USACO NOV12 Problem 'btree' Analysis

by Mark Gordon

We will first consider how to solve the problem where you're given a string of parenthesis and you want to find the substring that is the deepest balanced parenthesis expression. It's easier to envision solving the problem in terms of prefix sums. Let f(n) give the number of '(' characters minus the number of ')' characters. It should be clear that the range [a, b) is a balanced parenthesis expression iff f(a)=f(b)=min(f[a], f[a+1], ..., f[b]). Moreover the depth of the expression is at least f[z] - f[a] for a <= z <= b.

One way to approach this problem, then, is from the inside out. For each index z we'll want to find the deepest it could be nested in a balanced parenthesis expression. We can compute this directly by first computing g as max(min(f[0], f[1], ..., f[z]), min(f[z], f[z + 1], ..., f[n-1])). Then, the claim is, there must exist an a and b such that g=f(a)=f(b)=min(f[a], f[a+1], ..., f[b]) so that the deepest z could be nested is f(z) - g.

To see why the a and b must exist we can imagine starting a at z and decrementing it until f(a) equals g. Since f(a) starts out equal to f(z) > g and f(n) changes by at most 1 each index it follows that f(a) will eventually hit g when we'll terminate. Note that we introduced no elements into our range with f(x) smaller than g (or we would have stopped). Similarly we can do the same for b with incrementing. It should be clear that the balance conditions are now met.

We can apply this to trees in a similar way. Instead of picking an index, z, we will pick an edge, e. From one end we will calculate the maximum difference of '(' and ')' characters and from the other end we will calculate the maximum difference of ')' and '(' characters. And, similar to the linear case, the max depth the edge could be nested at is minimum of the two values.

These maximum differences of '(' and ')' characters (and vice versa) can be calculated using a tree DP. Our state is represented by the edge being processed and the orientation of the edge. Then the maximum difference is either 0 or the maximum for each outgoing edge from our first vertex (other than the edge we're processing) of the maximum difference including the character at our first vertex.

As an implementation note consider that the basic implementation may visit a node proportional to the number of edges it has and that it does work proportional to the number of edges it has. Without care this will lead to a O(N^2) solution in the worst case. To fix this you need to compute results for all edges incident a node at once in some cases. This is only possible the second time you visit a node, however, otherwise the dependency in calculations becomes cyclic.

```
#include <iostream>
#include <vector>
#include <cstdio>
#include <cstring>
#include <cassert>

using namespace std;

int A[200000];
int B[200000];
int C[200000][2];
int V[100000];
vector<int> E[100000];

bool vis[100000][2];
```

```
int solve(int x, int m) {
  int& ref = C[x][m == 1];
  if(ref != -1) return ref;

  int u = B[x];
  if(vis[u][m == 1]) {
    int r = 0;
    for(int i = 0; i < E[u].size(); i++) {
      C[E[u][i] ^ 1][m == 1] = max(r, C[E[u][i] ^ 1][m == 1]);
      r = max(r, m * V[u] + C[E[u][i]][m == 1]);
    }
    r = 0;
    for(int i = E[u].size() - 1; i >= 0; i--) {
      C[E[u][i] ^ 1][m == 1] = max(r, C[E[u][i] ^ 1][m == 1]);
      r = max(r, m * V[u] + C[E[u][i]][m == 1]);
    }
    return ref;
  }
  vis[u][m == 1] = true;

  ref = max(0, m * V[u]);
  for(int i = 0; i < E[u].size(); i++) {
    if(x == (E[u][i] ^ 1)) continue;
    ref = max(ref, m * V[u] + solve(E[u][i], m));
  }
  return ref;
}

int main() {
  freopen("btree.in", "r", stdin);
  freopen("btree.out", "w", stdout);

  int N; cin >> N;
  assert(1 <= N && N <= 40000);
  for(int u = 1, id = 0; u < N; u++) {
    int v; cin >> v; v--;
    assert(v < u);
    A[id] = u;
    B[id] = v;
    E[u].push_back(id++);
    A[id] = v;
    B[id] = u;
    E[v].push_back(id++);
  }
  for(int i = 0; i < N; i++) {
    char ch; cin >> ch;
    V[i] = ch == '(' ? 1 : -1;
  }

  int res = 0;
  memset(vis, 0, sizeof(vis));
  memset(C, -1, sizeof(C));
  for(int u = 0; u < N; u++) {
    for(int i = 0; i < E[u].size(); i++) {
      int id = E[u][i];
      res = max(res, min(solve(id, 1), solve(id ^ 1, -1)));
    }
  }
  cout << res << endl;
}
```