Team: Abdullah | Arsalan | Mahnoor | Sabir

Greedy Algorithm

In this algorithm design paradigm in which we try to find best and optimal solution at every step i.e. we are greedy at selections. For example lets take MST Prim's algorithm, it is also a greedy algorithm in which at each step we pick that edge with minimum wait so we are greedy at choice.

Prim's Algorithm :

In this algorithm basically we span all the vertices and at end we get a minimum spanning tree with all the vertices present and there is no cycle.
For applying prim's algorithm we must check whether graph is connected , weighted and undirected. These three conditions are must.

One Observation :

If we have dense graph i.e. if number of edges are pretty much more so its better to use prim's algorithm of MST rather than kruskal. Because in kruskal's algorithm we first sort all the edges in ascending order then pick edges one by one.
In prim's algorithm, at start we pick any vertex I.e put in bucket and then look for that edge which has minimum wait.

Correctness Of Algorithm:

We know prim's algorithm returns a MST and it's weight is minimum. If we want to prove it's correctness we have to show MST returned by prim has minimum weight. Suppose S is MST returned by prim's algorithm and T is another MST for w(S)=w(T).

For prim connected graph is must and it guarantees to span all vertices but by picking edge on minimum weight. We are also checking for not formation of cycle so it guarantees to return a spanning tree and as we are taking min edges it returns MST.

Also we know about "cut edges property" in which we cut edges with greater weight and then check if graph remains connected or not. So at end we are left with minimum spanning tree.

Time Complexity Of Prim's Algorithm:

inside while loop , it takes O(logv) while extracting the vertices from queue and updating minimum edges (it occurs inside loop iterations). And this will happen for all vertices and its complexity is determined by both loops. i.e. in some points we are also coming back to edges twice. So (2*E)
So time complexity of this phase become O(2*Elogv) and we can write O(ELogV).

```
# While queue not empty
while len(q)!=0:
    adjacentvertex=q.pop()
    if adjacentvertex[0] not in bucket:
        bucket.append(adjacentvertex[0])
        q.extend(graph[adjacentvertex[0]])
        mst=mst+adjacentvertex[1]
    q=sorted(q, key=lambda index: index[1], reverse=True)

return bucket,mst
```