Code Jam 2008 - Round 2

Analysis: PermRLE

Section A. The Hamiltonian cycle in a small world

A Hamilton cycle in a graph is a cycle that visits each node exactly once. Given a weighted, directed, complete graph on n nodes, there are (n-1)! distinct Hamiltonian cycles. It is well known that the problem of finding the shortest (or longest) Hamilton cycle is NP-hard. It is also known to many contestants that, for n as small as 20, dynamic programming makes a difference of $n*2^n$ vs. n!, which is the difference between a second and an eternity.

Let's have a look at the $n*2^n$ DP trick, in case you have not seen it before.

Without loss of generality, we may view node 0 as the start point of the cycle, as well as its end point. For any subset A of the node set V and any node x in A, we define

> dp[A][x] := The shortest path that starts from x, visits each point in A exactly once and ends up at node 0. (*)

To clarify, 0 does not necessary belong to A, but we do count the length of the edge from the last point to node 0. Thus the problem of finding the shortest Hamilton cycle is just dp[V][0]. (Convince yourself, maybe looking at (*).)

We need to compute dp[A][x]. For the easy cases where A = {x}, the answer is just the length of edge x→0. Otherwise, we focus on the first step of the path. If the first step is x→y, with edge length q, then we pay dp[A - x][y] + q. In general, dp[A][x] is

- length(x→0), if A = {x}.
- min { dp[A - x][y] + length(x→y) | y in A - x }, if |A|>1.

Section B. Wrap everything into a small world

For any string, define the *number of switches* to be the number of times adjacent characters are different in the string. We want to find a permutation that transforms **S** to one **S'** where the number of switches is minimal. Assume the length of **S** is **mk**. Then **S** can be viewed as a string with **m** blocks of length **k**.

Now we introduce a visual aid to simplify our writing. Let us draw the string **S** as **m** rows, each block on a single row. The key image is to count the number of switches one *column* at a time.

Let us take a semi-concrete example. Suppose that at one point we have decided that 5 is permuted to the 7th position, and that 2 goes to the 8th position. Then without knowing the rest of the permutation, we can inspect the 5th and 2nd characters in each block. Suppose that in Z of the blocks the 5th and the 2nd characters are different, then we know that in any such permutation, we will have to pay the price of Z.

The one exception is the last element of the permutation. In all cases but one, we simply wrap around to the beginning because the end of each k-block touches the beginning of the next k-block in the string, except for the last character in the string. We can handle both cases if we fix the last element of the permutation by trying all possibilities.

Next, we reduce our problem to the one in Section A. Suppose we fix **T** as the last element in the permutation. Define a weighted, directed, complete graph G on **k** vertices {1, 2, ..., k}. The

weight on the edge x→y is

- the number of blocks where the x-th character is different from the y-th character in the same block. (if x ≠ T)
- the number of blocks (excluding the last one) where the x-th character is different from the y-th character in the *next* block. (if x = T)

It is easy to check that for any permutation, the number of switches is the same as the length of the corresponding Hamiltonian cycle in G.

We have **k** different choices for T. For each T, finding the shortest Hamilton cycle takes $O(2^k k)$ time. The construction of the graph takes $O(k^2 m) = O(k |S|)$ time for each T; it is also easy to construct in $O(k^2 m)$ time the graphs for all the T's. The running time of the solution is $O(2^k k^2 + k |S|)$.