Data Structures

Shortest Path 2 (All Paths!)

CS 225

December 4, 2023

Brad Solomon & G Carl Evans

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Last Lecture!

Wednesday is review day. Prepare questions!

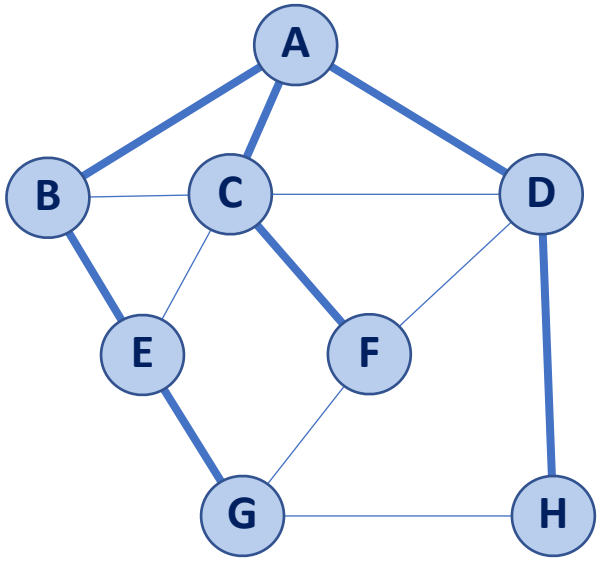
Can also post questions ahead of time (so I can prep slides)

Learning Objectives

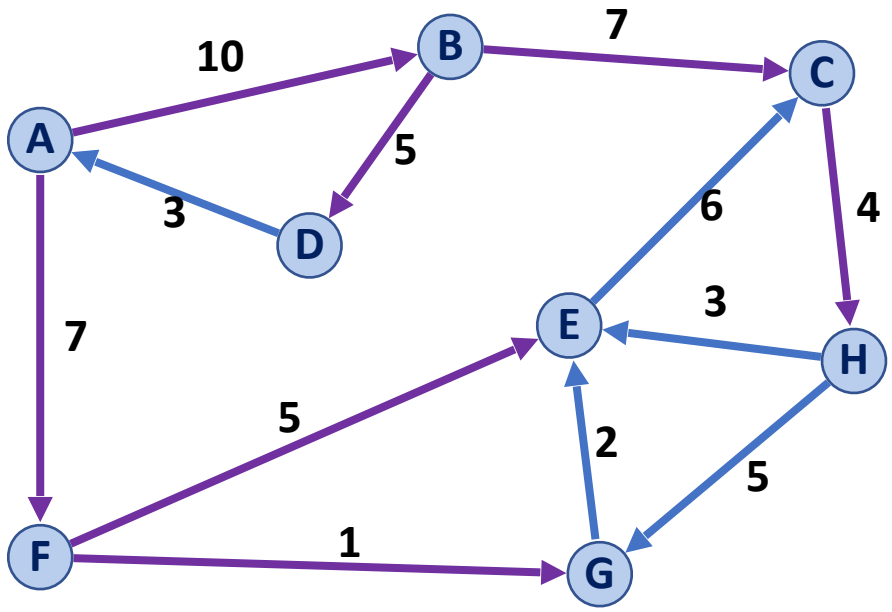
Calculate runtime of Dijkstras Algorithm

Introduce Bellman-Ford as an alternative to shortest path

Shortest Path



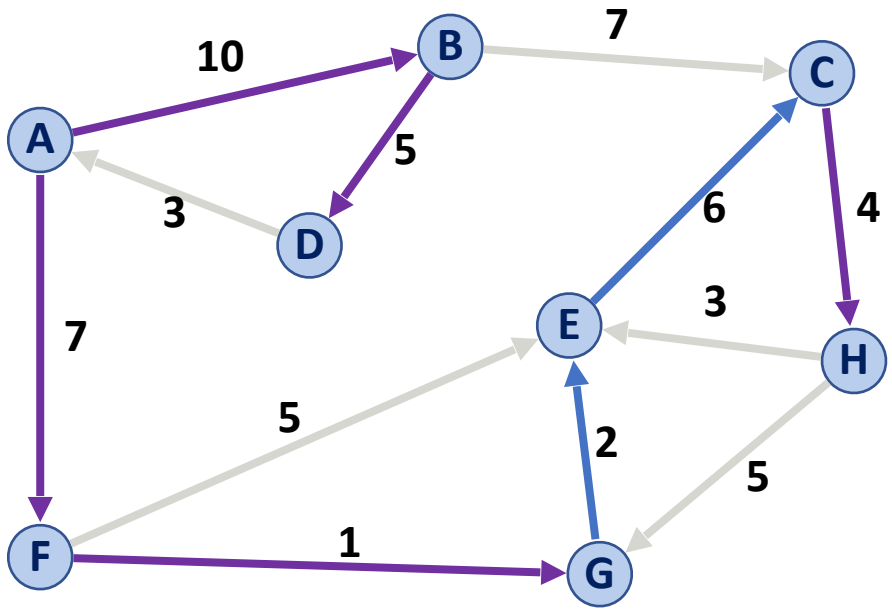
Dijkstra's Algorithm (SSSP)




```
    DijkstraSSSP(G, s):
 6    foreach (Vertex v : G.vertices()):
 7      d[v] = +inf
 8      p[v] = NULL
 9    d[s] = 0
10
11    PriorityQueue Q // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T           // "labeled set"
14
15    repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if cost(u, v) + d[u] < d[v]:
20          d[v] = cost(u, v) + d[u]
21          p[v] = u
```

