USACO NOV08 Problem 'cheer' Analysis

by David Benjamin

Given a choice of the edges and starting point, which form a rooted tree, the optimal path is an Euler tour from the root. That is, we visit nodes in order:

```
visit(node v):
    go to v
    for c in children of v:
        visit(c)
        go to v
```

A node with K children appears in a tree's Euler tour K+1 times. For all nodes except the root, K+1 is also the degree of the node (whereas a root with K children has degree K). Thus, a node of degree D is visited D times, with the root being visited once more. Also, each edge is traversed twice: once going down and once going up.

Our total visiting time, for a tree rooted at vertex R with edges of length L_i and vertices of degree D_j and cost C_j is 2(L_1 + ... + L_N-1) + (D_1*C_1 + ... + D_N*C_N) + C_R.

We can now simplify the problem by computing the cheapest tree and picking the smallest C_R independently. We also note that counting a vertex j D_j times is counting it once for every edge connected to it. Equivalently, this is attributing to each edge i, the costs of the vertices at either end, S_i and E_i.

If we relabel an edge from S to E with length L with a weight of 2*L + C_E + C_S, the cost of the tree is the sum of the edges. This is exactly the minimum-cost spanning tree problem, which can be solved in O(P log P) time with Kruskal's or Prim's algorithm.

```
#include <assert.h>
#include <stdio.h>
#include <limits.h>
#include <algorithm>

const int MAXV = 10000+5;
const int MAXE = 100000+5;

int v,e;
int cost[MAXV];
struct edge {
    int a,b;
    int l;
    bool operator<(const edge& e) const { return l < e.l; }
};
edge edges[MAXE];

int uf_rank[MAXV];
int uf_parent[MAXV];
/* Initialize the world with v elements: object i in set i */
```

```
void uf_init(int v) {
    for (int i=0;i<v;i++)
        uf_parent[i] = i;
}
/* Find the set containing v */
int uf_find(int v) {
    if (v == uf_parent[v]) return v;
    return (uf_parent[v] = uf_find(uf_parent[v]));
}
/* Merge the sets of a and b */
void uf_union(int a, int b) {
    a = uf_find(a); b = uf_find(b);
    if (uf_rank[a] > uf_rank[b]) {
        uf_parent[b] = a;
    } else {
        uf_parent[a] = b;
        if (uf_rank[a] == uf_rank[b])
            uf_rank[b]++;
    }
}

int main() {
    FILE * fin = fopen("cheer.in", "r");
    FILE * fout = fopen("cheer.out", "w");
    assert(fin != NULL); assert(fout != NULL);

    /* Input */
    int ans = INT_MAX;
    fscanf(fin, "%d %d", &v, &e);
    for (int i=0;i<v;i++) {
        fscanf(fin, "%d", cost+i);
        // sleep in the cheapest pasture
        ans = std::min(ans, cost[i]);
    }
    for (int i=0;i<e;i++) {
        fscanf(fin, "%d %d %d", &edges[i].a, &edges[i].b, &edges[i].l);
        edges[i].a--; edges[i].b--;
        // adjust the edge costs
        edges[i].l *= 2;
        edges[i].l += cost[edges[i].a];
        edges[i].l += cost[edges[i].b];
    }
    // visit edges in order of cost
    std::sort(edges, edges+e);

    /* Kruskal's Algorithm */
    uf_init(v);
    for (int i=0;i<e;i++) {
        // don't connect anything already connected
        if (uf_find(edges[i].a) == uf_find(edges[i].b)) continue;
        // add edge i
        ans += edges[i].l;
        uf_union(edges[i].a, edges[i].b);
    }

    fprintf(fout, "%d\n", ans);
}
```