Code Jam 2008 - World Finals

Analysis: Bridge Builders

This problem seems intimidating at first glance. There are many different routes between the islands and the costs are somewhat unintuitive. However, the problem is actually a disguised Minimum Spanning Tree between the forests; and the nice thing about MSTs is that they are very friendly to greedy approaches. Almost any reasonable greedy approach that focuses on connecting up the forests will get the right answer.

It is fairly easy to convince yourself that the correct approach to the Small dataset is to head directly for the other forest; it can't possibly hurt, since all the other islands pay their minimum possible cost (ie, the distance to the nearest forest). In fact, assuming that islands pay their minimum cost is key: you can ignore this "base" cost, and then the only time you actually have to pay "extra" is when connecting up the two forests.

So for the Large dataset, once again you ignore the "base" cost for each island and simply focus on reaching all the forests, at which point you're done. Based on this intuition, you might immediately think of building a MST across the forests. The "extra" cost of connecting forests increases with distance, so use Prim's Algorithm: always head towards the forest nearest to the forests you've already visited. After connecting all the forests, you can build minimal-cost bridges to the remaining islands.

This greedy approach always works, and is quite an intuitive answer when you think of the problem in these terms. However, the proof that it works turns out to be quite technical. The problem is that you have to be sure that the forest connections really do form a graph with well-defined, order-independent costs. If your choice of path between one pair of forests could somehow help you save cost later, this would be incorrect (and the specter of NP-Completeness would loom menacingly). It seems reasonable that this can't happen, especially if you try a few examples yourself. But writing a proof in contest time is probably impossible. In fact, it took us several days of collaboration to come up with a proper, correct proof. :) This is what you have to deal with as a competitor in the Finals of the Google Code Jam!

Here's an outline of this proof:

First, suppose that we already have a tree (no cycles) of bridges which connects up every island. We show that the cheapest way we could have built these bridges is given by our MST algorithm (where distances are measured along the given tree). Assume we have some cheapest ordering of the bridges. First, we can reorder at no cost to ensure that it consists of direct sequential paths from a previously-visited island to a forest. Now suppose there is ever an island A from which we build two bridges, A-B and A-C. Further, suppose A-B leads to a closer island than A-C, but we build A-C first. Finally, suppose this is the *last* such occasion; so we build directly to the closest islands in B's subtree and in C's subtree.

Let w be the distance, when A-C is built, from A to its nearest forest along built bridges. Let x be this distance when A-B is built (so x <= w). Let y be the distance from A to its closest island in B's subtree, and z to the closest island in C's subtree (so y <= z). Then the cost of building these A-C and A-B paths is (w+1 + w+2 + ... + w+z) + (x+1 + x+2 + ... + x+y). Instead, if we do the A-B path *first* and leave A-C and its subtree until later, we pay at most (w+1 + w+2 + ... + w+y) + (x+1 + x+2 + ... + x+z), and possibly less since intermediate steps may also cost less. The second subtracted from the first is (w+y+1)-(x+y+1) + (w+y+2)-(x+y+2) + ... + (w+z)-(x+z) = (z-y)*(w-x), which is non-negative.

Thus, without increasing the cost, we can rearrange the order of bridges to build A-B first. Repeating this, we determine that one cheapest way to build the tree of bridges is always to

head for the nearest forest first, which is what our algorithm does.

Now, for any graph (a set of islands with a set of possible bridges), we prove inductively that our algorithm always gives the cheapest cost. It's true for 1 island; assume it's true for k. Suppose we have a minimal plan to connect up k+1 islands. Let A-B be the final bridge built. B could not have been visited already; so consider this plan restricted to the first k islands. Inductively, we know this plan is no cheaper than our algorithm, when run on those k islands; furthermore -- a nice fact about our algorithm, if you have not noticed yet -- after running our algorithm A is as close as possible to a forest. So doing this then building A-B is also a cheapest plan, and, importantly, this forms a tree of bridges. Let's call it T.

Now, from above, we know the cost of building this tree of bridges: it's just the total cost based on running our algorithm on the forest-to-forest distances in T. But these distances are at least as large as the corresponding forest-to-forest distances along *all* the potential bridges between our k+1 islands. We've paid at least as much as a normal spanning tree on our forests - so at least as much as a MST. But our algorithm achieves this MST cost, which we now know to be minimal.

QED.

Interestingly, this result also holds for any graph of islands and potential bridges, not just a grid or a planar graph. And the distance-to-cost metric can be any nondecreasing function. These facets of the problem are artificial and not essential to the algorithm.

Finally, here is pseudocode for the solution, assuming you know any standard MST algorithm.

```
for each unordered pair of forests (a, b)
  x = distance(a, b)
  y = x / 2  // Assume integer division
  c = x * (x + 1) / 2  // Add cost of connecting forests
  c -= y * (y + 1)  // Subtract "base" cost of islands
  if x is even
    c += y // Avoid double-counting middle island
  add edge (a, b) at cost c to forest_graph

result = min_cost_spanning_tree(forest_graph)

for each island x
  d = INFINITY
  for each forest y
    d = min(d, distance(x, y))
  result += d  // Add "base" cost of island x
```