Shortest Path Algorithms

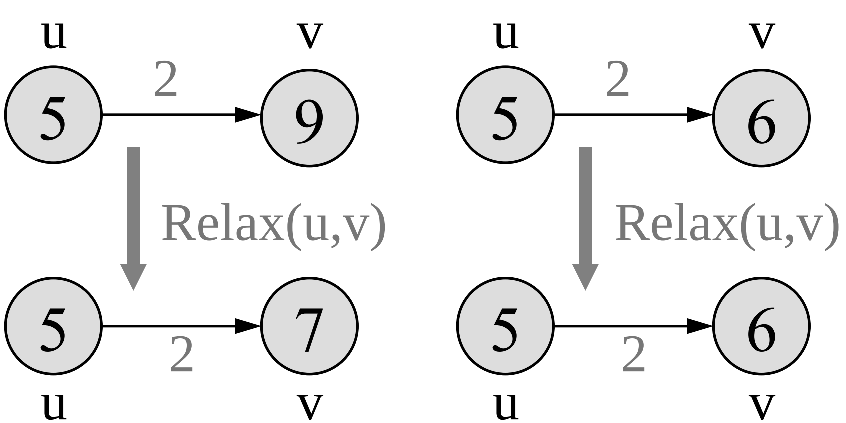
Manmath Narayan Sahoo

Shortest-Path Problems

- Shortest-Path problems
  - **Single-pair.** Given two vertices, find a shortest path between them.
  - **Single-source.** Find a shortest path from a given source (vertex *s*) to each of the vertices. Solution to single-source problem solves single-pair problem efficiently, too.
  - **All-pairs.** Find shortest-paths for every pair of vertices.

Relaxation

- For each vertex *v* in the graph, we maintain D_v, the estimate of the shortest path from ***s*** (initialized to ∞ at the start)
- Relaxing an edge *(u,v)* with cost $\mathbf{d_{uv}}$ means testing whether we can improve the shortest path to *v* found so far by going through *u*




```
Relax (u,v,G)
if Dv > Du+duv then
    Dv = Du+duv
    Parentv = u
```

Dijkstra's Algorithm

- Non-negative edge weights
- Use *Q*, a priority queue keyed by $\mathbf{D_v}$ (BFS used FIFO queue, here we use a PQ, which is re-organized whenever some **D** decreases)
- Basic idea
  - maintain a set ***S*** of solved vertices
  - at each step select "closest" vertex ***u***, add it to ***S***, and relax all edges from ***u***

Dijkstra’s Pseudo Code

- Input: Graph *G*, start vertex *s*

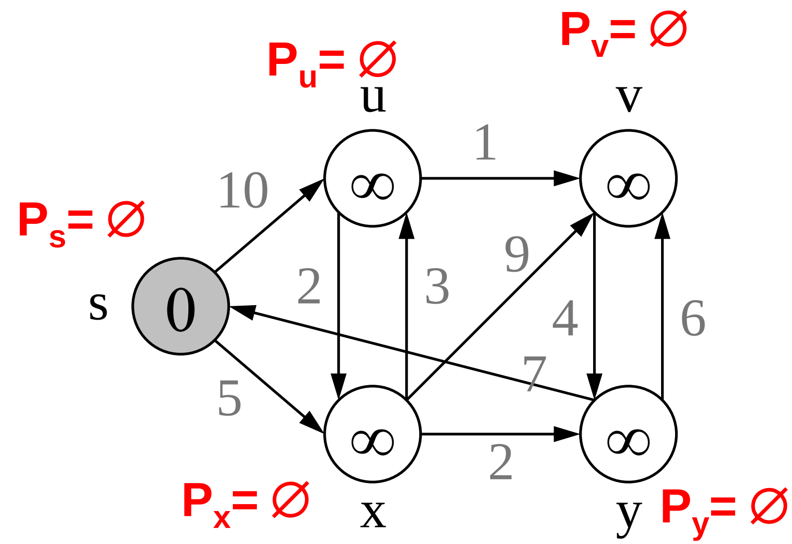
```
Dijkstra(G(V,E),s)
for each vertex u ∈ V
    Du ← ∞
    Parentu ← nil
Ds ← 0
S ←∅             // Set S : already solved vertices
Q.init(V)   // Q, intially contains all nodes in G
while not Q.isEmpty()
    u ← Q.extractMin()   //chose u with minimum Du
    S ←S ∪{u}
    for each v ∈ u.adjacent() do
        Relax(u, v, G)
```

