Design and Analysis of Algorithms
Single-source shortest paths, all-pairs shortest paths

Slides from: Haidong Xue and Dr. Fethi Jarray

Provided By: Muhammad Atif Tahir

Presented by: Farrukh Salim Shaikh

Shortest-Path Problems

– **Single-source.** Find a shortest path from a given source to each of the vertices

– **Single-pair.** Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too

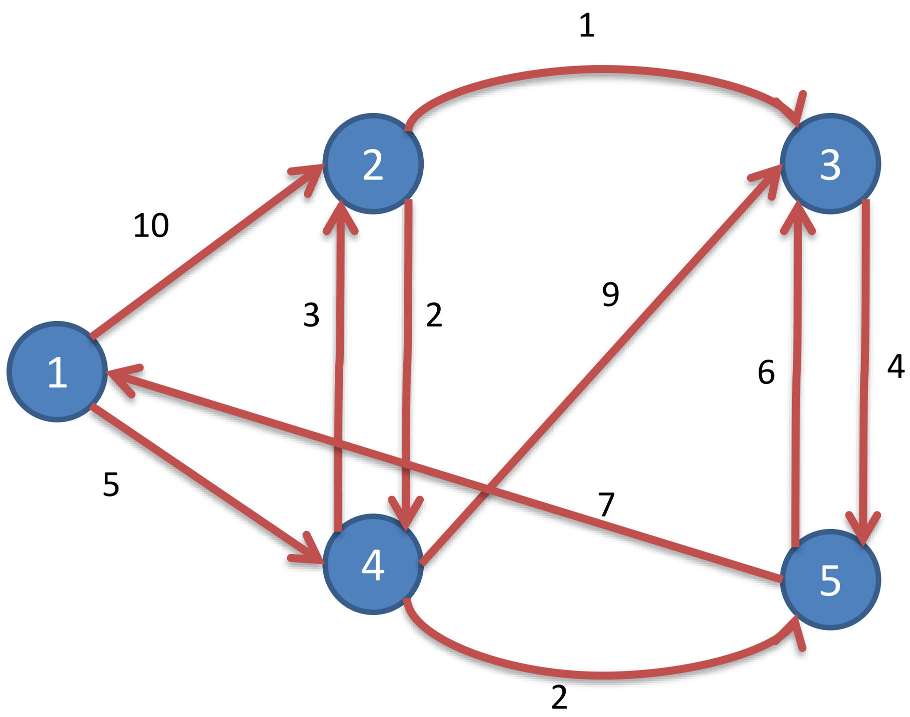
– **All-pairs.** Find shortest-paths for every pair of vertices

Single-source shortest paths

- A **path** of a weighted, directed graph is a sequence of vertices: $< v_1, v_2, \dots, v_k >$
- The **weight of a path** is the sum of weights of edges that make the path:

weight($< v_1, v_2, \dots, v_k >$)=$\sum_{i=1}^{k-1} w_{(v_i, v_{i+1})}$

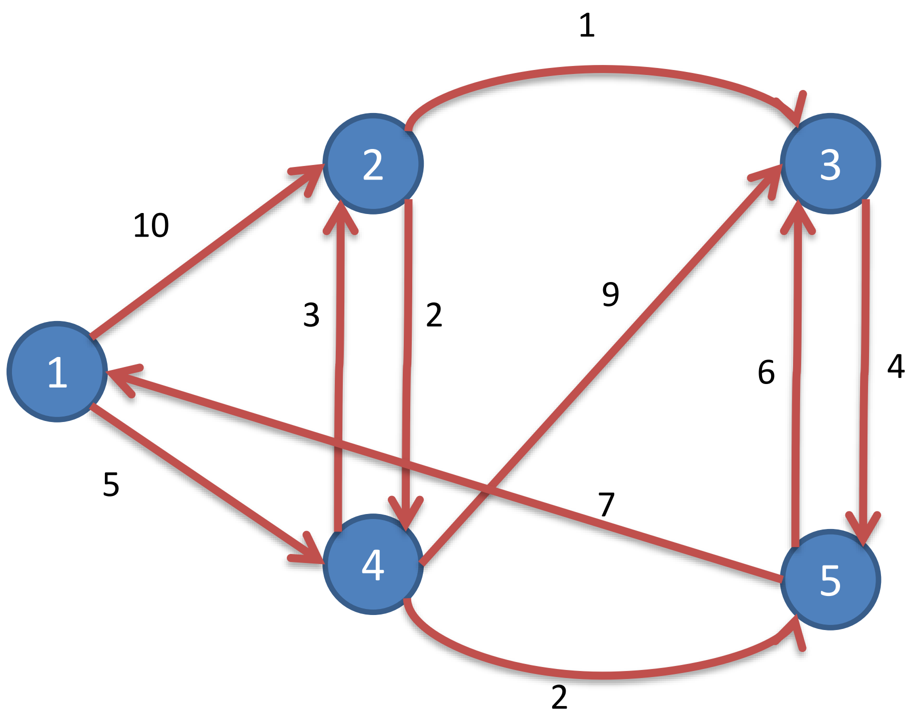
Single-source shortest paths



Given this weighted, directed graph, what is the weight of path: <1, 2, 4>?

10+2=12

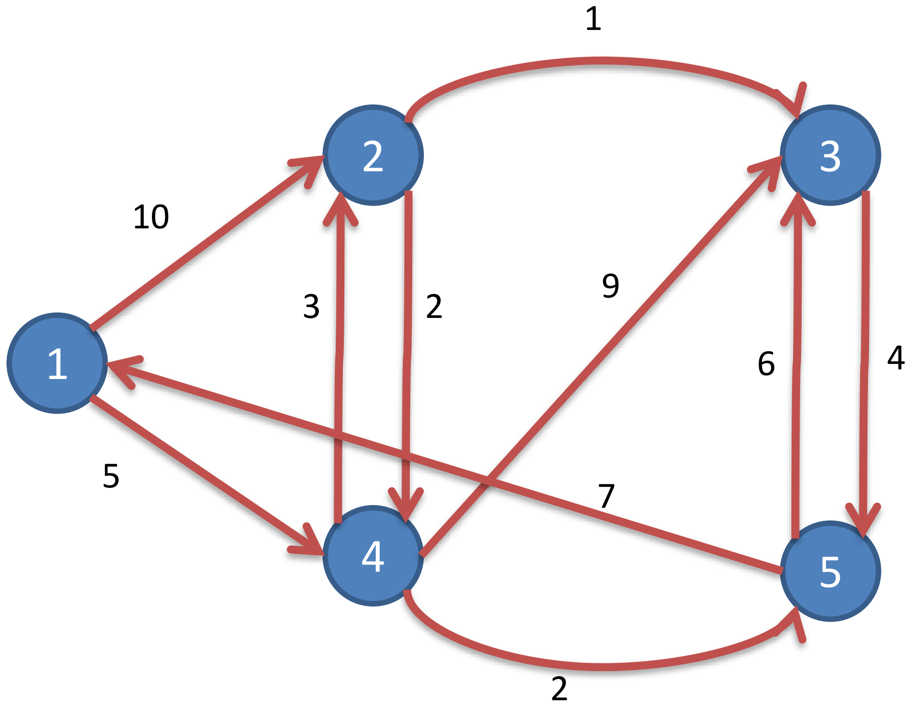
Single-source shortest paths



Given this weighted, directed graph, what is the weight of path: <1, 2, 4, 2, 4>?

10+2 + 3 + 2 = 17

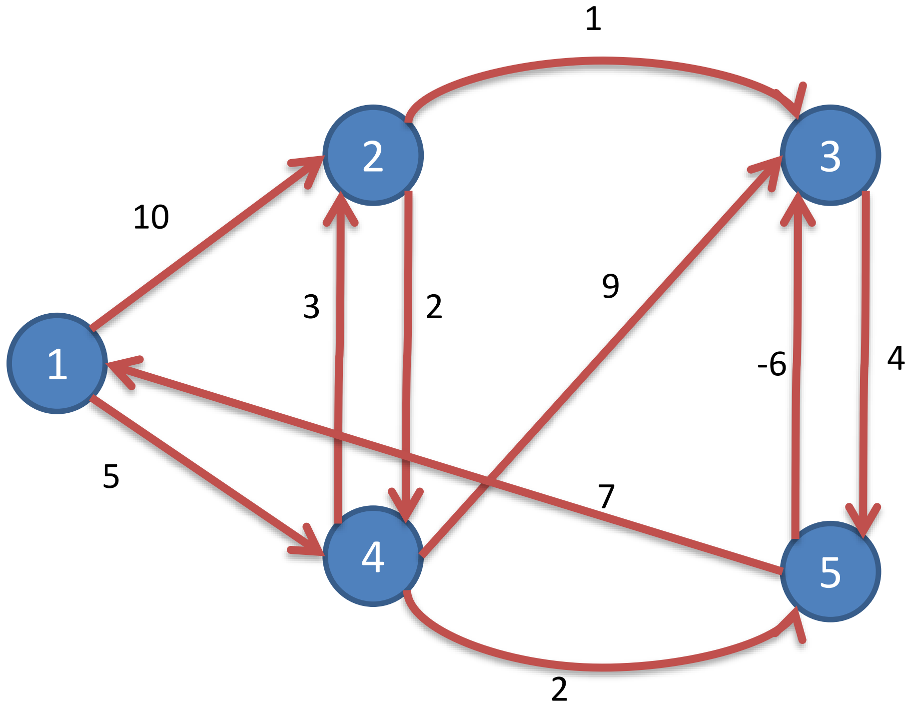
Single-source shortest paths



Given this weighted, directed graph, what is the weight of path: <1, 2, 4, 1>?

$10 + 2 + \infty = \infty$

Single-source shortest paths



Given this weighted, directed graph, what is the weight of path: <5, 3, 5> and <5, 3, 5, 3, 5>??

4-6 = -2 and -6+4-6+4 = -4

Negative cycle   There is no shortest path from 3 to 5

Single-source shortest paths

- **Shortest path** of a pair of vertices <u, v>: a path from u to v, with minimum path weight
- Applications:
  - Your GPS navigator
  - If weights are time, it produces the fastest route
  - If weights are gas cost, it produces the lowest cost route
  - If weights are distance, it produces the shortest route

Single-source shortest paths

- **Single-source shortest path problem:** given a **weighted**, **directed** graph G=(V, E) with source vertex s, find all the shortest (least weight) paths from s to all vertices in V.

Single-source shortest paths

- Two classic algorithms to solve single-source shortest path problem
  – Bellman-Ford algorithm
    - A dynamic programming algorithm
    - Works when some weights are negative
  – Dijkstra’s algorithm
    - A greedy algorithm
    - Faster than Bellman-Ford
    - Works when weights are all non-negative

Bellman-Ford algorithm

Observation:

- If there is a negative cycle, there is no solution
  - Add this cycle again can always produces a less weight path
- If there is no negative cycle, a shortest path has at most |V|-1 edges

Idea:

- Solve it using dynamic programming
- For all the paths have at most 0 edge, find all the shortest paths
- For all the paths have at most 1 edge, find all the shortest paths
- …
- For all the paths have at most |V|-1 edge, find all the shortest paths

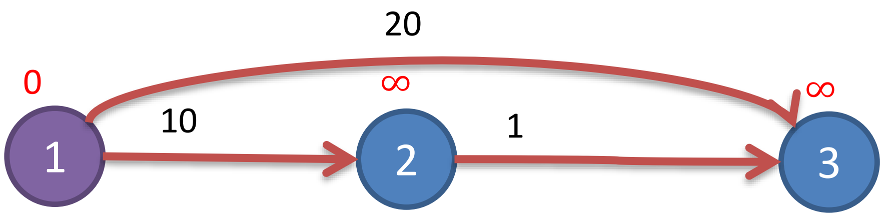
Bellman-Ford algorithm

```
Bellman-Ford(G, s)
    for each v in G.V{   //Initialize 0-edge shortest paths
        if(v==s) d_{s,v}=0; else d_{s,v}= ∞; //set the 0-edge shortest distance
                                               from s to v
        π_{s,v} = NIL;  //set the predecessor of v on the shortest path
    }
    Repeat |G.V|-1 times {   //bottom-up construct 0-to-(|V|-1)-edges shortest paths
        for each edge (u, v) in G.E{
            if(d_{s,v} > d_{s,u}+ w_{(u,v)}){
                d_{s,v} = d_{s,u}+ w_{(u,v)};
                π_{s,v} = u;
            }
    }
    for each edge (u, v) in G.E{   //test negative cycle
        If (d_{s,v} > d_{s,u}+ w_{(u,v)}) return false; // there is no solution
    }
    return true;
```

T(n)=O(VE)=O(V^3)

Bellman-Ford algorithm (Example 1)

e.g.



What is the 0-edge shortest path from 1 to 1?

<> with path weight 0

What is the 0-edge shortest path from 1 to 2?

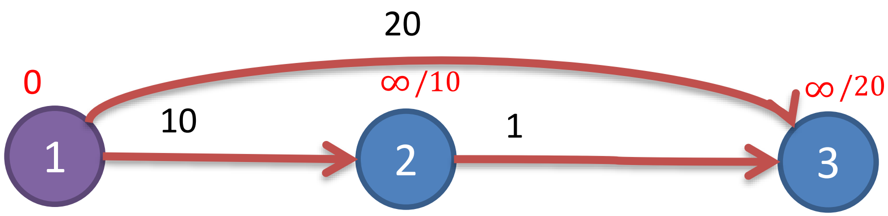
<> with path weight ∞

What is the 0-edge shortest path from 1 to 3?

<> with path weight ∞

Bellman-Ford algorithm (Example 1)

e.g.



$\infty > 0 + 20$

$d_{1,3} = 20$

$\infty > 0 + 10$

$d_{1,2} = 10$

$\infty = \infty+1$

$d_{1,3}$ *unchanged*

What is the at most 1-edge shortest path from 1 to 1?

<> with path weight 0

What is the at most 1-edge shortest path from 1 to 2?

<1, 2> with path weight 10

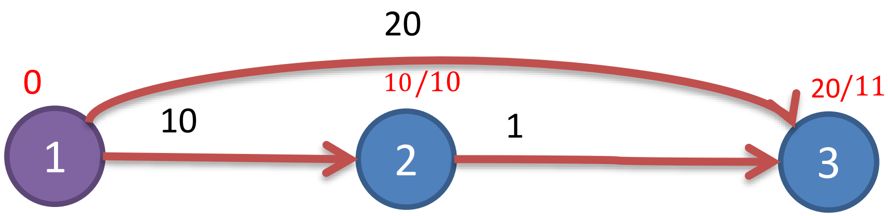
What is the at most 1-edge shortest path from 1 to 3?

<1, 3> with path weight 20

In Bellman-Ford, they are calculated by scan all edges once

Bellman-Ford algorithm (Example 1)

e.g.



$0 + 20 = 20$

$d_{1,3}\ unchanged$

$10 = 0 + 10$

$d_{1,2}\ unchanged$

$20 > 10+1$

$d_{1,3} = 11$

What is the at most 2-edges shortest path from 1 to 1?

<> with path weight 0

What is the at most 2-edges shortest path from 1 to 2?

<1, 2> with path weight 10

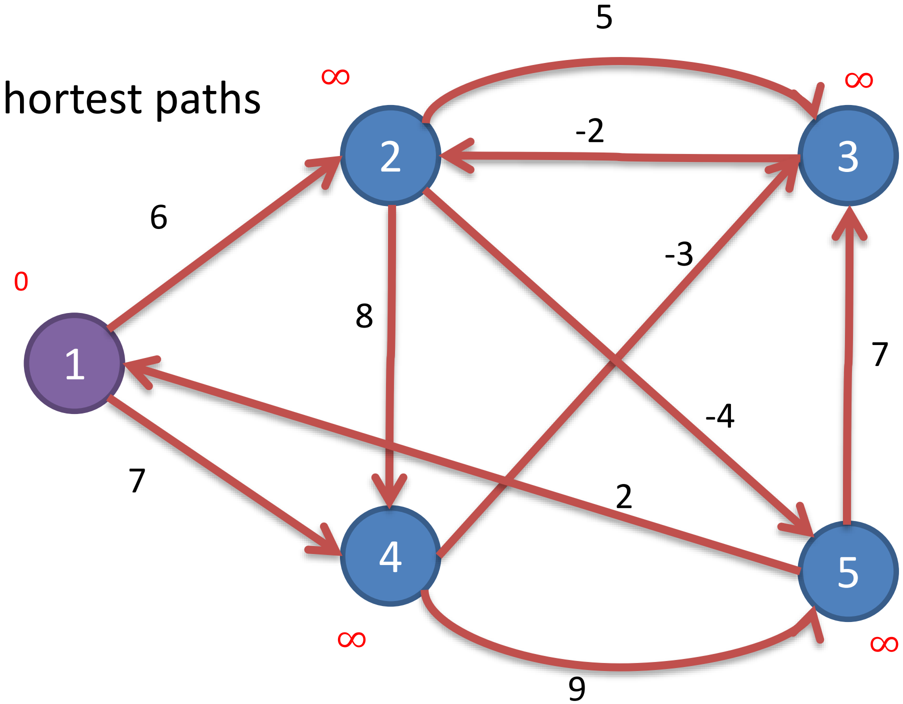
What is the at most 2-edges shortest path from 1 to 3?

<1, 2, 3> with path weight 11

In Bellman-Ford, they are calculated by scan all edges once

Bellman-Ford algorithm (Example 2)

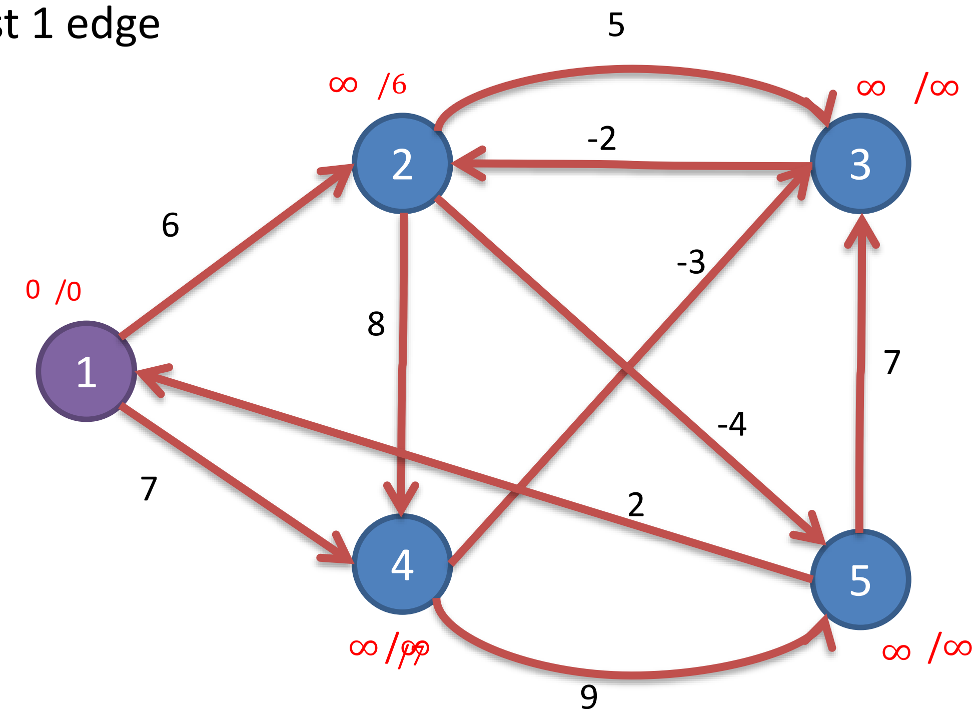
All 0 edge shortest paths



	1	2	3	4	5
d	0	∞	∞	∞	∞
π	/	/	/	/	/

Bellman-Ford algorithm (Example 2)

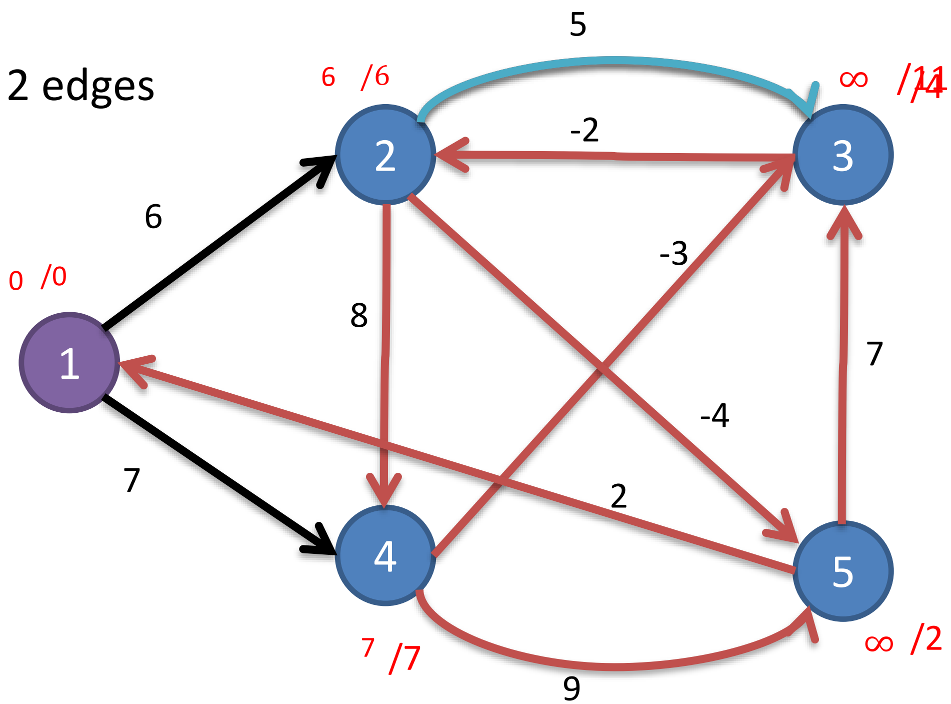
Calculate all at most 1 edge shortest paths



	1	2	3	4	5
d	0	6	∞	7	∞
π	/	1	/	1	/

Bellman-Ford algorithm (Example 2)

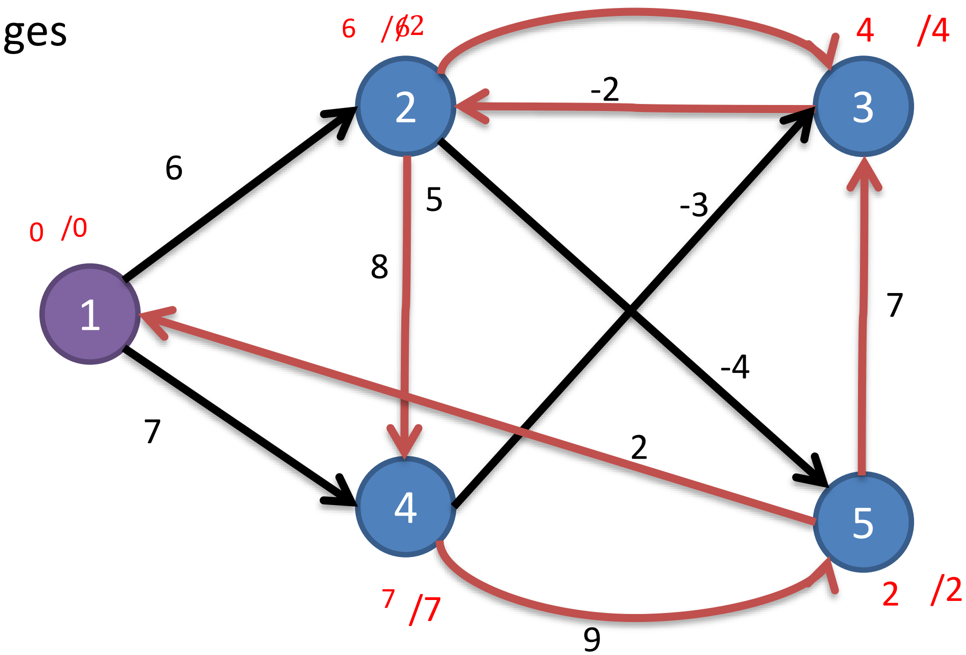
Calculate all at most 2 edges shortest paths



	1	2	3	4	5
d	0	6	4	7	2
π	/	1	4	1	2

Bellman-Ford algorithm (Example 2)

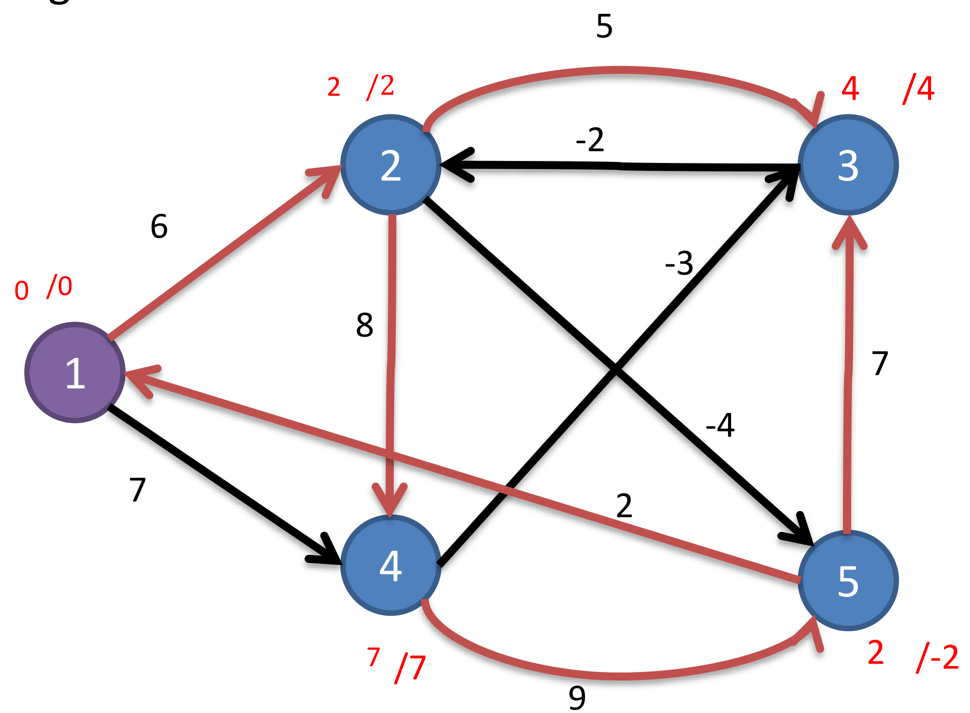
Calculate all at most 3 edges shortest paths



	1	2	3	4	5
d	0	2	4	7	2
π	/	3	4	1	2

Bellman-Ford algorithm (Example 2)

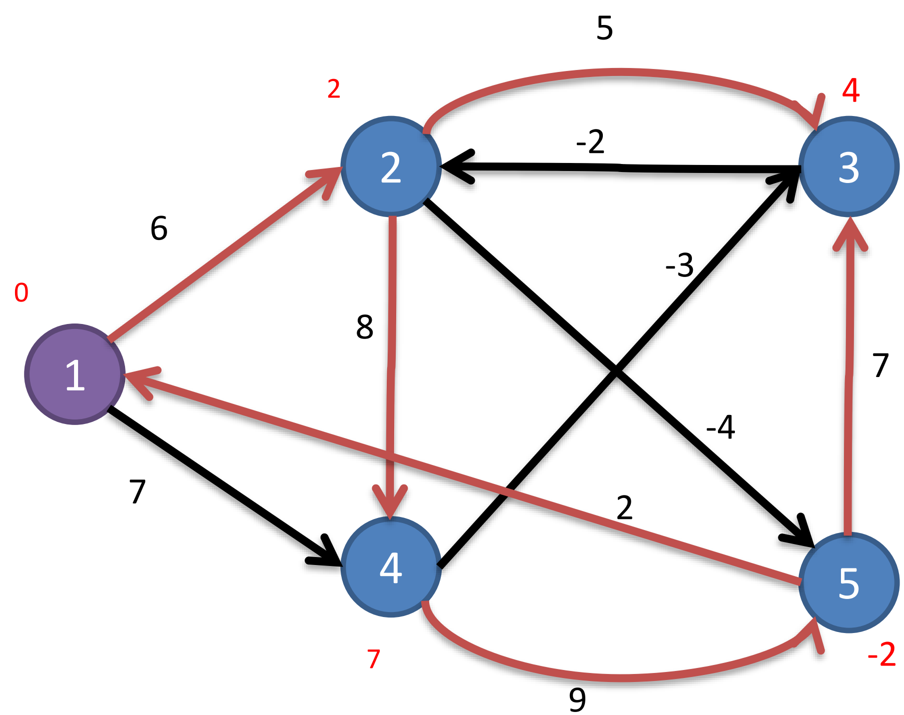
Calculate all at most 4 edges shortest paths



	1	2	3	4	5
d	0	2	4	7	-2
π	/	3	4	1	2

Bellman-Ford algorithm (Example 2)

Final result:



What is the shortest path from 1 to 5?

1, 4, 3, 2, 5

What is weight of this path? -2

What is the shortest path from 1 to 2, 3, and 4?

Dijkstra's Algorithm (Pseudocode 1)

- A greedy algorithm
- **Dijkstra** (G, s)

for each v in G.V{

  if(v==s) $d_{s,v}$=0; **else** $d_{s,v}$= ∞; //set the 0-edge shortest distance from s to v

  $\pi_{s,v} = \mathrm{NIL}$; //set the predecessor of v on the shortest path

}

S=∅;//the set of vertices whose final shortest-path weights have already been determined

Q=G.V;

while(Q ≠ ∅){

  u=Extract-Min(Q);

  S=S ∪ $\{u\}$;

  for all (u, v){//the greedy choice

    if($d_{s,v} > d_{s,u} + w_{(u,v)}$){

      $d_{s,v} = d_{s,u} + w_{(u,v)}$;

      $\pi_{s,v} = u$;

    }

  }

}

T(n)=O(V^2)

Dijkstra's Algorithm (Pseudocode 2)

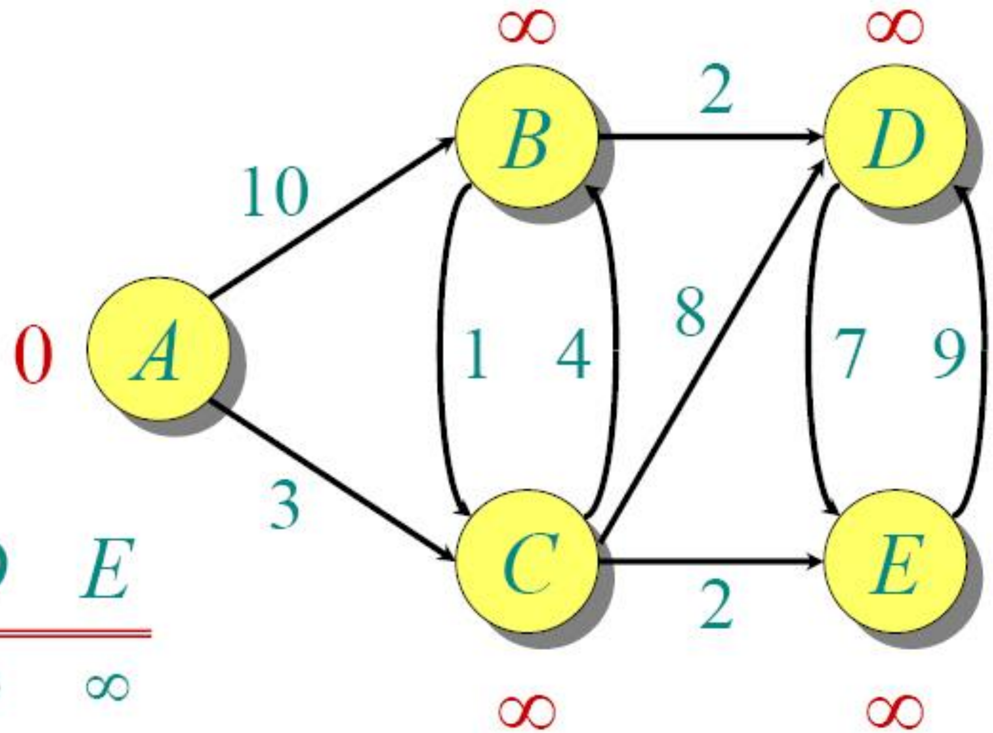
```
dist[s] ←o                                    (distance to source vertex is zero)
for  all v ∈ V–{s}
      do  dist[v] ←∞                          (set all other distances to infinity)
S←∅                                           (S, the set of visited vertices is initially empty)
Q←V                                           (Q, the queue initially contains all vertices)
while Q ≠∅                                    (while the queue is not empty)
do   u ← mindistance(Q,dist)                  (select the element of Q with the min. distance)
     S←S ∈{u}                                 (add u to list of visited vertices)
      for all v ∈ neighbors[u]  do
               dist[v]= min(dist[v], dist[u] + w(u, v))  (update distance)
return dist
```

Dijkstra Example

Initialize:

dist[s] ←o
for all v ∈ V–{s}
do dist[v] ←∞
S←Ø
Q←V

Q:	A	B	C	D	E
	0	∞	∞	∞	∞

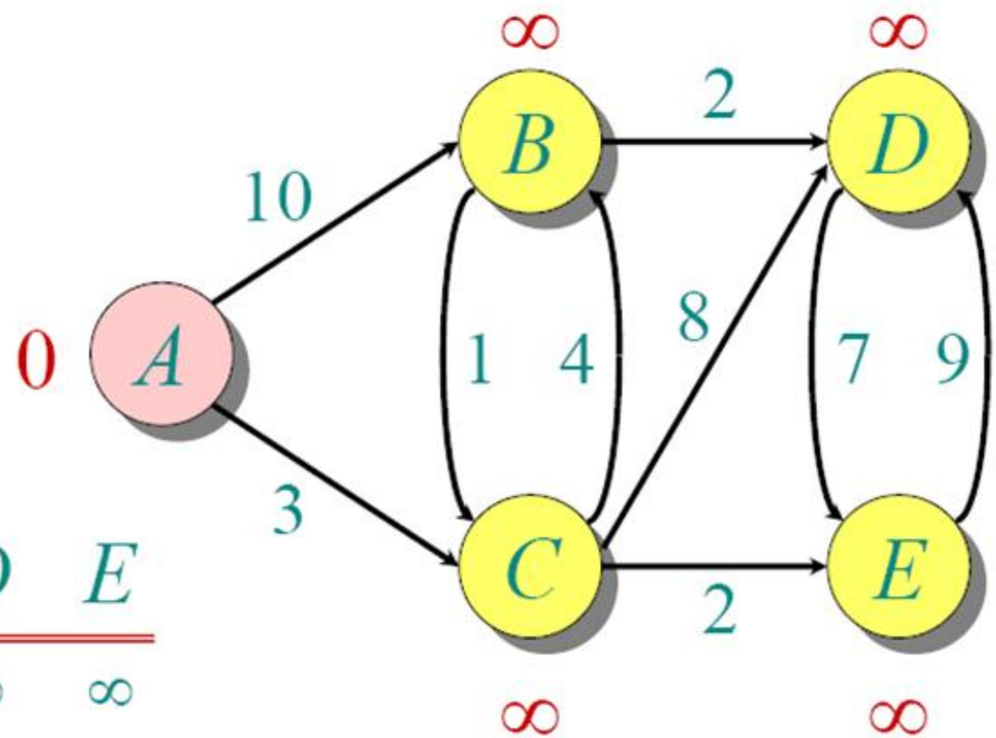


S: {}

Dijkstra Example

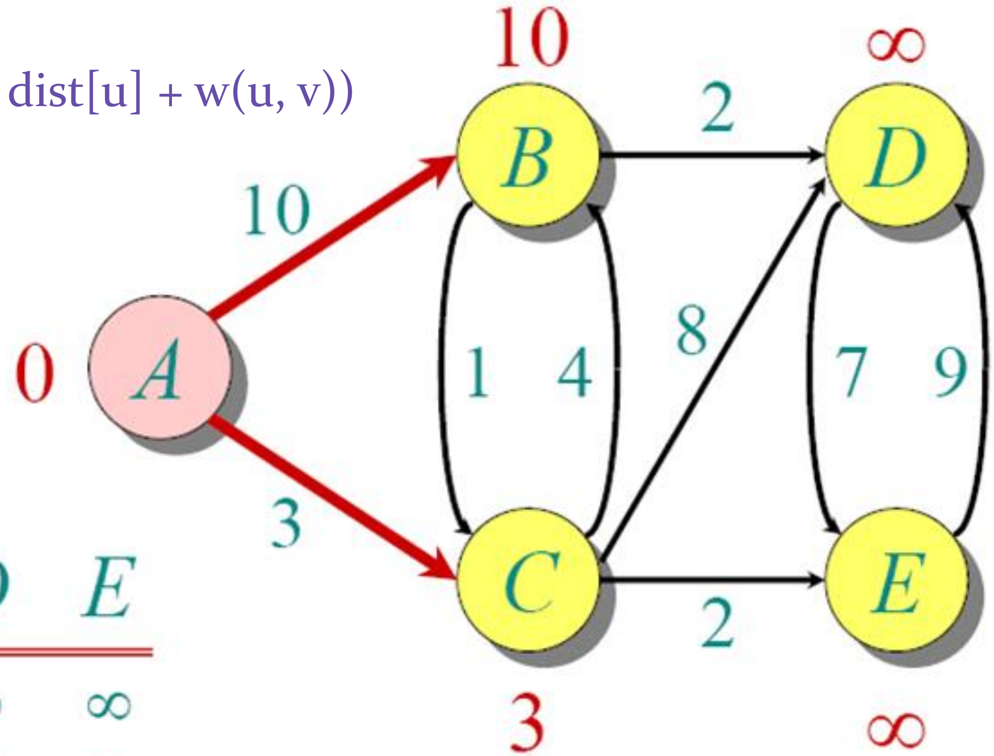
dist[s] ← 0
for all v ∈ V–{s}
    do dist[v] ← ∞
S ← ∅
Q ← V

Q:	A	B	C	D	E
	0	∞	∞	∞	∞



Dijkstra Example

while $Q \neq \emptyset$
do $u \leftarrow$ mindistance(Q,dist)
$S \leftarrow S \in \{u\}$
for all $v \in$ neighbors[u] do
dist[v]= min(dist[v], dist[u] + w(u, v))
return dist



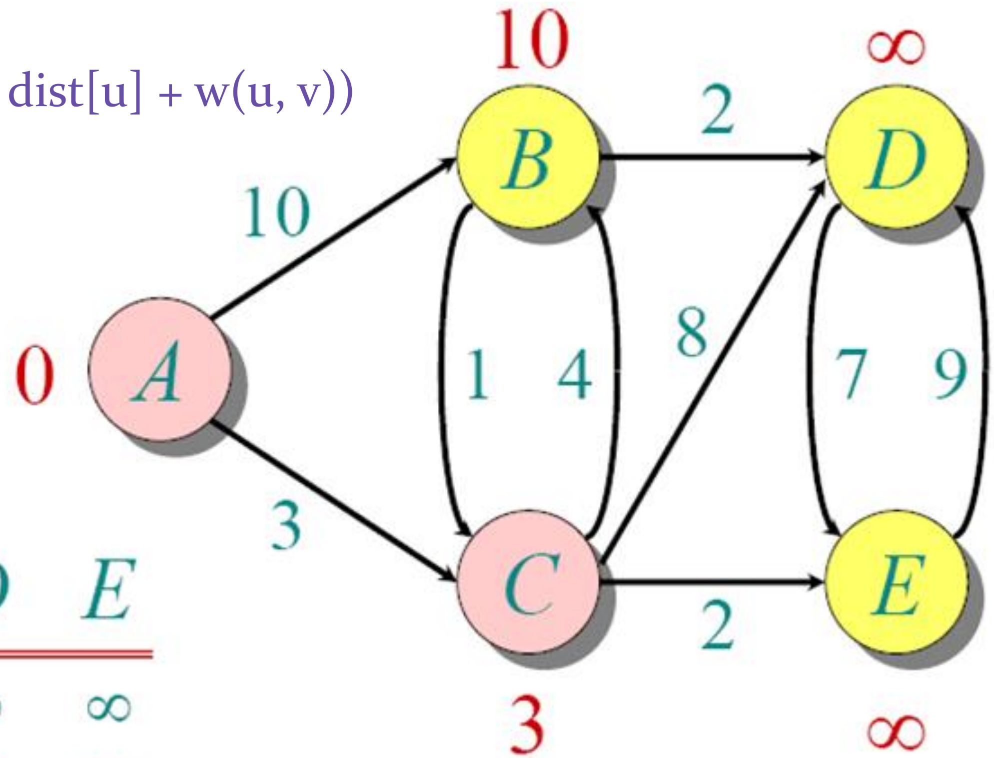
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

dist[s] $\leftarrow$o
for all $v \in V-\{s\}$
do dist[v] $\leftarrow \infty$
$S \leftarrow \emptyset$
$Q \leftarrow V$

Dijkstra Example

while $Q \neq \emptyset$
do $u \leftarrow$ mindistance(Q,dist)
$S \leftarrow S \in \{u\}$
for all $v \in$ neighbors[u] do
dist[v]= min(dist[v], dist[u] + w(u, v))
return dist

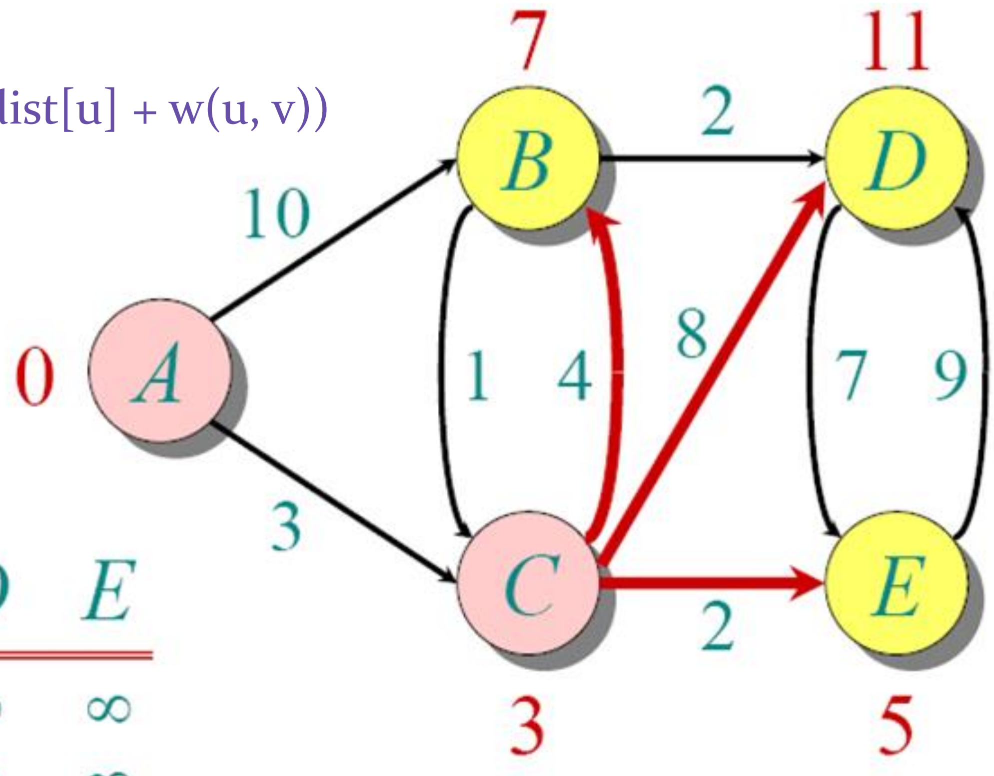


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

Dijkstra Example

while $Q \neq \emptyset$
do $u \leftarrow$ mindistance(Q,dist)
$S \leftarrow S \in \{u\}$
for all $v \in$ neighbors[u] do
dist[v]= min(dist[v], dist[u] + w(u, v))
return dist