A	B	C	D	E	F	G	H
--							
0							

Dijkstra's Algorithm (SSSP)




```
    DijkstraSSSP(G, s):
 6    foreach (Vertex v : G.vertices()):
 7      d[v] = +inf
 8      p[v] = NULL
 9    d[s] = 0
10
11    PriorityQueue Q // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T           // "labeled set"
14
15    repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if cost(u, v) + d[u] < d[v]:
20          d[v] = cost(u, v) + d[u]
21          p[v] = u
```

A	B	C	D	E	F	G	H
--	A	E	B	G	A	F	C
0	10	16	15	10	7	8	20

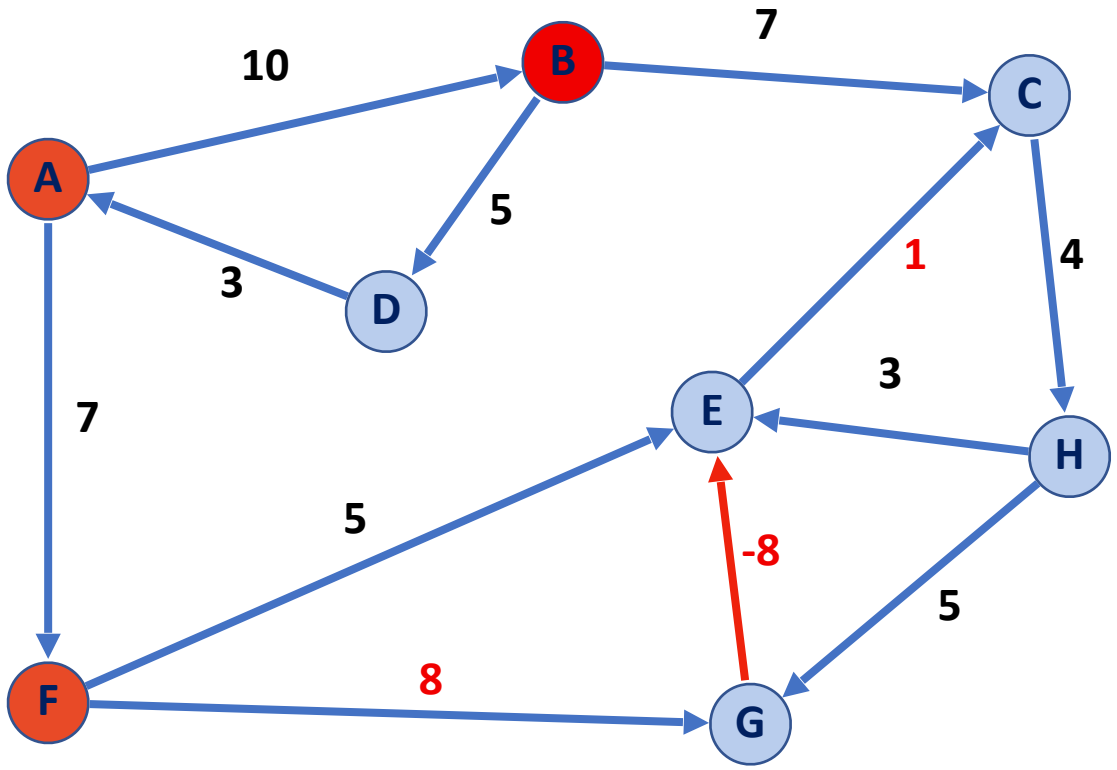
Dijkstra's Algorithm (SSSP)

What is the running time of Dijkstra's Algorithm?

```
    DijkstraSSSP(G, s):
 6    foreach (Vertex v : G):
 7      d[v] = +inf
 8      p[v] = NULL
 9    d[s] = 0
10
11    PriorityQueue Q // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T          // "labeled set"
14
15    repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if cost(u, v) + d[u] < d[v]:
20          d[v] = cost(u, v) + d[u]
21          p[v] = m
22
23    return T
```

Dijkstra's Algorithm (SSSP)

How does Dijkstras handle a negative weight edge without a cycle?