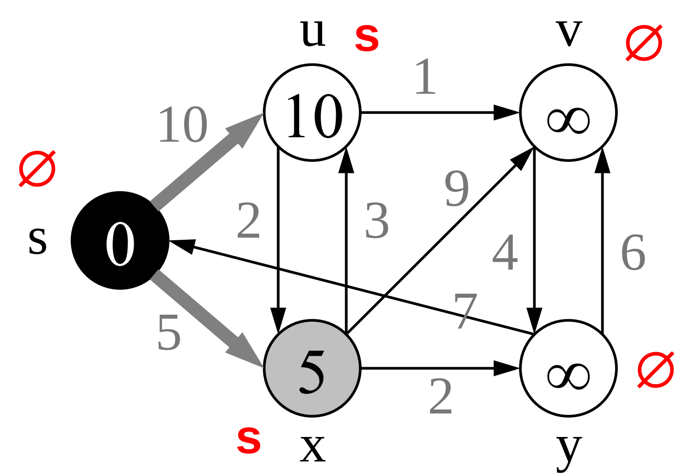
relaxing edges

Dijkstra's Example

```
Dijkstra(G(V,E),s)
for each vertex u ∈ V
     D_u ← ∞
     Parent_u ← nil
D_s ← 0
S ←∅
Q.init(V)
while not Q.isEmpty()
    u ← Q.extractMin() //grey
    S ←S ∪{u}           //black
    for each v ∈ u.adjacent() do
       Relax(u, v, G)
```

```
Relax (u,v,G)
if D_v > D_u+d_uv then
   D_v = D_u+d_uv
   Parent_v = u
```

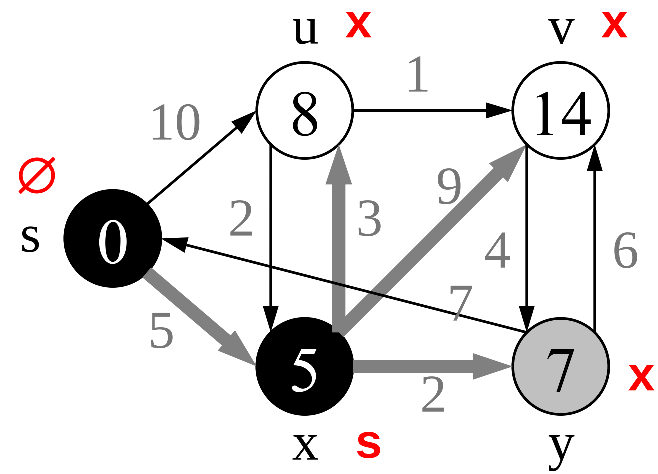


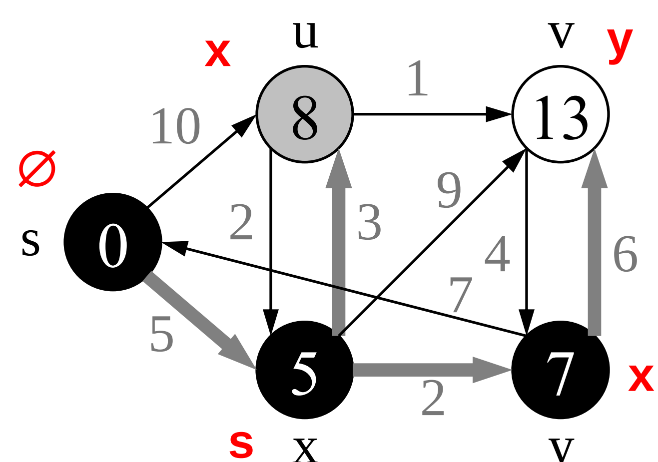


Dijkstra’s Example (2)

```
Dijkstra(G(V,E),s)
for each vertex u ∈ V
      D_u ← ∞
      Parent_u ← nil
D_s ← 0
S ←∅
Q.init(V)
while not Q.isEmpty()
     u ← Q.extractMin() //grey
     S ←S ∪{u}              //black
     for each v ∈ u.adjacent() do
        Relax(u, v, G)
```

```
Relax (u,v,G)
if D_v > D_u+d_uv then
    D_v = D_u+d_uv
    Parent_v = u
```

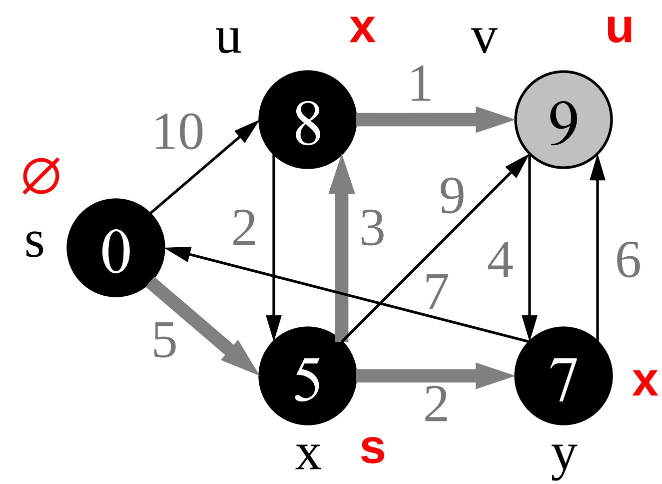


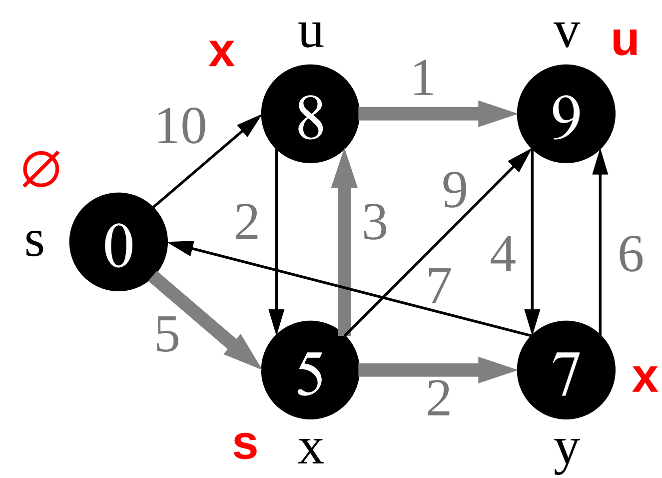


Dijkstra's Example (3)

```
Dijkstra(G(V,E),s)
for each vertex u ∈ V
     Du ← ∞
     Parentu ← nil
Ds ← 0
S ←∅
Q.init(V)
while not Q.isEmpty()
    u ← Q.extractMin() //grey
    S ←S ∪{u}            //black
    for each v ∈ u.adjacent() do
       Relax(u, v, G)
```

```
Relax (u,v,G)
if Dv > Du+duv then
   Dv = Du+duv
   Parentv = u
```



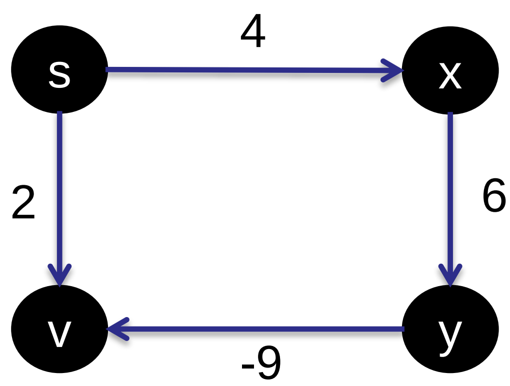


Dijkstra's Algorithm

- Dijkstra's Complexity:
  - O(|V| x |E|)

Bellman-Ford Algorithm

- Dijkstra’s doesn’t work when there are negative edges:

