CS 161 Winter 2020 Section 8

March 5-6, 2020

Mice to Holes

There are n mice and n holes along a line. Each hole can accommodate only 1 mouse. A mouse can stay at its position, move one step right from x to $x + 1$, or move one step left from x to $x - 1$. Any of these moves consumes 1 minute. Mice can move simultaneously. Assign mice to holes such that the time it takes for the last mouse to get to a hole is minimized, and return the amount of time it takes for that last mouse to get to its hole.

Example:
Mice positions: 4, -4, 2
Hole positions: 4, 0, 5
Best case: The last mouse gets to its hole in 4 minutes ($\{4 \to 4, -4 \to 0, 2 \to 5\}$ and $\{4 \to 5, -4 \to 0, 2 \to 4\}$ are both possible solutions)

Sort the mice locations and the hole locations. For $0 \le i < n$, have the ith mouse go to the ith hole. The maximum distance will be the max distance between each mouse and its corresponding hole.

Justification:

We'll first prove a lemma that will be useful in our induction proof later
Lemma: For $i_1 < i_2$ $j_1 < j_2$ and $dist(x, y) = |x - y|$.

$$\max(dist(i_1, j_1), dist(i_2, j_2)) \le \max(dist(i_1, j_2), dist(i_2, j_1))$$

Without loss of generality, let's say $i_1 \le j_1$. Our cases are then that $i_1 \le i_2 \le j_1 \le j_2$, $i_1 \le j_1 \le i_2 \le j_2$, or $i_1 \le j_1 \le j_2 \le i_2$. In any of these cases, the lemma holds.

Now, we proceed with induction.
Inductive hypothesis. By sending the i^{th} (sorted) mouse to the i^{th} (sorted) hole, there is a minimal solution that extends the current solution.
Base case. If we haven't sent any mice to any holes, we haven't eliminated the ideal solution.
Inductive Step. Suppose that we have sent the first $k - 1$ sorted mice to the first $k - 1$ sorted holes. Now suppose there is an optimal solution where the k^{th} mouse is sent to the p_0^{th} hole, where $k < p_0 < n$, the p_0^{th} mouse is sent to the p_1^{th} hole, and so on until the p_d^{th} (for some d) mouse is sent to the k^{th} hole. We could then swap the p_0^{th} hole with the k^{th} hole; we know by our lemma that the result will not be worse than the optimal solution. Therefore, by sending the k^{th} mouse to the k^{th} hole, we have not eliminated an optimal solution.
Conclusion. By the n^{th} step, we have not ruled out the optimal solution. Therefore, the solution we chose is optimal.

MST With Leaf Requirements

We are given an undirected weighted graph $G = (V, E)$ and a set $U \subset V$. Describe an algorithm to find a minimum spanning tree such that all nodes in U are leaf nodes. (The result may not be an MST of the original graph G.)

Let $T = V \setminus U$ be the set of nodes we *don't* require to be leaves, and let $D \subset E$ be the edges between nodes in T. Create an MST on (T, D), using Prim's algorithm for example. Then, add nodes in U to this tree by taking the lightest edge from a node $u \in U$ to any node $t \in T$.

This gives us a minimum-weight solution because any such solution having U as leaves must have an MST on T as a sub-graph. Otherwise there would be a lower-weight solution to the original problem.

If it's not clear why the MST on T must be part of the answer, consider some optimal MST S' satisfying the leaf requirements. Since each leaf by definition has degree 1, removing the leaf nodes of U along with their attached edges does not affect the connectedness of the remaining nodes. Once we remove from S' all the leaf nodes of U and their attached edges, we are left with a spanning tree over the nodes of T. If this spanning tree was not a MST over T, then taking the MST over T in conjunction with the leaf edges we just removed would yield a lower cost spanning tree than S'. Since this is not possible, the MST on T must be part of the solution S'.

Roads and Airports

Given a set of n cities, we would like to build a transportation system such that there is some path from any city i to any other city j. There are two ways to travel: by driving or by flying. Initially all of the cities are disconnected. It costs r_{ij} build a road between city i and city j. It costs a_i to build an airport in city i. For any two cities i and j, we can fly directly from i to j if there is an airport in both cities. Give an efficient algorithm for determining which roads and airports to build to minimize the cost of connecting the cities.

To find the roads and airports to build, we first note that there are two cases: either we do not build any airports or we build at least one airport.

To consider the case where we do not build any airports, we construct an undirected graph where the cities are the nodes and the roads are the edges with weights corresponding to the cost of building that road. We then construct the MST of this graph. This gives us the minimum construction cost using no airports. (If we constructed a non-tree connected graph, we could always remove a road to decrease cost without disconnecting the graph, so the optimal solution must be a tree.)

