COP 3503 - 3/17/22

Notes about Dijkstra's

1) Works only w/ non-neg edge weights
2) Greedy
3) Run-time $O(E \lg V)$ → sparse
   $O(V^2)$ → dense

Neg edge weight single source shortest path

Bellman - Ford

```
for (int i = 0; i < n-1; i++) {          // n = # vertices
    for (Edge e : g) {                   // relaxation
        dist[e.v] = min(dist[e.v],
                        dist[e.u] + e.w);
    }
}
// Edge object  u = start
//              v = end
//              w = weight
```

dist[e.u] + e.w: old, this edge

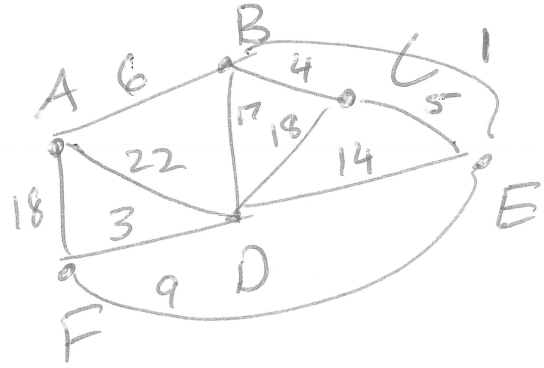
O(VE)

TODAY: Minimum Spanning Tree

1) Define the Problem
2) Critical fact about MSTs
3) Prim's
4) Kruskal's
5) Live Code (Kattis)

Input: Weighted undirected graph

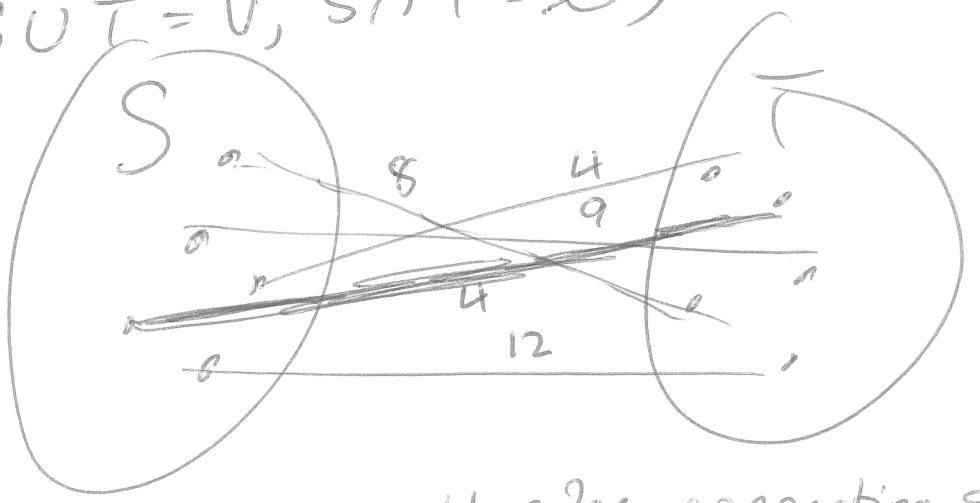
Output: set of edges which connect all the vertices in the graph while minimizing the sum of edge weights.



tree connected graph w/o cycles!

Critical Fact about an MST

Given a graph with vertex set V, for any partition of V into sets S and $\overline{T}$ ($S \cup \overline{T} = V$, $S \cap \overline{T} = \emptyset$)



the minimum weight edge connecting some vertex in S to some vertex in $\overline{T}$ will be in some minimum spanning tree.

If I am building an MST, it's ALWAYS safe to add an edge that connects to "unconnected sets" that is of minimum weight of all the edges btw those sets.

Prim's Algorithm

(1) Pick start vertex, v. Add this to our set S.

(2) Add to a priority queue all edges that touch v.

(3) while (|S| < |V|) {

edge cur = pq.poll(); // Get next edge P.Q.

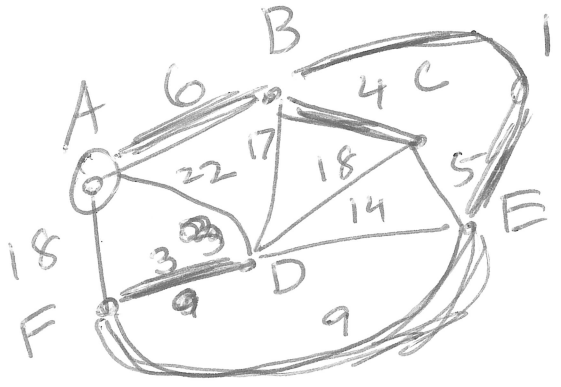
if (cur connects 2 vertices in S)
continue;

Add cur to MST.

Add the new vertex, t, it reaches to S.

Add to the priority queue all edges that touch t.

}



v = A

1) add AB, pq add next B

2) add BE, pq add next E

3) add BC, pq add next C

4) CE NOT added

5) Add EF, pq add next F

6) Add FD

Prim's: Vine growing via path of least resistance.

Kruskal's Alg

1. Sort edges by weight
2. Go through edges in order + add them to MST as long as they don't cause a cycle!

Question: How do I detect a cycle?

Answer: Disjoint Set