

Data Structures

Single Source Shortest Path

CS 225

November 4, 2024

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Exam 4 (11/13 — 11/15)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam will be on PL

Topics covered can be found on website

Registration started October 31

<https://courses.engr.illinois.edu/cs225/fa2024/exams/>

Learning Objectives

Compare Kruskal and Prim MST Algorithms

Introduce Single-Source Shortest Path Problem

Discuss Dijkstra's Algorithm

Extend to All-Paths Shortest Path (if time)

Kruskal's Algorithm

$$|V| = n, |E| = m$$

Priority Queue:	Total Running Time
Heap	$O(n) + O(m) + O(m \log n)$
Sorted Array	$O(n) + O(m \log n) + O(m)$

```
1 KruskalMST(G):
2   DisjointSets forest
3   foreach (Vertex v : G.vertices()):
4     forest.makeSet(v)
5
6   PriorityQueue Q // min edge weight
7   Q.buildFromGraph(G.edges())
8
9   Graph T = (V, {})
10
11  while |T.edges()| < n-1:
12    Vertex (u, v) = Q.removeMin()
13    if forest.find(u) != forest.find(v):
14      T.addEdge(u, v)
15      forest.union( forest.find(u),
16                  forest.find(v) )
17
18  return T
19
```

Prim's Algorithm

Sparse Graph: $m \sim n$

Adj List Heap best

Dense Graph: $m \sim n^2$

Unsorted Array best

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

	Adj. Matrix	Adj. List
Heap	$O(n^2 + m \lg(n))$	$O(n \lg(n) + m \lg(n))$
Unsorted Array	$O(n^2)$	$O(n^2)$

MST Algorithm Runtime:

Kruskal's Algorithm:
 $O(n + m \log(n))$

Prim's Algorithm:
 $O(n \log(n) + m \log(n))$

Sparse Graph: $m \sim n$

Dense Graph: $m \sim n^2$

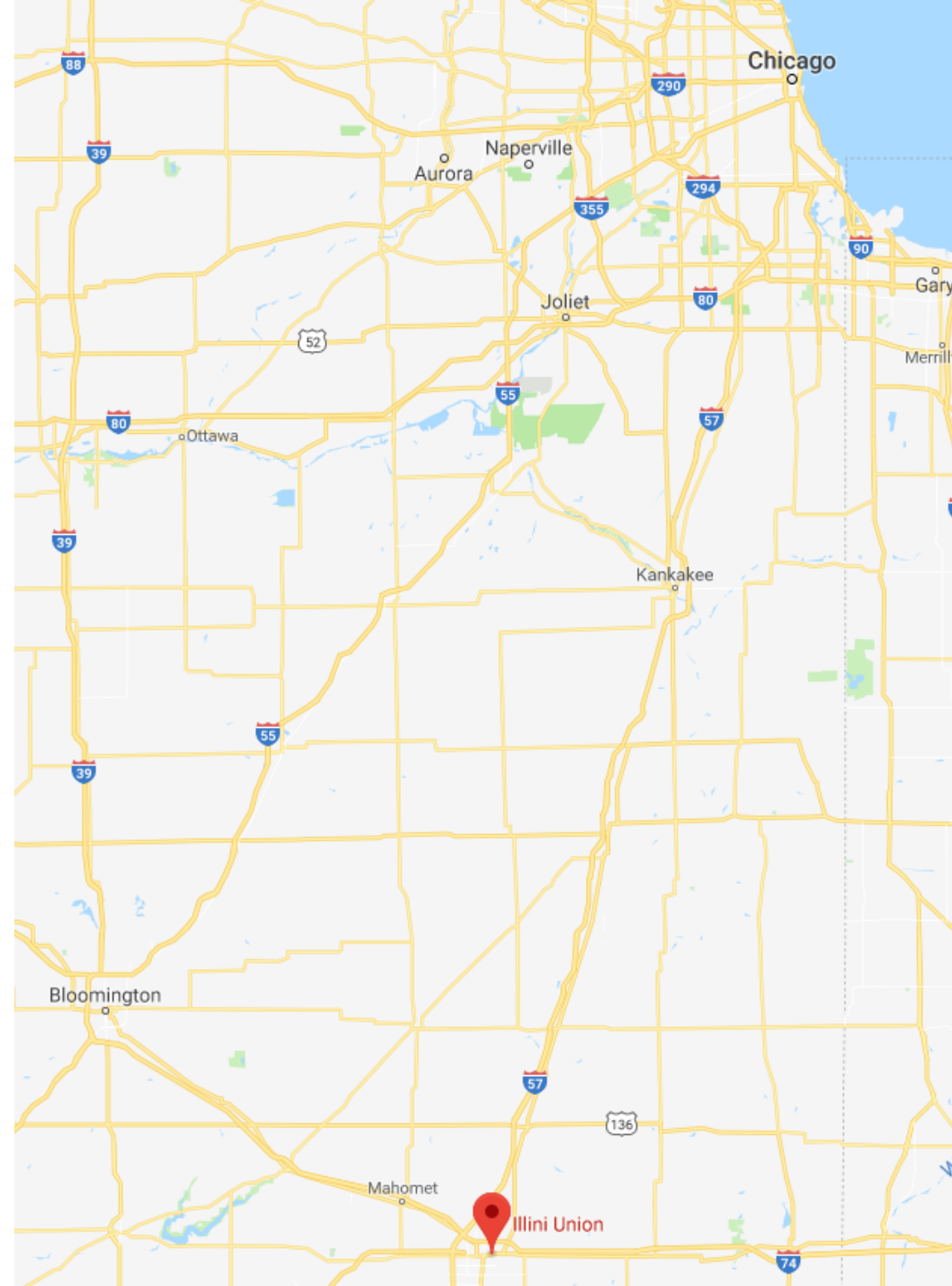
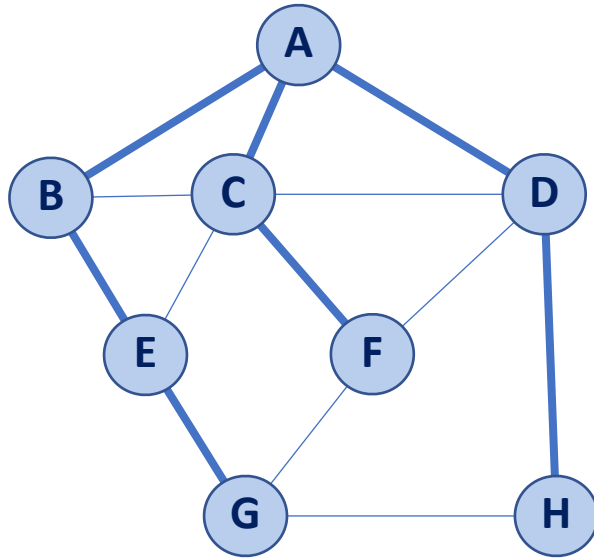
Suppose I have a new heap:

