Design and Analysis of Algorithms Assignment - 6

Department of Information Technology,

Indian Institute of Information Technology, Allahabad 211015, India

SUBHASH BALLA(IIT2019207), DHANUSH VASA (IIT2019208)

INTRODUCTION

We are given an un-directed weighted graph.our goal is to find build minimum spanning tree using kruskal's algorithm.

Minimum Spanning Tree- A sub graph(basically a tree)made from set of edges of input graph G(m,n) such that the graph still remains connected and sum of weights of all edges is minimum. MST's have a lot of practical applications in real world.

ALGORITHM DESIGN

Given graph is, **G(m,n)**
m -set of edges with correspong weighrs
n -set of vertices

Kruskal's Algorithm is based on **cyclic property of graphs**.
Firstly,we sort all edges with respect to weights in **ascending order**.Then we consider each edge sequentially and check if those 2 vertices forming that edge are in same set or not.
If they are in same set,then including that edge creates a cycle which we don't want, since those 2 vertices are already connected,including that edge is useless.Hence we exclude that edge.
If they belong to 2 different sets ,we include that edge in our MST and combine those 2 sets.
We use **disjoint-set-union(DSU) data structure** to work with sets.

Code Implementation

```
#include<bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(false),cin.tie(NULL)
#define mod 1000000007
#define  pb push_back
typedef long long ll;
typedef unsigned long long ull;
typedef  long double ld;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<vi >vvi;
typedef vector<ll> vll;
typedef vector<vll >vvll;


struct edge_ {
        ll v1, v2, wt;
};

ll max_vertices = 25, max_weights = 10;
ll n, m, u, v, w, cost;
ll _rank[1001], par[1001];
vector<edge_>edges, result;


void create_random_graph() {

        n = 50 + rand() % max_vertices;
```

```
        ll max_edges = n * (n - 1) / 2;
        m = 1 + rand() % max_edges;


        ll edge_count = m;
        set<pair<ll, ll> >s;

        while (edge_count) {
                ll v1 = 1 + rand() % n, v2 = 1 + rand() % n;
                if (v1 == v2)continue;
                if (s.find({v1, v2}) != s.end() || s.find({v2, v1}) != s.end())continue;
                ll wt = 1 + rand() % max_weights;
                edges.pb({v1, v2, wt});
                s.insert({v1, v2});
                edge_count--;
        }

}


bool comp(const edge_ &e1, const edge_ &e2) {
        return (e1.wt < e2.wt);
}

ll find(ll v1) {

        if (par[v1] == -1)return v1;

        return par[v1] = find(par[v1]);
}

void combine(ll v1, ll v2) {

        ll a = find(v1), b = find(v2);

        if (_rank[a] == _rank[b])_rank[b]++, par[a] = b;
        if (_rank[a] < _rank[b])par[a] = b;
        else par[b] = a;
}


int main() {

        srand(time(0));

        create_random_graph();

        cout << "n  ---> " << n << "\n";
        cout << "m  ---> " << m << "\n";
        cout << "\ninput graph--->\n";
        for (auto ele : edges)cout << ele.v1 << " " << ele.v2 << " " << ele.wt << "\n";


        for (ll i = 0; i <= n; i++)par[i] = -1;

        sort(edges.begin(), edges.end(), comp);


        for (ll i = 0; i < m; i++) {
                ll v1 = edges[i].v1, v2 = edges[i].v2;
                ll a = find(v1), b = find(v2);
                if (a != b) {
                        combine(a, b);
                        result.pb({v1, v2, edges[i].wt});
                        cost += edges[i].wt;
                }
        }

        cout << "\ncost ---> " << cost << "\n";
        cout << "\nminimum spanning forest --->\n";

        for (auto ele : result)cout << ele.v1 << " " << ele.v2 << " " << ele.wt << "\n";
```

```
}
```

ALGORITHM ANALYSIS

Time-
sorting step -$O(m * log(m))$.
For each check if both vertices belong to same set or not(find() in dsu) and combine 2 sets (union() in dsu) if they belong to different sets.

In our implementation of DSU,we used both **rank-heursitic** and **path compression technique**.
Hence find() and union() works in $\alpha(m, n)$ time,where $\alpha(m, n)$ is inverse ackermann function.
It doesn't cross '5' for any practically large real input.
Hence **time taken=**$O(m * log(m) + m * \alpha(m, n))$.

Space-
We used rank[] and parent[] arrays to work with sets ,which are linear.
Hence,**Auxillary space** =$O(m + n)$

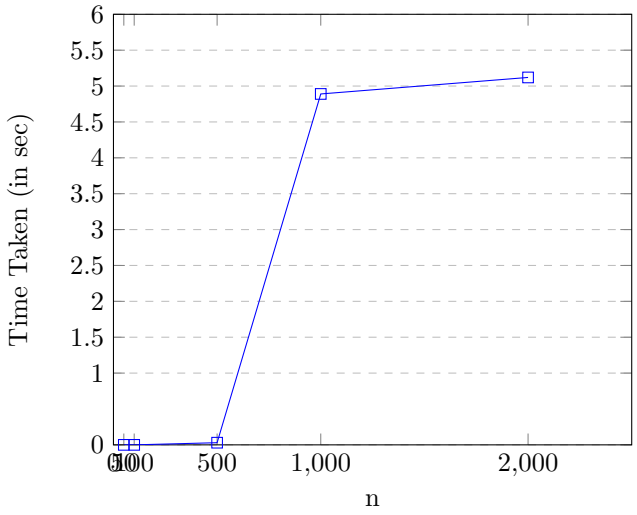
---

1 Experimental Analysis

In the following table some cases are plotted for the first algorithm,

m	n	Time Taken (in ms)
50	662	0.0000
100	4011	0.0000
500	22238	0.03
1000	497566	4.89
2000	900937	5.12

Algorithm 1



Discussion on Parallelising Kruskal's Algorithm

Kruskal's algorithm involves
1. sorting step.

2. determing if 2 vertices belong to same set or not and combine if belong to different sets.
Second task is difficult to parallelise .Because ,say 2 edges are being checked for that condition and calling a union() function by 2 different processors simultaneously,we could get incorrect results for find() and there is possibility that we combine 2 incorrect sets.
This is due to concurrency issues when working with multi processor systems.

However,**sorting step can be parallelised.**
Sorting can be done in **linear time** if worked with **O(log(m)) processors**.

Hence **Time taken** = O(n+m*$\alpha(m,n)$) = $O(m * \alpha(m,n))$.

This is better than usual serial kruskal's algorithm.

CONCLUSION

We can conclude that the given problem of finding minimum spanning tree(MST) of a given graph works in $O(m * log(m) + m * \alpha(m,n))$ time and consumes $O(m+n)$ space.

ACKNOWLEDGMENT

We are very much grateful to our Course instructor Mr.Rahul Kala and our mentor, Ms.Tejasvee, who have provided the great opportunity to do this wonderful work on the subject of Design and Analysis of Algorithms, specifically on MST's and kruskal's algorithm.

REFERENCES

1.https://en.wikipedia.org/wiki/Kruskal%27s_$algorithm$