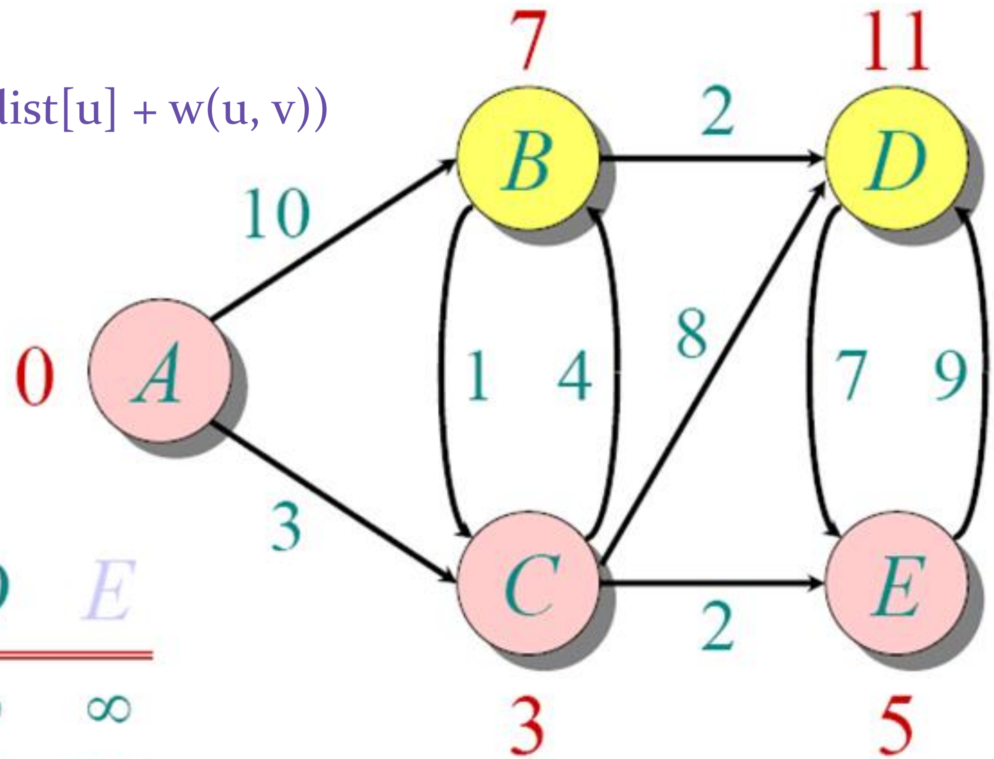


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

$S: \{ A, C \}$

Dijkstra Example

while Q ≠∅
do u ← mindistance(Q,dist)
  S←S ∈{u}
  for all v ∈ neighbors[u] do
    dist[v]= min(dist[v], dist[u] + w(u, v))
return dist

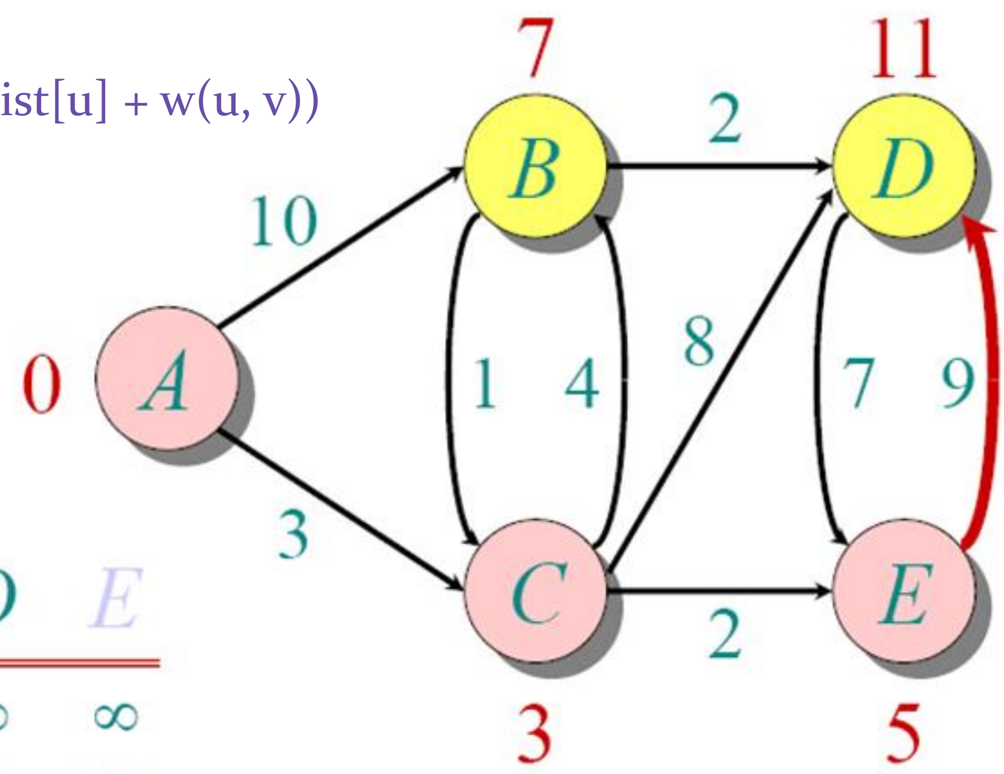


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

Dijkstra Example

```
while Q ≠∅
do   u ← mindistance(Q,dist)
     S←S ∈{u}
      for all v ∈ neighbors[u]  do
              dist[v]= min(dist[v], dist[u] + w(u, v))
return dist
```

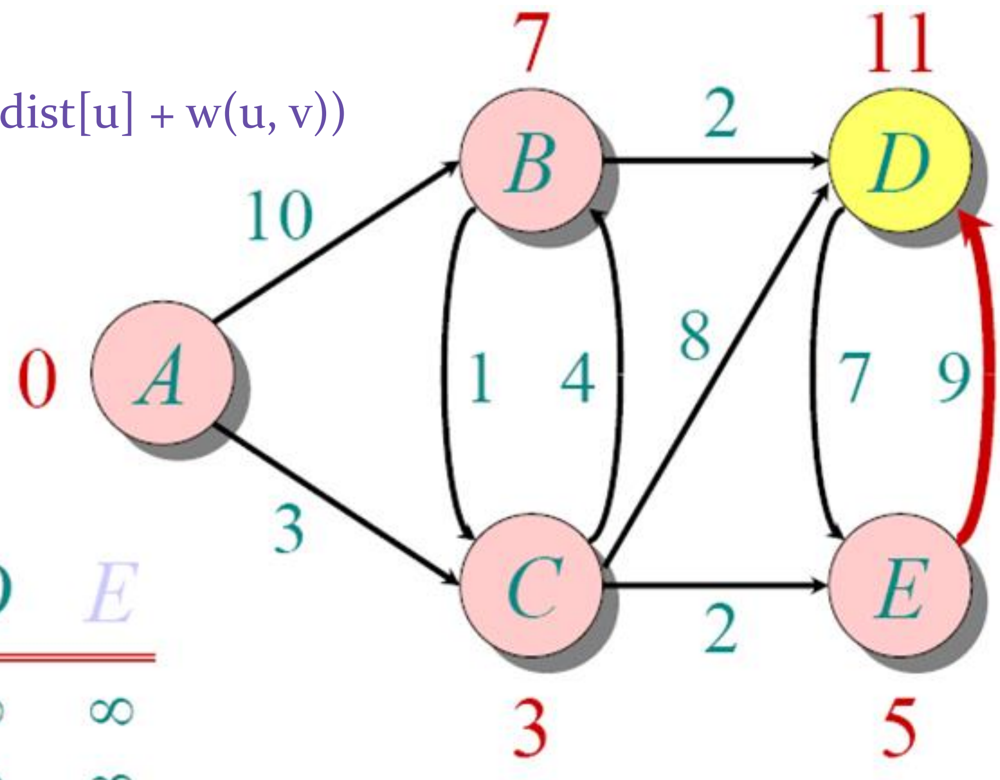


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

Dijkstra Example

while $Q \neq \emptyset$
do $u \leftarrow$ mindistance(Q,dist)
$S \leftarrow S \in \{u\}$
for all $v \in$ neighbors[u] do
dist[v]= min(dist[v], dist[u] + w(u, v))
return dist

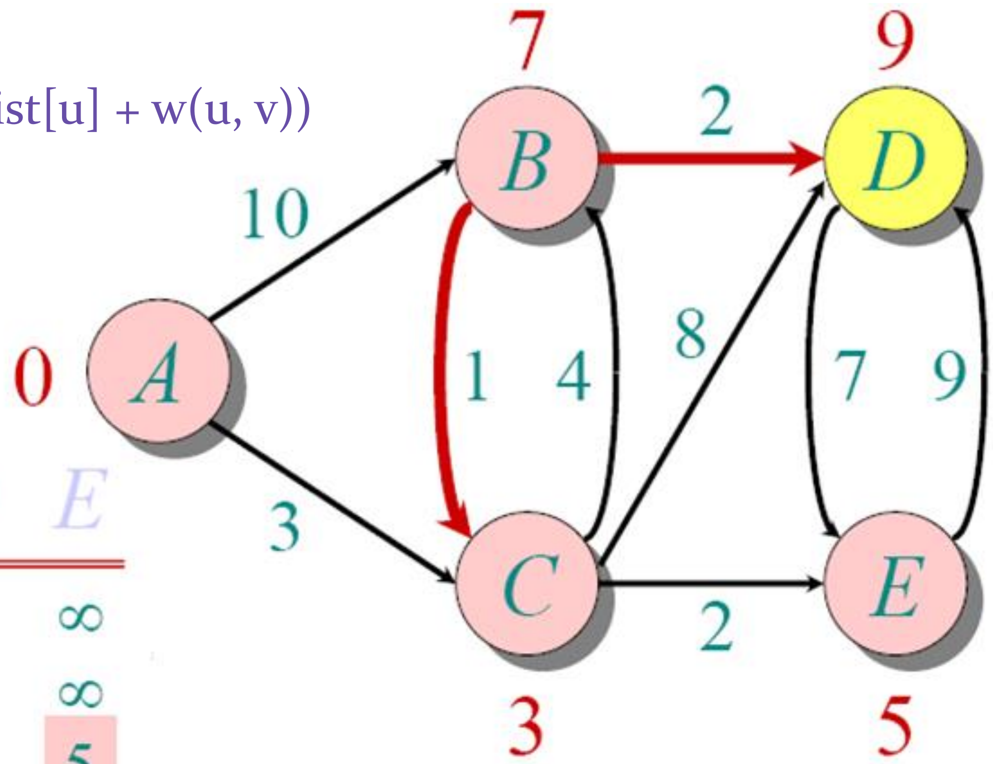


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

Dijkstra Example

while Q ≠∅
do   u ← mindistance(Q,dist)
    S←S ∈{u}
    for all v ∈ neighbors[u] do
        dist[v]= min(dist[v], dist[u] + w(u, v))
return dist

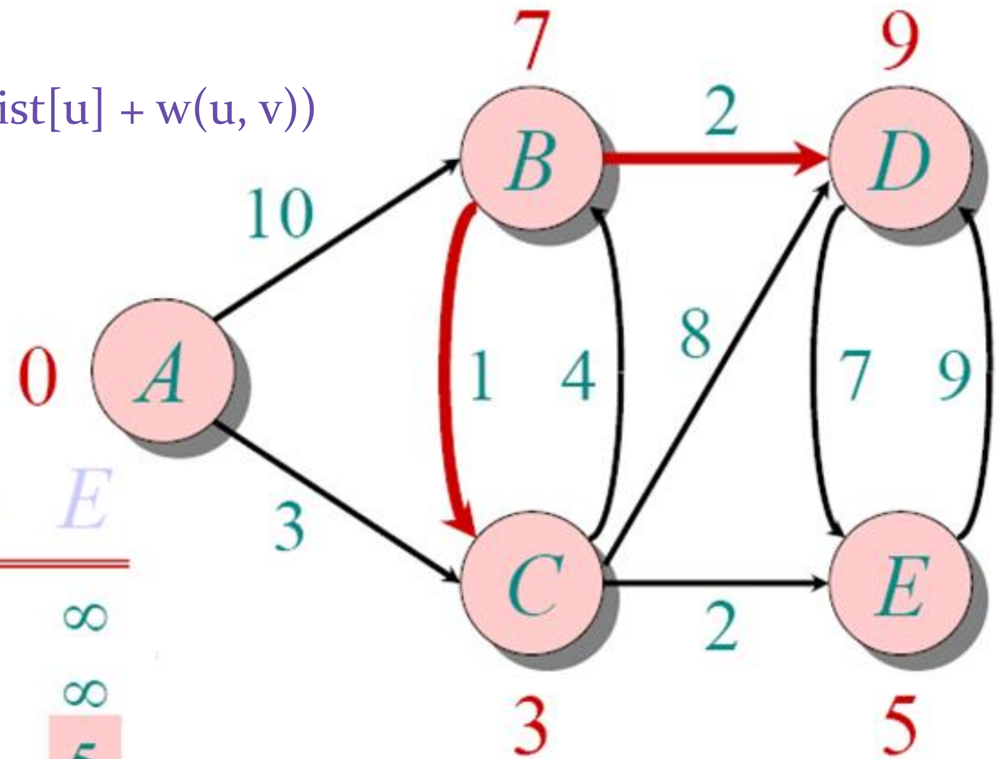


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B }

Dijkstra Example

while Q ≠∅
do u ← mindistance(Q,dist)
    S←S ∈{u}
    for all v ∈ neighbors[u] do
        dist[v]= min(dist[v], dist[u] + w(u, v))
return dist



