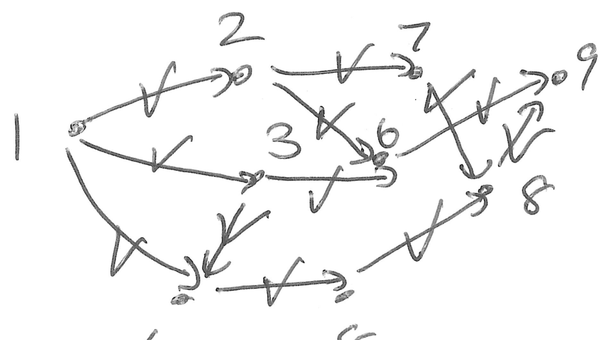
COP 4516 2/11/2025

Weighted Graph Algorithms

Last Week: Topological Sort

1) Top Sort
2) MST (Minimum Spanning Tree)
   a) Prim's
   b) Kruskal's
3) Single Source Shortest Distance
   a) Dijkstra's
   b) Bellman-Ford
4) All Pairs Shortest Path - Floyd Warshalls

Top Sort



order: 1, 2, 3, 7, 4, 6, 5, 8, 9.

	1	2	3	4	5	6	7	8	9
indeg	0	~~1~~ 0	~~1~~ 0	~~2~~ ~~1~~ 0	~~1~~ 0	~~2~~ ~~1~~ 0	~~1~~ 0	~~2~~ ~~1~~ 0	~~2~~ ~~1~~ 0

queue will store all items indeg 0

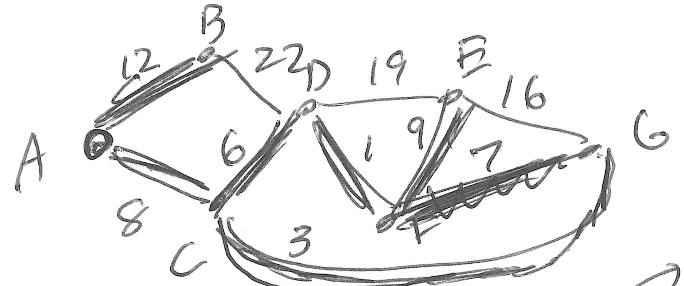
q: ~~1~~, ~~2~~, ~~3~~, ~~7~~, ~~4~~, ~~6~~, ~~5~~, ~~8~~, 9

while true:
- nextitem = q.poll()
- order.append(nextitem)
- update indegrees based on edge leaving nextitem
- if indeg == 0
  - add queue

Minimum Spanning Tree

Input: UNDIRECTED, WEIGHTED GRAPH

Goal: Pick subset of edges of minimal total cost such that those edges ~~completely~~ connect all the vertices in the graph



connected[v]=true

Prims (Graph G, ~~$~~Vertex v)

keep list of connected vertices (boolean list)

PQ of edges

add all edges incident to v to PQ.

loop until graph is connected:

look at PQ dequeue next item

if it connects to a new vertex add it to the MST.

Otherwise continue.

Add all edges adjacent to the new vertex to the PQ.

1) AC [CD, CG]
2) CG [EG, FG]
3) CD [DB, DF, DE]
4) DF [FE, FG]
~~5) FG~~
5) FG → throw out
6) FE → [ED, EG]
7) AB

Kruskal's

1) Sort edges by weight
2) Loop through list
   add to MST so long as
   doing so doesn't cause a cycle!

?

Disjoint Set Data Structure
Path Compression ~ near O(1) time
for our 2 ops: union, find
Union-Find.

Single Source Shortest Distance

1) Dijkstras

Weighted, Directed
No neg edge weights!!!

$O(E \lg V)$

↑ # edges ↑ vertices

You give it a starting vertex, it will calculate all shortest distances <u>from</u> that vertex to all the others.

Note: If you need all shortest distances from lots of places to one ending place, reverse all edges in the graph + make the end vertex the start vertex for Dijkstra's.

It's really a modified Breadth First Search

BFS	Dijkstras
queue add start v	PQ add start
dist[start] = 0	dist[start] = 0
while (queue isn't empty){	while PQ isn't empty:
v = queue.poll()	v = PQ.poll()
for (next x: v){	for (next edge v->x){
if (dist[x] == -1){	if (dist[x] == -1 || dist[v] + weight(v,x) < dist[x] {
dist[x] = dist[v]+1	//prev[x]=v dist[x] = dist[v] + weight(v,x)
queue.offer(x)	PQ(x, dist[x])
}	}
}	
return dist	

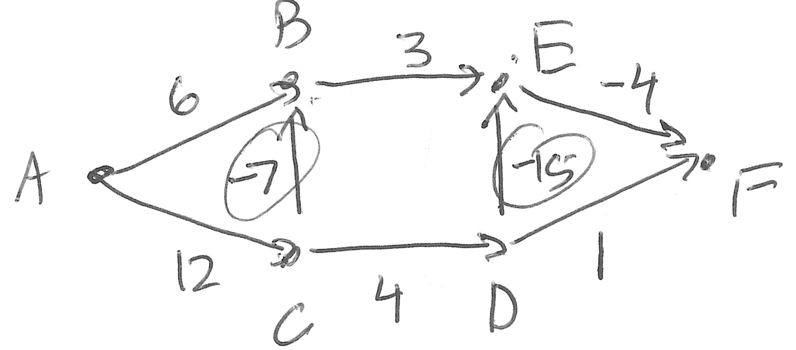
Bellman- Ford (source

dist[start] = 0
dist[else] = ∞

edge
u to
v weight w

```
for (int i = 0; i < V; i++)
    for (Edge e : Graph)
        dist[e.v] = (dist[e.u] + e.w, dist[e.v])
                    min
```

O(EV)



	A	B	C	D	E	F
Iter 0	0	∞	∞	∞	∞	∞
Iter 1	0	6	12	∞	∞	∞
Iter 2	0	5	12	16	9	∞
Iter 3	0	5	12	16	~~8~~	5
					1	
	0	5	12	16	1	-3

Floyd - Warshalls

All Pairs Shortest Path, Neg Edge Weights Allowed

m[i][j] = edge weight btw i and j.

= large # if no edge

Initially we store only shortest paths that DON'T VISIT any intermediate vertices.

```
//k   is intermediate vertex
//i   is start vertex
//j   is end vertex

for (int k = 0; k<n; k++)
  for (int i = 0; i<n; i++)         } all poss estimates
    for (int j = 0; j<n; j++)
      m[i][j] = min(m[i][j], m[i][k] +
                               m[k][j]);
```

$O(V^3)$

at end m[i][j] will store shortest distance ~~for~~ from vertex i to vertex j.