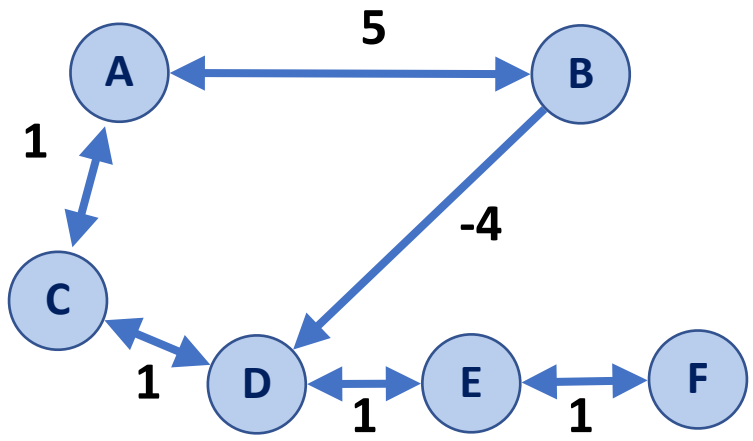
A	B	C	D	E	F	G	H
--	A	B	B	F	A	F	--
0	10	17	15	12	7	15	∞

Dijkstra's Algorithm (SSSP)

We assume that item pulled out of priority queue is **the next smallest item**

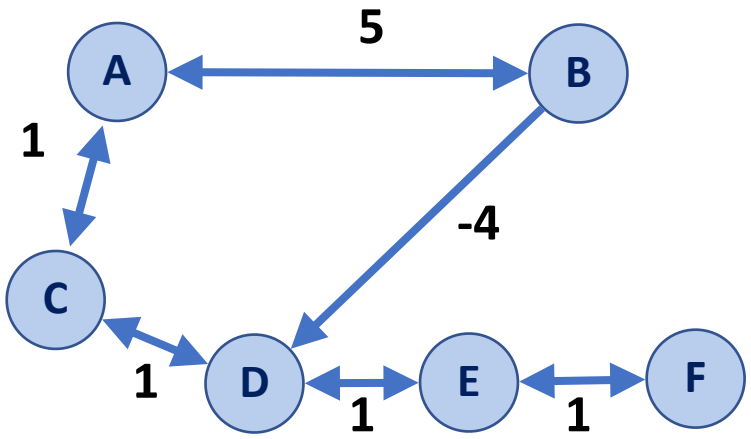
Negative weights break this assumption!

A	B	C	D	E	F
--					
0					



Dijkstra's Algorithm (SSSP)

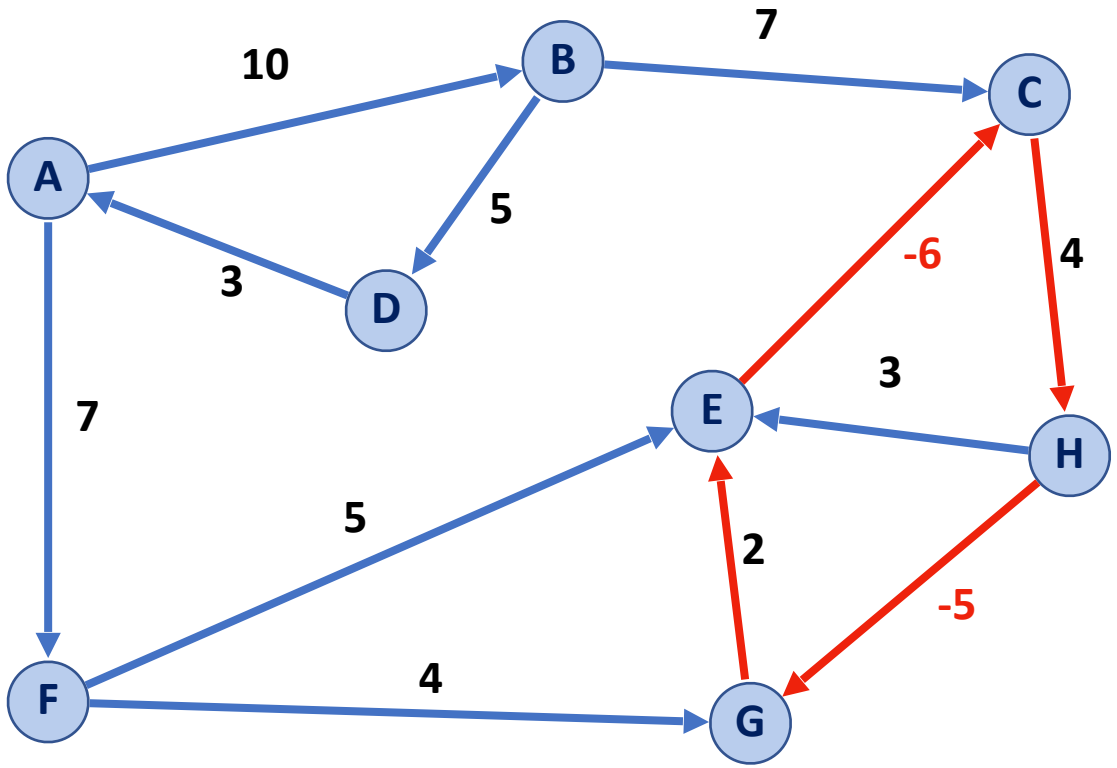
Recalculating all distances is possible, but algorithm runtime is very bad!




```
     DijkstraSSSP(G, s):
 6     foreach (Vertex v : G):
 7       d[v] = +inf
 8       p[v] = NULL
 9     d[s] = 0
10
11     PriorityQueue Q // min distance, defined by d[v]
12     Q.buildHeap(G.vertices())
13     Graph T          // "labeled set"
14
15     repeat until Q.empty():
16       Vertex u = Q.removeMin()
17       T.add(u)
18       foreach (Vertex v : neighbors of u not in T):
19         if cost(u, v) + d[u] < d[v]:
20           d[v] = cost(u, v) + d[u]
21           p[v] = m
22           if v not in Q:
23              Q.push(v)
24     return T
```

