Code Jam 2018 - World Finals

Analysis: Jurisdiction Restrictions

Test set 1

We begin modeling the problem as a flow network. We construct a directed graph G where the set of nodes is {source, sink} ∪ S ∪ B where S is the set of blocks that contain a station and B is the set of blocks that do not contain a station and are within reach of some station. The edges of the graph are {source → s : s ∈ S} ∪ {s → b : s ∈ S, b ∈ B} ∪ {b → sink : b ∈ B}. Notice that each path from source to sink links a specific station with a specific block. If we refine G by assign capacity |B| to each edge source → s, and capacity 1 to each other edge, a valid flow of the network is a union of paths from source to sink, and due to the capacities, each station will belong to at most one of those paths. Moreover, a maximum flow will cover all stations, so there is a bijection between maximum flows and station assignments. In this way, we can reframe the problem as minimizing the difference between maximum and minimum flows going through edges that come out of the source in a maximum flow of G.

If we want to set an upper bound U for the flow going through edges coming out of the source, we can adjust the capacities of those edges to U. If we want to set a lower bound L on those quantities, we can add a second copy of each of those edges with capacity L and have a cost of 0 in the new edges and 1 in the max-capacity version (plus, cost 0 for all other types of edges). Using min-cost max-flow instead of max flow will give preference to flows that use at least L capacity through each station, and if the retrieved flow doesn't use the full capacity, it means that one doesn't exist.

With the above, we can just try every possibility for L and U, discarding the combinations that don't achieve a max flow of |B| that saturates all the cost 0 edges coming out of the source. The non-discarded combination with minimum U-L is the answer. Both U and L are bounded by |B|, which is bounded by $\mathbf{R} \times \mathbf{C}$. The size of G is $O(\mathbf{S} \times \mathbf{R} \times \mathbf{C})$, and min-cost max-flow on a graph with only 0 and 1 costs takes only linear time per augmenting path, so quadratic time overall. This means the described algorithm takes time $O(\mathbf{S}^2 \times \mathbf{R}^4 \times \mathbf{C}^4)$. This can be too slow even for test set 1, but there are some optimizations that will shed some of that time, and you only need one of them to pass test set 1.

- Instead of trying all combinations of L and U, you can try only the ones that have a chance to improve: if a combination (L, U) doesn't work, then we know (L+1,U), (L+2,U), etc will also not work, so we can move on to (L, U+1). If a combination (L, U) works, we know that (L, U+1), (L, U+2), etc will also work but will yield a larger difference, so we can skip them and move directly to (L+1, U). This requires us to test a linear (instead of quadratic) number of combinations of L and U.
- An even simpler version of the optimization above is to try all possibilities for only one of L or U, and then use binary search to find the optimal choice for the other. This doesn't take the number of combinations down to linear in |B|, but it does make it quasilinear.
- A flow for the combination (L, U) is also a valid flow for the combination (L, U+1), so instead of starting the flow from scratch, we can use it, dramatically reducing the number of augmenting paths we need to find overall.

Test set 2

For test set 2, the values of **R** and **C** are too high to appear linearly in the running time of the algorithm. That means we have to improve upon the solution above in at least two fronts: the size of G, which is linear on **R** and **C** and ultimately impacts any algorithm we want to run on G, and the number of combinations of L and U we try, which is also linear or more.

We start by defining G for test set 2 a little differently, using a technique commonly known as coordinate compression. We define B' as a set of sets of blocks that form a partition of B above. The way to partition is by taking the full grid and cutting through up to 2**S** horizontal and 2**S** vertical lines at the boundaries of the jurisdiction of each station. The resulting rectangles have the property that the set of stations that can reach any block of the rectangle is the same. Rectangles only represent the blocks within them that do not contain a station. We can now define G as having nodes {source, sink} ∪ S ∪ B' and edges {source → s : s ∈ S} ∪ {s → b' : s ∈ S, b' ∈ B'} ∪ {b' → sink : b' ∈ B'}. Note that each node in B' now represents a set of blocks, so we also have to fiddle with capacities. Edges source → s have capacity |B| (effectively infinity) as before, and so do edges s → b'. Edges b' → sink, however, have capacity |b'|. In this way, each node b' is the union of many nodes b from the previous version of G, so its outgoing edge has the sum of the capacities of the outgoing edges from those nodes b. The size of this G is $O(\mathbf{S}^3)$ with $O(\mathbf{S}^2)$ nodes.

Now, in G, each path source → s → b' → sink that carries flow X represents that X blocks from set b' are assigned to station s. Notice that there is no longer a bijection between flows and station assignments, but there is a bijection if we consider permutations of blocks within a rectangle to be equivalent. Since all those blocks are equivalent for the purpose of this problem, this bijection is enough. As before, we now have to find a maximum flow in G such that the difference between the maximum and minimum flow in edges going out of the sink is minimized.

Since we cannot try all possibilities for L and U, we will use a little theory. Let G_C be a copy of G, but with the capacities of all edges source → s changed to C. $G = G_{|B|}$. Notice that a valid flow in G_C is also a valid flow in any $G_{C'}$ with C' ≥ C.

Now, let U be minimum such that G_U allows a flow of total size |B|. Let L be maximum such that there exists a maximum flow of size |S| × L in G_L, that is, there is a flow that saturates all edges of type source → s. Any maximum flow in G has at least one edge of type source → sink carrying flow at least U (otherwise, such maximum flow is a valid flow in G_{U-1}, contradicting the definition of U). Also, any maximum flow in G has at least one edge of type source → sink carrying at most L (otherwise, we can subtract flow from any path sink to source that carries more than L+1 to obtain a flow that carries exactly L+1 through all nodes, and thus it would be a valid flow in G_{L+1} of size |S| × (L+1), contradicting the definition of L). Finally, if we start with the flow in G_L that justifies the definition of L and use it as a valid flow in G_U, we know that it can be extended to a maximum flow F in G_U via augmenting paths. Since the augmenting paths don't contain cycles, they don't decrease flow in edges of type source → s. Therefore, F is a maximum flow in G_U such that the difference between the maximum and minimum flow going through edges coming out of the source is exactly U - L. And by definition, a maximum flow of G_U is a maximum flow of G. The previous observations show that there is no flow in G with such a difference less than U - L, so U - L is the answer to the problem. We can binary search for L and U independently using their original definitions, which yields an algorithm that takes time $O(\mathbf{S}^5 \times \log(\mathbf{R} \times \mathbf{C}))$.