USACO DEC11 Problem 'grassplant' Analysis

by Albert Gu

There are many ways to do this problem, some of which are faster than others. One solution uses a technique called heavy-light decomposition. Root the tree arbitrarily. Consider an edge between vertices u and v (let u be the parent of v). Call this edge heavy if the size of the subtree rooted at v is at least half the size of the subtree rooted at u, and light otherwise. Note that each parent has at most one heavy edge to a child, so the heavy edges form a set of chains. We will keep track of each heavy chain separately.

Finally, any path from a vertex to the root has at most log n light edges, because advancing up a light edge at least doubles the size of the subtree. This path also has at most log n heavy chains, since a light edge separates every two heavy chains. This means we can divide this path into log n pieces, so adding grass along a path from a vertex to a root will take O(log n) time by adding it to each light edge and heavy chain on the path. Note that we will need to augment each heavy chain with a structure such as a range tree or BIT (binary indexed tree) because we might need to plant grass on only a prefix of a heavy chain.

To update an arbitrary path, we note the following: Planting 1 grass on each edge along the path from A to B is the same as planting 1 grass on each edge from A to the root, 1 grass on each edge from B to the root, and -2 grass on each edge from LCA(A, B) to the root, where LCA denotes the least common ancestor.

(As a note, the memory limits for this problem were extremely tight --- probably a bit too tight; conserving memory was one of the main problems faced by most competitors).

Here is Mark Gordon's code:

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

using namespace std;

/* Heavy Light Implementation */
#define MAXN 100000

int N; // Number of nodes

/* Node fields. */
int VC[MAXN];  // Vertex chain index
int VPOS[MAXN]; // Vertex chain position

/* Chain fields. */
int CP[MAXN]; // Chain parent (-1 if none)
int CLN[MAXN]; // Number of vertexes in chain.
```

```
vector<int> E[MAXN];

pair<int, int> hang(int u, int p) {
  int nodes = 1;
  pair<int, int> result(0, -1);
  for(int i = 0; i < E[u].size(); i++) {
    int v = E[u][i];
    if(v == p) continue;
    pair<int, int> res = hang(v, u);
    nodes += res.first;
    result = max(result, res);
  }
  E[u].clear();

  if(result.second == -1) result.second = u;
  E[result.second].push_back(u);
  CP[result.second] = p;

  result.first = nodes;
  return result;
}

/* Before calling E[i] represents the edges of vertex i.  After calling E[i]
is
 * a list (from root to child) of vertexes in chain i. */
void prep_heavylight() {
  hang(0, -1);
  for(int i = 0; i < N; i++) {
    CLN[i] = E[i].size();
    for(int j = 0; j < E[i].size(); j++) {
      VC[E[i][j]] = i;
      VPOS[E[i][j]] = j;
    }
  }
}

/* Represents the nodes in the range [lo, hi) of chain. */
struct ChainPart {
  ChainPart(int chain, int lo, int hi) : chain(chain), lo(lo), hi(hi) {}
  int chain;
  int lo; // lo node on the chain.
  int hi; // hi node on the chain. hi=CLN[chain] means chain connects to
parent
};

/* Returns a list of all the sections of the chains on the path from u to v.
*/
vector<ChainPart> chain_path(int u, int v) {
  vector<pair<int, int> > cu, cv;
  for(; u != -1; u = CP[VC[u]]) cu.push_back(make_pair(VC[u], VPOS[u]));
  for(; v != -1; v = CP[VC[v]]) cv.push_back(make_pair(VC[v], VPOS[v]));
  reverse(cu.begin(), cu.end());
  reverse(cv.begin(), cv.end());
  if(cv.size() < cu.size()) cu.swap(cv);

  int i;
  for(i = 0; i < cu.size() && cu[i] == cv[i]; i++);

  vector<ChainPart> ret;
  if(i == cu.size()) {
    ret.push_back(
        ChainPart(cu.back().first, cu.back().second, cu.back().second));
  } else if(cu[i].first == cv[i].first) {
    ret.push_back(ChainPart(cu[i].first,
                            min(cu[i].second, cv[i].second),
                            max(cu[i].second, cv[i].second)));
    ++i;
  }
  for(int j = i; j < cu.size(); j++) {
    ret.push_back(ChainPart(cu[j].first, cu[j].second, CLN[cu[j].first]));
  }
  for(int j = i; j < cv.size(); j++) {
    ret.push_back(ChainPart(cv[j].first, cv[j].second, CLN[cv[j].first]));
  }
  return ret;
}

/* Get rid of empty chain parts. */
vector<ChainPart> filter(const vector<ChainPart>& ch) {
  vector<ChainPart> res;
  for(int i = 0; i < ch.size(); i++) {
    if(ch[i].lo != ch[i].hi) {
      res.push_back(ch[i]);
    }
  }
  return res;
}

vector<int> bit[MAXN];

void bit_add(vector<int>& A, int x, int v) {
  for(int i = x | A.size(); i < (A.size() << 1); i += i & -i) {
    A[i ^ A.size()] += v;
  }
}

int bit_get(vector<int>& A, int x) {
  int ret = A[0];
  for(int i = x; i; i &= i - 1) ret += A[i];
  return ret;
}

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

  int M;
  scanf("%d%d", &N, &M);
  for(int i = 1; i < N; i++) {
    int u, v; scanf("%d%d", &u, &v); u--; v--;
    E[u].push_back(v);
    E[v].push_back(u);
  }
  prep_heavylight();

  for(int i = 0; i < N; i++) {
    if(!CLN[i]) continue;
    bit[i] = vector<int>(1 << (32 - __builtin_clz(CLN[i] - 1)), 0);
  }

  for(int i = 0; i < M; i++) {
    char op[2]; int u, v; scanf("%1s%d%d", op, &u, &v); u--; v--;
    vector<ChainPart> ch = filter(chain_path(u, v));
    if(op[0] == 'P') {
      for(int i = 0; i < ch.size(); i++) {
        bit_add(bit[ch[i].chain], ch[i].lo, 1);
        if(ch[i].hi < CLN[ch[i].chain]) {
          bit_add(bit[ch[i].chain], ch[i].hi, -1);
        }
      }
    } else {
      assert(ch.size() == 1 && ch[0].lo + 1 == ch[0].hi);
      printf("%d\n", bit_get(bit[ch[0].chain], ch[0].lo));
    }
  }
}
```