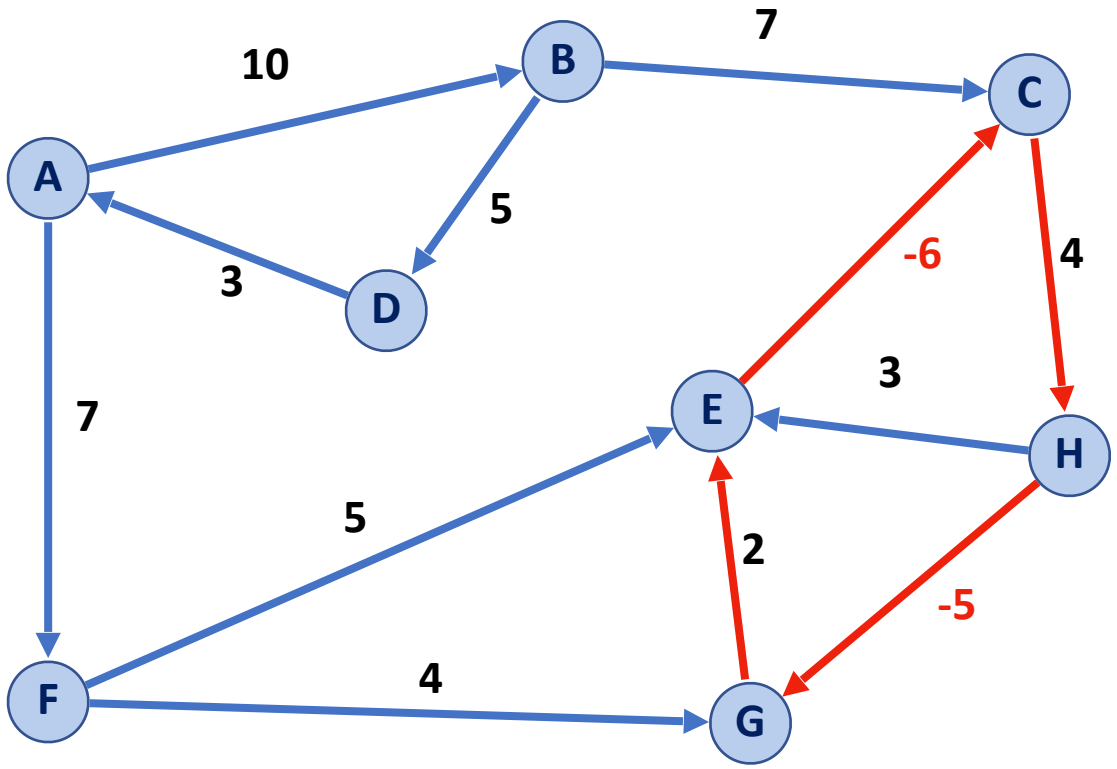
Dijkstra's Algorithm (SSSP)

How does Dijkstras handle a negative weight cycle?



Dijkstra's Algorithm (SSSP)

How does Dijkstras handle a negative weight cycle?



