Walchand College of Engineering, Sangli
Computer Science & Engineering
Third Year

Course: Design and analysis of algorithm Lab

Lab course coordinator:
Mrs A M Chimanna- Batch: - T1, T2,T3,T4

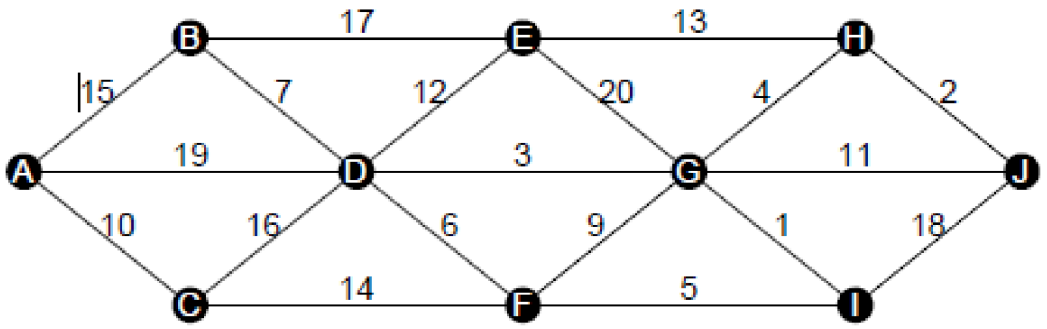
Week 7 Assignment

Greedy method

PRN: 21510111.

Name: Siddharth Shrinivas Salunkhe

Implement Kruskal's algorithm & Prim's algorithm to find **M**inimum **S**panning **T**ree (*MST*) of the given an undirected, connected and weighted graph.



Q) How many edges does a minimum spanning tree have for the above example?

Solution: Number of Edges in MST for above example=Number of Node-1

=10-1

Number of Edges in MST for above example=9.

Q) In a graph *G*. Let the edge *u v* have the least weight. Is it true that *u v* is always part of any minimum spanning tree of *G*?.Justify your answers.

Solution:

Yes,it is true that the edge uv with least weight is always part of the minimum spanning tree of G.

To justify this statement, let's consider the properties of a minimum spanning tree and the definition of the MST:

1. A minimum spanning tree is a spanning tree (a subgraph that includes all vertices of the original graph) that has the smallest possible total edge weight among all spanning trees of the graph.

2. In any connected graph, a spanning tree exists, and the number of edges in a spanning tree is equal to the number of vertices minus one.

3. When you add an edge to a spanning tree, it creates a cycle.

Now, let's consider a graph G, and we have an edge (u, v) with the least weight. If we exclude this edge from the graph, it will disconnect the graph into two components. Let's assume that the two components are A and B.

If (u, v) is not included in the MST, then there must be some other edge(s) that connects the components A and B in the MST. Let's call one of these edges (x, y).

Now, we have two cases to consider:

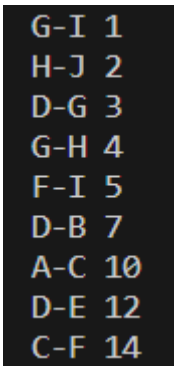
Case 1: If the weight of (x, y) is greater than or equal to the weight of (u, v), then we can replace (x, y) with (u, v) in the MST to obtain a spanning tree with a smaller total weight, which contradicts the definition of the MST.

Case 2: If the weight of (x, y) is less than the weight of (u, v), we can create a cycle in the MST by adding (u, v) and removing (x, y). This cycle will contain the edges (x, y), (u, v), and at least one other edge in the MST. We can remove one of the other edges from the cycle to obtain a new spanning tree with a smaller total weight, which again contradicts the definition of the MST.

In both cases, we have a contradiction, which means that the edge (u, v) with the least weight must be part of any minimum spanning tree of the graph G.

Therefore, it is always true that the minimum-weight edge is part of any minimum spanning tree of the graph.

Also for given above example, edge GI has least weight and is part of MST



Q) Let *G* be a graph and T be a minimum spanning tree of *G*. Suppose that the weight of an edge e is decreased. How can you find the minimum spanning tree of the modified graph? What is the runtime of your solution?

Solution: When the weight of an edge e in the graph G is decreased, you can find the minimum spanning tree (MST) of the modified graph using the following approach:

1. Remove the edge e from the original MST T. This will disconnect the MST into two subtrees, say, T1 and T2, where T1 contains one endpoint of e, and T2 contains the other endpoint of e.