Q:

A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

S: { A, C, E, B, D }

Dijkstra’s algorithm Running Time

- Initialization : $\Theta(V)$
- *|V| iterations*
- *Each iteration:* $\Theta(V)$
- Total time: $O(V^2)$

Dijkstra's algorithm Running Time

- Extract-Min executed $|V|$ time
- Decrease-Key executed $|E|$ time
- Time = $|V|\ T_{\text{Extract-Min}} + |E|\ T_{\text{Decrease-Key}}$
- T depends on different Q implementations

Q	T(Extract -Min)	T(Decrease-Key)	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$ (amort.)	$O(V \lg V + E)$

- Decrease key means: ~~update better (lower) value~~
  - Means: Removing value
- Binary heap: O((V + E) log V)

Dijkstra's Algorithm (Pseudocode 2)

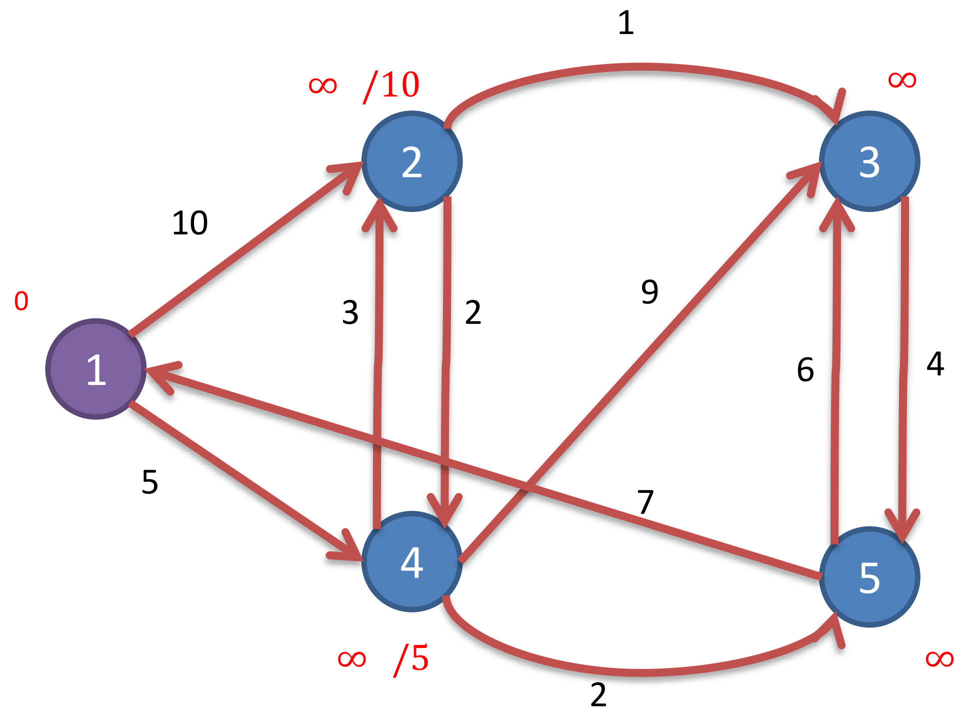
$dist[s] \leftarrow 0$
for all $v \in V-\{s\}$
    do $dist[v] \leftarrow \infty$
    prev [v] = nil
$S \leftarrow \emptyset$
$Q \leftarrow V$ // Make Queue using distance values as key
while $Q \neq \emptyset$
do $u \leftarrow$ mindistance$(Q, dist)$
    $S \leftarrow S \in \{u\}$ DeleteMin or ExtractMin
    for all $v \in$ neighbors$[u]$ do
        $dist[v] = \min(dist[v], dist[u] + w(u, v))$
return dist

$dist[v] = \min(dist[v], dist[u] + w(u, v))$

If $dist[v] > dist[u] + w(u, v)$
    $dist[v] = dist[u] + w(u, v)$
    prev [v] = u
    decreaseKey(Q, v)

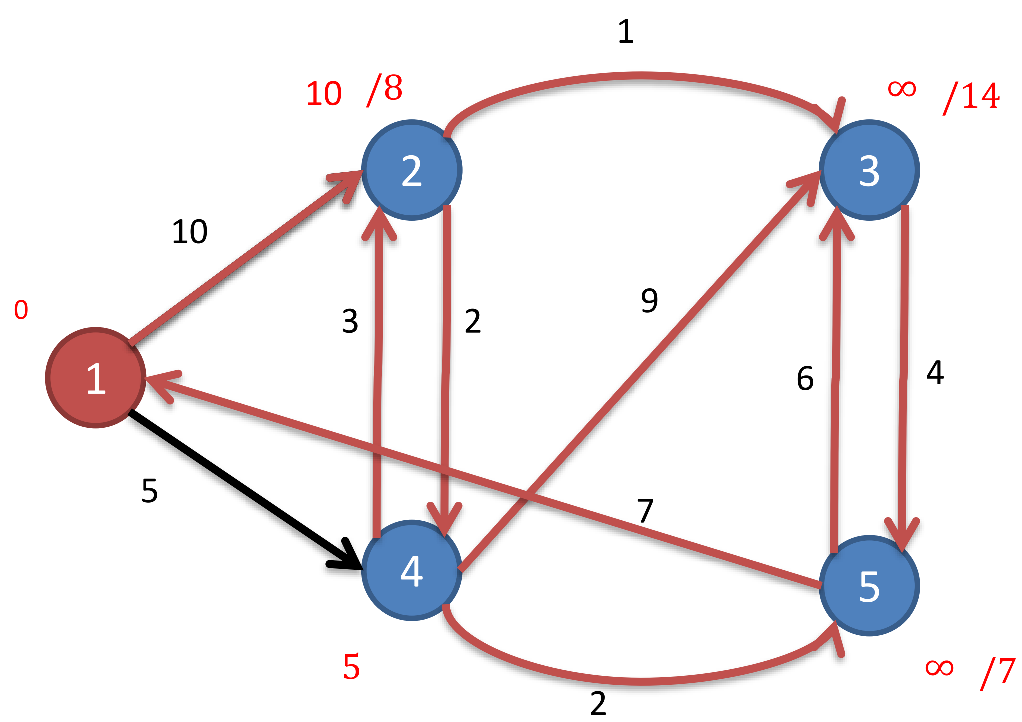
Dijkstra's Algorithm (Example 2)

The shortest path of the vertex with smallest distance is determined



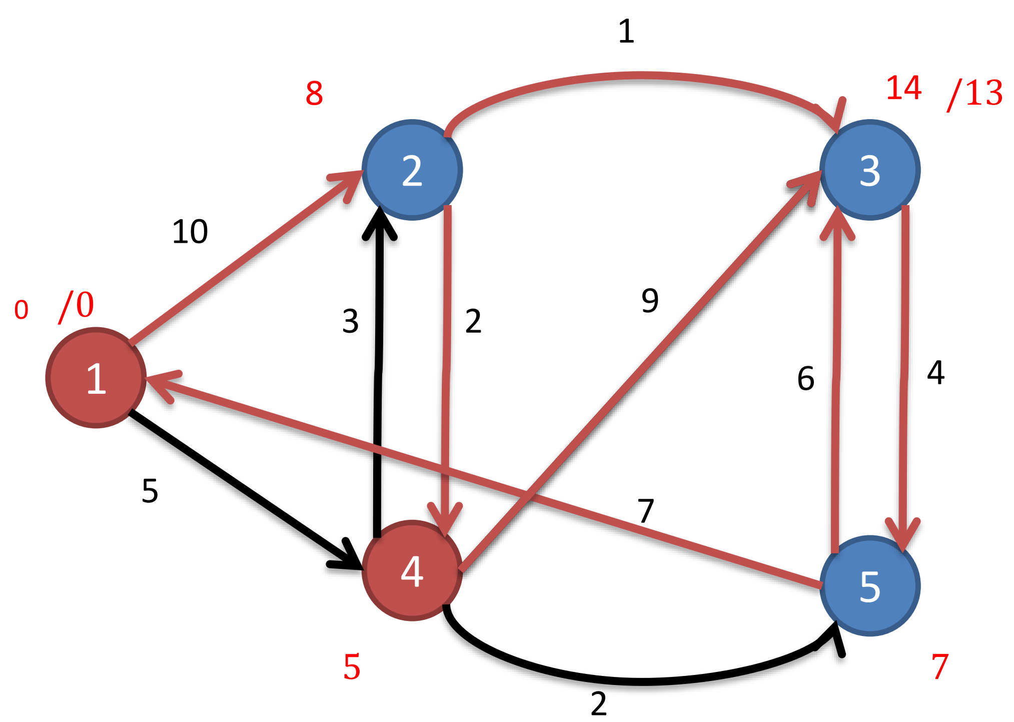
Dijkstra’s Algorithm (Example 2)

Choose a vertex whose shortest path is now determined



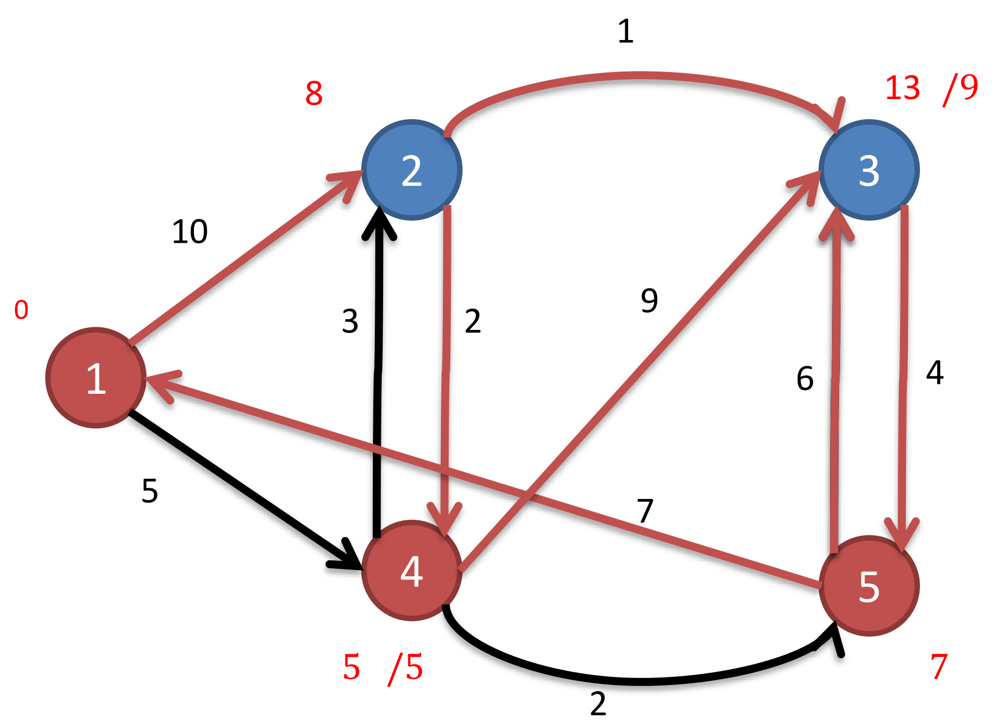
Dijkstra’s Algorithm (Example 2)

Choose a vertex whose shortest path is now determined



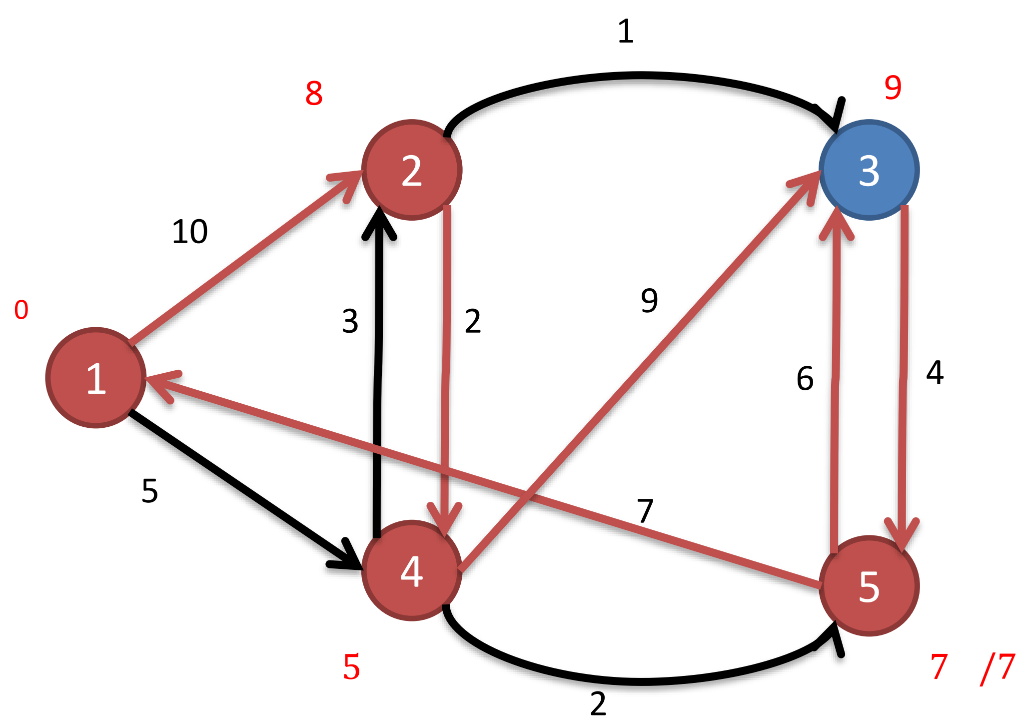
Dijkstra's Algorithm (Example 2)

Choose a vertex whose shortest path is now determined



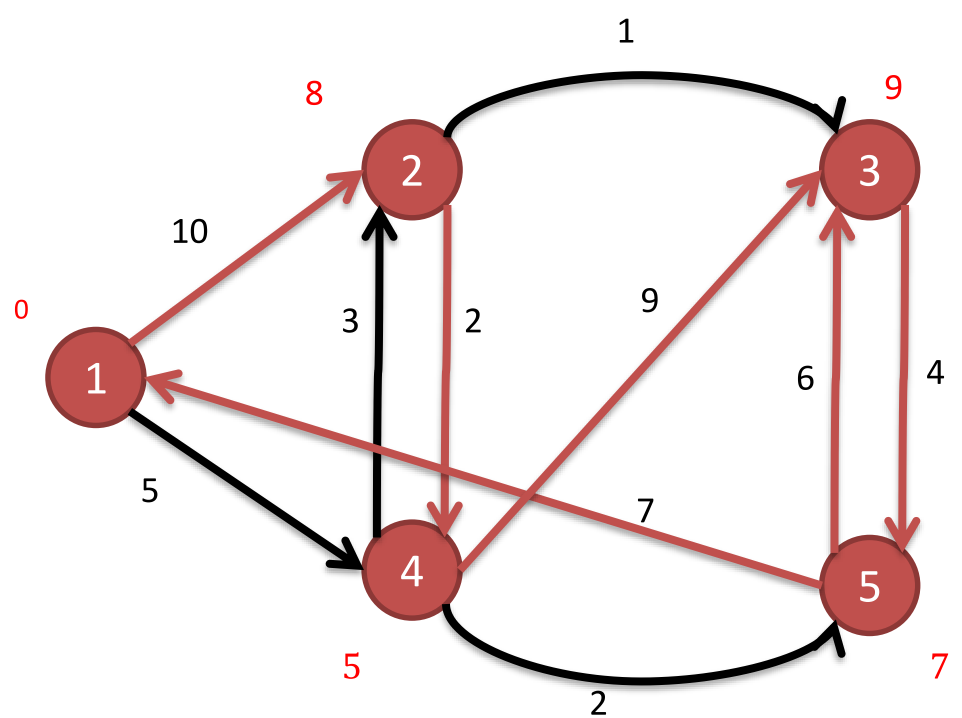
Dijkstra’s Algorithm (Example 2)

Choose a vertex whose shortest path is now determined



Dijkstra’s Algorithm (Example 2)

Final result:



What is the shortest path from 1 to 5?