Shortest Path (A → E): A → F → E → (C → H → G → E)*

Length: 12

Length: -5 (repeatable)

Dijkstra's Algorithm (SSSP)

Dijkstras Algorithm works only on non-negative weights

Optimal implementation:

Optimal runtime:

```
    DijkstraSSSP(G, s):
 6    foreach (Vertex v : G):
 7      d[v] = +inf
 8      p[v] = NULL
 9    d[s] = 0
10
11    PriorityQueue Q // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T          // "labeled set"
14
15    repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if cost(u, v) + d[u] < d[v]:
20          d[v] = cost(u, v) + d[u]
21          p[v] = m
22
23    return T
```

Floyd-Warshall Algorithm

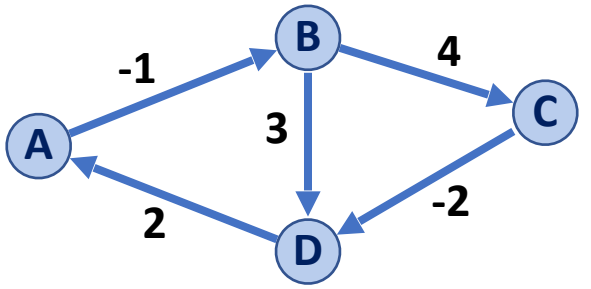
Floyd-Warshall's Algorithm is an alternative to Dijkstra in the presence of negative-weight edges (not negative weight cycles).

```
 1  FloydWarshall(G):
 2    Let d be a adj. matrix initialized to +inf
 3    foreach (Vertex v : G):
 4      d[v][v] = 0
 5    foreach (Edge (u, v) : G):
 6      d[u][v] = cost(u, v)
 7
 8    foreach (Vertex w : G):
 9      foreach (Vertex u : G):
10        foreach (Vertex v : G):
11          if (d[u, v] > d[u, w] + d[w, v])
12            d[u, v] = d[u, w] + d[w, v]
```

Floyd-Warshall Algorithm

```
1  FloydWarshall(G):
2    Let d be a adj. matrix initialized to +inf
3    foreach (Vertex v : G):
4      d[v][v] = 0
5    foreach (Edge (u, v) : G):
6      d[u][v] = cost(u, v)
```

	A	B	C	D
A				
B				
C				
D				

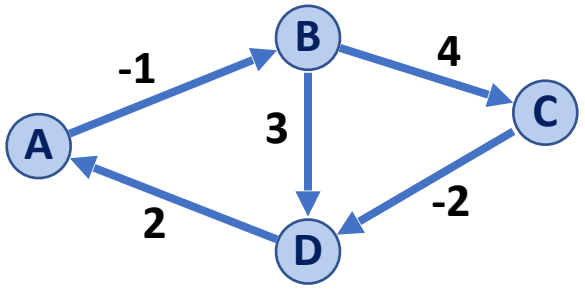


Floyd-Warshall Algorithm

```
 8   foreach (Vertex w : G):
 9     foreach (Vertex u : G):
10       foreach (Vertex v : G):
11         if (d[u, v] > d[u, w] + d[w, v])
12           d[u, v] = d[u, w] + d[w, v]
```

Let us consider comparisons where w = A:

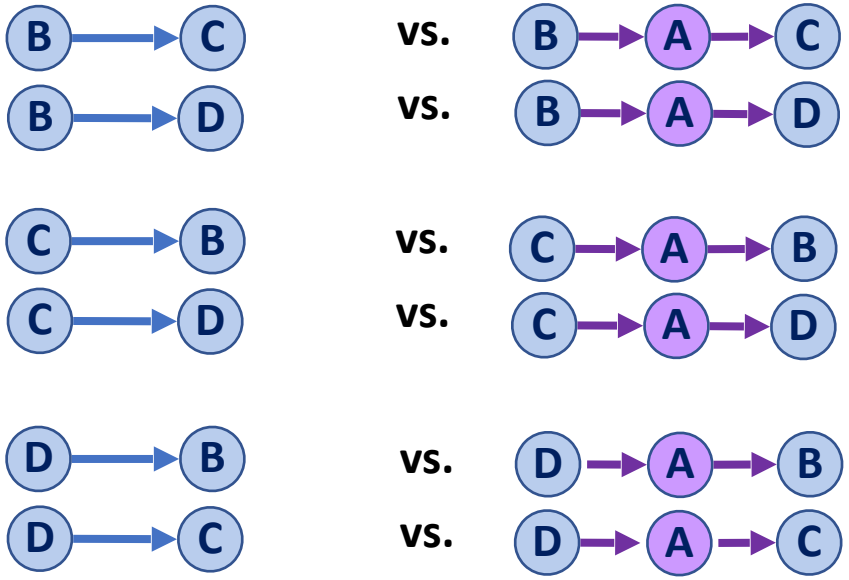
	A	B	C	D
A	0	-1	∞	∞
B	∞	0	4	3
C	∞	∞	0	-2
D	2	∞	∞	0