	Binary Heap	Fibonacci Heap
Remove Min	$O(\lg(n))$	$O(\lg(n))$
Decrease Key	$O(\lg(n))$	$O(1)^*$

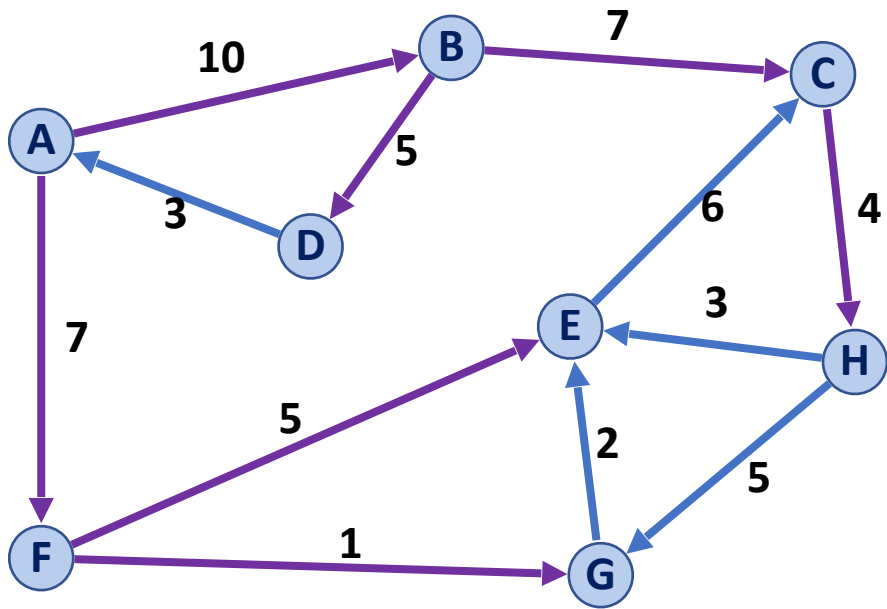
What's the updated running time?

```
PrimMST(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 m = Q.removeMin()
17       T.add(m)
18       foreach (Vertex v : neighbors of m not in T):
19           if cost(v, m) < d[v]:
20               d[v] = cost(v, m)
21               p[v] = m
```

Shortest Path



Dijkstra's Algorithm (SSSP)



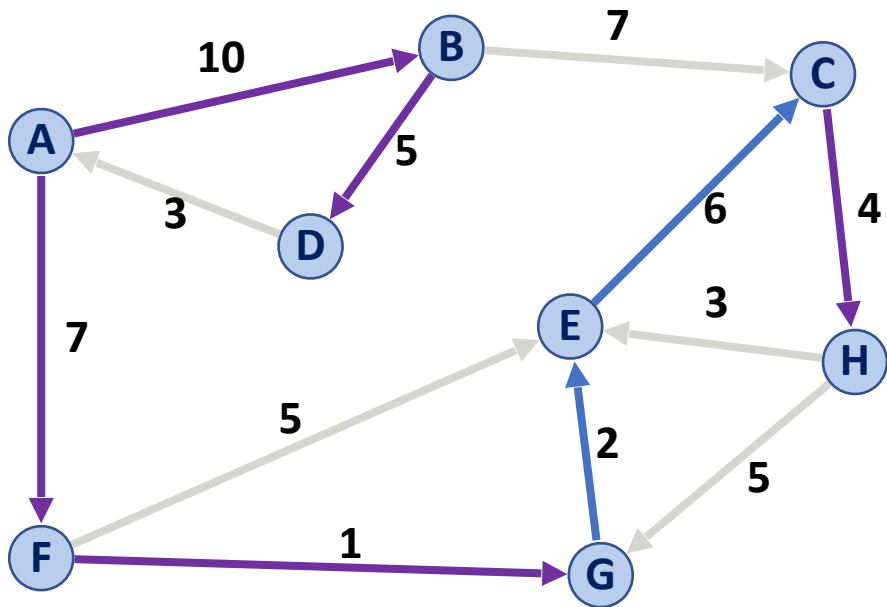
```

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 _____ < d[v]:
20        d[v] = _____
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

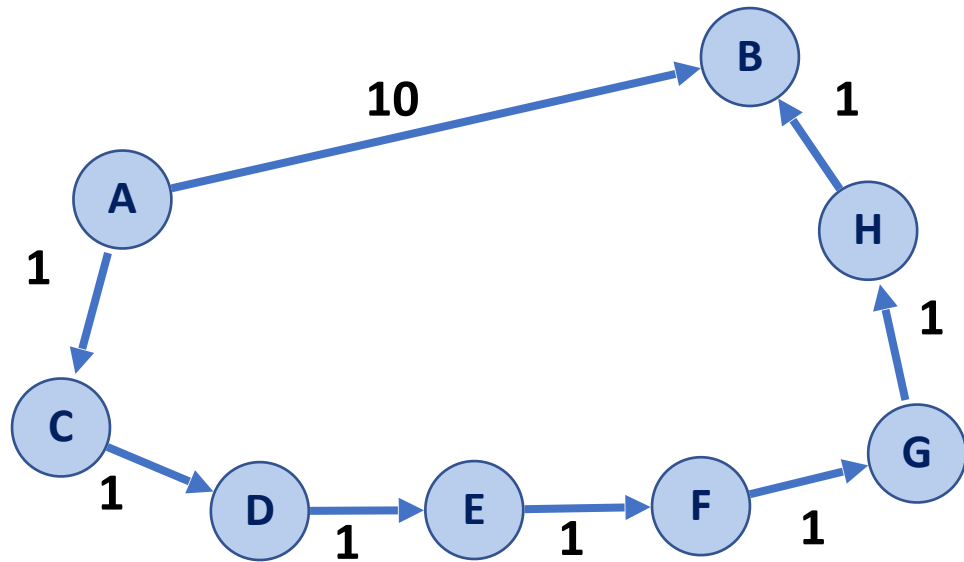
Dijkstra's Algorithm (SSSP)

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
```

