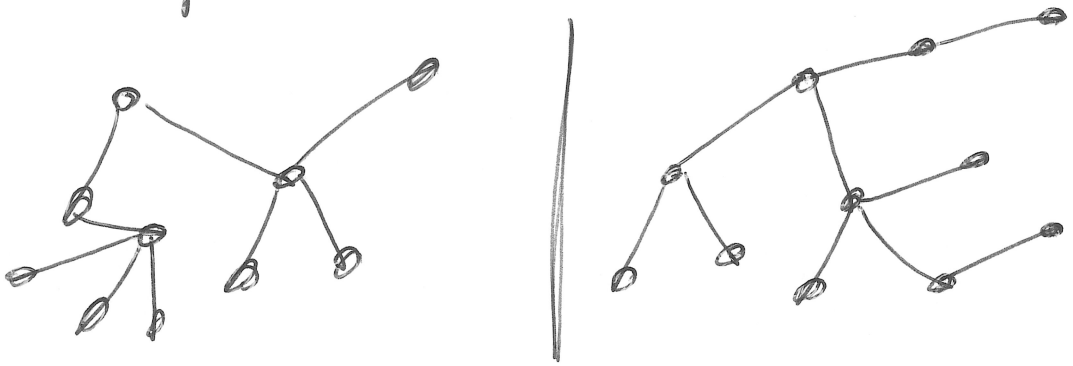
COP 3503 3/5/2024

Today: Minimum Spanning Tree Algorithms

Input: Weighted Undirected Graph

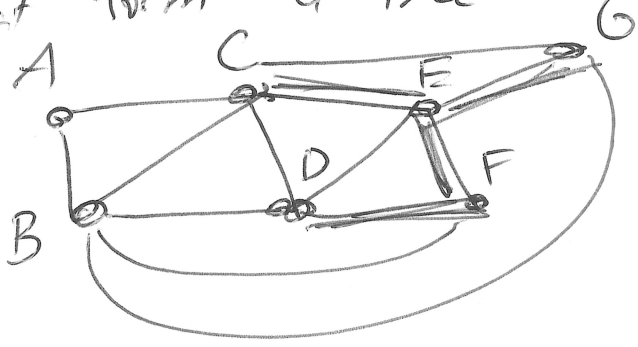
Output: Minimum Spanning Tree (if it exists)

Tree: Graph with no cycles. (Unique path btw any pair of vertices.) Connected.



V vertices $\Rightarrow$ V-1 edges

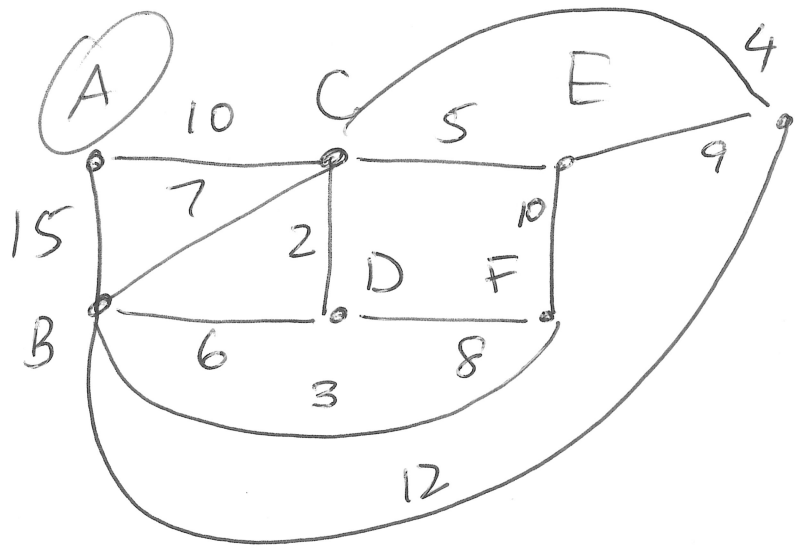
Tree within a larger graph is a set of edges that form a tree



Tree
EG
CE
EF
DE

Spanning - contains all vertices from the graph

Can make the selection above spanning tree by adding CB and BA.



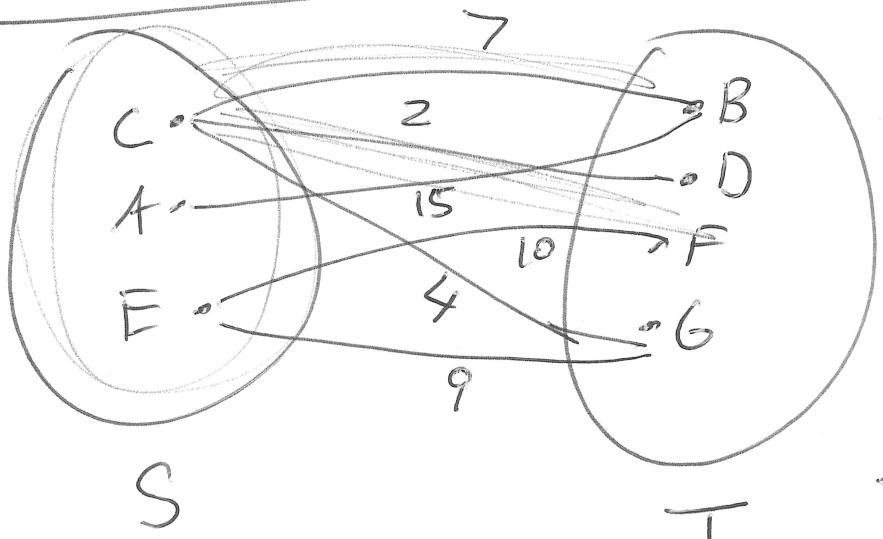
Min Span Tree

Of all possible spanning trees find one that minimizes the sum of the weights of the edges in the tree.

