WEEK 3, LECTURE 6 ON 4 SEPTEMBER 2021 CS1.301.M21 ALGORITHM ANALYSIS AND DESIGN

DIVIDE AND CONQUER REVIEW

- Computing the Nth Fibonacci Number
- Karatsuba Integer Multiplication
- Merge Sort
- Strassen's Matrix Multiplication
- Fast Fourier Transform
- Median

The divide and conquer paradigm isn't exactly defined i.e.. we just divide and conquer isn't an exact list of steps. But the philosophy behind it is the most important and so when we encounter a problem in which the Divide and Conquer method is applicable we can use the method and are able to solve.

GREEDY ALGORITHMS

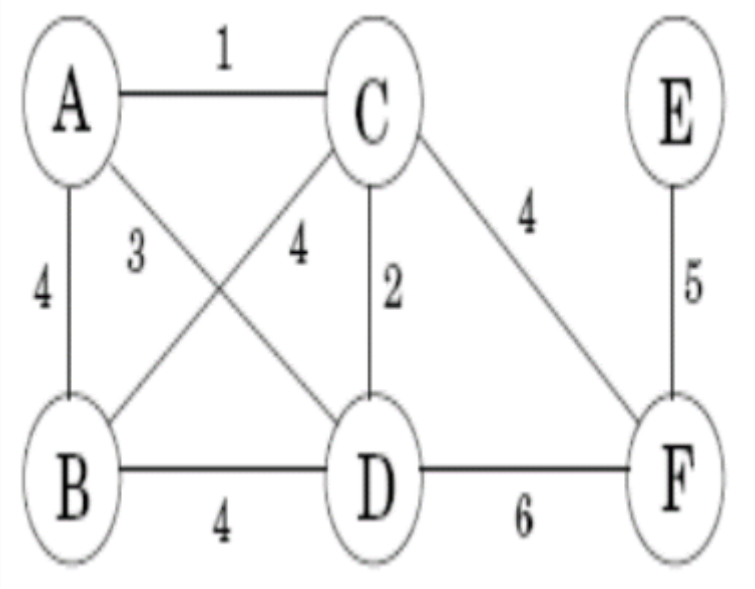
Minimum Spanning Tree Problem

Input: An undirected graph $G = (V, E)$; Edge weights are w_e.

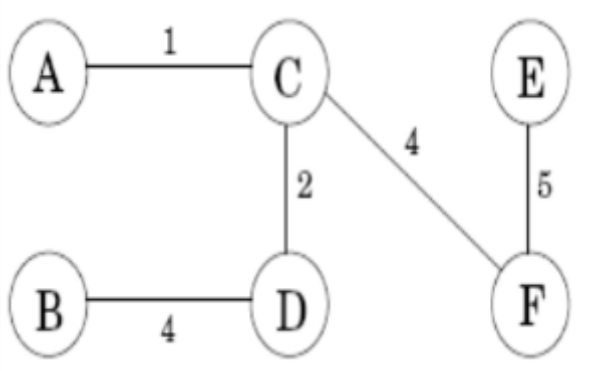
Output: A tree $T = (V, E')$, with $E' \subseteq E$, that minimizes the the weight $\sum_{e \in E'} w_e$.

Example:

For the graph,



An MST is

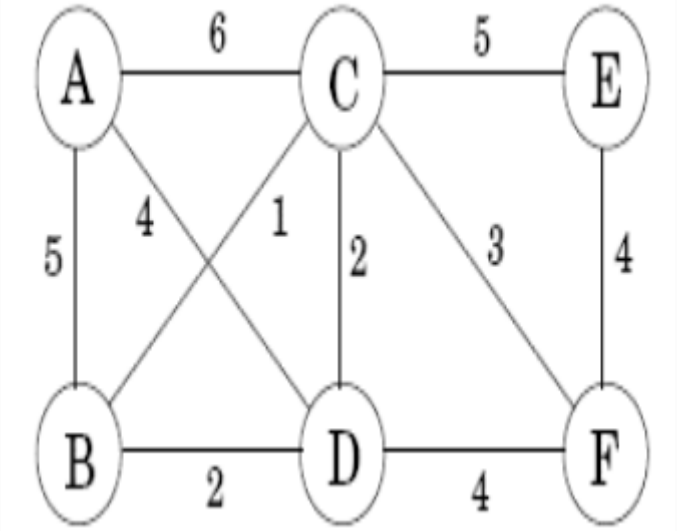


Kruskal's Algorithm

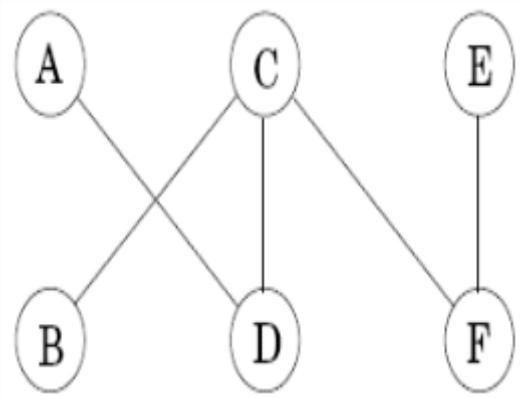
Repeatedly add the next lightest edge that doesn't produce a cycle.

i.e. Take the edge with the least weight and see if it produces a cycle. Then add or don't add.

Consider:



First we add BC then CD and so on...



Greedy strategies, when they work, are often the most applicable, most efficient and easiest to implement.

Is there a condition that's true when a greedy strategy is applicable?

Matroid Theory gives a sufficient condition such that when it does say greedy strategies are applicable, they are and when it doesn't say so, it **needn't be**.

Proof of Kruskal's Algorithm

Usually greedy algorithms are proved by induction

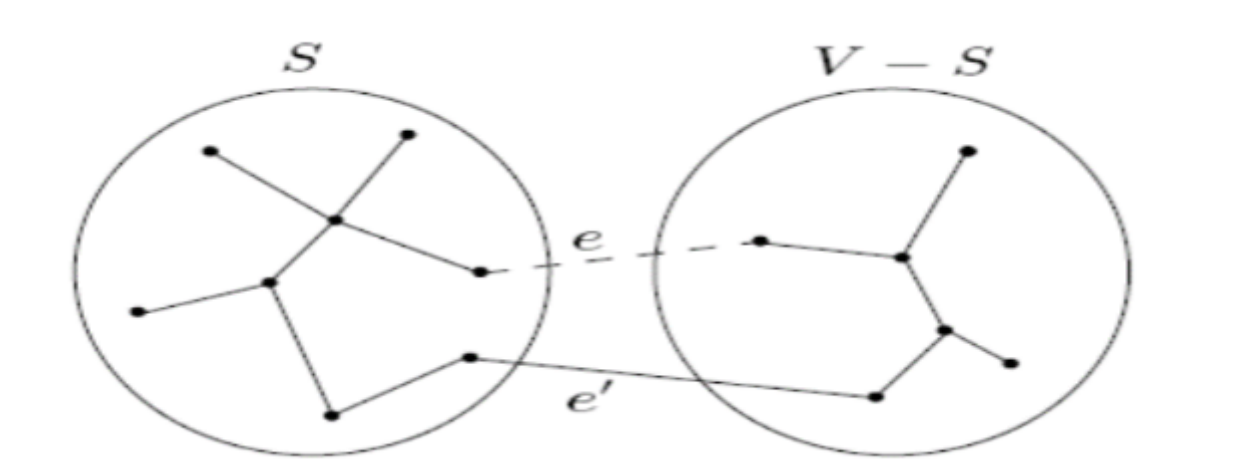
i.e. Step 1 isn't a mistake therefore, all succeeding steps are correct.

The CUT property:

Not only used for Kruskal's proof. Many algorithms can be proved with the cut property.

Suppose edges X are part of an MST of graph G. Pick any subset of nodes S for which X does not cross between S and $V - S$, and let e be the lightest edge across this partition. Then $X \cup \{e\}$ is part of some MST.

- Edges X are part of the MST T
- Assume new edge e is not in T
- Construct a different MST T' containing $X \cup \{e\}$
- Add edge to T, and this creates a cycle having another edge e' between V and V-S.



If we remove e', we are left with $T' = T \cup \{e\} - \{e'\}$, which will be shown to be a tree

Suppose e' has a weight lower than e, then a Tree containing e is no longer of minimal weight. hence e' should rather be taken.

T' is connected since e' is a cycle edge and a deleting a cycle edge cannot disconnect a graph

T' will have the same number of edges as T so it is also a tree since a connected graph which has $|V| - 1$ is tree.