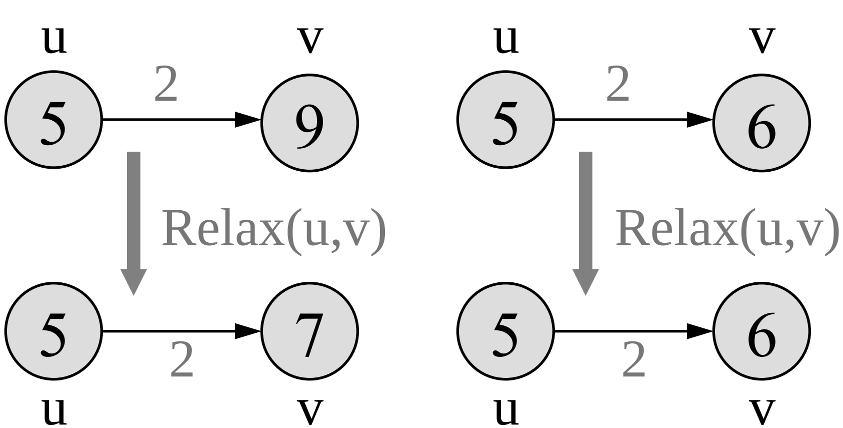


- Shortest path from s to v is s→v (cost=2), but actually the shortest path is s→x→y→v (cost=1)

- Bellman-Ford algorithm detects negative cycles (returns *false*) or returns the shortest path

Relaxation

- For each vertex *v* in the graph, we maintain D_v, the estimate of the shortest path from ***s*** (initialized to ∞ at the start)
- Relaxing an edge *(u,v)* with cost $\mathbf{d_{uv}}$ means testing whether we can improve the shortest path to *v* found so far by going through *u*



Relax (u,v,G)

if $D_v > D_u + d_{uv}$ **then**

  $D_v = D_u + d_{uv}$

  $\text{Parent}_v = u$

Bellman-Ford Algorithm

```
Relax (u,v,G)
if D_v > D_u+d_uv then
    D_v = D_u+d_uv
    Parent_v = u
```

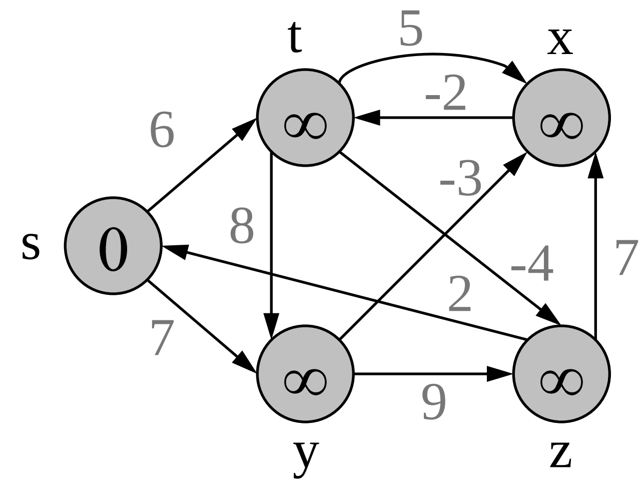
```
Bellman-Ford(G,s)
for each vertex u ∈ V
    D_u ← ∞
    Parent_u ← NIL
D_s ← 0
for i ← 1 to |V|-1 do
    for each edge (u,v) ∈ E do
        Relax (u,v,G)
for each edge (u,v) ∈ E do
    if D_v > D_u+d_uv then
        return false
return true
```

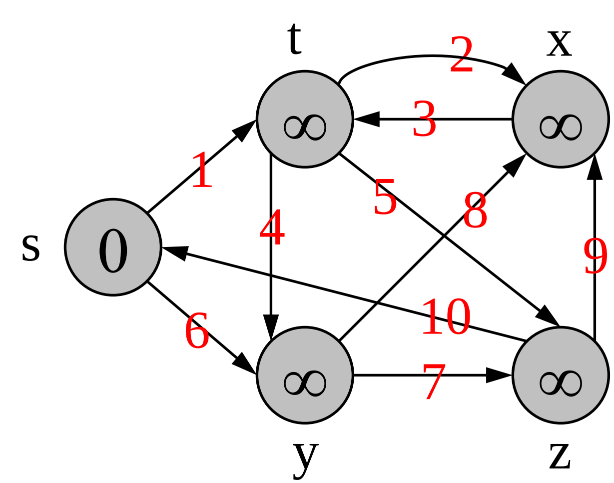
Cycle detection

Bellman-Ford Example(1)

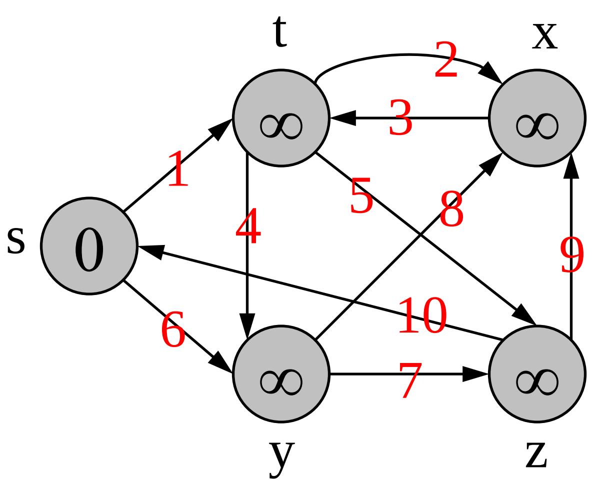
```
Bellman-Ford(G,s)
for each vertex u ∈ V
    D_u ← ∞
    Parent_u ← NIL
D_s ← 0
for i ← 1 to |V|-1 do
    for each edge (u,v) ∈ E do
        Relax (u,v,G)
for each edge (u,v) ∈ E do
    if D_v > D_u+d_uv then
        return false
return true
```

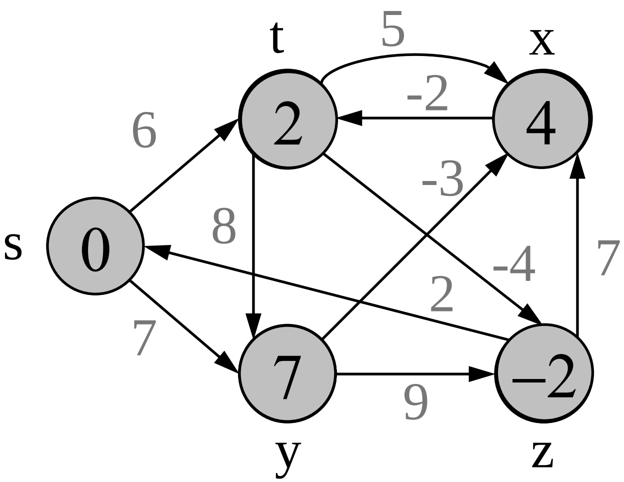
```
Relax (u,v,G)
if D_v > D_u+d_uv then
    D_v = D_u+d_uv
    Parent_v = u
```





red: edge orders





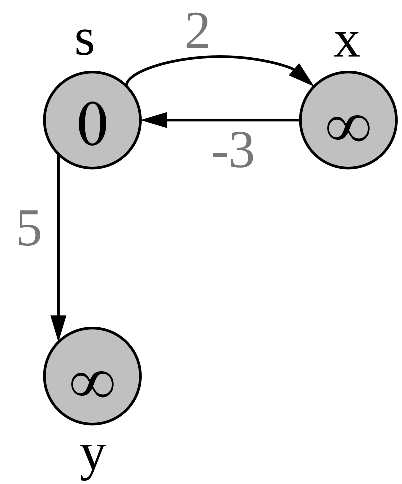
Iteration 1

Iteration 2

Iteration 3 (no relax)

Iteration 4 (no relax)

Negative Length cycle example



Bellman-Ford Algorithm

- Bellman-Ford Complexity:
  - $(|V|-1)|E| + |E| = O(|V| \times |E|)$

Floyd-Warshall Algorithm

- Find all pair shortest paths

- Can be solved by executing Bellman Ford algorithm |V| times with different sources
  - Complexity: $O(|V|^2 \times |E|)$
  - For dense network/graph $O(|V|^4)$

- Floyd-Warshall algorithm finds in $O(|V|^3)$ complexity

Floyd-Warshall Algorithm - Idea

$d_{s,t}^{(i)}$ – the shortest path from s to t containing only vertices $v_1, ..., v_i$

$$d_{s,t}^{(0)} = w(s,t)$$

$$d_{s,t}^{(k)} = \begin{cases} w(s,t) & \text{if } k = 0 \\ \min\{d_{s,t}^{(k-1)}, d_{s,k}^{(k-1)} + d_{k,t}^{(k-1)}\} & \text{if } k > 0 \end{cases}$$

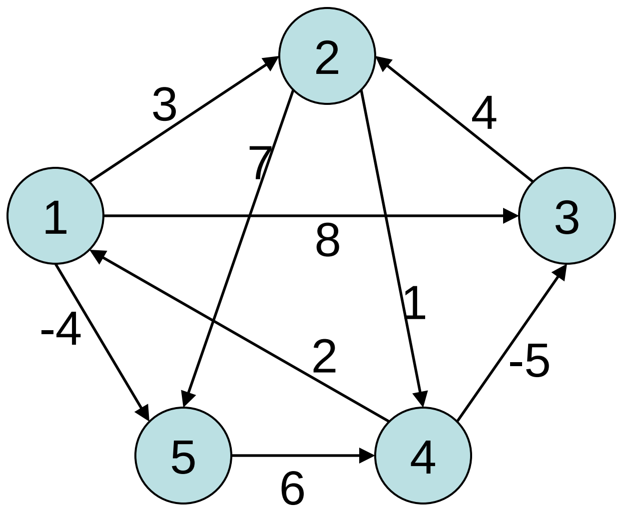
$$D^{(k)} = [d_{s,t}^{(k)}]$$

Floyd-Warshall Algorithm

FloydWarshall(**matrix** W, **integer** *n*)

for $k \leftarrow 1$ **to n do**
  for $i \leftarrow 1$ **to n do**
    for $j \leftarrow 1$ **to n do**
      $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
    end
  end
  // $D^{(k)}$ is formed
end
return $D^{(n)}$

Floyd-Warshall Algorithm – Example(1)



W

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

Floyd-Warshall Algorithm – Example(2)

$D^{(0)}=$

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

$\Pi^{(0)}=$

0	0	0		0
	0		0	0
	0	0		
0		0	0	
			0	0

$D^{(1)}=$

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	5	-5	0	-2
∞	∞	∞	6	0

$\Pi^{(1)}=$

0	0	0		0
	0		0	0
	0	0		
0	1	0	0	1
			0	0

Floyd-Warshall Algorithm – Example(3)

$D^{(1)}=$

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	5	-5	0	-2
∞	∞	∞	6	0

$\Pi^{(1)}=$

0	0	0		0
	0		0	0
	0	0		
0	1	0	0	1
			0	0

$D^{(2)}=$

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	5	-5	0	-2
∞	∞	∞	6	0

$\Pi^{(2)}=$

0	0	0	2	0
	0		0	0
	0	0	2	2
0	1	0	0	1
			0	0

Floyd-Warshall Algorithm – Example(4)

$D^{(2)}=$

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	5	-5	0	-2
∞	∞	∞	6	0

$\Pi^{(2)}=$

0	0	0	2	0
	0		0	0
	0	0	2	2
0	1	0	0	1
			0	0

$D^{(3)}=$

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	-1	-5	0	-2
∞	∞	∞	6	0

$\Pi^{(3)}=$

0	0	0	2	0
	0		0	0
	0	0	2	2
0	3	0	0	1
			0	0

Floyd-Warshall Algorithm – Example(5)

$D^{(3)}=$

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	-1	-5	0	-2
∞	∞	∞	6	0

$\Pi^{(3)}=$

0	0	0	2	0
	0		0	0
	0	0	2	2
0	3	0	0	1
			0	0

$D^{(4)}=$

0	3	-1	4	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

$\Pi^{(4)}=$

0	0	4	2	0
4	0	4	0	1
4	0	0	2	1
0	3	0	0	1
4	3	4	0	0

Floyd-Warshall Algorithm – Example(6)

$D^{(4)}=$

0	3	-1	4	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

$\Pi^{(4)}=$

0	0	4	2	0
4	0	4	0	1
4	0	0	2	1
0	3	0	0	1
4	3	4	0	0

$D^{(5)}=$

0	3	-1	2	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

$\Pi^{(5)}=$

0	0	4	5	0
4	0	4	0	1
4	0	0	2	1
0	3	0	0	1
4	3	4	0	0