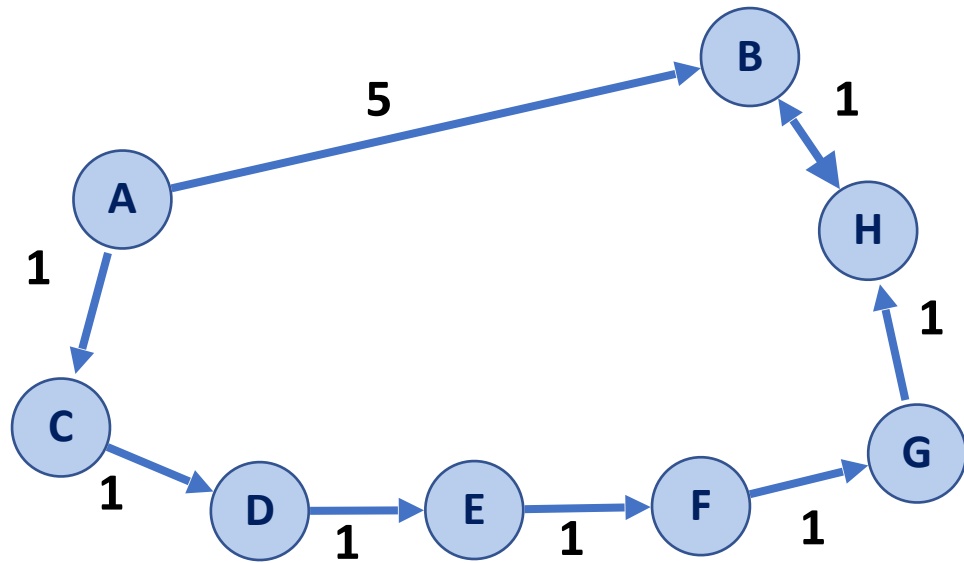
Dijkstra's Algorithm (SSSP)

When we will visit B in the following graph?



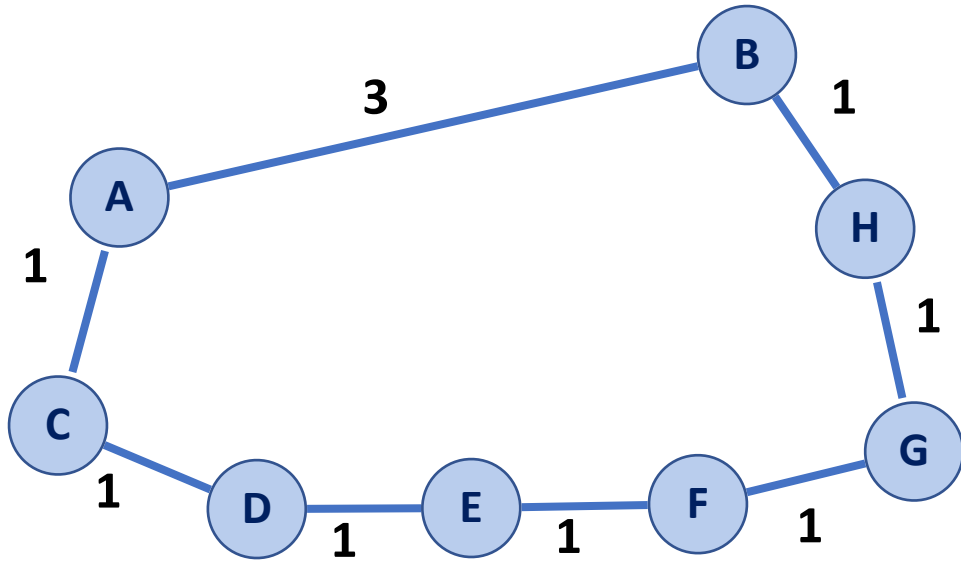
Dijkstra's Algorithm (SSSP)

When we will visit H in the following graph?



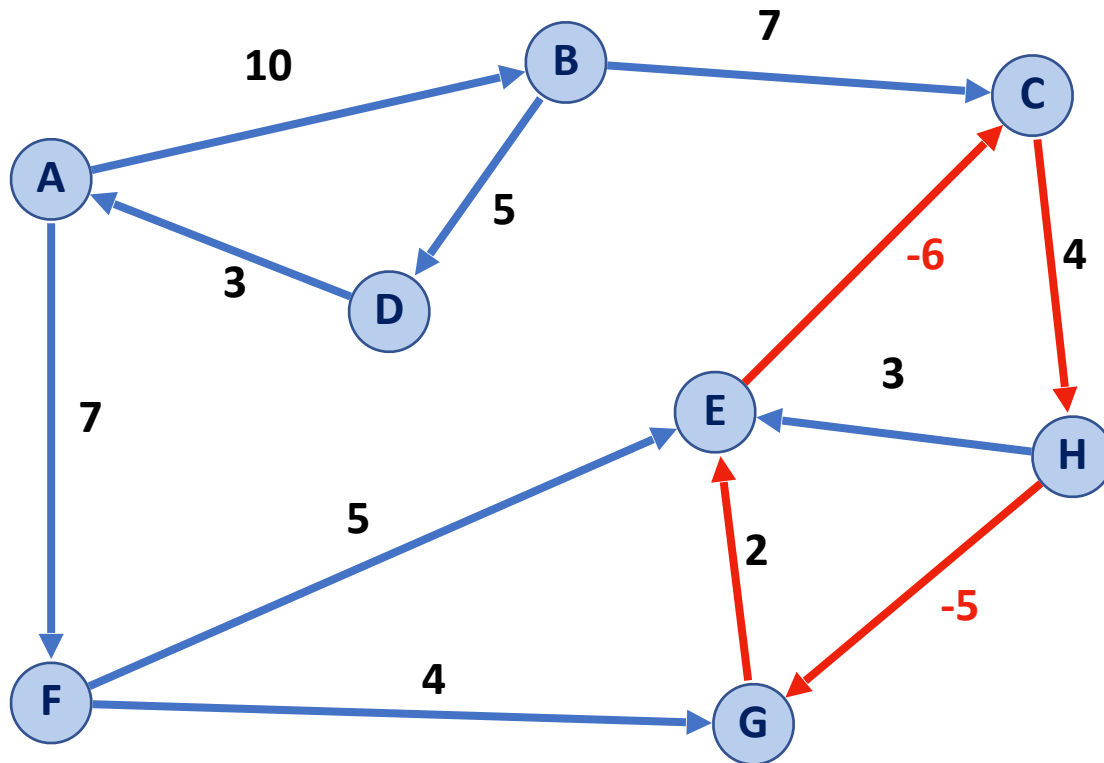
Dijkstra's Algorithm (SSSP)

How does Dijkstra's algorithm handle undirected graphs?



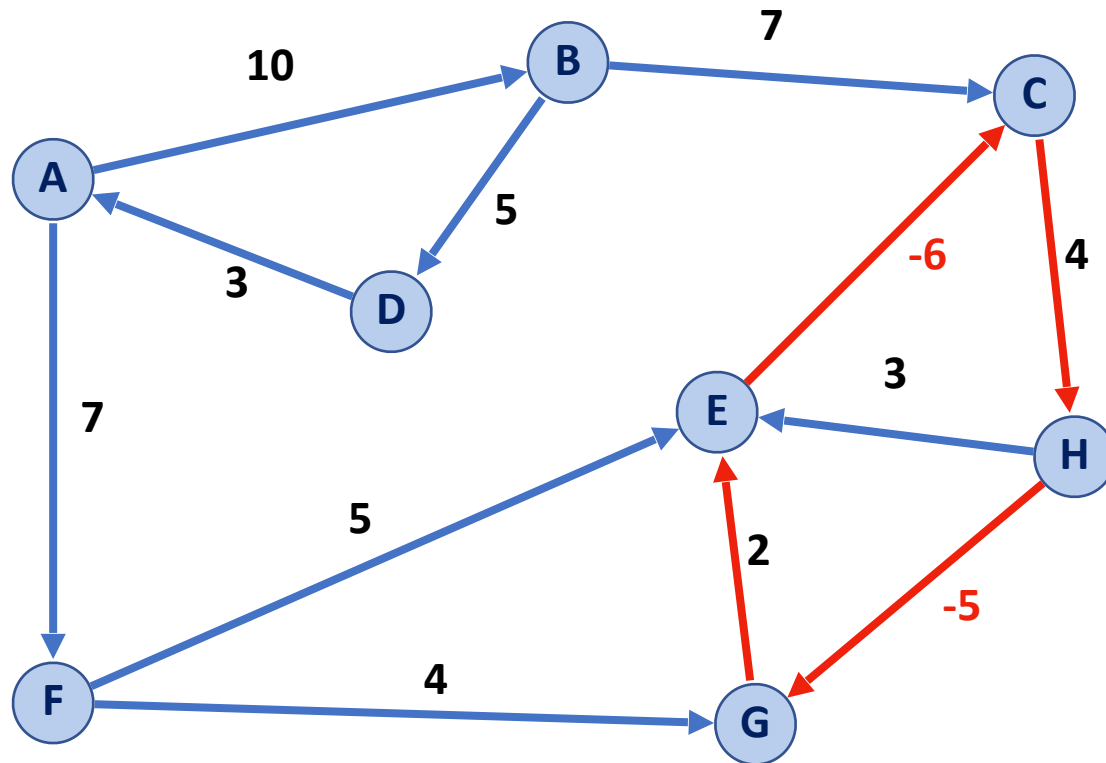
Dijkstra's Algorithm (SSSP)

How does Dijkstra's handle a negative weight cycle?



