ASSUMPTION UNIVERSITY
Vincent Mary School of Science and Technology
Department of Computer Science

SC 6310
DESIGN AND ANALYSIS OF ALGORITHMS

TERM PROJECT REPORT

1416 Confidential
Timus Online Judge

Submitted to
Asst Prof Dr Thitipong Tanprasert
by
Chayapol Moemeng
Semester 1 2017
1. Introduction
We aim to solve an algorithm problem on TIMUS [1]. The problem original text is shown in Appendix B. In summary, the problem asks to find the routes across set of planets where each transition between planets has a cost. We are to find the route with the smallest cost as well as the second smallest cost from the given planetary structures in a limited time.

2. Problem Analysis
We reduce this problem to a minimum spanning tree (MST) problem; where each planet is a vertex, each transition is an edge, and the transition cost is the weight of the edge.

2.1 Minimum Spanning Tree
Kruskal’s algorithm is used to find the minimum spanning tree $T$ with disjoin-set operations. The running time of Kruskal’s algorithm is $O(E \lg E)$, where $E$ is the number of edges [2] [3]. In our C++ implementation, we use built-in sort algorithms whose running time is $O(n \lg n)$, where $n$ is the container size to sort the edge [4].

2.2 The Second MST
To discover the second MST, we repeatedly remove an edge from $T$ and search for the alternate path (set of edges) which connects all vertices. This process is a Kruskal’s applying to the modified G. The algorithm of MST2 is shown in Code Listing 1 MST2 algorithm. Unlike Kruskal’s which halts when all vertices are connected, MST2 performs exhaustive search on every edge which is not in the MST.

2.3 Running Time Analysis
Disjoint set is implemented with array without path compression, therefore find-set runs in $\Theta(\lg n)$ [3]. As a result, all-connected runs at $\Theta(n \lg n)$. Assume $E$ is the number of all edges in $G$ and $T$ is the number of edges of the MST. MST2 Iteration at line 4 run exactly $T$ times and line 8 for $E - T$ times, note that $T + (E - T) = E$; thus these two loops run at most $\left(\frac{E}{2}\right)^2 $ times. Line 8 to 11 are Kruskal’s algorithm running on a modified graph. And all-connected calls are varied by the following analysis.

2.3.1 Worst Case
MST2 discovers the second MST at the last iteration. Each iteration $\text{mst2cost}$ gets updated to a lower value. Therefore, all-connected is run every iteration in line 12. In worst case, $E \geq T$, therefore the running time only consider term $E$.

$$T(E) = \left(\frac{E}{2}\right)^2 \left(\left(\frac{E}{2}\right) \lg E + \left(\frac{E}{2}\right) \lg E\right)$$
$$= \left(\frac{E}{2}\right)^2 \left(2 \left(\frac{E}{2}\right) \lg E\right)$$
$$= \frac{E^2}{2} \lg E$$
$$= O(E^2 \lg E)$$
2.3.2 Best Case

MST2 discovers the second MST in the first iteration and update mst2cost for the first time, mst2cost is used to discard the rest of all edge adding, hence all-connected is run exactly one time. Another best situation is when the difference between T and E is minimal;

1. T and E are the same, where no further edge elimination is required.
2. |T − E| = 1 then the number of iteration required is E, because (E)(1) = E.

From these observation, we calculate the lower bound as follow. The first (E)((1)(lgE)) is the iteration for eliminating edge e from T and run Kruskal's on a single available edge, and the latter E lg E is the only all-connected call.

\[
T(E) = (E)((1)(lgE)) + (E lg E) \\
= 2E lg E \\
= \Omega(E lg E)
\]

MST2(T):
1: T\text{new} = T
2: T2 = ∅
3: mst2cost = ∞
4: for each edge e ∈ T:
5: cost = w(T)
6: T\text{new} = T\text{new} − e
7: cost = cost − w(e)
8: for each edge (u,v) ∈ T:
9: if find-set(u) ≠ find-set(v):
10: T\text{new} = T\text{new} ∪ {(u,v)}
11: cost = cost + w(u,v)
12: if cost < mst2cost and all-connected(T\text{new}):
13: mst2cost = min(cost, mst2cost)
14: T2 = T\text{new}
15: return T2

Code Listing 1 MST2 algorithm

all-connected(T):
1: v∅ = V(T)[∅]
2: p = find-set(v∅)
3: for each vertex v ∈ (V(T) − v∅):
4: if find-set(v) ! = p:
5: return False
6: return True

Code Listing 2 all-connected algorithm
3. Proof of Correctness

The proof consists of three parts:

(1) the proof that shows that the algorithm produces a spanning tree,
(2) the proof that shows that the spanning tree is of minimal weight, and
(3) the proof that shows that there exists the spanning tree is of next to minimal weight.

Kruskal’s algorithm has been commonly discussed in Graph Theory. And so the proof for Part 1 and 2 of Kruskal’s algorithm are also shown in [2]. Let’s shift our focus to Part 3.