1, 4, 5

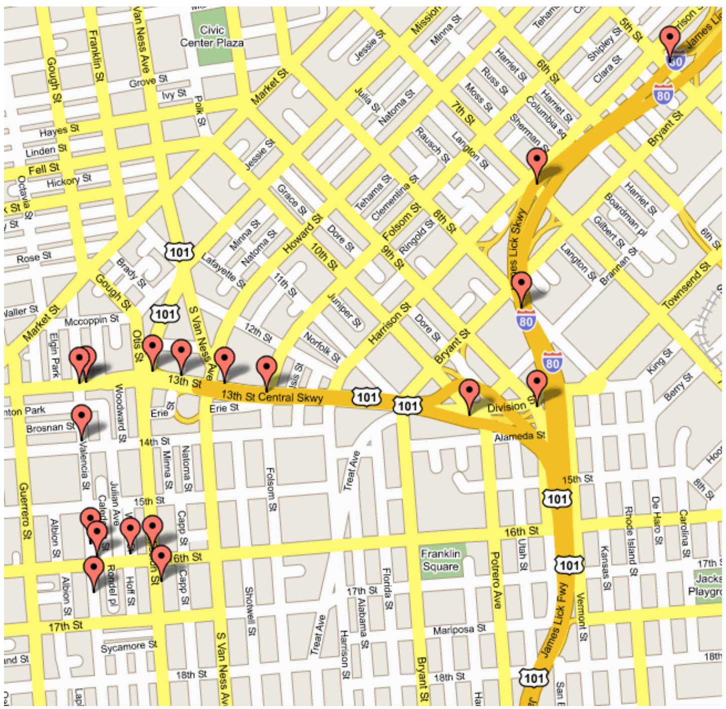
What is weight of this path?

7

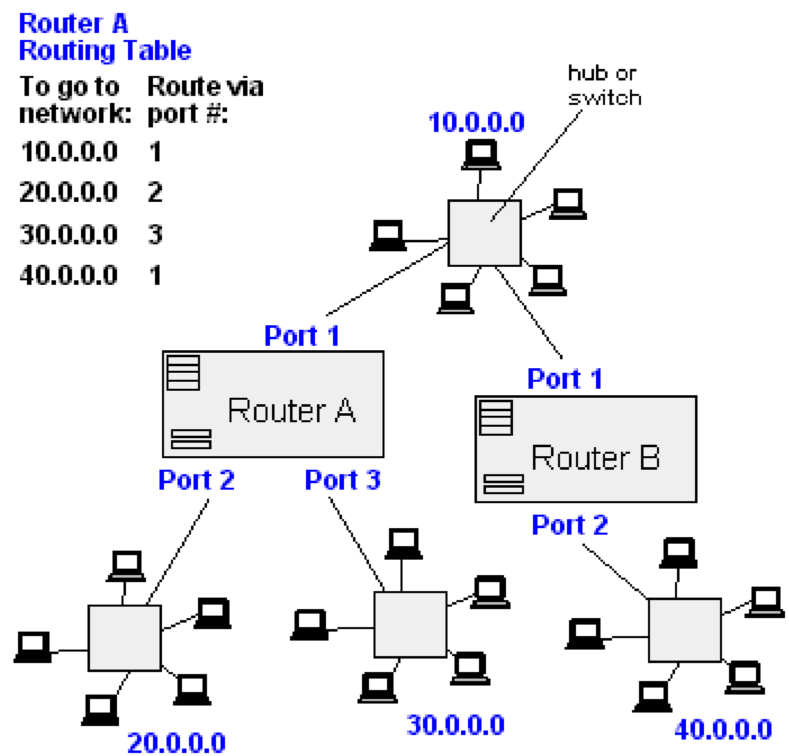
What is the shortest path from 1 to 2, 3, and 4?

Applications

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems



From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.



All-pairs shortest paths

- **All-pairs shortest path problem**: given a weighted, directed graph G=(V, E), for every pair of vertices, find a shortest path.
- If there are negative weights, run **Bellman-Ford** algorithm |V| times
  - T(n)=$|V|O(|V|^3) = O(|V|^4)$
- If there are no negative weights, run **Dijkstra's** algorithm |V| times
  - T(n)=$|V|O(|V|^2) = O(|V|^3)$

All-pairs shortest paths

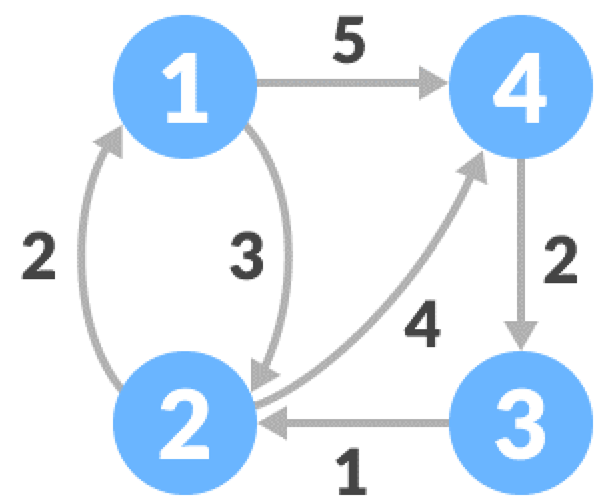
- There are other algorithms can do it more efficient, such like **Floyd-Warshall** algorithm
- **Floyd-Warshall** algorithm
  - Negative weights may present, but no negative cycle
  - T(n)= $O(|V|^3)$
  - A dynamic programming algorithm

Floyd Warshall (Method No. 2)

- Create a matrix A^0 of dimension n*n where n is the number of vertices. The row and the column are indexed as i and j respectively. i and j are the vertices of the graph.

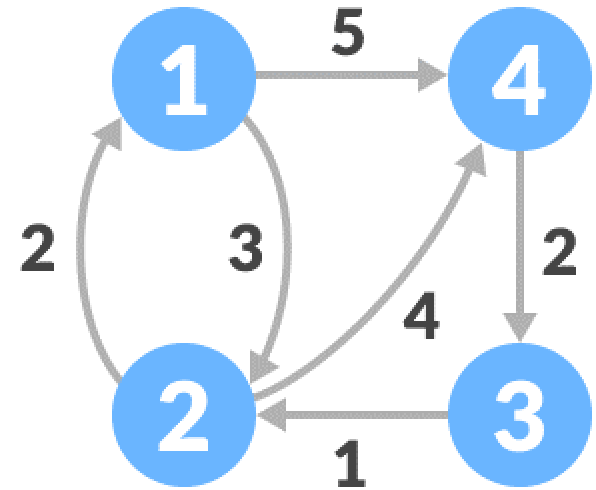
- Each cell A[i][j] is filled with the distance from the ith vertex to the jth vertex. If there is no path from ith vertex to jth vertex, the cell is left as infinity.

$$A^0 = \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$



Floyd Warshall (Method No. 2)

$$A^0 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

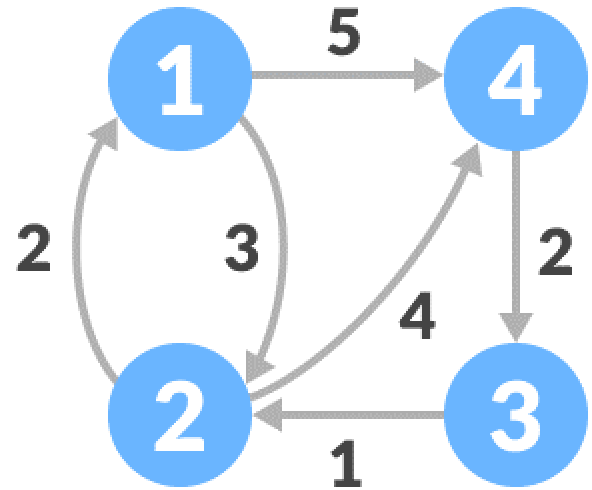


if (A[i][j] > A[i][k] + A[k][j])

$$A^1 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & & \\ \infty & & 0 & \\ \infty & & & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Floyd Warshall (Method No. 2)

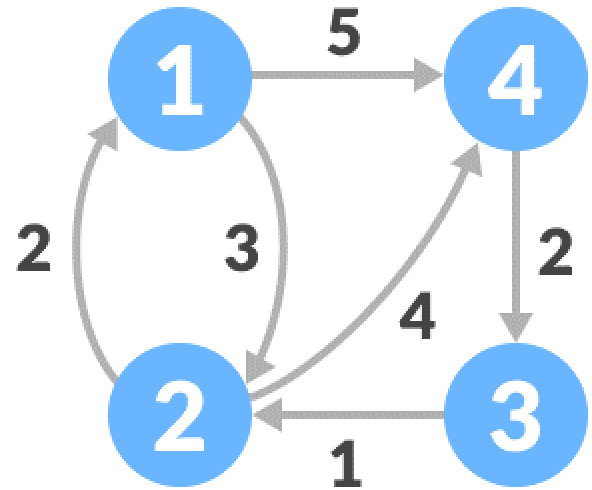
$$A^1 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$



$$A^2 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & & \\ 2 & 0 & \infty & 4 \\ & 1 & 0 & \\ & \infty & & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ 3 & 1 & 0 & 5 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Floyd Warshall (Method No. 2)

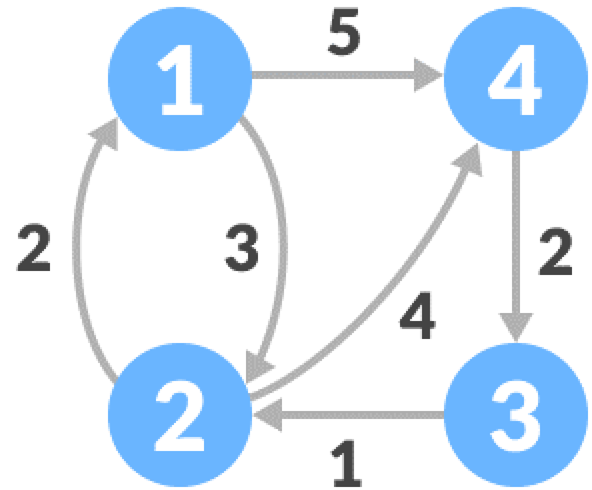
$$A^2 = \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ 3 & 1 & 0 & 5 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$



$$A^3 = \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & & \infty & \\ & 0 & \infty & \\ 3 & 1 & 0 & 5 \\ & & 2 & 0 \end{bmatrix} \end{array} \rightarrow \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Floyd Warshall (Method No. 2)

$$A^3 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

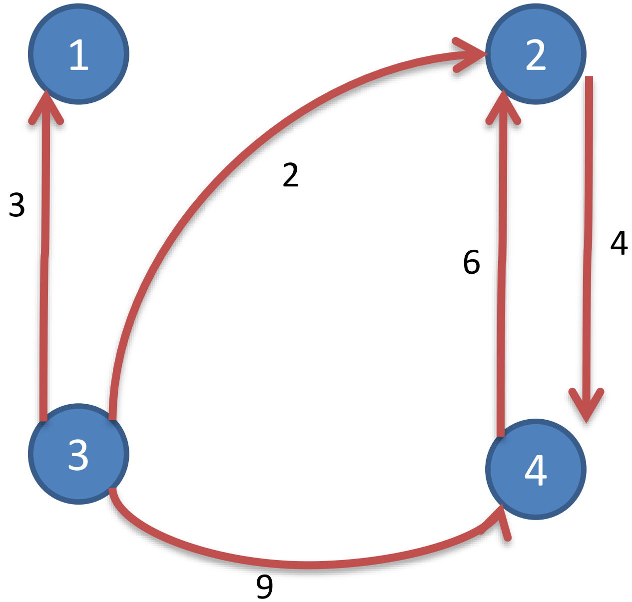


$$A^4 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & & & 5 \\ & 0 & & 4 \\ & & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & 7 & 5 \\ 2 & 0 & 6 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Floyd Warshall (Method No. 2)

```
for(int k = 1; k <= n; k++){
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            dist[i][j] = min( dist[i][j], dist[i][k] + dist[k][j] );
        }
    }
}
```

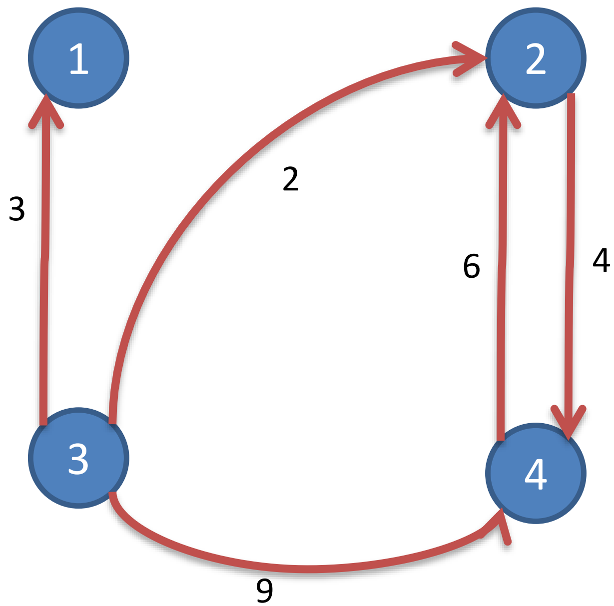
Floyd Warshall (Example)



What are the weights of shortest paths with no intermediate vertices, D(0)?

