USACO JAN08 Problem 'phoneline' Analysis

by Spencer Liang & Parham Razaghi

We construct the following graph G: each pole is a vertex, and each possible connection between poles is an edge between corresponding vertices with weight equal to the distance between the poles.

Now imagine we have a function f(lim) that tells us if there exists a path from vertex 1 to vertex N using no more than K edges whose weights are greater than lim. If we have such a function f, we can perform a binary search for the answer: the smallest lim that works (in other words, the smallest k such that f(k) is true) is the minimum amount Farmer John must pay.

So the problem now is implementing function f(lim) efficiently, and to do so, we consider the graph H, which has the same vertices and edges as G but different edge weights. More precisely, an edge between vertices a and b in H has weight 0 if the corresponding edge in G has weight w <= lim, and weight 1 otherwise (if the corresponding edge in G has weight w > lim), so the shortest path between two vertices a and b in H represents the minimum number of edges with weight greater than lim on a path between a and b in G. Thus computing f(lim) is equivalent to checking if the shortest path between 1 and N in H is less than or equal to K, and we can do this in O(E log V) time with Dijkstra's.

In the worst case, we will need to evaluate function f O(log V) times (because of the binary search), so the total running time of the entire algorithm is $O(E \log^2 V)$. (It's actually possible to compute the shortest path between two vertices in a graph where all edges have weight 0 or 1 in linear time, but that's not needed here.)

Below is the short solution of Iran's Parham Razaghi:

```
#include<fstream>
#include<vector>

using namespace std;

ifstream fin ("phoneline.in");
ofstream fout ("phoneline.out");

const int MAX = 1000 + 5;

vector <int> a[MAX], b[MAX];
int     e[MAX * 10];

bool    mark[MAX];
int     dis [MAX], saf[MAX], head, tail;

int     n, k, D, M;

void dfs (int u) {
    dis[u] = D;
    mark[u] = true;
    saf[tail++] = u;
    for (int i = 0; i < a[u].size (); i++) {
        if (!mark[a[u][i]] && b[u][i] <= M)
            dfs (a[u][i]);
    }
}
void Bfs (int MM) {
    M = MM;
    memset (mark, 0, sizeof mark);
    head = tail = 0;
```

```
    D = 0;
    dfs (n - 1);
    while (head < tail) {
        int k = saf[head++];
        for (int i = 0; i < a[k].size (); ++i) {
            if (!mark[a[k][i]]) {
                D = dis[k] + 1;
                dfs (a[k][i]);
            }
        }
    }
}

void bs (int x, int y) {
    if (y == x + 1) {
        fout << e[y] << endl;
        exit (0);
    }
    int     mid = (y + x) / 2;
    Bfs (e[mid]);
    if (dis[0] <= k)
        bs (x, mid);
    else
        bs (mid, y);
}
int main () {
    int ee;
    fin >> n >> ee >> k;
    int     u, v, w;
    for (int i = 0; i < ee; ++i) {
        fin >> u >> v >> w;
        u--;
        v--;
        a[u].push_back (v);
        b[u].push_back (w);
        a[v].push_back (u);
        b[v].push_back (w);
        e[i + 1] = w;
    }
    sort (e, e + 1 + ee);
    Bfs (0);
    if (!mark[0]) {
        fout << "-1" << endl;
        return 0;
    }
    if (dis[0] <= k) {
        fout << "0" << endl;
        return 0;
    }
    bs (0, ee);
    return 0;
}
```