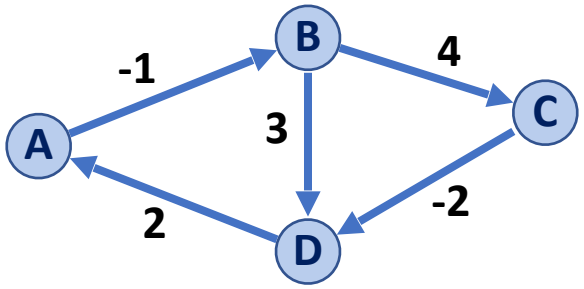
Floyd-Warshall Algorithm

```
 8   foreach (Vertex w : G):
 9     foreach (Vertex u : G):
10       foreach (Vertex v : G):
11         if (d[u, v] > d[u, w] + d[w, v])
12           d[u, v] = d[u, w] + d[w, v]
```

Let us consider w = A (and u != w and v != w):



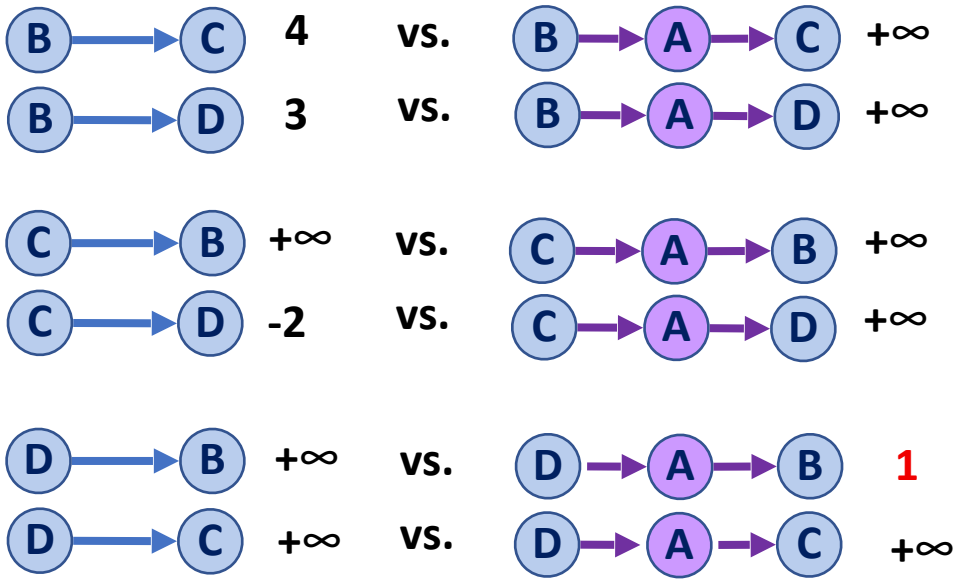
	A	B	C	D
A	0	-1	∞	∞
B	∞	0	4	3
C	∞	∞	0	-2
D	2	∞	∞	0



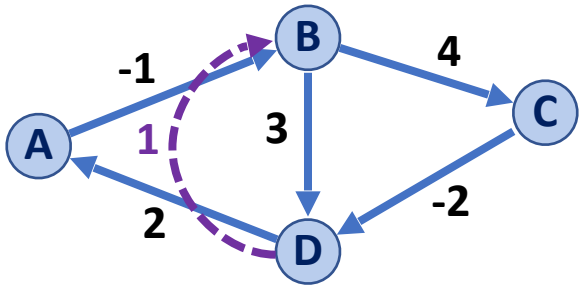
Floyd-Warshall Algorithm

```
 8   foreach (Vertex w : G):
 9     foreach (Vertex u : G):
10       foreach (Vertex v : G):
11         if (d[u, v] > d[u, w] + d[w, v])
12           d[u, v] = d[u, w] + d[w, v]
```

Let us consider w = A (and u != w and v != w):



