Week 3 - Lecture 6

Introduction:

In this lecture,we studied some of the Greedy algorithms and how to solve them effectively.

We primarily discussed the following problems:

- What is Minimum Spanning Tree?
- What is Kruskal's Algortihm and Cut Property?
- Disjoint Set Union Data Structure?

Minimum Spanning tree:

Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together.

There can be multiple spanning tree but the one among them with the minimum weight of the all the edge weights is known as the Minimum Spanning Tree.

So for a given graph with n vertices and m edges , the MST would have n vertices and n-1 edges to ensure the graph remains connected and these n-1 edges would be a subset of the original m edges.

The minimum Spanning problem is thus stated formally as:

Given an undirected graph G = (V,E); edge weights , find a tree T = (V,E'), with , that minimises total edge weights.

Cut Property:

In graph Theory, a cut is defined as a partition that divides a graph into two disjoint subsets.This means that say we have two sets of edges S_1 and S_2 ,then cut property ensures there won't be any edge between the two sets .

Let us assume we have already picked up some edges (x) that are part of the MST and now want to add another edge into it.

So the cut property says that that pick any subset of nodes S for which X does not cross between S and (V-S) , if e is the lightest edge in this partition that joins both the sets , then according to cut property, that edge will be a part of MST.

Proof of CUT Property:

- Say we have already chosen X edges in our MST T.
- Consider some edge e joining some sets (S,V-S) , If it is part of the MST , then we have nothing to prove.
- Else,let's consider a different MST T' which contains X union {e}.
- When we add an edge e to T , T will contain a cycle.This is because T was already connected so adding an edge would crate a cycle . But since it is cycle, it must have some edge e' across the partition (S,V-S).
- If we remove this edge , then again we would have a tree T' which consists of the edges X+(e}-{e'}. Since we removed e' , T' won't contain a cycle and since it has n-1 edges, it would also remain connected.
- So we have another contender for MST , that is T'. Now according to cut Property , edge e is the lightest edge between the cut (S,V-S).
- Therefore, edge weight of e' has to be to greater than equal to e'.
- So tree T' has less weight than T. But this is a contradiction since we initially assumed that X belongs to MST.
- Hence, we have proved the cur property.

Kruskal's Algorithm:

This is a greedy algorithm. Before we start, we must sort all edges according to the edge weight.

We start with an empty graph and repeatedly add the next lightest edge that doesn't produce a cycle.

At some moment of time, say we have included set S of vertices in our MST. Now Kruskal says that we add the edge between the partitions (S,V-S) that does not add a cycle in our graph and has minimum weight.

So from the cut property , we know that this is the most optimal thing to do. Hence proved

```
function Kruskal(G,w):
for all u in V:
makeset(u)
X = {}
sort all edges E by weight
for all edges {u,v} in E, in increasing order of weight:
if find(u) != find(v):
add edge {u,v} to X
union(u,v)
```

Disjoint Set Union :

To implement the find and union function, we would need to take help of the DSU data structure.

It is used to do union between two disjoint sets and assign a parent as a representative to each of those sets.

The find function returns the parent of the current node and it can be implemented as follows:

Find Function:

```
int root(int x) {
if (x == parent[x]) return x;
return root(parent[x]);
}
```

This can run O(n) time in worst case when we have a linear tree.

So to fasten it up ,We can do Path compression:

Path Compression:

If we have found the parent of some vertex v (say p), then while going to the parent of v (p) using the recursive call, we can simultaneously update parent of all intermediate vertices as p.

```
int find(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find(parent[v]);
}
```

Union Function:

This function is used for unifying two disjoint sets. This can be implemented in a naive way as follows:

```
void union(int a, int b) {
    a = find(a);
    b = find(b);
    if (a != b) {
        if (size[a] < size[b])
            swap(a, b);
        parent[b] = a;
        size[a] += size[b];
    }
}
```

But this would be too slow , so to optimise it we can do Union by rank.

Union by Rank:

Union by Rank is used to optimise the union process, so instead of merging two sets in a random manner, we make the parent of the set with lesser cardinality equal to parent of the set with more cardinality.

```
void union(int a, int b) {
    a = find(a);
    b = find(b);
```

```
    if (a != b) {
        if (size[a] < size[b])
            swap(a, b);
        parent[b] = a;
        size[a] += size[b];
    }
}
```

Both of these optimisations make the DSU implementation almost constant time.

Kruskal's Time Complexity:

$$|E|log|E|$$

The complexity of Kruskal's is dominated by the time required to sort all the edges because we only need to go though those m edges and we have alraedy optimised our DSU to almost contant time.

So final complexity is dominated by the complexity of sorting.

Extra Materials:

I referred to cp algorithms to study more about DSU and Kruskal algorithm.

https://cp-algorithms.com/data_structures/disjoint_set_union.html

https://cp-algorithms.com/graph/mst_kruskal_with_dsu.html

I also solved the following problems to improve my efficiency in these topics:

- https://codeforces.com/problemset/problem/1513/D
- https://codeforces.com/contest/472/problem/D