AB + AC + CE + EG + EF + DF

15 + 10 + 5 + 9 + 10 + 8 = 57

Crucial Fact About MSTs



① Partition vertices into 2 nonempty sets S, T.

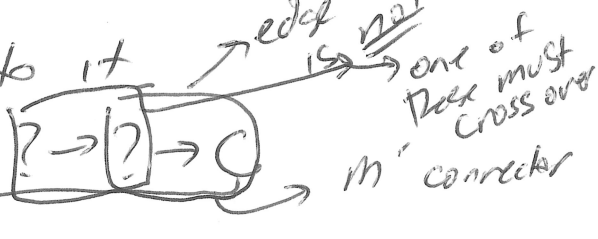
② Look at all edges from a vertex is S to a vertex in T

The minimum edge in this list belongs to some minimum spanning tree of the graph

Assum opposit min edge across isn't in MST. We know at least one edge is. MST M' doesn't have min edge. M' has a different edge in it btw S and T.

Take M' and add min edge to it ⟹ creates cycle

ADD CD Remove the cross edge in cycle



Prim's Alg

We'll "grow" our MST, starting with those sets S and T I previously described. S will start as contain 1 vertex (input function)

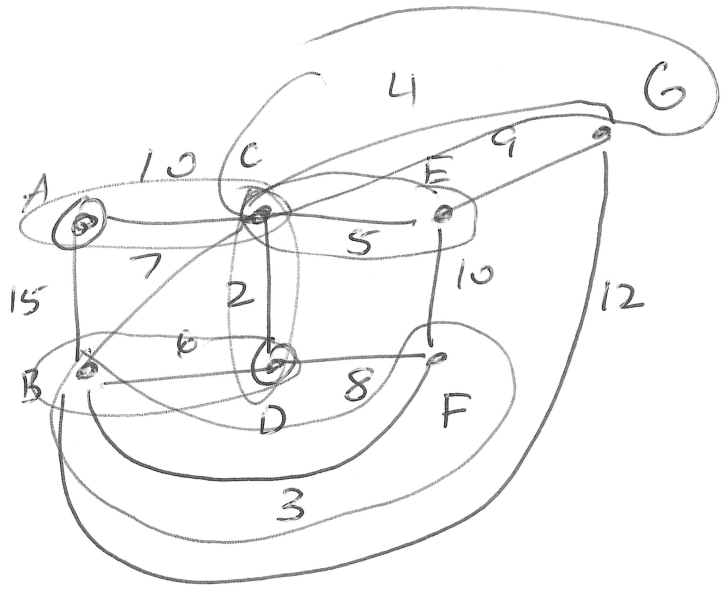
```
Prims(vertex v, Graph G) :
    S = {v}, Priority Queue of edges PQ( add all edges that leave v )
    while ( S != V ) {
        if (PQ is empty) break;
        edge e = PQ.poll(); //get next edge
        if( both e.u and e.v are in S)
            continue;
        newe = !in[e.u] ? e.u : e.v;
        Add newe to S
        Add edge e to MST
        Add each edge leaving newe
        to the PQ.

    }
    if (S == V) the MST is what
    we built.
    else
        there's no MST
}
```

Kruskals Algorithm

1) Sort edges by weight
2) One by one go through the edges, add to MST so long as it doesn't cause a cycle!

Cycle Detection?
Disjoint set!

```
int kruskals( ArrayList<edge> elist, int n) {
    Collections.sort(elist);
    djset dj = new djset(n);
    int res = 0, numE = 0;
    for (int i=0; i< elist.size(); i++) {
        boolean ok= dj.union(elist.get(i).U,
                             elist.get(i).V);
        if (!ok) continue;
        res += elist.get(i).w;
        numE++;
        if (numE == n-1) break;
    }
    return numE == n-1 ? res : -1;
}
```



Prim's

S

A , pq 10, 15

C , edge AC
pq: CD, CB, CA, CE, CG

D , edge CD
pq: BD, ~~CD~~, DF

G, edge CG
pq: EG, BG

E, edge CE
pq: EF, EC, EG

D, edge BD
pq: BF, BG
BA, BC

AC	10
CG	4
CD	2
CE	5
BD	6
BF	3
	30

Kruskal's:

CD	2	✓
BF	3	✓
CG	4	✓
CE	5	✓
BD	6	✓
CB		X
DF		X
EG		X
AC		✓

Red Blue Tree

Run MST w/ red edges = 0
blue edge = 1

then MST sum is fewest # of blue edges necessary.

Run MST w/ no edge = 1
blue edge = 0

⇒ max blue edges

if min ≤ k ≤ max yes!