2. Now, you have two disjoint subtrees, T1 and T2, and the modified graph after decreasing the weight of edge e.

3. Calculate the minimum-weight edge (let's call it e') between T1 and T2 in the modified graph. This can be done using a standard minimum spanning tree algorithm like Kruskal's or Prim's algorithm with a few modifications.

4. Add the edge e' to the MST T. This will form a new MST in the modified graph.

This approach ensures that the new MST of the modified graph still spans all the vertices and has the minimum possible total weight. It's essentially an incremental approach where you update the existing MST to adapt to the changes in the graph.

The runtime of this solution depends on the MST algorithm used. The most common algorithms for finding the MST are Kruskal's algorithm and Prim's algorithm. The runtime complexities of Kruskal's Algorithm: O(E log E), where E is the number of edges in the graph.

In the case of updating the MST after decreasing the weight of an edge, the runtime complexity will be determined by the MST algorithm you choose and the specific data structures you use for maintaining the MST. Therefore, it would typically be O(E log E) or O(E + V log V) in practice.

CODE FOR UPDATION:

```
#include<bits/stdc++.h>
using namespace std;
class Ds{
    vector<int> rank,parent;
    public:
    Ds(int n){
        rank.resize(n+1,0);
        parent.resize(n+1);
        for(int i=0;i<=n;i++)parent[i]=i;
    }
    int findUparent(int node){
        if(parent[node]==node)return node;
        return parent[node]=findUparent(parent[node]);
    }
    void unionbyrank(int u,int v){
        int up=findUparent(u);
        int vp=findUparent(v);
        if(rank[up]>rank[vp])parent[vp]=up;
        else if(rank[up]<rank[vp])parent[up]=vp;
        else{
            parent[up]=vp;
            rank[vp]++;
        }
    }
};
```

```
void
mst(vector<pair<int,pair<char,char>>>&ori_graph,vector<pair<int,pair<ch
ar,char>>>&ans_graph,int &sum,int numNodes){

    sort(ori_graph.begin(),ori_graph.end());

    Ds *mst=new Ds(numNodes);

    for(int i=0;i<ori_graph.size();i++){

        int u=ori_graph[i].second.first-'A';

        int v=ori_graph[i].second.second-'A';

        if(mst->findUparent(u)!=mst->findUparent(v)){

            mst->unionbyrank(u,v);

            sum += ori_graph[i].first;

            ans_graph.push_back(ori_graph[i]);

        }

    }

}

int main(){

    vector<pair<int,pair<char,char>>>ori_graph,ans_graph,updated_graph;

    int numNodes=10,numEdges=19;

    ori_graph={{15,{'A','B'}},

            {19,{'A','D'}},

            {10,{'A','C'}},

            {7,{'D','B'}},

            {16,{'C','D'}},

            {17,{'B','E'}},

            {14,{'C','F'}},

            {12,{'D','E'}},

            {6,{'D','F'}},

            {3,{'D','G'}},

            {20,{'E','G'}},

            {9,{'F','G'}},
```

```
            {13,{'E','H'}},
            {5,{'F','I'}},
            {11,{'G','J'}},
            {4,{'G','H'}},
            {18,{'I','J'}},
            {2,{'H','J'}},
            {1,{'G','I'}}};
    int sum=0;
    mst(ori_graph,ans_graph,sum,numNodes);
    cout<<"sum: "<<sum<<endl;
    for(auto x:ans_graph) cout<<x.second.first<<"-"<<x.second.second<<"
"<<x.first<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
    char a,b;
    int x;
    cout<<"Enter nodes of edge you want to update:";
    cin>>a>>b;
    cout<<"Enter Updated weight of that edge:";
    cin>>x;
    for(int i=0;i<numEdges;i++)
    {
                                        if(ori_graph[i].second.first==a
&&ori_graph[i].second.second==b)ori_graph[i].first=x;
                                  else    if(ori_graph[i].second.first==b
&&ori_graph[i].second.second==a)ori_graph[i].first=x;
    }
    cout<<"Updated MST:"<<endl;
```

```
    sum=0;

    mst(ori_graph,updated_graph,sum,numNodes);

    cout<<"sum: "<<sum<<endl;

                                        for(auto          x:updated_graph)
cout<<x.second.first<<"-"<<x.second.second<<" "<<x.first<<endl;

    return 0;

}
```

OUTPUT:

```
sum: 58
G-I 1
H-J 2
D-G 3
G-H 4
F-I 5
D-B 7
A-C 10
D-E 12
C-F 14



Enter nodes of edge you want to update:A C
Enter Updated weight of that edge:8
Updated MST:
sum: 56
G-I 1
H-J 2
D-G 3
G-H 4
F-I 5
D-B 7
A-C 8
D-E 12
C-F 14
PS C:\Users\ASUS\Documents\Ty\vait_divas\Daa_lab\Assignment 7>
```

Q) Find order of edges for Kruskal's and Prim's?