Dijkstra's Algorithm (SSSP)

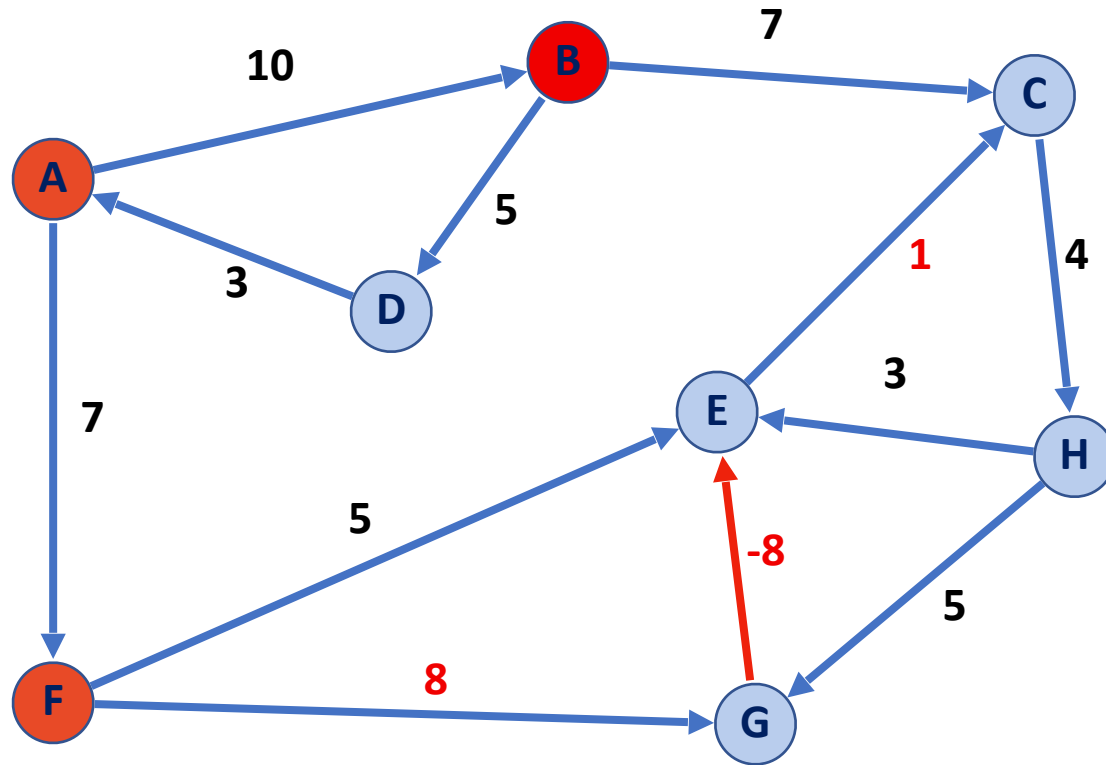
How does Dijkstra's handle a negative weight cycle?



Shortest Path (A → E): A → F → E → (C → H → G → E)*
Length: 12 Length: -5 (repeatable)

Dijkstra's Algorithm (SSSP)

How does Dijkstra's handle a negative weight edge without a cycle?



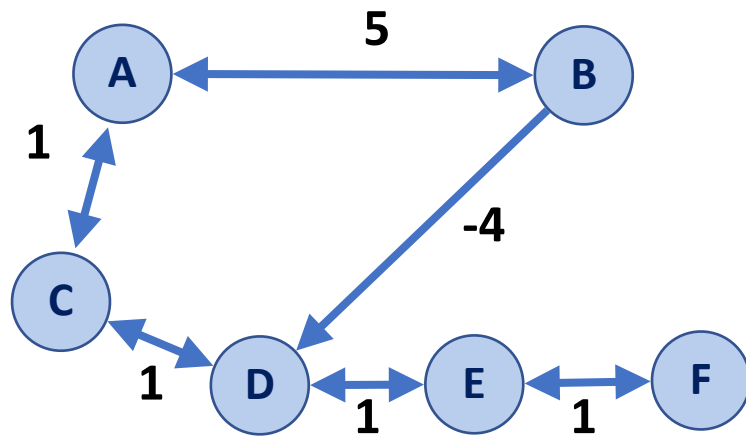
A	B	C	D	E	F	G	H
--	A	B	B	F	A	F	--
0	10	17	15	12	7	15	∞

