1. Computability, Complexity and Algorithms

Problem (Olympic training).

Two GeorgiaTech runners are training to compete in the next Olympic Games marathon competition. They need a route to train on. They got a map of Atlanta divided into N neighborhoods connected by $N-1$ roads in a way such that it is possible to go from any neighborhood to another. They noticed the map also shows M trails connecting some neighborhoods and decide to set a route to train using roads and trails.

(a) The runners find out that trails are nicer to exercise and decide to choose a running map that connects all neighborhoods and has the longest total distance over trails, without creating cycles. Design an algorithm that finds such optimal route, or report no if it does not exist.

Your input is a list L_1 of neighborhoods, a list L_2 of pairs of neighborhoods connected by a road, and a list L_3 of pairs of neighborhoods connected by a trail. Furthermore, for each trail a nonnegative number is given, representing its length. Make sure to justify the correctness of your design and state and analyze its running time.

(b) Now that the team knows the city, they agree to set a new route that does not visit a neighborhood nor travel a road or trail twice. The training route can start at any neighborhood, does not need to visit every neighborhood, but must end where it started. Also, for training purposes, they need to pick a route with an even total number of roads plus trails. The issue is that a rival team hears of the plan and set to block certain trails (roads cannot be blocked!) in a way that it is impossible to build a valid training route. There is a cost c_i to block trail i, for each $1 \leq i \leq M$. Design an algorithm that finds the smallest total cost needed to block some trails such that no training route exists satisfying the above requirements.

As input you get the same lists $L_i, i = 1, 2, 3$, from part **(a)**, and the cost $c_i > 0$ for blocking the i^{th} trail. *You also notice that every neighborhood has at most four roads/trails connecting to it.* Make sure to state and briefly analyze the running time of your design in terms of N and M.

Solution. (a) Consider the graph given by the N neighborhoods and the trails. Get a spanning forest F of this graph of maximal weight. This can be done by running a classical algorithm such as Kruskal's, or simply describing a procedure to keep track of the cycle free property, likely using Union-Find, making sure the heaviest edge/trails are added first. Add the roads back to the map and get a spanning tree containing F. Again, this can be done by either running Kruskal's algorithm, but initializing the spanning tree at F, or continuing the Union-Find procedure. The resulting tree is the desired route.

To see why this is correct, note that the maximum distance over trails is given by the forest F, and adding another trail will create a cycle. Also, since we find a spanning tree, the route will visit every neighborhood.

The running time is $O((M+N)\log N)$, as we need to check $N+M$ many roads and trails, and it is possible to maintain the datastructure performing each update in time $O(\log N)$.

* * *

(b) Use dynamic programming. Pick a neighborhood and consider the rooted tree given by the neighborhoods and the roads. Call this tree T. The question is equivalent to finding the maximum total cost of trails that can be added to this tree such that no cycle has even length.

Let $t_1, t_2, \ldots, t_M$ be the trails. We can start by deleting those t_i such that the cycle they create when added to T has even length. Note that, if two trails form odd cycles that share some tree-edges (roads) then their union forms an even-length cycle. So the problem reduces to finding the max cost of trails we can add while creating odd-length *disjoint* cycles. The states of our dynamic programming are indexed by subtrees of T. Each subtree can be rooted at the highest vertex (highest means closer to the root). To get a recursion among states, consider the two cases:

1. No cycle passes through o, the root of the subtree. In this case, we simply add the cases corresponding to the subtrees rooted at the children of o.
2. A cycle passes through o. We delete the tree-edges in such a cycle, and proceed to add the states corresponding to the resulting subtrees.

If we encode each tree by its root and the children we delete, we can see that the number of states is $O(N \times 2^4) = O(N)$. The number of transitions is bounded by M, which is the number of trails, and every transition takes $O(N)$, as a result of deleting the tree-edges in the cycle of every trail (this is equivalent to finding the lowest common ancestor). The running time is then $O(MN)$.

2. Theory of Linear Inequalities

Problem. Let $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ be a polytope and let V be the set of vertices of P. Let S be a subset of V with the property that, for every $v \in V \setminus S$, the set S contains all the vertices adjacent to v. Recall, two vertices are adjacent if the segment joining them is an edge (face of dimension 1) of P. Consider the following algorithm.

1. Compute a vertex v of P minimizing $\{c^T x : x \in P\}$. If $v \in S$, return v.
2. Otherwise return a vertex v' of P minimizing $\{c^T x : x \text{ is adjacent to } v\}$.

Show the following for the algorithm.

