accepting $\iff$ $[q_1 \in A(D_1)] + [q_2 \in A(D_2)] = 1$, where $A(D_i)$ is the set of accepting states of D_i, and $[q_i \in A(D_i)]$ is one if $q_i \in A(D_i)$ and zero otherwise.

Deciding if the language of D is empty or not, is just equivalent to checking if any accepting state of D is reachable from the start position, which can easily be done using **DFS**, **BFS**, etc.

30.D. (25 pts.)

$$L_4 = \{\langle M\rangle \mid L(M) \text{ contains some word of even length, where } M \text{ is a Turing machine}\}.$$

Solution:

Same old, same old. The reduction is from Halting. So, let $\langle M, w\rangle$ be an input to halting, and the question is whether M stops on w. We construction a new machine M_w, which simulates M on w, and if it halts, then it stops and accepts whatever the real input is. As such, the language of $L(M_w) = \Sigma^*$ if M halts on w, and $L(M_w) = \emptyset$ otherwise. Namely, $\langle M_w\rangle \in L_4 \iff M$ stops on w.

Now, if L_4 was decidable then we could decide if $\langle N_w\rangle \in L_4$, which is equivalent to deciding if M stops on w, which is impossible, since halting is undecidable.