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] = u
22              if v not in Q:
23                  Q.push(v)
24  return T
```



Dijkstra's Algorithm (SSSP)

Dijkstras Algorithm works only on non-negative weights

Optimal implementation:

Fibonacci Heap

If dense, unsorted list ties

Optimal runtime:

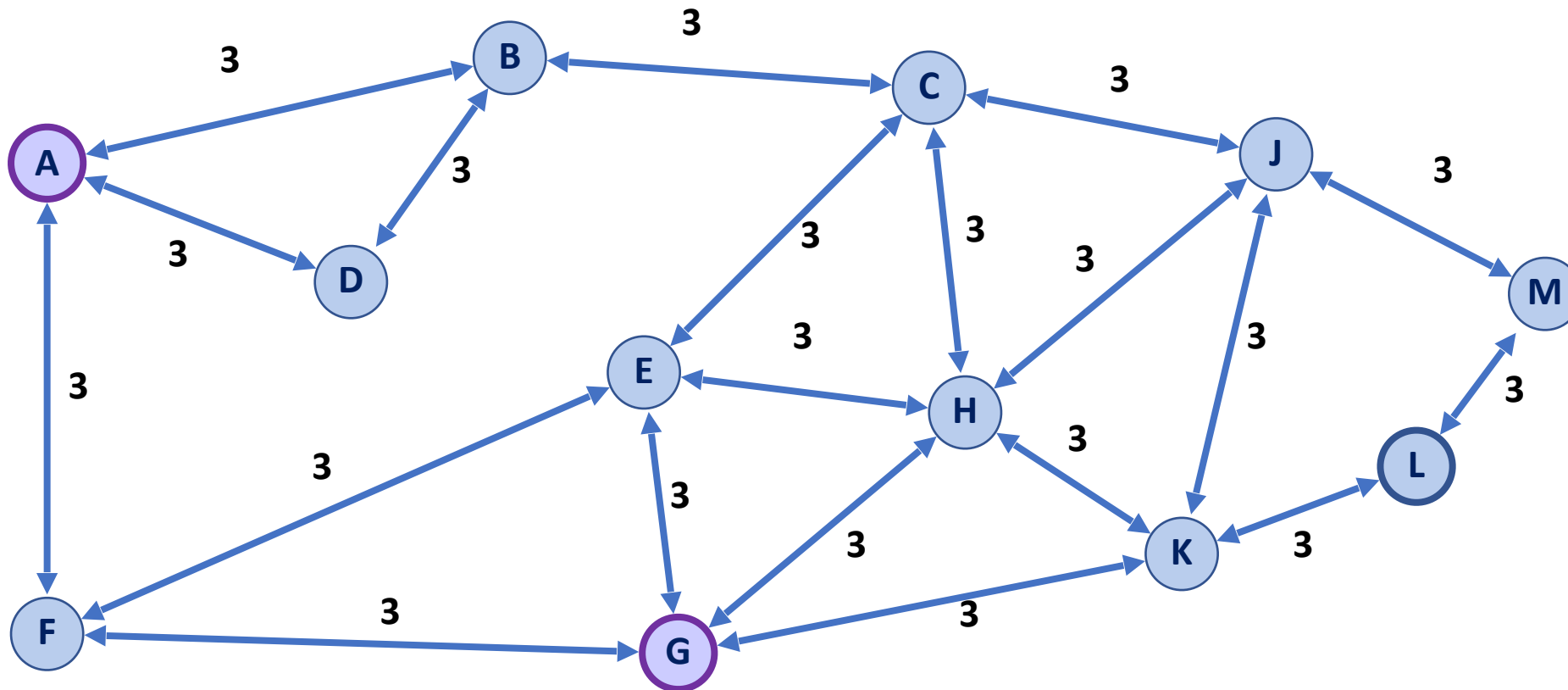
Sparse: $O(m + n \log n)$

Dense: $O(n^2)$

```
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
```

Landmark Path Problem

What if I wanted to get the shortest path from A to G but stopping at L along the way?



Floyd-Warshall Algorithm

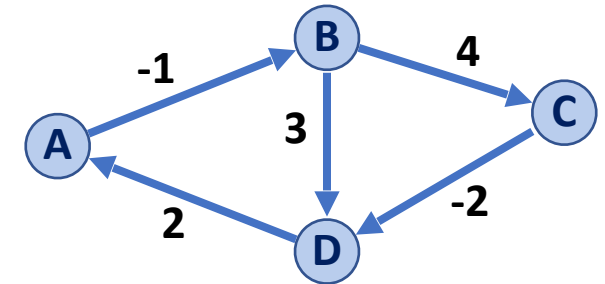
Floyd-Warshall's Algorithm is an alternative to Dijkstra in the presence of **negative-weight edges (not negative weight cycles)**.

```
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 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]
```

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				

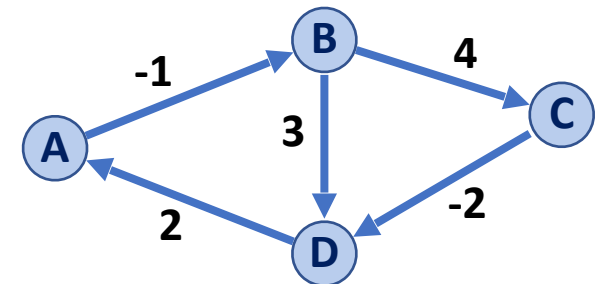


Floyd-Warshall Algorithm

```
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

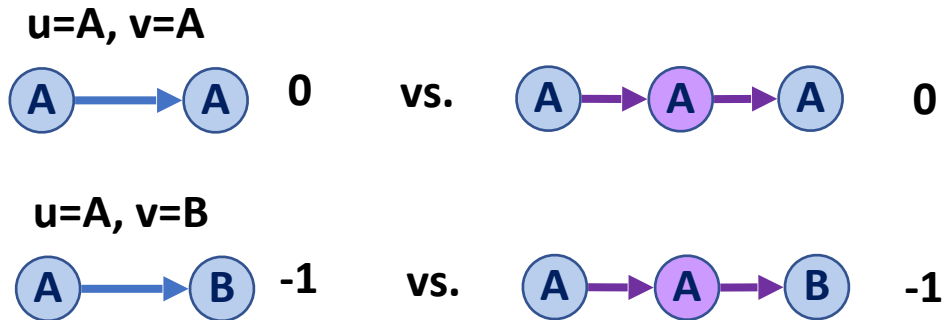


Floyd-Warshall Algorithm

```
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]
```

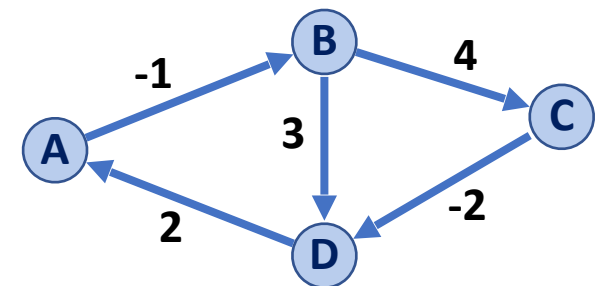
Let **w** be midpoint
Let **u** be start point
Let **v** be end point
Is our distance shorter now?

Let us consider comparisons where $w = A$:



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

Don't waste time if $u=w$ or $v=w$!



