USACO FEB14 Problem 'rblock' (Gold) Analysis

by Neal Wu

This problem is a standard occurrence of the shortest path problem, except that we can now choose a single edge to be doubled and would like to choose the edge that maximizes the new shortest path.

Notice that once we have chosen an edge, we can compete the shortest path easily in either O(M log N) or O(N^2) time, by simply modifying the edge length and performing Dijkstra's shortest path algorithm. However since there are M edges this gives an O(M^2) overall complexity, which was intended to be too slow.

To improve the complexity, we can notice that if the edge we choose to double is not on the original shortest path from 1 to N, then the final shortest path length stays the same. This means we only need to try doubling the edges on the original shortest path from 1 to N, and there are only O(N) of them. This gives us a better complexity of either O(NM log N) or O(N^3), the latter of which is implemented below.

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;

FILE *in = fopen ("rblock.in", "r"), *out = fopen ("rblock.out", "w");

const int MAXN = 505;

int N, M, edge [MAXN][MAXN], dist [MAXN], prev [MAXN];
bool visited [MAXN];

int best_path (int start, int end)
{
    memset (dist, 63, sizeof (dist));
    memset (visited, false, sizeof (visited));
    memset (prev, -1, sizeof (prev));
    dist [start] = 0;

    while (true)
    {
        int close = -1;

        for (int i = 0; i < N; i++)
            if (!visited [i] && (close == -1 || dist [i] < dist [close]))
                close = i;

        if (close == -1)
            break;

        visited [close] = true;

        for (int i = 0; i < N; i++)
        {
            int ndist = dist [close] + edge [close][i];

            if (ndist < dist [i])
            {
                dist [i] = ndist;
                prev [i] = close;
            }
```

```
        }
    }

    return dist [end];
}

int main ()
{
    memset (edge, 63, sizeof (edge));
    fscanf (in, "%d %d", &N, &M);

    for (int i = 0; i < M; i++)
    {
        int a, b, len;
        fscanf (in, "%d %d %d", &a, &b, &len);
        a--; b--;
        edge [a][b] = edge [b][a] = len;
    }

    int original = best_path (0, N - 1);
    vector <int> path;

    for (int i = N - 1; i != -1; i = prev [i])
        path.push_back (i);

    int most_doubled = original;

    for (int i = 0; i + 1 < (int) path.size (); i++)
    {
        int a = path [i], b = path [i + 1];
        edge [a][b] *= 2;
        edge [b][a] *= 2;
        most_doubled = max (most_doubled, best_path (0, N - 1));
        edge [a][b] /= 2;
        edge [b][a] /= 2;
    }

    fprintf (out, "%d\n", most_doubled - original);
    return 0;
}
```