1) Whiteboard Sign Up

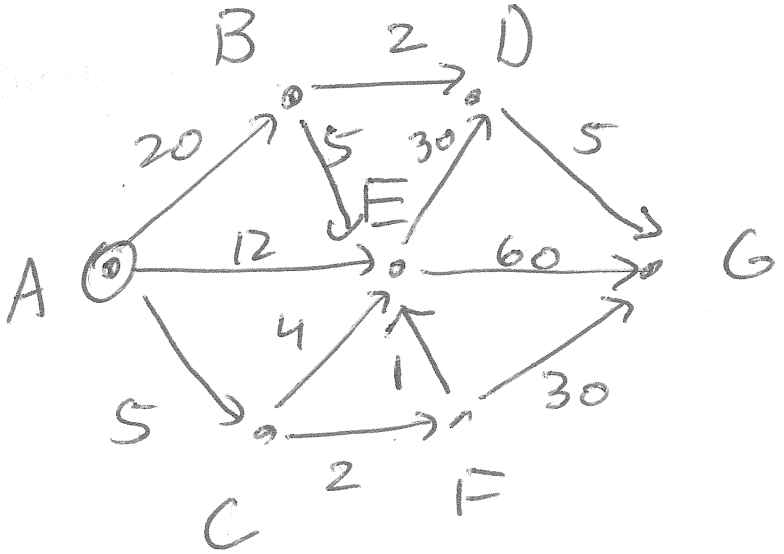
2) Shortest Distance Algs
   - Weighted
   - directed
   } graphs

a) Dijkstra's SSSP no neg edge weight

b) Bellman Ford SSSP neg edge weight

c) Floyd Warshall's All Pairs SP

d) Debug my SSSP3



In unweighted undirected graph edge {u,v} set u/2 vertices

Directed weighted Graph edge ordered triplet (u, v, w)

u: start, v: end, w: cost/weight

Dijkstra's

Single Source Shortest Path No Neg Edge Weights

Run-Time if efficiently implemented

$O(E \lg V)$

$E \lg E$, but since $E \leq V^2$ and $\lg V^2 = 2 \lg V$ we typically write it as above.

One Source Vertex → finds all shortest path from source to the other vertices.

maintain distance estimates to each vertex (from source).

PQ of estimates

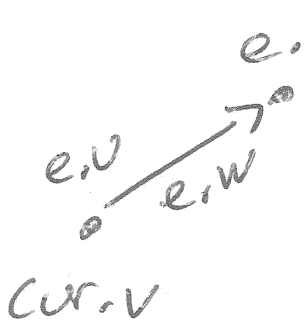
	A	B	C	D	E	F	G
D	0	∞	∞	∞	∞	∞	∞
Set = A	0	20/A	5/A	∞	12/A	∞	∞
C	0	20	5	∞	9/C	7/C	∞
F	0	20	5	∞	8/F	7	37/F
E	0	20	5	38	8	7	37
B	0	20	5	22/B	8	7	37
D	0	20	5	22	8	7	27/D

pq.add(source,0)
while pq.size()>0
  u=estimate
  add estimates based on edges leaving current vertex

path[i] = last vertex visited on shortest path to i

BUILD PATH

```
path = list<>   path.add(end);
while ( prev[end] != start ) {
    path.add(prev[end])
    end = prev[end];
}
path.add(start)
reverse(path)
```



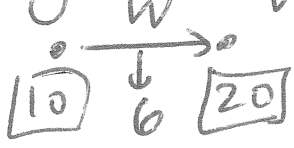
What about negative edge weights?

Dijkstra's DOES edge relaxation

Does edge (u, v, w) help me?

dist[v] - OLD WAY

dist[u] + w - NEW WAY



10+6 < 20

YES!

relaxed this edge!

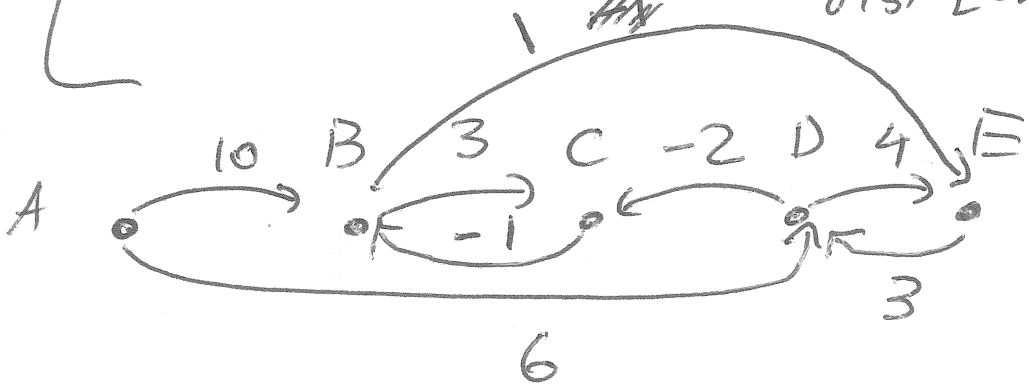
Bellman-Ford → # ve

Repeat n-1 times:
Relax every graph edge

Stl w/ distance array dist[source] = 0
dist[other] = ∞

works: no neg weight cycles

```
for (int i = 0; i < n-1; i++)
  for (edge e : list)
    dist[e.v] = min(dist[e.v],
                    dist[e.u] + e.w);
```

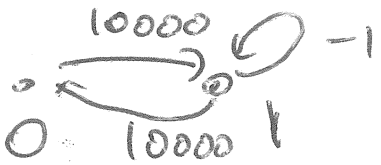


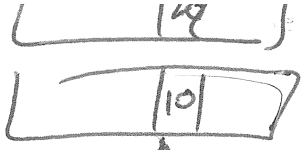
A	B	C	D	E
0	∞	∞	∞	∞
0	10	∞	6	∞
0	10	4	6	10

worst case 1 iteration

Shortest dist 2 edges

After k iterations, distances are the shortest ones that use k edges or fewer!