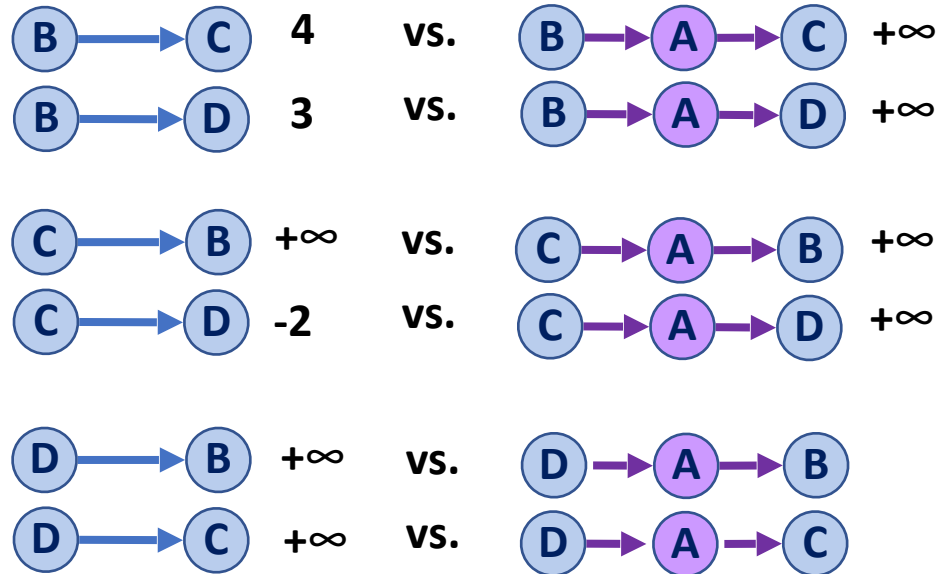
Floyd-Warshall Algorithm

Let **w** be midpoint
 Let **u** be start point
 Let **v** be end point
 Is our distance shorter now?

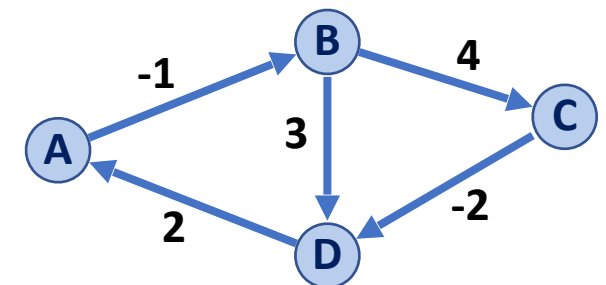
```

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 $u \neq w$ and $v \neq w$):



	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]
    
```

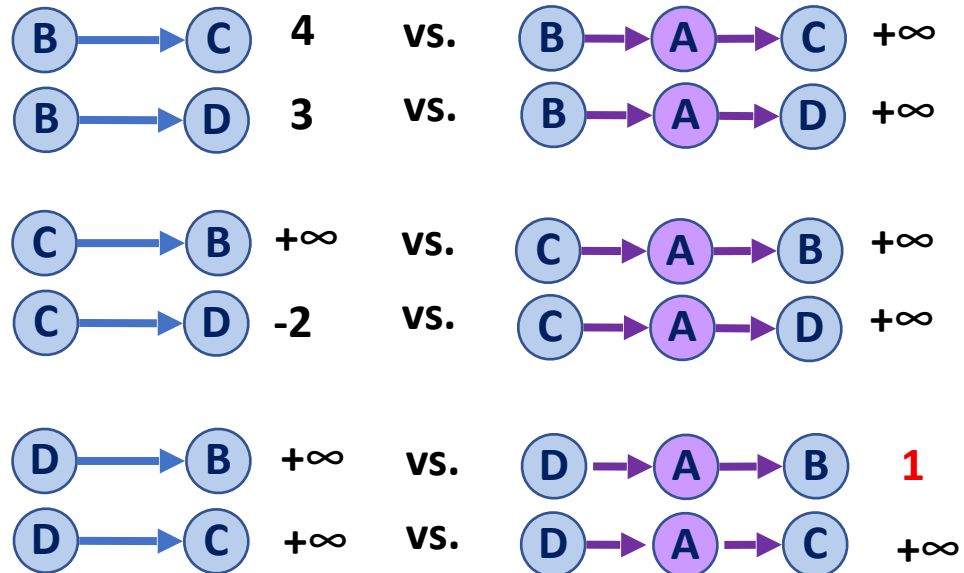
Let **w** be midpoint

Let **u** be start point

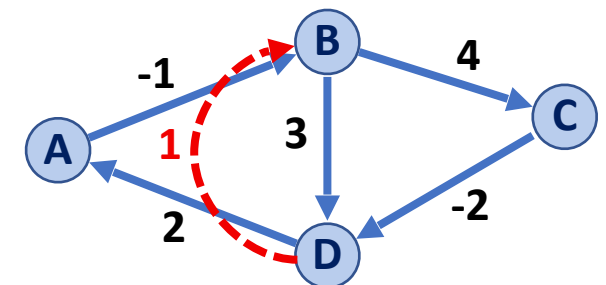
Let **v** be end point

Is our distance shorter now?