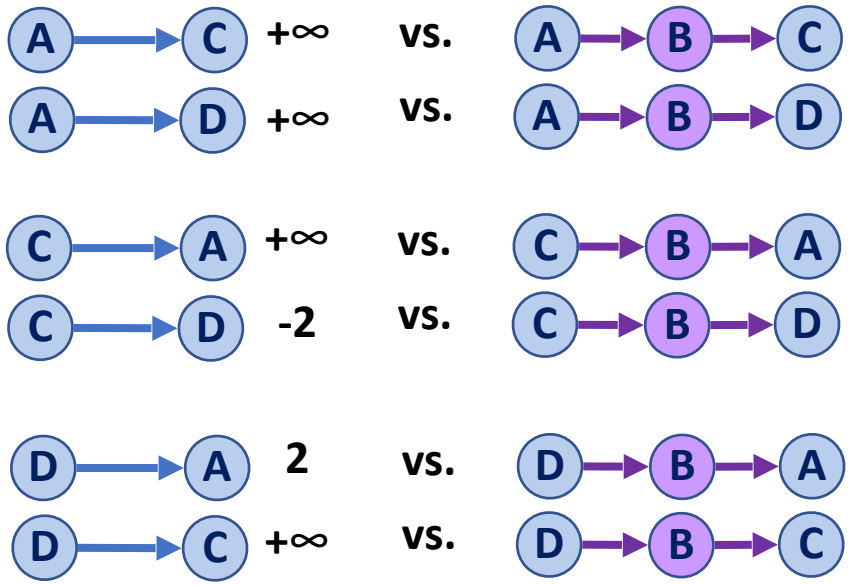
	A	B	C	D
A	0	-1	∞	∞
B	∞	0	4	3
C	∞	∞	0	-2
D	2	1	∞	0



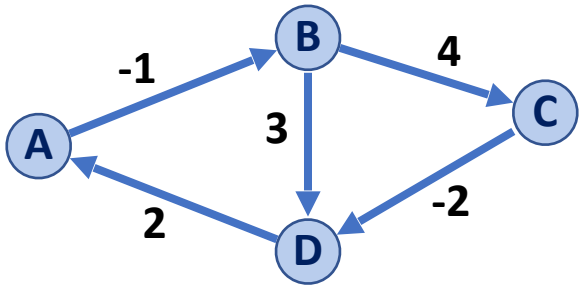
Floyd-Warshall Algorithm

```
 8   foreach (Vertex w : G):
 9     foreach (Vertex u : G):
10       foreach (Vertex v : G):
11         if (d[u, v] > d[u, w] + d[w, v])
12           d[u, v] = d[u, w] + d[w, v]
```

Let us consider w = B (and u != w and v != w):



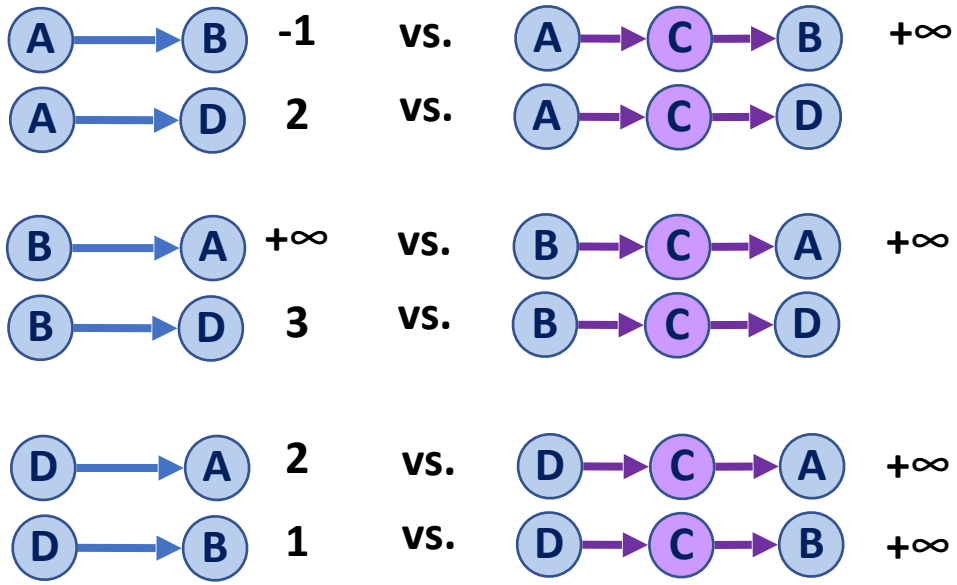
	A	B	C	D
A	0	-1	∞	∞
B	∞	0	4	3
C	∞	∞	0	-2
D	2	1	∞	0