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	4
3	3	2	0	9
4	∞	6	∞	0

Floyd Warshall (Example)



D(0)

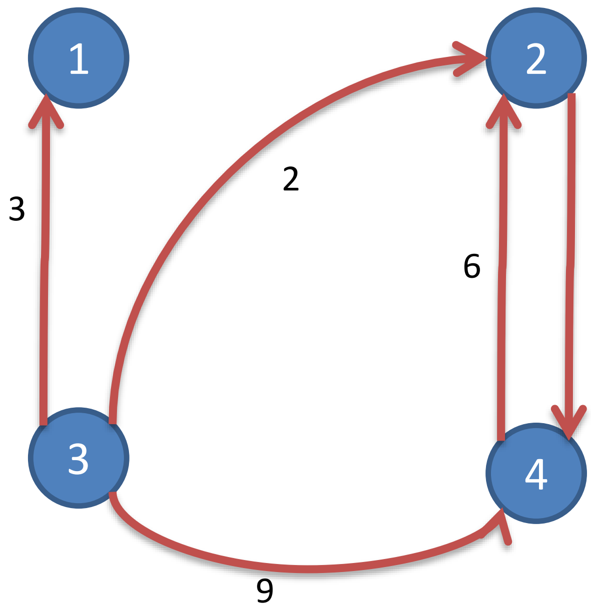
	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	4
3	3	2	0	9
4	∞	6	∞	0

What are the weights of shortest paths with intermediate vertex 1?

D(1)

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	4
3	3	2	0	9
4	∞	6	∞	0

Floyd Warshall (Example)



D(1)

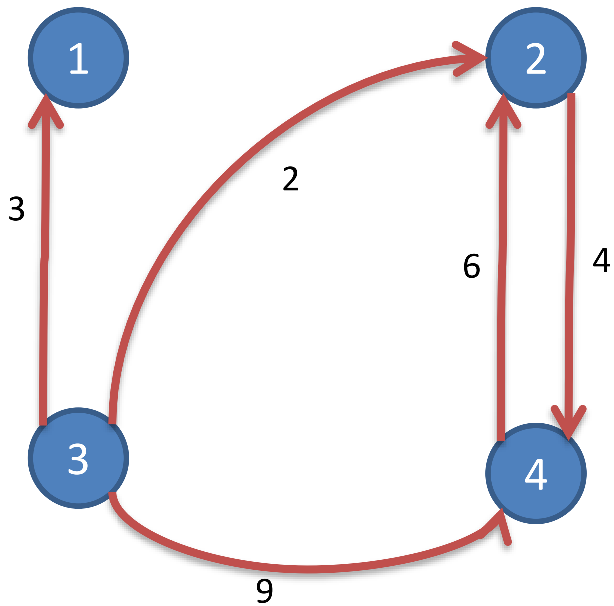
	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	4
3	3	2	0	9
4	∞	6	∞	0

What are the weights of shortest paths with intermediate vertices 1 and 2?

D(2)

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	4
3	3	2	0	6
4	∞	6	∞	0

Floyd Warshall (Example)



D(2)

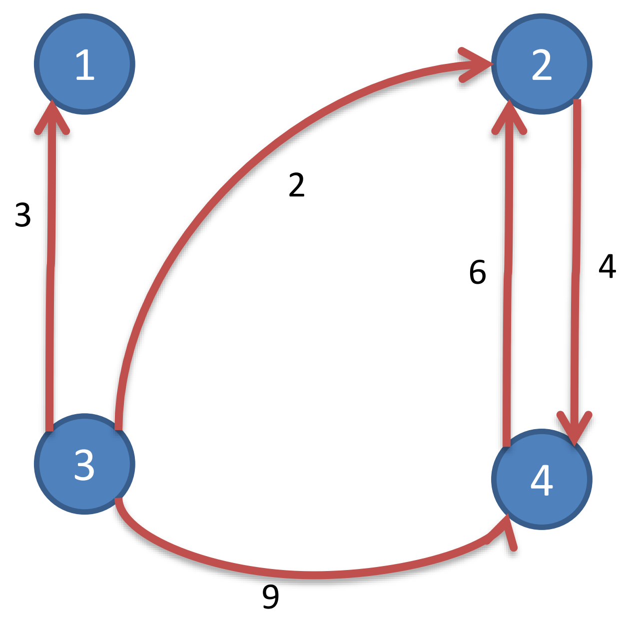
	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	4
3	3	2	0	6
4	∞	6	∞	0

What are the weights of shortest paths with intermediate vertices 1, 2 and 3?

D(3)

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	4
3	3	2	0	6
4	∞	6	∞	0

Floyd Warshall (Example)



D(3)

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	4
3	3	2	0	6
4	∞	6	∞	0

What are the weights of shortest paths with intermediate vertices 1, 2, 3 and 4?

D(4)

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	4
3	3	2	0	6
4	∞	6	∞	0

Floyd Warshall (Example)

Add predecessor information to reconstruct a shortest path

If updated the predecessor i-j in D(k) is the same as the predecessor k-j in D(k-1)

D(0)

	1	2	3	4
1	0/n	∞/n	∞/n	∞/n
2	∞/n	0/n	∞/n	4/2
3	3/3	2/3	0/n	9/3
4	∞/n	6/4	∞/n	0/n

D(1)

	1	2	3	4
1	0/n	∞/n	∞/n	∞/n
2	∞/n	0/n	∞/n	4/2
3	3/3	2/3	0/n	9/3
4	∞/n	6/4	∞/n	0/n

D(2)

	1	2	3	4
1	0/n	∞/n	∞/n	∞/n
2	∞/n	0/n	∞/n	4/2
3	3/3	2/3	0/n	6/2
4	∞/n	6/4	∞/n	0/n

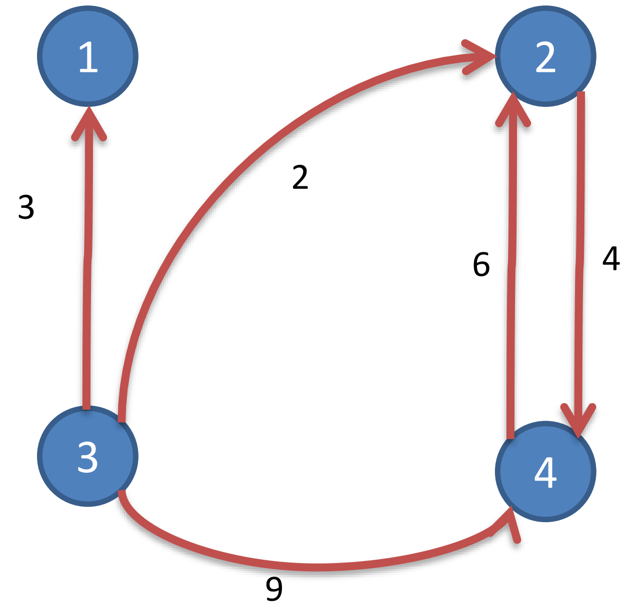
D(3)

	1	2	3	4
1	0/n	∞/n	∞/n	∞/n
2	∞/n	0/n	∞/n	4/2
3	3/3	2/3	0/n	6/2
4	∞/n	6/4	∞/n	0/n

Floyd Warshall (Example)

D(4)

	1	2	3	4
1	0/n	∞/n	∞/n	∞/n
2	∞/n	0/n	∞/n	4/2
3	3/3	2/3	0/n	6/2
4	∞/n	6/4	∞/n	0/n



What is the shortest path from 3 to 4? 3, 2, 4

What is weight of this path? 6

All-pairs shortest paths

- **Floyd-Warshall**(G)

  Construct the shortest path matrix when there is no intermediate vertex, D(0);

  for(i=1 to |G.V|){

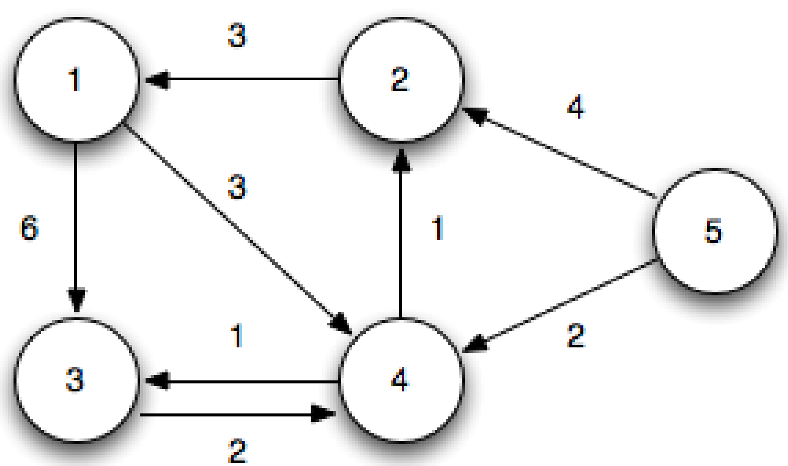
    //D(i) is the shortest path matrix when the intermediate
    //vertices could be: $v_1, v_2, \ldots, v_i$

    Compute D(i) from D(i-1);

  }

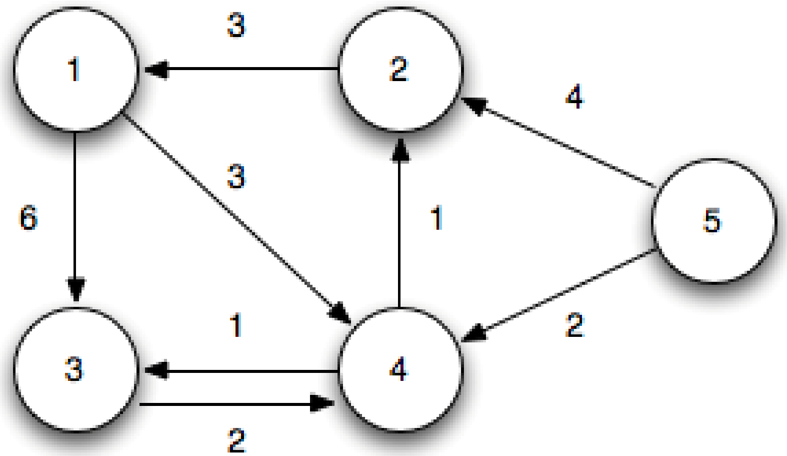
Floyd-Warshall Algorithm

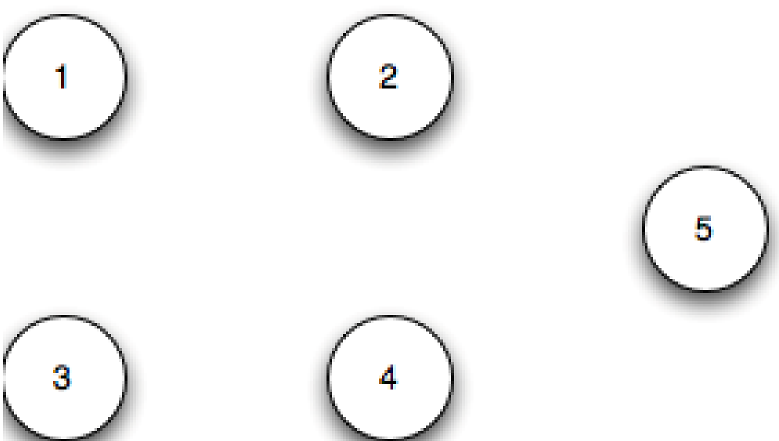
- The Floyd-Warshall algorithm works based on a property of intermediate vertices of a shortest path.