1. (3 points) The algorithm can be implemented in polynomial time. (Here A, b is the input to the problem. $|V|$ and $|S|$ can be exponential in size of input. Moreover you are also given a polynomial time algorithm to check if $x \in conv(S)$ for any x.)

2. (5 points) Show that the algorithm solves the problem $\min\{c^T x : x \in S\}$.

3. (2 points) Give a polynomial algorithm to optimize a linear function over the set of 0,1 vectors in $\mathbb{R}^n$ with an even number of 1's.

Solution.

1. Clearly, the first step of the algorithm can be implemented using any polynomial time algorithm for linear programming (say Ellipsoid algorithm). For the second step, we can use one iteration of the simplex algorithm to find the best neighboring vertex, or even be more profligate and check for all possible neighboring bases (at most n^2 of such bases) and pick the best one.

2. If the algorithm returns $v \in S$ in the first step, it minimizes $c^T x$ over all $x \in P \supseteq conv(S)$ and therefore it is the optimum solution. In the other case, suppose the optimum vertex v for $\min\{c^T x : x \in P\}$ is not in S. Then the algorithm returns the best neighboring vertex v' of v. Firstly, $v' \in S$ by the hypothesis of the problem. Now suppose that the optimum vertex for $\min\{c^T x : x \in S\}$ is $\tilde{v}$ that is not a neighbor of v. Consider running the simplex algorithm starting from $\tilde{v}$ to solve $\min\{c^T x : x \in P\}$. The algorithm will improve in each step and end at v. In the last step of the algorithm it visits a neighbor of v that must be in S and must have objective at least as good as $\tilde{v}$. Thus there is a neighbor of v that is optimum to $\min\{c^T x : x \in S\}$ as required.

3. Let P be the convex hull of all $\{0,1\}$-vectors, i.e., $P = \{x \in R^n : 0 \leq x \leq 1\}$. Let S be the set of all vectors that have even number of 1's. Apply the above algorithm.

3. Graph Theory

Problem. Let G be a connected graph and assume that G admits a $\mathbb{Z}_2 \times \mathbb{Z}_2$-flow.

(1) Use a $\mathbb{Z}_2 \times \mathbb{Z}_2$-flow in G to construct a simple cycle cover of G, i.e., a collection $\mathcal{C}$ of cycles in G such that every edge of G appears in at least one but at most two cycles in $\mathcal{C}$.

(2) Show that the $\mathcal{C}$ in (1) may be chosen to satisfy $\sum_{C \in \mathcal{C}} |E(C)| \leq |E(G)| + |V(G)| - 1$.

Solution. Since we are dealing with $\mathbb{Z}_2 \times \mathbb{Z}_2$, we will ignore the orientations of edges. For each vertex $v \in V(G)$, let $E(v)$ denote the set edges of G incident with v.

For (1), let f be a $\mathbb{Z}_2 \times \mathbb{Z}_2$-flow in G. Let $E_1 = \{e \in E(G) : f(e) = (1,0) \text{ or } f(e) = (1,1)\}$ and $E_2 = \{e \in E(G) : f(e) = (0,1) \text{ or } f(e) = (1,1)\}$. Since f is a $\mathbb{Z}_2 \times \mathbb{Z}_2$-flow, $\sum_{e \in E(v)} f(e) = (0,0)$ for all $v \in V(G)$. Hence, $|E(v) \cap E_i| \equiv 0 \pmod 2$ for $i = 1, 2$ and $v \in V(G)$. Thus, $G_i := G[E_i]$, $i = 1, 2$, are even subgraphs of G. Now each G_i admits a cycle decomposition, say $\mathcal{C}_i$. Clearly, $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ gives the desired collection for (1).

For (2), we choose the flow f in (1) so that the set $S := \{e \in E(G) : f(e) = (1,1)\}$ is minimal. We claim that $G[S]$ is acyclic. For, suppose C is a cycle in $G[S]$. Then let $g(e) = f(e) - (0,1)$ for all $e \in e(C)$, and let $g(e) = f(e)$ for all $e \notin E(C)$. We see that g is also $\mathbb{Z}_2 \times \mathbb{Z}_2$-flow in G. This is a contradiction, as $\{e \in E(G) : g(e) = (1,1)\}$ is properly contained in S.

Define $E_1, E_2, G_1, G_2, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}$ as in (1). Then $E_1 \cap E_2 = S$. Since $G[E_1 \cap E_2]$ is acyclic, $|E_1 \cap E_2| \leq |V(G)| - 1$. We have $\sum_{C \in \mathcal{C}} |E(C)| \leq |E(G)| + |V(G)| - 1$.