Then we consider the case where we choose to build at least one airport. To model this, we construct a slightly different graph. We start with the same graph from the previous case: an undirected graph where the cities are the nodes and the roads are the edges with weights corresponding to the cost of building that road. We then add another node to the graph, representing the air. Call this node a. We add an undirected edge between every city i and a with weight a_i. We then construct the MST of this graph.

To find the overall minimum cost set of roads and airports to build, we use either the MST from the first case or the MST from the second case, whichever has lower total cost. For every edge in the MST between two cities i and j, we build a road between i and j, and for every edge between a city i and a the airport node, we build an airport in city i.

Telephone Line

A hot new Silicon Valley start-up, Blockain.AI.ML.IoT (BAMI), is moving into its new offices, which consists of two buildings. The head of HR, Jacques, needs to decide which engineers to place in each building. The buildings are very old and made of concrete, so there is no cell reception, and there is only one desk in each building that has a landline phone. In order for two friends to be able to talk to each other, they either (1) both have to be in the same building or, (2) both have to be sitting at the two desks with phones so they can call each other.

Jacques must find a way to assign engineers to buildings while still ensuring every engineer can talk to all their friends. Unfortunately, it is not possible to seat all engineers with their friends in the same building, so one pair of friends will need to talk to each other through the phone. Jacques' goal is to find a pair of friends that can sit at the desks with phones so they can still talk with each other, while also ensuring that all other engineers are in the same building as their friends.

Give a short but clear English description of your algorithm.

Data Format
Input
n: the number of engineers.
F: a mapping from an engineer to their list of friends (e.g. F_{John} would refer to the friend list of John).
Output
(E_1,E_2): a tuple containing the names of the two friends that will sit at the desks with phones.
Assumptions:
Your input will always lead to a valid solution (no need to handle the case where there is no solution), and the solution will always require the use of the phone. Every engineer has $\theta(\sqrt{n})$ friends.

Example
Input:
n=5
F = Alice: [Bob, John], Bob: [Alice, John], Jane: [Kathy, Susan], John: [Alice, Bob, Susan], Kathy: [Jane, Susan], Susan: [Jane, John, Kathy]
Output: (John, Susan)
Explanation: If John and Susan are at the desks with phones, it directly implies that Alice, Bob, and John are in one building, and Susan, Kathy, and Jane are in the other building. All friends would be able to talk with each other as the problem requires.

There are a number of different ways to approach this problem. Below we describe a few such approaches

Karger Solution: $O(n^4)$ **expected time.**
Run Karger's algorithm to find a global minimum cut in the graph. Bonus: make the algorithm Las Vegas (instead of Monte Carlo) by checking that our cut has only 1 edge.

Connected Components Solution: $O(n^3)$ **time.**
For every edge in the graph, attempt to remove the edge and runs BFS/DFS to detect if the graph is still fully connected. If the removal of an edge disconnects the graph, return the people corresponding to the edge.

Karger-Stein Solution: $O(n^2 \log^2(n))$ **expected time.**
Run Karger-Stein's algorithm to find a global minimum cut in the graph.

Modified Karger-Stein Solution
In the analysis of Karger-Stein, we say that at every iteration, every meta-vertex has degree at least the size of the min-cut. When the min-cut has only 1 edge, every meta- vertex that is not incident on that 1 edge has degree at least 2, i.e. twice the min-cut. Therefore the probability of contracting the min-cut is almost twice is small.

Optimal algorithm, Modified SCC: $O(n^{3/2})$ **time**
(This algorithm is based on exam solutions of a few students, including Koby Chan and Daniel C Lee!) This algorithm first chooses a direction for each edge of the graph, then runs the Shortest Connected Components (SCC) algorithm. We will choose the direction in such a way that all the engineers that have to be on the same side of the cut will be in the same SCC.
Directing the edges: Run DFS from some arbitrary vertex s. Because the original graph is undirected, for every edge (u,v), u is either a descendant or ancestor of v in the DFS tree. We direct all the edges on the DFS tree from child to parent. We direct all other edges the opposite way (i.e. from ancestor to its descendant).
SCC: Find all vertices reachable from s in the directed graph. Let (y,x) be an edge in the DFS tree such that, x is reachable from s, but y isn't. Then there are no directed edges from x to its descendants in the DFS tree. Hence (y, x) must be a cut edge.
Runtime: Run DFS twice. Since ever engineer has $\theta(\sqrt{n})$ friends, the runtime is $O(m + n) = O(n^{3/2})$.