TopCoder Competitions

SRM 638

Single Round Match 638
Wednesday, November 3rd, 2014

Archive
Match Overview
Discuss this match

Match summary

The Problems

NamingConvention | NarrowPassage2Easy | CandleTimerEasy | ShadowSculpture | NarrowPassage2 | CandleTimer

CandleTimerEasy

Rate It | Discuss it

Used as: Division Two - Level Three:

Value	1000
Submission Rate	20 / 493 (4.06%)
Success Rate	2 / 20 (10.00%)
High Score	**mxh7k7k** for 824.38 points (13 mins 44 secs)
Average Score	631.22 (for 2 correct submissions)

There are at most 20 nodes in the tree made of candles. That means there will be at most 19 leaves. In order to count the total number of times, we can try all 2^{19} subsets of leaves of the trees for candles we lit initially. We can pick all those subsets using backtracking or bitmasks

The difficult part of the problem is to, once the leaves are picked, conduct the simulation to find the final time after which all the candles are consumed. 2^{19} is not very small so we should aim to do this simulation in $O(n)$ or $O(n \log(n))$ time.

Furthest point

The flames start at some of the leaves in the tree. I think the easiest way to understand the simulation is to notice how similar the movement of the flames is to Dijkstra's algorithm. Points are visited by the flames in increasing order of distance to the flames' starting points. When two flames meet, the visited points continue the same logic.

From this strange idea comes the solution: The final point the flames will visit is the point that is furthest from the initial flame points - the leaves we picked in the previous step. Therefore we just need to find the furthest point.

Two possibilities:

- The furthest point is a node in the tree. This case is easy, just calculate the minimum distances between the flames' initial positions and each of the nodes of the tree (A single Dijkstra execution is good enough to do this task). And find the node with the maximum distance.
- The furthest point is between two nodes. Then we need to find the minimum distances to these intermediary points. Consider an edge $A \leftrightarrow B$, if you wanted to know if the flames meet inside this edge, we would need u and v to be connected to distinct flame starting points. We can confirm this by using the Dijkstra idea and keeping in mind the predecessors of A and B in the shortest path we find using Dijkstra, if A is not the predecessor of B nor vice versa, then we can imagine the flames will meet in this edge.

The question is at what time will the flames meet? This depends on the times at which the flames will be at point A and B respectively, imagine that of those two times, x is the largest and y is the smallest. The distance between A and B is L.

- At time x, flame 1 is at distance 0 from A.
- At time y, flame 2 is at distance L from A.
- At time x, flame 2 is at distance $u = L - (x - y)$ from A.
- Then the flames need $\frac{u}{2}$ seconds to meet , in addition to the x seconds: $x + \frac{u}{2}$.

If u is odd, $x + \frac{u}{2}$ may be fractionary, we can avoid this slight complication by multiplying each candle length by 2, this doesn't alter the final result, but it makes all final time results integers.

For each selection of initial flame positions, run a Dijkstra algorithm and find the maximum distance to a node OR a point between two nodes. This maximum is the duration for the given flame position selection. The total number of different durations is the final result.

Code

```
int differentTime(vector<int> A, vector<int> B, vector<int> len)
{

    const int INF = 1000000000;

    int n = A.size() + 1;
    vector<list<int>> g(n);
    vector<vector<int>> w(n, vector<int>(n, INF));

    for (int i = 0; i < n - 1; i++) {
        len[i] *= 2;
        for (int j = 0; j < 2; j++) {
            g[A[i]].push_back(B[i]);
            w[A[i]][B[i]] = len[i];
            swap(A[i],B[i]);
        }
    }
    vector<int> leaves;
    for (int i = 0; i < n; i++) {
        if ( g[i].size() == 1 ) {
            leaves.push_back(i);
        }
    }
    int t = leaves.size();

    set<int> times;
    for (int mask = 1; mask < (1<< t); mask++) {
        vector<int> dist(n, INF);
        vector<int> Q;
        vector<int> parent(n, -1);

        for (int i = 0; i < t; i++) {
            if (mask & (1<<i)) {
```

Alternative solutions and additional comments.

<Place your comments here>

Next problem: ShadowSculpture

By **vexorian**

TopCoder Member

5 Comments | Add Comment