WeAreAllConnected Writeup

I modeled the problem as a weighted undirected graph. The cities are vertices and the communication links are edges. Given a list of possible edges, I needed to select the edge that, if added to the graph, would minimize the total pairwise weight of connections between vertices. I do not know a special name for this problem.

The brute force algorithm I first discovered for the solution works as follows: First, I found the shortest path between each city. I did this by using Floyd-Warshall's Shortest Path algorithm from each city. For the remainder of this writeup, S(a, b) refers to the shortest path between a and b.

Next, for each possible segment addition p connecting cities a and b, I examined each pair of cities l and r (I considered one left of the addition and the other to its right). I calculated [S(l, a) + p.duration + S(b, r)] – S(l, r), the difference between the path from l to r that utilized the new segment and the shortest path with no additions. If the difference was positive, it meant that communication between l and r is faster with segment p. I ignored negative differences, where even with the new segment, these cities would communicate faster with the old segment. I summed the positive differences for each pair x and y to get the total improvement caused by p. I returned the p with the maximum improvement.

The brute force algorithm's order of growth is O(n^3 + pn^2), where n is the number of cities, s is the number of segments currently in the graph, and p is the number of possible additions. Floyd-Warshall's order of growth is O(V^3), and so the stage's order of growth is O(n^3). For each suggested segment, my algorithm examines each pair of cities. There are p possible segments and n^2 pairs of cities, so that stage has an order of growth of O(pn^2). The stages are each performed once, so I add their order of growths to get O(n^3 + pn^2). I can't drop either term, because p can be O(1), in which case the first term dominates, or it can be proportional to n^2, in which case the second term dominates.

My improved algorithm is mostly the same as the brute-force algorithm but with one change to each stage. In the first stage, I used Dijkstra's algorithm instead of Floyd-Warshall for sparse graphs, calling it from each city to get the shortest paths from every city to every city. Dijkstra's order of growth is O(ElogV), and it is called for each city, so that stage's order of growth is now O(VElogV), or O(nslog$n$), for sparse graphs. For dense graphs, I continued to use Floyd-Warshall, as when s becomes proportional to n^2, Dijkstra becomes proportional to O(n^3log$n$), worse than Floyd Warshall.

In the second stage, for each segment p connecting cities a and b, I add each city c to either list L, list R, or neither. A city goes to list L if S(c, a) + p.duration < S(c, b). This means that the distance from the vertex to b is shorter going through p than around it, which means it might benefit from being on city a's "left" side of the new segment. Likewise, a city goes to list R if S(c, b) + p.duration < S(c, a). Because it can get to city a quicker going through the new segment, it might benefit from being on city b's "right" side of the new segment. For each pair of cities $l \in L$ and $r \in R$, I compare the new path, $[S(l, a) + p.\text{duration} + S(b, r)] - S(l, r)$, with the old path, $S(l, r)$, just as I did in the brute force algorithm.

If a city c does not make it into a list, it is because it can get to the other side of p at least as quickly by ignoring p than by going through it. This means that c can never get any improvement from p, because doing so would require going through a and b en route to another city, but c could already get there at least as quickly by going to a, taking its detour around p to b, and continuing to its destination.

It is possible for neither city to be on any list, but no city can be on both lists. Because (proof by contradiction) if a city were on both lists, then $S(c, b) + p.\text{duration} < S(c, a)$ and $S(c, a) + p.\text{duration} < S(c, b)$. This can be rearranged to yield $p.\text{duration} < S(c, a) - S(c, b)$ and $p.\text{duration} < S(c, b) - S(c, a)$. However, $S(c, a) - S(c, b) = -[S(c, b) - S(c, a)]$. So, one of these differences must be <= 0, so $p.\text{duration} < 0$. But all segments have non-negative durations! QED.

Because of this, the second stage's order of growth is improved to $O(p|L||R|)$. Since no city can be in both lists, $|L||R|$ is maximized when each is equal to $n/2$. This means that while the order of growth is unchanged, the stage's efficiency is improved by a factor of 4, so the total order of growth is now $O(ns\log n + pn^2)$. This is more effective than improving the order of growth of lower-order terms (which I couldn't figure out), because in asymptotic analysis like what Sedgewick uses with his tilde notation, lower order terms fall away, but constants of the highest-order term remain.