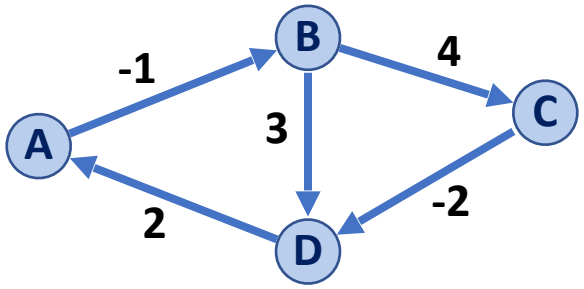
Floyd-Warshall Algorithm

```
 8   foreach (Vertex w : G):
 9     foreach (Vertex u : G):
10       foreach (Vertex v : G):
11         if (d[u, v] > d[u, w] + d[w, v])
12           d[u, v] = d[u, w] + d[w, v]
```

Let us consider w = C (and u != w and v != w):



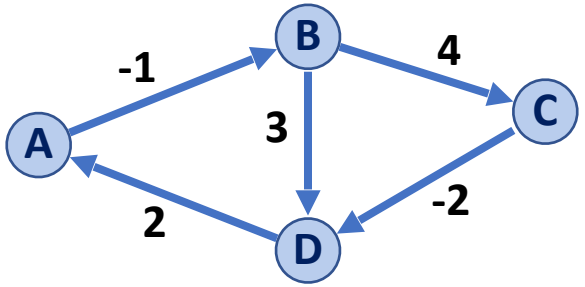
	A	B	C	D
A	0	-1	3	2
B	∞	0	4	3
C	∞	∞	0	-2
D	2	1	5	0



Floyd-Warshall Algorithm

```
 1  FloydWarshall(G):
 2    Let d be a adj. matrix initialized to +inf
 3    foreach (Vertex v : G):
 4      d[v][v] = 0
 5    foreach (Edge (u, v) : G):
 6      d[u][v] = cost(u, v)
 7
 8    foreach (Vertex u : G):
 9      foreach (Vertex v : G):
10        foreach (Vertex w : G):
11          if (d[u, v] > d[u, w] + d[w, v])
12            d[u, v] = d[u, w] + d[w, v]
```

	A	B	C	D
A	0	-1	3	1
B	5	0	4	2
C	0	-1	0	-2
D	2	1	5	0



Floyd-Warshall Algorithm

Running time?

```
    FloydWarshall(G):
 6    Let d be a adj. matrix initialized to +inf
 7    foreach (Vertex v : G):
 8      d[v][v] = 0
 9    foreach (Edge (u, v) : G):
10      d[u][v] = cost(u, v)
11
12    foreach (Vertex w : G):
13      foreach (Vertex u : G):
14        foreach (Vertex v : G):
15          if (d[u, v] > d[u, w] + d[w, v])
16            d[u, v] = d[u, w] + d[w, v]
```

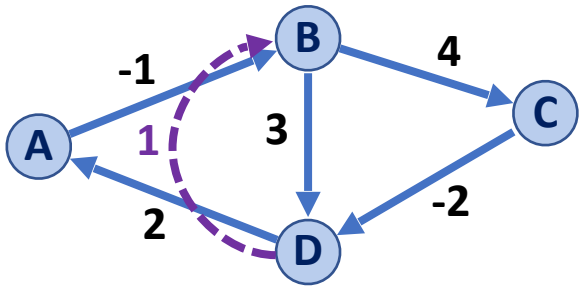
Floyd-Warshall Algorithm

We aren't storing path information! Can we fix this?

```
    FloydWarshall(G):
 6    Let d be a adj. matrix initialized to +inf
 7    foreach (Vertex v : G):
 8      d[v][v] = 0
 9    foreach (Edge (u, v) : G):
10      d[u][v] = cost(u, v)
11
12    foreach (Vertex w : G):
13      foreach (Vertex u : G):
14        foreach (Vertex v : G):
15          if (d[u, v] > d[u, w] + d[w, v])
16            d[u, v] = d[u, w] + d[w, v]
```

Floyd-Warshall Algorithm

```
    FloydWarshall(G):
 6    Let d be a adj. matrix initialized to +inf
 7    foreach (Vertex v : G):
 8      d[v][v] = 0
 9      s[v][v] = 0
10    foreach (Edge (u, v) : G):
11      d[u][v] = cost(u, v)
12      s[u][v] = v
13
14    foreach (Vertex w : G):
15      foreach (Vertex u : G):
16        foreach (Vertex v : G):
17          if (d[u, v] > d[u, w] + d[w, v])
18            d[u, v] = d[u, w] + d[w, v]
19            s[u, v] = s[u, w]
```



	A	B	C	D
A	0	-1	∞	∞
B	∞	0	4	3
C	∞	∞	0	-2
D	2	∞	∞	0

	A	B	C	D
A		B		
B			C	D
C				D
D	A			

CS 225 In Review

Lists

Stacks and Queues

Trees

Heaps

Disjoint Sets

Probability

Hash Tables

Bloom Filters

MinHash

Graphs

The End - Questions?