We claim that there exists a MST $T$ for graph $G$. Removing an edge $e$ in $T$ from $G(E)$ creates a new graph $G’$. Kruskal’s algorithm yields a MST $T’$ for $G’$ from the prior claim. Since we will generate multiple $G’$ for any edge removal from $G$, thus the lightest MST $T’$ on $G’$ is the second MST of $G$, where $T’ \neq T$.

4. The Implemented Program

The followings are MST (Kruskal’s algorithm) and MST2 functions. The implementation of supplementary classes: Path and DisjointSets are omitted and the full source code is shown in Appendix A.

```cpp
void mst()
{
    // first best
    int cost = 0;

    list<Path *> E = paths;
    E.sort(comp_w_asc);
    DisjointSets d(n);

    for (Path *p : E)
    {
        if (d.findset(p->a) != d.findset(p->b))
        {
            bestRoute.insert(p);
            d.unionset(p->a, p->b);
            cost += p->w;
        }
    }

    cost1 = cost;
}
```

**Code Listing 3 Kruskal’s algorithm implemented in C++**
void mst2()
{
    // second best
    int cost = 0;
    int minCost = INT_MAX;
    list<Path *> bestRouteSorted = bestRoute.begin(), bestRoute.end();
    bestRouteSorted.sort(comp_w_asc);

    list<Path *> E = paths;
    DisjointSets d(n);

    for (Path *pbr : bestRouteSorted)
    {
        cost = 0;
        d.reset();
        for (Path *p : bestRouteSorted)
        {
            if (p != pbr)
            {
                d.unionset(p->a, p->b);
                cost += p->w;
            }
        }

        for (Path *p : E)
        {
            if (p != pbr)
            {
                if (d.findset(p->a) != d.findset(p->b))
                {
                    d.unionset(p->a, p->b);
                    cost += p->w;
                    if (cost >= minCost)
                    {
                        break;
                    }
                }
            }
        }
    }

    if (cost >= minCost || !d.allConnected())
    {


```cpp
    cost = -1;
    }

    if (cost != -1 && cost < minCost) {
        minCost = cost;
    }
    }

    if (minCost == INT_MAX) {
        cost2 = -1;
    } else {
        cost2 = minCost;
    }
```

**Code Listing 4 MST2 algorithm implemented in C++**

### 5. The Example Executions

Our solution successfully passed the TIMUS Online Judge as shown in Figure 1.

**Figure 1 Time Judge Result**

### 6. Test Cases

The list of edges is sorted once for Kruskal’s algorithm and then it is used again in MST2. In this way, we feed into MST2 with lightest edge first. The first discovery of the mst2cost then helps skip calling all-connected entirely. However, it cannot be certain that the mst2cost discovered this way will be the only answer. The following graph shows the running time of the solution where X and Y axes are number of edges and time in seconds. We generate test cases with maximum possible random-weight edges which connects all vertices; thus $E = \frac{(V-1)(V)}{2}$. Figure 2 shows the performance on the max-edges data set. The growth is linear without quadratic nor logarithmic behaviours however it is between our calculated lower and upper bounds. The maximum input range to the problem is $V \leq 500$ which the maximum number of $E$ is only 124,750 edges. But without pruning to prevent calling unnecessary all-connected, the solution performs significantly slower and exceeded time limit on TIMUS Online Judge.
Possible Improvement

The proposed solution has rooms to improve when comparing to other solutions published on TIMUS. Some recommendations are (1) improve it further by optimising the `find-max` with path compression, and (2) use other problem solving techniques such as dynamic programming which TIMUS discussion claims that it yields better performance.

Conclusions

We have shown a solution to the problem 1416: Confidential of TIMUS. Kruskal’s algorithm and its variation have been applied to the solution. The proof of correctness is established and we also calculated the running time and run against a set of test cases.

References


Appendix A: Source Code

```cpp
#include <iostream>
#include <list>
#include <set>
#include <climits>
#include <ctime>

using namespace std;

class Path
{
public:
    int a;
    int b;
    int w;

    Path(int a, int b, int w)
    {
        this->a = a;
        this->b = b;
        this->w = w;
    }
};

int cost1 = 0;
int cost2 = 0;
int n, m;
list<Path *> paths;
set<Path *> bestRoute;

bool comp_w_asc(const Path *a, const Path *b)
{
    return a->w < b->w;
}

class DisjointSets
{
    public:
        int *p;
        int *rank;
        int size;

        DisjointSets(int size)
        {
            this->size = size;
            reset();
        }

        void reset()
        {
            p = new int[size + 1];
            rank = new int[size + 1];
            for (int i = 1; i < size + 1; i++)
            {
                p[i] = i;
                rank[i] = 0;
            }
        }
};
```
int findset(int u)
{
    if (p[u] == u)
    {
        return u;
    }
    else
    {
        p[u] = findset(p[u]);
        return p[u];
    }
}

