Algorithms Analysis and Design

Week 3 - Diary

Ayan Agrawal (2020101034)

Lecture 6 : Greedy Algorithms

In this lecture, we discussed the following topics :-

1. ***Minimum Spanning Trees***
    - Kruskal's Algorithm
    - Cut Property
    - Pseudo code for Kruskal's Algorithm
2. ***Disjoint Set - Data structure***
    - Makeset, find and union functions
    - Properties of Rank(x)

1. Minimum Spanning trees

- *Given an undirected graph G = (V,E) with edge weights w_e , we need to find a tree T = (V,E') such that $E' \subseteq E$ and it minimises the weight of the tree.*

1.1 Kruskal's Algorithm

- It is a Greedy Algorithm which is used to find out a graph's MST.
- *We start with an empty graph and repeatedly keep adding the next least weighted edge along with keeping in check that it doesn't produce a cycle. We do this until it is not possible further. By the help of Cut property, we will prove its correctness.*

1.2 Cut Property

Theorem :

- *Suppose edges X are part of a MST of $G = (V, E)$. Pick any subset of nodes say S for which X doesn't cross between S and $V - S$, and let e be the lightest edge across this partition, then $X \cup \{e\}$ is a part of some MST.*

Proof :

- It is trivial if e is a part of T. Now, if e isn't a part of T, then let's construct another MST say T' , which contains $X \cup \{e\}$. Now, since T is a spanning tree, it has exactly 1 edge connecting S and $V - S$. Let this edge be e' , now since e is the lightest edge connecting S and $V - S$, then $w_{e'} = w_e$.

$$\text{weight } (T \cup \{e\} - \{e'\}) = \text{ weight } (T) + w_e - w_{e'} = weight(T)$$

$T \cup \{e\} - \{e'\}$ is also a tree because it contains no cycle and contains $n-1$ edges. Therefore, $T \cup \{e\} - \{e'\}$ is also a MST of G. This proves the *Cut property* and along with that, *Kruskal's algorithm* also.

1.3 Pseudo code for Kruskal's algorithm

- We use Disjoint sets here, which is a data structure.

```
# G is an un-directed graph
# w is weights of edges of G

def kruskal(G,w):
    for node in V:
        makeset(node)

    X = {}
    sort(G,w) # function to sort edges in E in ascending order of
weights.

    for edges{u,v} in E:
        if find(u) != find(v):
            X.insert({u,v})
            union(u,v)
```

- Functions find, makeset and union will be discussed further. Now, computing the time complexity of Kruskal's algorithm,

$$\underbrace{O(|E| \log |E|)}_{\text{sorting } |E| \text{ edges}} + \underbrace{O(|E| \log^* |V|)}_{\text{find()}}$$

where, $\log^* |V|$ is the min. number of $\log$ steps required to make $|V| \leq 1$.

Now, we will discuss the Disjoint set - data structure because it played an important role in Kruskal's algorithm, also we need to explore the functions find, union and makeset.

2. Disjoint Set Union - Data structure

- Basically, a DSU comprises of 3 functions which are makeset, find and union. Now, we will discuss these functions one-by-one and also see their pseudo codes.

1. $makeset(v)$:

- It is a function which creates a new singleton set say S which comprises of element v. Rank of a node is the height of the sub-tree hanging from that node.

```
def makeset(v):
    parent(v) = v # parent pointer
    rank(v) = 0 # rank of v is height of subtree from v
```

2. $find(v)$:

- Return the *root* of the set containing v, which is an element in set same as v.

```
def find(v):
    if v == parent(v):
        return v
    return parent(v)
```

Above algorithm has a worst-case complexity of $O(n)$ and average complexity of $O(\log n)$.

We can definitely do better than this. ***Path Compression*** optimization would bring it down to an amortised $O(1)$ when used along with union-rank optimization.

```
def find(v):
    if v == parent(v):
        return v
    parent(v) = find(parent(v))
    return parent(v)
```

3. $union(u, v)$:

- It unifies 2 sets and returns the unified set.

```
def union(u,v):
    p = find(u)
    q = find(v)

    if p == q :
        return
    if rank(p) > rank(q):
        parent(q) = parent(p)
    else:
        parent(p) = parent(q)
        if rank(p) == rank(q):
            rank(q) = rank(q) + 1
```

These 3 functions are used in DSU data structure.

Analysis :

1. $makeset(v) : O(1)$

2. $find(v) : O(\log n)$

3. $union(u, v) : O(\log n)$

- We can also apply ***Path compression*** optimization which will improve complexity of the $find(v)$ function.

*Therefore, **Overall complexity** is :* $O((|E| + |V|) \log |V|)$

Properties of Rank(x) :

- **Property 1** : *For any x,* $rank(x) < rank(parent(x))$.
- **Property 2** : *Any root node of rank* k *has atleast* 2^k *nodes in its tree since a root node of rank* k *is formed from the merging of two root nodes of rank* $k-1$ *and we we achieve the former result by recursing this process. This property applies to all nodes, not only the root.*
- **Property 3** : *If there are* n *elements overall, there can be atmost* $\frac{n}{2^k}$ *nodes of rank* k.

Final Analysis :

- The time taken by a specific find operation is simply the number of pointers followed.
- Ranks are divided into $\log^* n$ intervals.
- Nodes x on the chain (to root) fall into two categories
  - either the rank of parent(x) is in a higher interval than the rank of x,
  - or else it lies in the same interval.
- Atmost $\log^* n$ nodes are present of 1st type.
- If x is of 2nd type,
  - its parent changes to one of higher rank.
  - As a result, rank of x always lies between $k+1$ and 2^k, also it will be of this kind atmost 2^k times before its parent's rank is in a higher interval, after which it will never become of 2nd type.
- Therefore,
  - Overall time for m find is

$$O(m \log^* n) + 2^k * (\text{ the number of nodes with rank} > \text{k which is } \leq nlog^*n)$$

This is because $\frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \frac{n}{2^{k+3}} + \frac{n}{2^{k+4}} \ldots\ldots \leq \frac{n}{2^k}$

> ***Hence, Final complexity of Kruskal's algorithm using DSU with Path Compression is***
>
> $$\text{Sorting } |E| \text{ elements } + O(E * \log^* |V|)$$