**Code Jam 2010 - World Finals**

# Analysis: City Tour

This proved to be the easiest problem in the finals for the experienced competitors.

We get some insight into the problem by noticing the graph resembles a tree. These graphs are actually called partial 2-trees. We can actually build a tree if we follow the way the graph is constructed. We build another graph that associates a node to each new added cycle of length this cycle shares an edge to an old cycle so we add an edge between the two respective nodes in the second graph. This way we have built the tree decomposition of these graphs. As on trees many problems that are hard to solve on general graphs have polynomial algorithms on this type of graphs.

Let's solve the related problem of finding the longest path in a tree. We can use dynamic programming and depth first search. We mark a node as a root. Every path in the tree has exactly one node that is closest to the root and this node splits the path in two downward paths. Now for each node in the tree we compute the longest downwards path that starts in it. To find the largest path in the tree we look at each node and at the two longest paths that start in it's children. This solves the problem in linear time.

The solution for our original problem is pretty similar. For each edge (x,y), we compute the cycle that contains it and and all the other nodes in the cycle have larger indexes. Let's call this a downward cycle since it goes the opposite direction of where the initial three nodes are. To find that number we have to look at all higher indexed nodes that were connected to this edge and try to use them as intermediary points in the cycle. So for a given intermediary point z we can build a cycle by looking at the longest downward cycle that contains the edge (x,z) and the longest downward cycle that contains the edge (z,y), use all the edges, add edge (x,y) and remove the edges (x,z) and (z,y).

We also compute the largest downward cycle which contains these two nodes but doesn't contain this edge, this is a union of the cycle that goes through these nodes and the second largest path from which we remove the edge (x,y).

And here is some code that does implements this solution:

```
int best_so_far = 0;

int best(int x, int y, int N, int[][] a) {
    int max_len = 2;
    int second_max_len = -1;
    for (int i = Math.max(x, y) + 1; i < N; i++) {
      if (a[x][i] * a[y][i] > 0) {
        int len =  best(x, i, N, a) + best(y, i, N, a) - 1;
        if (len > max_len) {
          second_max_len = max_len;
          max_len = len;
        } else if (len > second_max_len) {
          second_max_len = len;
        }
      }
    }
    best_so_far = Math.max(max_len, best_so_far);
    best_so_far = Math.max(max_len + second_max_len - 2,
                           best_so_far);
    return max_len;
}
```

Another cool solution is based on the idea of contracting each node of degree two. We replace it with an edge which has the weight equal to the sum of the weights of the two incoming edges. It's a pretty neat idea and we'll let you figure out the details on your own.