void unionset(int u, int v)
{
    int a = findset(u);
    int b = findset(v);
    if (rank[a] < rank[b])
    {
        p[a] = b;
    }
    else
    {
        p[b] = a;
        if (rank[a] == rank[b])
        {
            rank[a] += 1;
        }
    }
}

bool allConnected()
{
    int root = findset(1);
    for (int i = 2; i < size + 1; i++)
    {
        if (findset(i) != root)
        {
            return false;
        }
    }
    return true;
}

void mst()
{
    // first best
    int cost = 0;
    list<Path *> E = paths;
    E.sort(comp_w_asc);
    DisjointSets d(n);
    for (Path *p : E)
    {
        if (d.findset(p->a) != d.findset(p->b))
        {
            bestRoute.insert(p);
            d.unionset(p->a, p->b);
            cost += p->w;
        }
    }
    cost1 = cost;
}
void mst2()
{
    // second best
    int cost = 0;
    int minCost = INT_MAX;
    list<Path *>
bestRouteSorted(bestRoute.begin(),
    bestRoute.end());
    bestRouteSorted.sort(comp_w_asc);

    list<Path *> E = paths;
    DisjointSets d(n);

    for (Path *pbr : bestRouteSorted)
    {
        cost = 0;
        d.reset();
        for (Path *p : bestRouteSorted)
        {
            if (p != pbr)
            {
                d.unionset(p->a, p->b);
                cost += p->w;
            }
        }
        for (Path *p : E)
        {
            if (p != pbr)
            {
                if (d.findset(p->a) != d.findset(p->b))
                {
                    d.unionset(p->a, p->b);
                    cost += p->w;
                    if (cost >= minCost)
                    {
                        break;
                    }
                }
            }
        }
        if (cost >= minCost || !d.allConnected())
        {
            cost = -1;
        }
        if (cost != -1 && cost < minCost)
        {
            minCost = cost;
        }
    }
}
if (minCost == INT_MAX)
{
    cost2 = -1;
}
else
{
    cost2 = minCost;
}

int main()
{
    clock_t start = clock();
    cin >> n >> m;

    int a, b, w;
    for (int i = 0; i < m; i++)
    {
        cin >> a >> b >> w;
        paths.push_back(new Path(a, b, w));
    }

    // Solve
    mst();
    mst2();

    // Output
    cout << "Cost: " << cost1 << endl;
    cout << "Cost: " << cost2 << endl;

    // Time
    clock_t end = clock();
    double seconds = double(end - start) / CLOCKS_PER_SEC;
    cout << "Elapsed Time: " << seconds << endl;
}
Appendix B: The Problem

Zaphod Beeblebrox — President of the Imperial Galactic Government. And by chance he is an owner of enterprises that trade in secondhand pens. This is a complicated highly protable and highly competitive business. If you want to stay a leader you are to minimize your expenses all the time. And the president's high post helps in those affairs. But he is to keep this business in secret. As a president Zaphod has access to the top secret and important information an exact value of power loss in the hyperspace transition between the planets. Of course, this information is very useful to his company Zaphod is to choose the minimal possible set of trans-planet passages so that he could pass from any planet to any other one via those passages and their total cost would be minimal. The task won't be complicated if Zaphod was not to keep in secret that he helps his company with the secret information. Thus, Zaphod decided to find not the cheapest passages set but the next one. As a real businessman he wants to estimate the value of his conspiracy expenses.

Input
The first line contains integers \( n \) and \( m \) that are a number of planets in the Galaxy and an amount of passages between them \( (2 \leq n \leq 500) \). The next \( m \) lines contain integers \( a_i, b_i, \) and \( w_i \) that are the numbers of the planets connected with the passage and the transition cost \( (1 \leq a_i, b_i \leq n; 0 \leq w_i \leq 1000) \). If an \( A \) to \( B \) transition is possible then a \( B \) to \( A \) transition is possible too, and the cost of these transitions are equal. There is no more than one passage between any two planets. One can reach any planet from any other planet via some chain of these passages.

Output
You should find two different sets of transitions with the minimal possible cost and output theirs costs. Print the minimal possible cost first. If any of those sets of transitions does not exist denote it's cost by \(-1\).

Samples

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 6</td>
<td>Cost: 4</td>
</tr>
<tr>
<td>1 2 2</td>
<td>Cost: 4</td>
</tr>
<tr>
<td>2 3 2</td>
<td></td>
</tr>
<tr>
<td>3 4 2</td>
<td></td>
</tr>
<tr>
<td>4 1 2</td>
<td></td>
</tr>
<tr>
<td>1 3 1</td>
<td></td>
</tr>
<tr>
<td>2 4 1</td>
<td></td>
</tr>
<tr>
<td>3 2</td>
<td>Cost: 4</td>
</tr>
<tr>
<td>1 2 2</td>
<td>Cost: -1</td>
</tr>
<tr>
<td>2 3 2</td>
<td></td>
</tr>
</tbody>
</table>