Let us consider $w = A$ (and $u \neq w$ and $v \neq w$):



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



Floyd-Warshall Algorithm

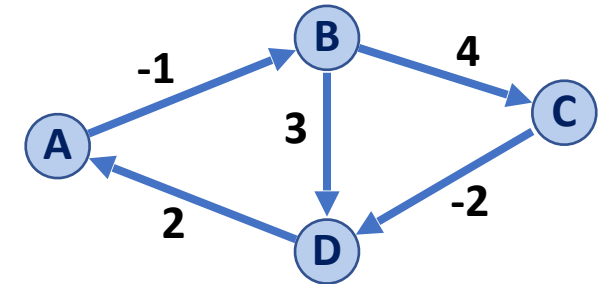
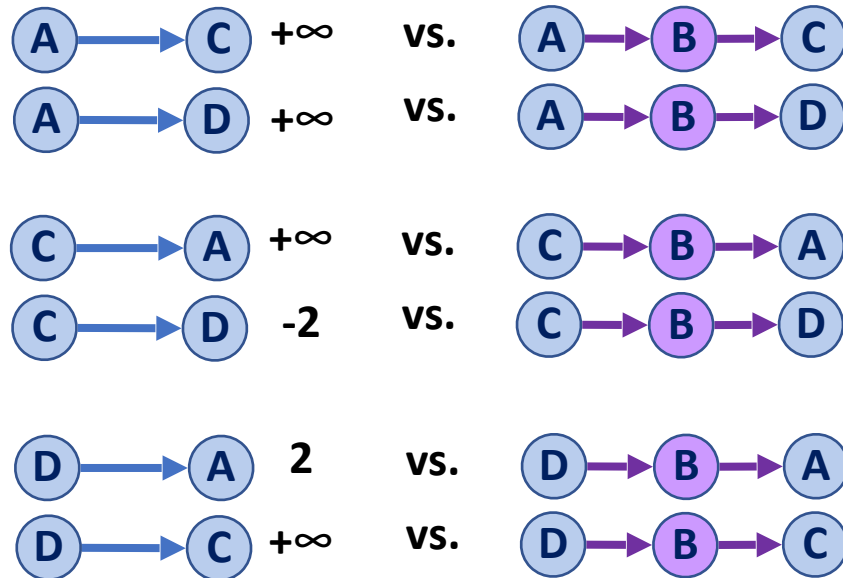
```

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]

```

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

Let us consider $w = B$ (and $u \neq w$ and $v \neq w$):



Floyd-Warshall Algorithm

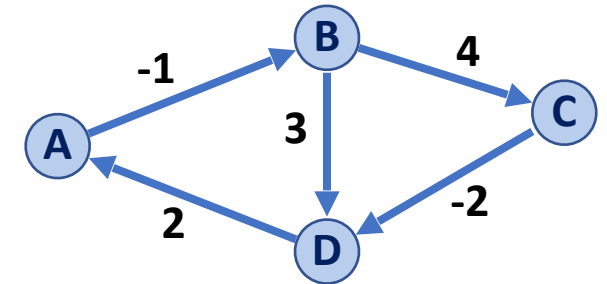
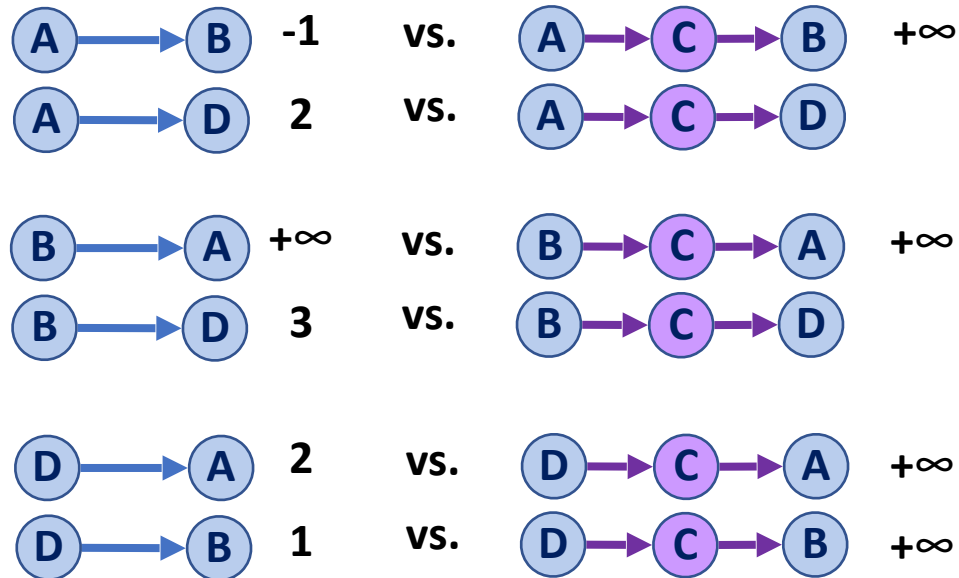
```

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]

```

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

Let us consider $w = C$ (and $u \neq w$ and $v \neq w$):

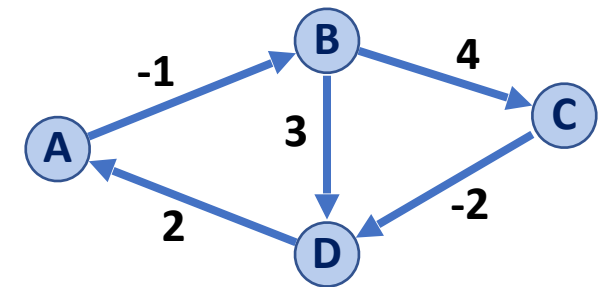


Floyd-Warshall Algorithm



```
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 u : G) :  
13    foreach (Vertex v : G) :  
14      foreach (Vertex w : 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

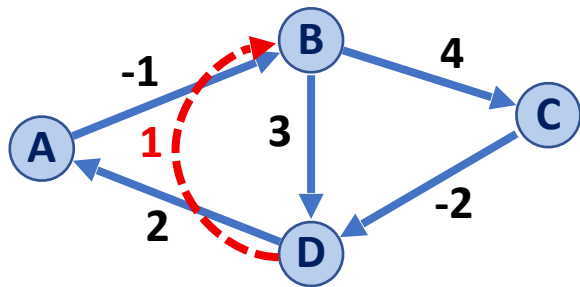
We aren't storing path information! Can we fix this?

```
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			