USACO MAR10 Problem 'gather' Analysis

by Jeffrey Wang

We can think of this problem as optimizing some objective function (inconvenience) over the nodes in a tree, i.e. we would like to evaluate this function at each node in the tree and find the minimum value. Naively, this gives an O(N^2) algorithm since it takes O(N) time to evaluate the function at any one node. It is fairly straightforward to improve this by realizing that we can traverse the tree in depth-first order and update the value of the objective as we go instead of recomputing each time.

- First, arbitrarily root the tree and, for each node X, compute down(X) as the total number of cows in the subtree rooted at X and up(X) as the total number of cows not in the subtree. Note that we can compute up/down at every X in O(N) time by a simple depth-first traversal.
- Then, compute the objective value at the root (we can do this in the same traversal).
- Suppose we know the objective value is C at some node U. If we are then going to one of its children, V, we can update the objective value to be C-down(V)*w(U,V)+up(V)*w(U,V) since all of the cows in V's subtree get w(U,V) closer to the chosen location while the rest of the cows get further away.

Thus with two depth-first traversals of the tree we can evaluate the objective function at every node in O(N) time. Below is my code.

```
#include <iostream>
#include <vector>
using namespace std;

int N;
long long cost[1<<17], cows[1<<17], down[1<<17], up[1<<17];
vector<vector<long long> > e, w;

long long dfs1(int cur, int prev) {
  down[cur] = cows[cur];
  long long c = 0;
  for(int i = 0; i < e[cur].size(); i++) {
    if(e[cur][i] == prev) continue;
    c += dfs1(e[cur][i], cur);
    c += down[e[cur][i]]*w[cur][i];
    down[cur] += down[e[cur][i]];
  }
  return c;
}

long long dfs2(int cur, int prev) {
  long long c = cost[cur];
  for(int i = 0; i < e[cur].size(); i++) {
    if(e[cur][i] == prev) continue;
    cost[e[cur][i]] = cost[cur]-
down[e[cur][i]]*w[cur][i]+up[e[cur][i]]*w[cur][i];
    c = min(c, dfs2(e[cur][i], cur));
  }
  return c;
}

int main() {
  FILE* in = fopen("gather.in", "r");
  FILE* out = fopen("gather.out", "w");

  fscanf(in, "%d", &N);
  e.resize(N); w.resize(N);
  for(int i = 0; i < N; i++) fscanf(in, "%lld", &cows[i]);
  for(int i = 0; i < N-1; i++) {
    long long a, b, c;
    fscanf(in, "%lld %lld %lld", &a, &b, &c);
    a--; b--;
    e[a].push_back(b); w[a].push_back(c);
    e[b].push_back(a); w[b].push_back(c);
  }

  cost[0] = dfs1(0, -1);
  for(int i = 0; i < N; i++) up[i] = down[0] - down[i];
  fprintf(out, "%lld\n", dfs2(0, -1));

  fclose(in);
  fclose(out);
}
```