Floyd-Warshall Algorithm

Initialization: (k = 0)





D

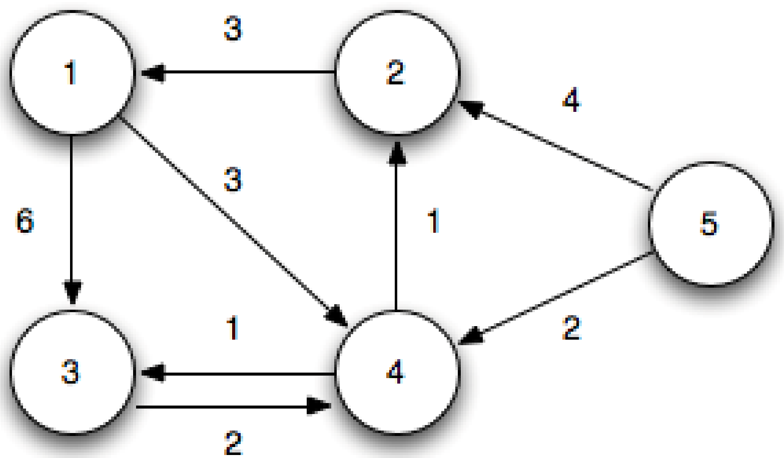
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	∞	∞	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	/	/	/
3	/	/	/	3	/
4	/	4	4	/	/
5	/	5	/	5	/

Floyd-Warshall Algorithm

Iteration 1: (k = 1) Shorter paths from $2 \rightsquigarrow 3$ and $2 \rightsquigarrow 4$ are found through vertex 1

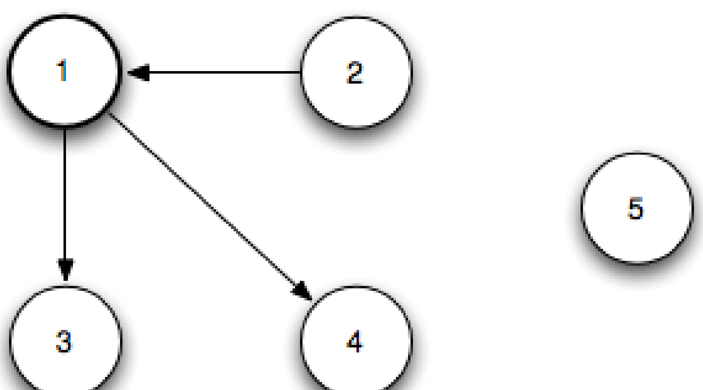


D

	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	∞	∞	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	/	/	/
3	/	/	/	3	/
4	/	4	4	/	/
5	/	5	/	5	/



D

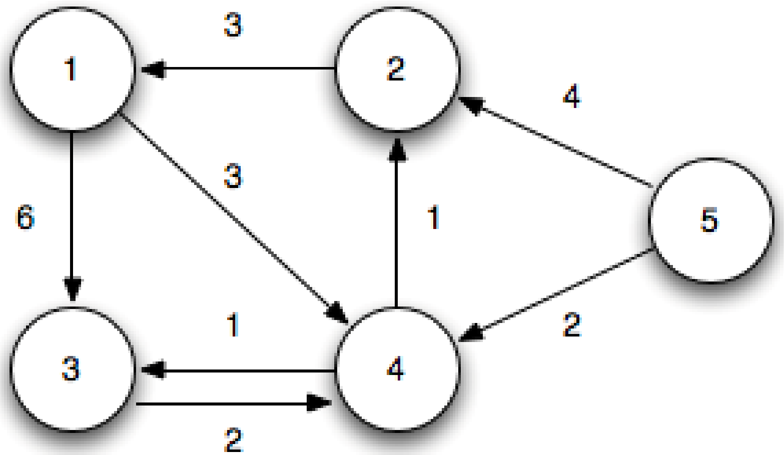
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	**9**	**6**	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	**1**	**1**	/
3	/	/	/	3	/
4	/	4	4	/	/
5	/	5	/	5	/

Floyd-Warshall Algorithm

Iteration 2: (k = 2) Shorter paths from 4 $\rightsquigarrow$ 1, 5 $\rightsquigarrow$ 1, and 5 $\rightsquigarrow$ 3 are found through vertex 2

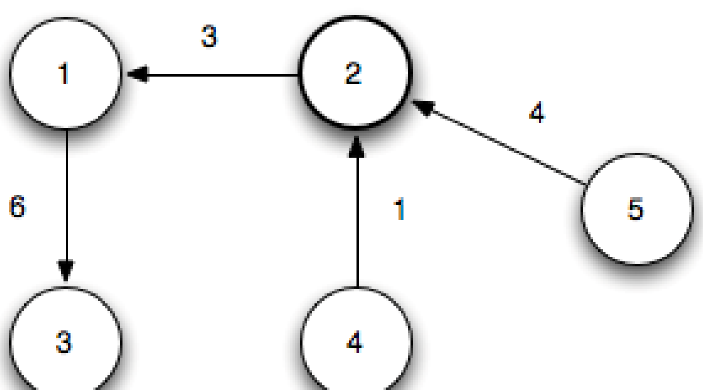


D

	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	**9**	**6**	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	**1**	**1**	/
3	/	/	/	3	/
4	/	4	4	/	/
5	/	5	/	5	/



D

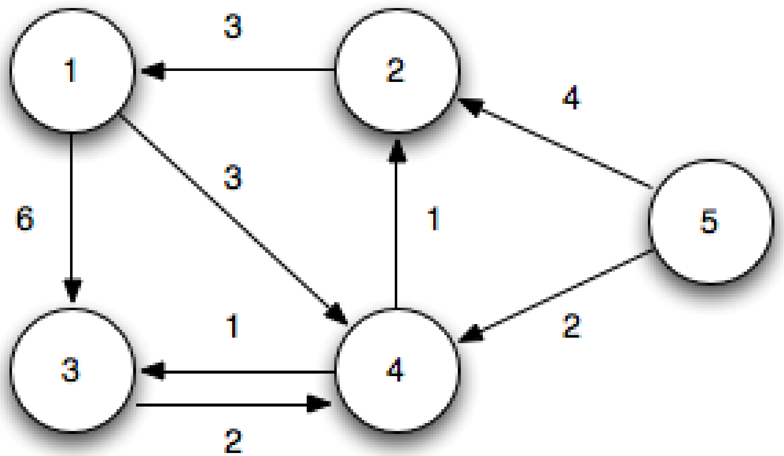
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	**4**	1	1	0	∞
5	**7**	4	**13**	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	1	1	/
3	/	/	/	3	/
4	**2**	4	4	/	/
5	**2**	5	**2**	5	/

Floyd-Warshall Algorithm

- Iteration 3: (k = 3) No shorter paths are found through vertex 3

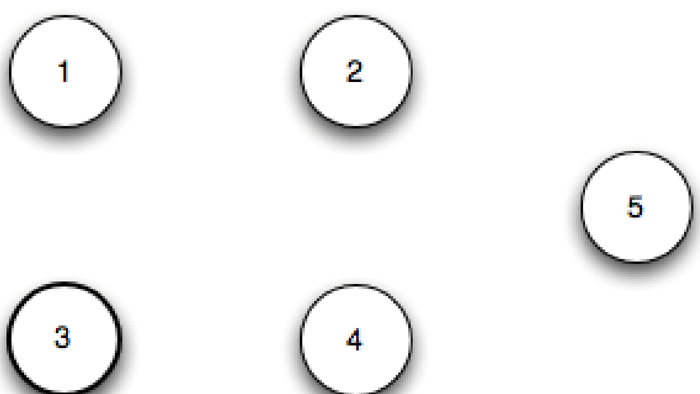


D

	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	**4**	1	1	0	∞
5	**7**	4	**13**	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	1	1	/
3	/	/	/	3	/
4	**2**	4	4	/	/
5	**2**	5	**2**	5	/



D

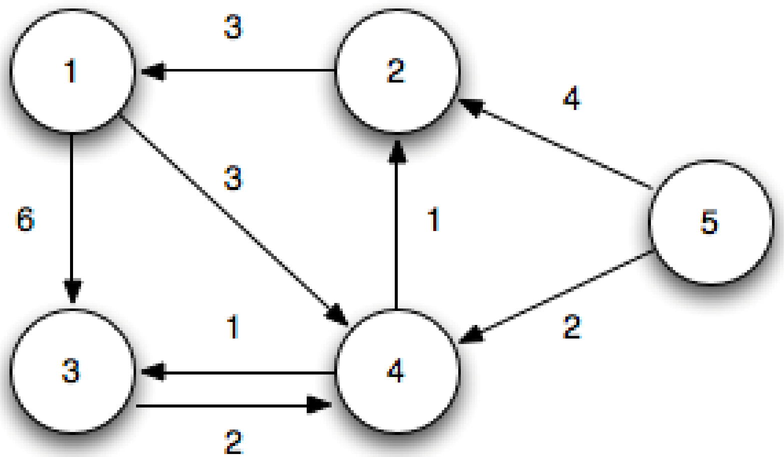
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	4	1	1	0	∞
5	7	4	13	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	1	1	/
3	/	/	/	3	/
4	2	4	4	/	/
5	2	5	2	5	/

Floyd-Warshall Algorithm

Iteration 4: (k = 4) Shorter paths from 1 $\rightsquigarrow$ 2, 1 $\rightsquigarrow$ 3, 2 $\rightsquigarrow$ 3, 3 $\rightsquigarrow$ 1, 3 $\rightsquigarrow$ 2, 5 $\rightsquigarrow$ 1, 5 $\rightsquigarrow$ 2, 5 $\rightsquigarrow$ 3, and 5 $\rightsquigarrow$ 4 are found through vertex 4

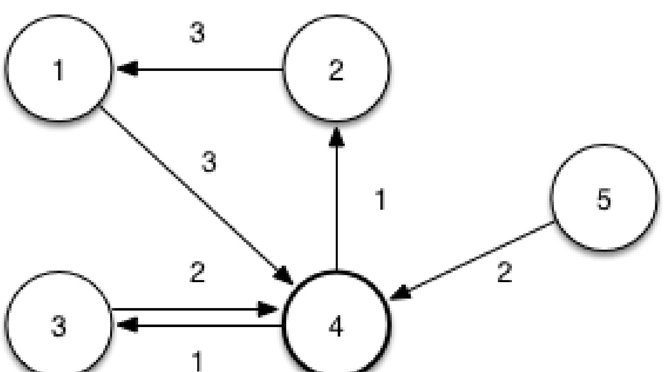


D

	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	4	1	1	0	∞
5	7	4	13	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	1	1	/
3	/	/	/	3	/
4	2	4	4	/	/
5	2	5	2	5	/



D

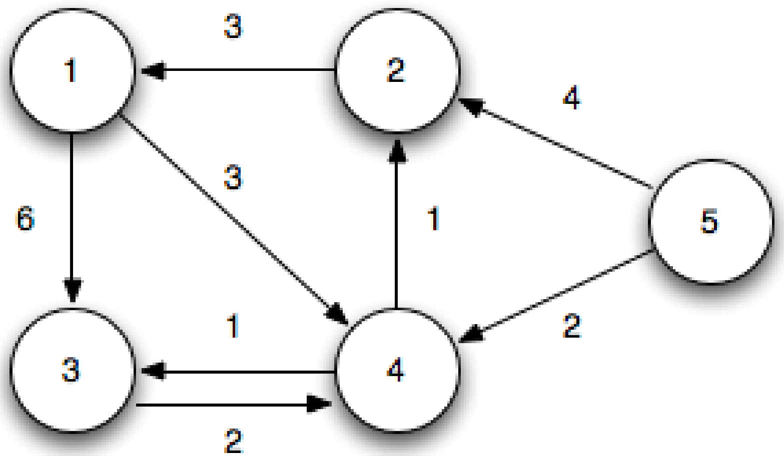
	1	2	3	4	5
1	0	**4**	**4**	3	∞
2	3	0	**7**	6	∞
3	**6**	**3**	0	2	∞
4	4	1	1	0	∞
5	**6**	**3**	**3**	2	0

Π

	1	2	3	4	5
1	/	**4**	**4**	1	/
2	2	/	**4**	1	/
3	**2**	**4**	/	3	/
4	2	4	4	/	/
5	**4**	**4**	**4**	5	/

Floyd-Warshall Algorithm

Iteration 5: (k = 5) No shorter paths are found through vertex 5

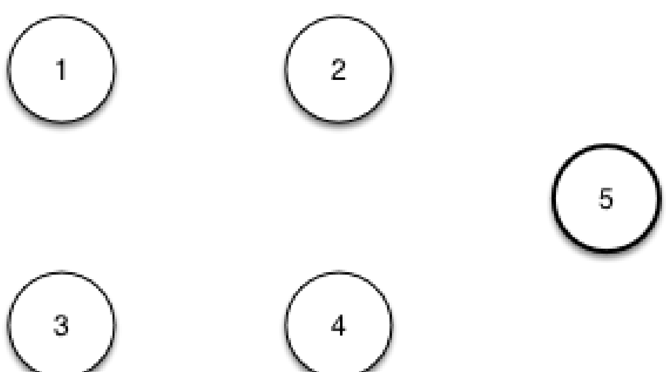


D

	1	2	3	4	5
1	0	**4**	**4**	3	∞
2	3	0	**7**	6	∞
3	**6**	**3**	0	2	∞
4	4	1	1	0	∞
5	**6**	**3**	**3**	2	0

Π

	1	2	3	4	5
1	/	**4**	**4**	1	/
2	2	/	**4**	1	/
3	**2**	**4**	/	3	/
4	2	4	4	/	/
5	4	**4**	**4**	5	/



D

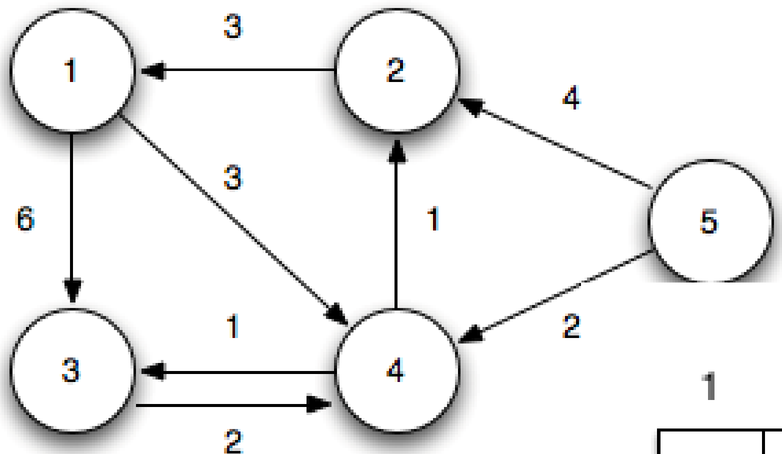
	1	2	3	4	5
1	0	4	4	3	∞
2	3	0	7	6	∞
3	6	3	0	2	∞
4	4	1	1	0	∞
5	6	3	3	2	0

Π

	1	2	3	4	5
1	/	4	4	1	/
2	2	/	4	1	/
3	2	4	/	3	/
4	2	4	4	/	/
5	4	4	4	5	/

Floyd-Warshall Algorithm

The final shortest paths for all pairs is given by



D

	1	2	3	4	5
1	0	4	4	3	∞
2	3	0	7	6	∞
3	6	3	0	2	∞
4	4	1	1	0	∞
5	6	3	3	2	0

Π

	1	2	3	4	5
1	/	4	4	1	/
2	2	/	4	1	/
3	2	4	/	3	/
4	2	4	4	/	/
5	4	4	4	5	/

```
FLOYD-WARSHALL(W)
n = W.rows
D^(0) = W
Π^(0) = π^(0)_ij = NIL if i = j or w_ij = ∞
                 = i   if i ≠ j and w_ij < ∞
for k = 1 to n
    let D^(k) = (d^(k)_ij) be a new n x n matrix
    for i = 1 to n
        for j = 1 to n
            d^k_ij = min(d^(k-1)_ij, d^(k-1)_ik + d^(k-1)_kj)
            if d^(k-1)_ij ≤ d^(k-1)_ik + d^(k-1)_kj
                π^(k)_ij = π^(k-1)_ij
            else
                π^(k)_ij = π^(k-1)_kj
return D^(n)
```

Remarks

- Dijkstra’s algorithm is a <u>single source</u> one
- Bellman’s algorithm consider negative weights but for acyclic graphs
- Floyd’s algorithm solves for the shortest path among all pairs of vertices.