T' is an MST so comparing it's weight with weight of T:

$$\text{weight}(T') = \text{weight}(T') + w(e) - w(e')$$

We can infer from this that Kruskal's algorithm is correct.

Textbook implementation of Kruskal's:

Figure 5.4 Kruskal's minimum spanning tree algorithm.

```
procedure kruskal(G, w)
Input:   A connected undirected graph G = (V, E) with edge weights w_e
Output:  A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

makeset(x): create a singleton set containing just x.

find(x): to which set does x belong?

union(x, y): merge the sets containing x and y.

Analysis of Kruskal's

In `makeset(x)` suppose, rank is the height of the sub-tree hanging from that node.

for any x, $rank(x) < rank(\pi(x))$

any root node k has at least 2^k nodes in its tree

Cost per operation:

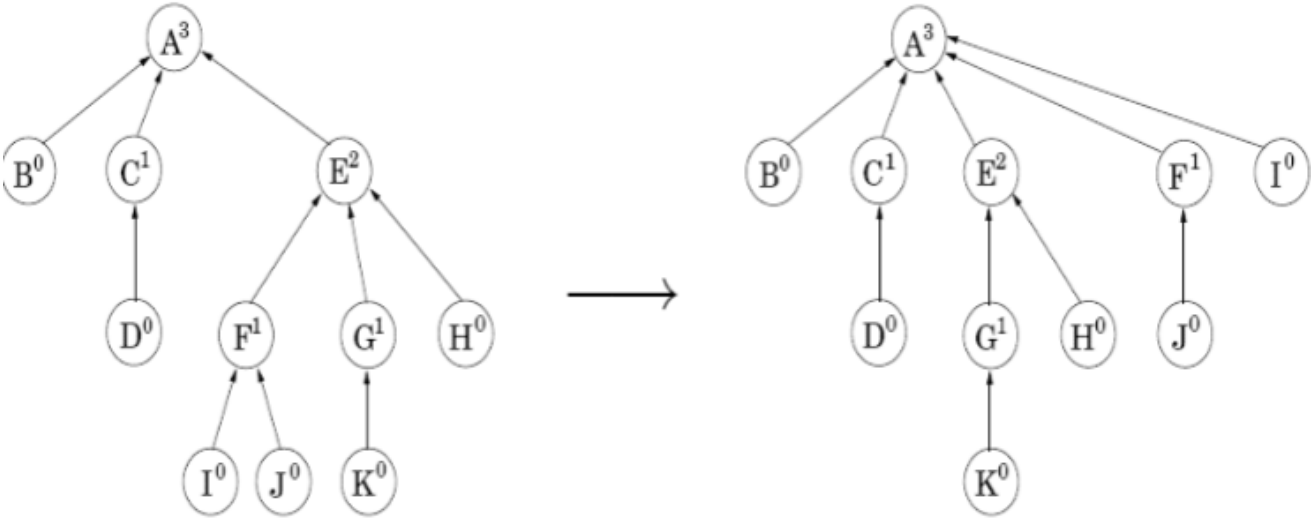
- Makeset: $O(1)$
- Find: $O(logn)$
- Union: $O(logn)$

Overall: $O((|E| + |V|)log|V|)$

Can We Do Better?

What if the edges are given to us in a sorted order?

Modifying Find: we modify all the pointers going to a root, we can change it so that it directly points to the root.




```
function find(x)
if x ≠ π(x): π(x) = find(π(x))
return π(x)
```

Ranks are divided into log^*n

> No of logs to get less than 1 is the log^*n function

$$\{1\},\{2\},\{3,4\},\{5,6,\ldots,16\},\{17,18,\ldots,2^{16}=65536\},\{65537,65538,\ldots,2^{65536}\},\ldots$$

So the time taken by a find operation becomes the number of pointers followed.

Node x on the chain to the root falls into two categories:

- rank of parent of x is in a higher interval than rank(x)
- or it lies in the same interval

So there are only at most log^*n nodes of the first type.

If x is of the second type, then when a find occurs, its parent changes to one of a higher rank.

Therefore, if rank(x) lies in $\{k+1,\ldots 2^k\}$

and it will only be of this category 2^k times and will then never be of this category again since its parent will be in a higher interval.

so find() becomes $O(m \cdot log^* n) + 2^k \times \text{number of nodes with rank}>k$. And the number of nodes with rank is $\leq n \cdot log^* n$.
