COP 4516 3/31/2026

1) Codeforces - mostly I'll put in grades, need to talk to few(3) people - email

2) USACO - Recheck (ChatGPT Experiment thinking people will finish before 3 hrs → I'll come by to recheck)

3) Weighted Graph Algs

MST (minimum spanning tree)

Kruskal's, Prim's } $O(E \lg V)$

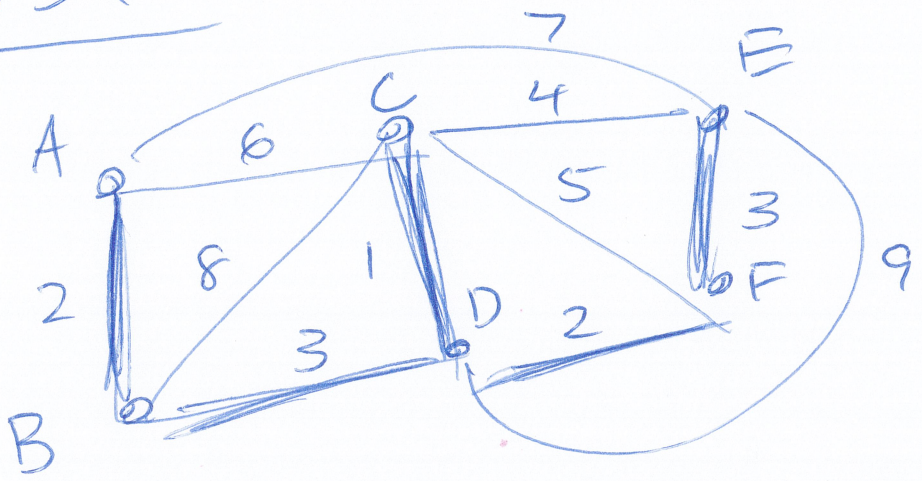
V = # vertices
E = # edges

(All-Pairs Shortest Path
↳ Floyd-Warshalls } $O(V^3)$

Single Source Shortest Path

$O(E \lg V)$ ✓ Dijkstra's → non-neg edge weight
$O(EV)$ ✓ Bellman Ford ok neg edge weight

MST

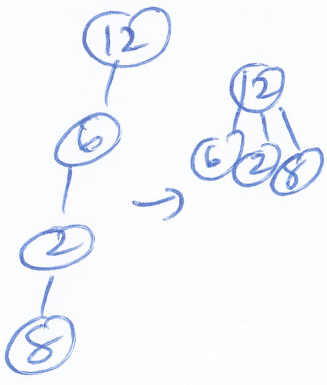


goal: pick subset of edges that connect all vertices in a tree with minimum possible sum

Kruskal's: ① Sort edges by weight
② Add to MST } Loop the edges in this order
if an edge doesn't cause a cycle

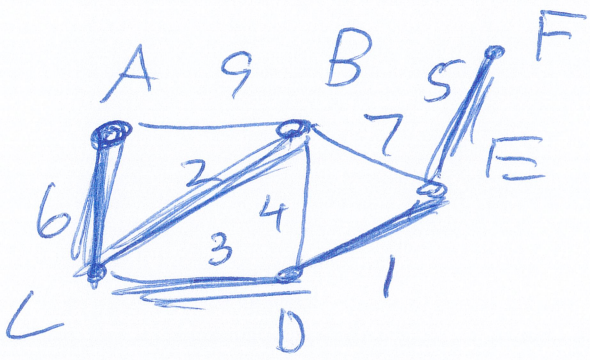
Disjoint Set w/path compression

Path Comp



Prim's

Starts at a single vertex grows 1 connected tree



PQ: 9, ~~3~~

PQ: 9, ~~7~~, 7, 4

PQ: 9, 7, 4, ~~5~~

PQ: 9, 7, ~~4~~, 5

↓

causes cycle (used array)

Prim's

Add all edges incident to A (starting vertex) to a Priority Queue.

while not done:

get next edge from PQ

skip if both vertices are connected

otherwise

add to MST.

add new edges to queue adjacent to the next edge

Floyd Warshalls

Directed Weighted Graph

Store edges in 2D array

	0	6	2	3	7
		0			
			0		
				e	
					0

no edge

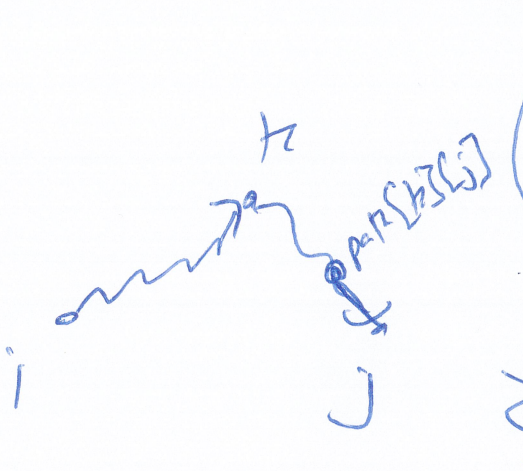
-1, if non neg edge

null, object

10^9, big number

~~for (int k=0; k<n; k++)~~

```
for (int k=0; k<n; k++) //mid
    for (int i=0; i<n; i++) //start
        for (int j=0; j<n; j++) //end) {
            d[i][j] = min(d[i][j], d[i][k] +
                                   d[k][j]);
            if ( d[i][k] + d[k][j] < d[i][j]){
                path[i][j] = path[k][j]
            }
    }
```



↓
last vertex visited on the shortest path from i to j.

* Watch out for negative cycle detection

Paths : allpairspath

Dijkstra's

It's a BFS with a PriorityQueue to get shortest distances from a single starting point to all other vertices in a directed weighted graph w/ no negative edge weights.

BFS

```
dist[s] = 0
queue.offer(s)
while (s.size() > 0) {
    v = queue.poll()
    for(int next : g[v]) {
        if ~~used~~
            dist[next] != -1)
                continue
        dist[next] = dist[v]+1;
        queue.offer(next)
    }
}
```

Dijkstra's

```
dist[s] = 0
PQ<Edge> → add in all edges incident to s.
while (s.size() > 0) {
    e = pq.poll();
    (e.u start, e.v end)
    for (edge next : g[v])
        pq.offer(
            new edge(to next.v
            weight dist[v] +
            ~~edge~~ next.w)
}
```

technically extends to other vertices

if you already pulled this vertex skip it!