↑ put a neg weight

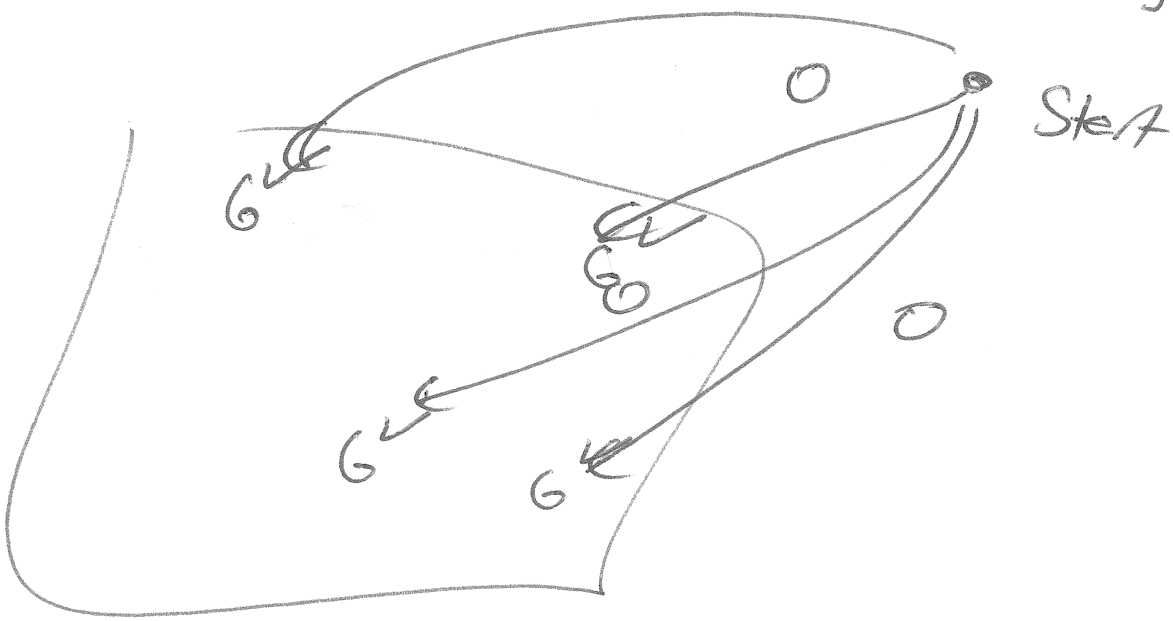
BFS from every edge on a neg







All as source BFS + see where I can go:



Floyd-Warshall's

All-Pairs Shortest Path

$O(V^3)$

Input: $V \times V$ adjacency matrix with every possible edge weight

adj[i][j] weight of the edge from vertex i to vertex j.

Beginning the adjacency matrix stores ALL shortest paths of 1 edge.

In F-W, we'll add in intermediate vertices 1 by 1.

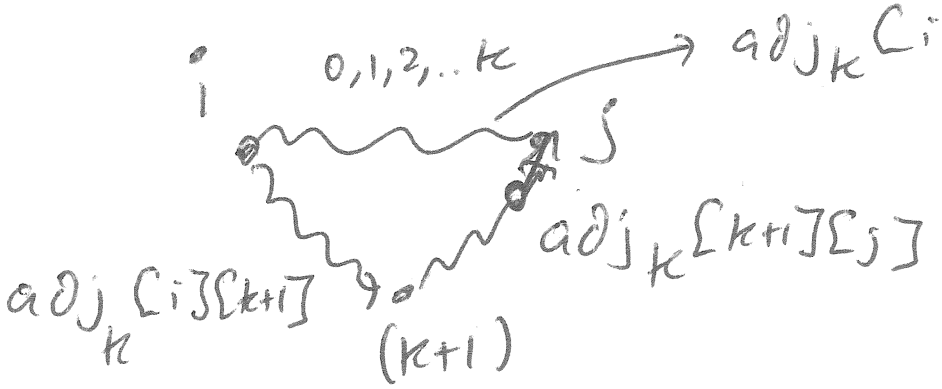
Old enough for intersection 0.

OLD WAY adj[i][j]

NEW WAY adj[i][0] + adj[0][j]

Do this for all pairs (i,j), then we'll have shortest distances that use only k=0 as an intermediate point.

$adj_k[i][j]$ - shortest distance from i to j using intermediate vertices 0, 1, 2, .. k.



$$adj_{k+1}[i][j] = \min\left(adj_k[i][j],\ adj_k[i][k+1] + adj_k[k+1][j]\right)$$

If we update

$path[i][j] = path[k+1][j]$;