For Kruskal's Algorithm:

```
#include<bits/stdc++.h>
using namespace std;
class Ds
{
    vector<int> rank,parent;
    public:
    Ds(int n)
    {
        rank.resize(n+1,0);
        parent.resize(n+1);
        for(int i=0;i<=n;i++)parent[i]=i;
    }
    int findUparent(int node)
    {
        if(parent[node]==node)return node;
        return parent[node]=findUparent(parent[node]);
    }
    void unionbyrank(int u,int v)
    {
        int up=findUparent(u);
        int vp=findUparent(v);
        if(rank[up]>rank[vp])parent[vp]=up;
        else if(rank[up]<rank[vp])parent[up]=vp;
        else
        {
            parent[up]=vp;
```

```
            rank[vp]++;
        }
    }
};

int main()
{
    vector<pair<int,pair<char,char>>>ori_graph,ans_graph;
    int numNodes=10,numEdges=19;
    ori_graph={{15,{'A','B'}},
            {19,{'A','D'}},
            {10,{'A','C'}},
            {7,{'D','B'}},
            {16,{'C','D'}},
            {17,{'B','E'}},
            {14,{'C','F'}},
            {12,{'D','E'}},
            {6,{'D','F'}},
            {3,{'D','G'}},
            {20,{'E','G'}},
            {9,{'F','G'}},
            {13,{'E','H'}},
            {5,{'F','I'}},
            {11,{'G','J'}},
            {4,{'G','H'}},
            {18,{'I','J'}},
            {2,{'H','J'}},
            {1,{'G','I'}}};
```

```
    sort(ori_graph.begin(),ori_graph.end());
    Ds *mst=new Ds(numNodes);
    int sum=0;
    for(int i=0;i<ori_graph.size();i++)
    {
        int u=ori_graph[i].second.first-'A';
        int v=ori_graph[i].second.second-'A';
        if(mst->findUparent(u)!=mst->findUparent(v))
        {
            mst->unionbyrank(u,v);
            sum += ori_graph[i].first;
            ans_graph.push_back(ori_graph[i]);
        }
    }
    cout<<"sum: "<<sum<<endl;
    for(auto x:ans_graph) cout<<x.second.first<<"-"<<x.second.second<<"
"<<x.first<<endl;
    return 0;
}
```

Order of Edges:

```
sum: 58
G-I 1
H-J 2
D-G 3
G-H 4
F-I 5
D-B 7
A-C 10
D-E 12
C-F 14
```

For Prim’s Algorithm

```
#include<bits/stdc++.h>

using namespace std;

int main(){

    int numNodes=10,numEdges=19;

    map<char,vector<pair<int,char>>> mp;

    vector<pair<int,pair<char,char>>> ori_graph;

    vector<int>visited(numNodes,0);

    ori_graph={{15,{'A','B'}},

            {19,{'A','D'}},

            {10,{'A','C'}},

            {7,{'D','B'}},

            {16,{'C','D'}},

            {17,{'B','E'}},

            {14,{'C','F'}},

            {12,{'D','E'}},

            {6,{'D','F'}},

            {3,{'D','G'}},

            {20,{'E','G'}},

            {9,{'F','G'}},

            {13,{'E','H'}},

            {5,{'F','I'}},

            {11,{'G','J'}},

            {4,{'G','H'}},

            {18,{'I','J'}},

            {2,{'H','J'}},

            {1,{'G','I'}}};
```

```
    for(auto x:ori_graph)
    {
        mp[x.second.first].push_back({x.first,x.second.second});
        mp[x.second.second].push_back({x.first,x.second.first});
    }

priority_queue<pair<int,char>,vector<pair<int,char>>,greater<pair<int,c
har>>> pque;
    pque.push({0,'A'});
    int sum = 0;
    while(pque.size())
    {
        pair<int,char> p=pque.top();
        pque.pop();
        if(!visited[p.second-'A'])
        {
            sum += p.first;
             cout<<"Next node included is:- '"<<p.second<<"'  and Sum of
weight of MST till now: "<<sum<<endl;
        }
        visited[p.second-'A'] ++;
        for(auto x:mp[p.second])
        if(!visited[x.second-'A'])
        pque.push({x.first,x.second});
    }
    cout<<"Total sum:"<<sum<<endl;
    return 0;
}
```

Order of Edges: