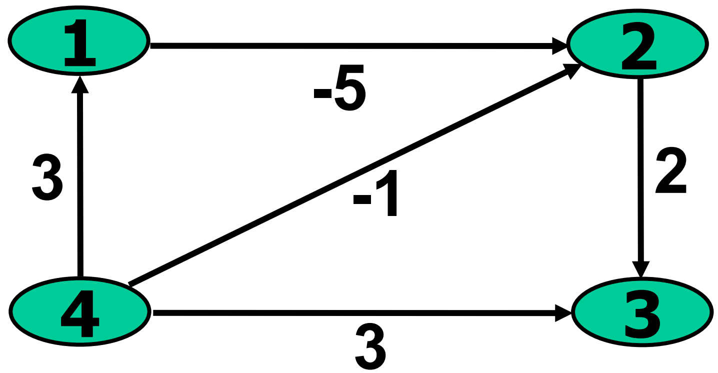
All-Pairs Shortest Paths

- Given a weighted digraph G=(V,E), compute the length of a shortest path between every pair of vertices in G
- Weights could indicate profit/loss of travelling along a certain route



G

Shortest path distance matrix for G:

$$D^* = \begin{pmatrix} 0 & -5 & -3 & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 0 & 0 \end{pmatrix}$$

Single-source shortest paths

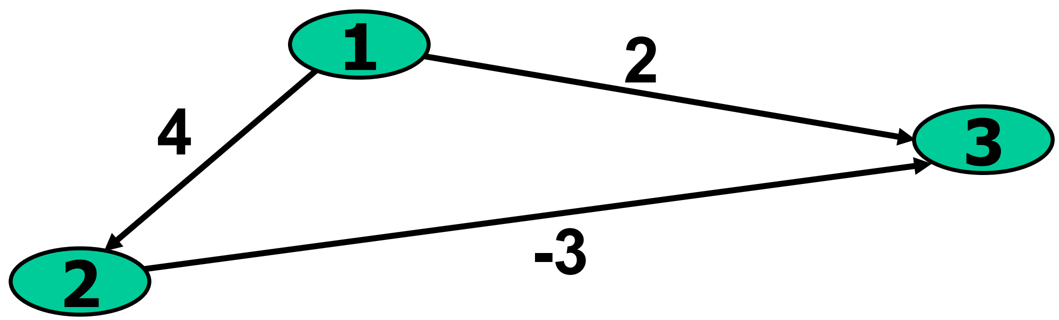
- Recall Dijkstra's algorithm:
  - given a single source vertex s, it computes a shortest path from s to every other vertex in the graph
  - it maintains a set S of vertices for which shortest path from s is known
  - for each vertex v, it maintains a value d(v) representing the length of a shortest path from s to v passing only through vertices of S
  - assume that wt(v,w) represents the weight of the edge (v,w)
  - assume that adj(v) represents the adjacency list of vertex v

Dijkstra's algorithm: pseudocode

```
public void dijkstra(Vertex s)
{   S = {s};
    for (v : V)
        if (v==s)
            d(v)=0;
        else if (s,v)∈E
            d(v) = wt(s,v);
        else
            d(v) = ∞;
    while (S != V)
    {   find v not in S with d(v) minimum;
        S = S ∪ {v};
        for (w : adj(v)\S)
            if (d(v) + wt(v,w) < d(w))
                d(w) = d(v) + wt(v,w);
    }
}
```

Complexity, and negative edge weights

- Dijkstra's algorithm runs in $O(n^2)$ time, where $n=|V|$
  - or $O(n\log n + m)$ using a Fibonacci Heap, where $m=|E|$
- We could use it for each source vertex in the graph to obtain all-pairs shortest paths - $O(n^3)$ time (or $O(n^2\log n + nm)$ time)
- *But*, Dijkstra's algorithm doesn't work for negative edge weights!



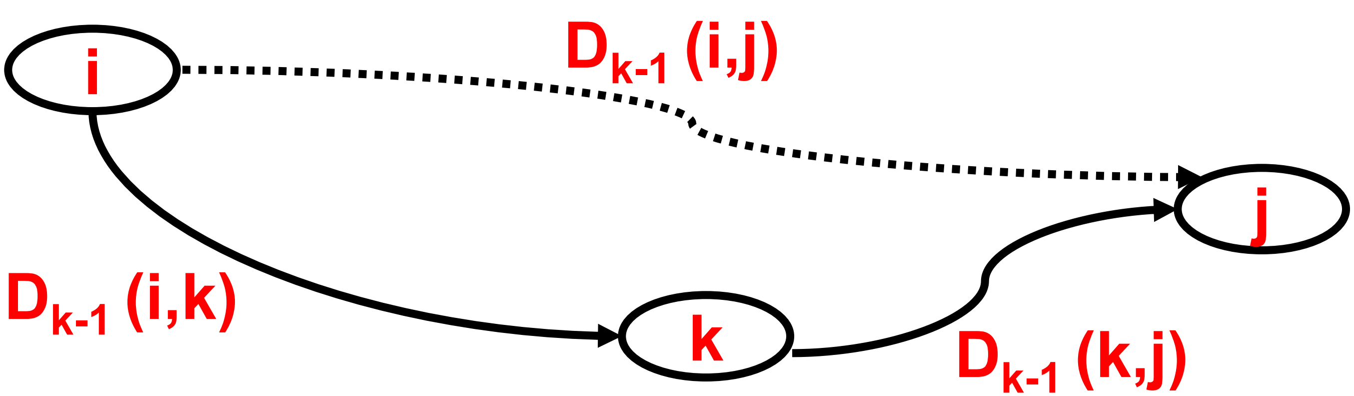
- E.g. take source vertex to be 1

Single-source shortest paths

- **Alternative: Bellman-Ford algorithm**
  - given a single source vertex s, compute a shortest path from s to every other vertex in the graph
  - algorithm works even if there are negative edge weights
  - runs in $O(nm)$ time, where $m=|E|$
  - could use it for each source vertex in the graph to obtain all-pairs shortest paths - $O(n^2m)$ time
- **If G is dense (i.e. $|E|\sim n^2$), Bellman-Ford algorithm is no better than $O(n^4)$**
- **We will see an $O(n^3)$ algorithm**
  - the Floyd-Warshall algorithm
  - based on dynamic programming
  - may give unexpected results if negative weight cycles exist

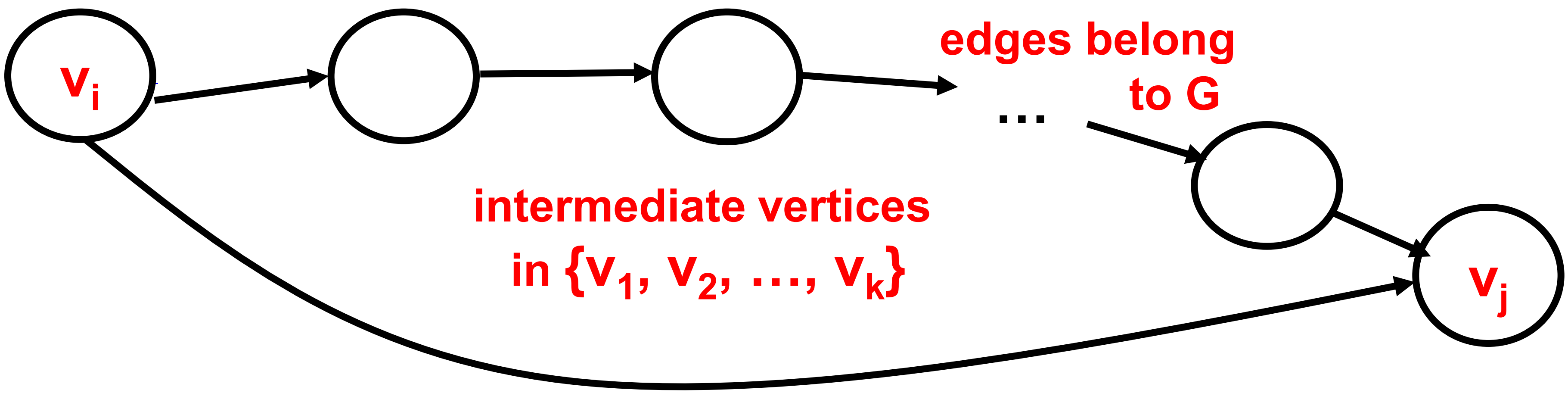
Constructing D*

- Begin by numbering the vertices of G as $v_1, v_2, \ldots, v_n$
- Construct a sequence of matrices $D_0 , D_1 , \ldots , D_n$
  - $D_0 (i,j) = \begin{cases} 0, \text{ if } i=j \\ wt(v_i, v_j), \text{ if } (v_i, v_j) \in E \\ \infty, \text{ otherwise} \end{cases}$
  - for $k \geq 1$, $D_k (i,j) = \min \{ D_{k-1} (i,j), D_{k-1} (i,k) + D_{k-1} (k,j) \}$
- Constructing $D_k (i,j)$:



The key property

For $k \geq 0$, $D_k(i,j)$ contains the shortest path distance from v_i to v_j whose intermediate vertices (if any) belong to $\{v_1, v_2, \ldots, v_k\}$.



- So $D_n = D^*$!
- Proof that the key property above holds is a tutorial exercise
- Floyd-Warshall algorithm computes D_n based on the rules from the previous slide
- Assumes adjacency matrix representation of G

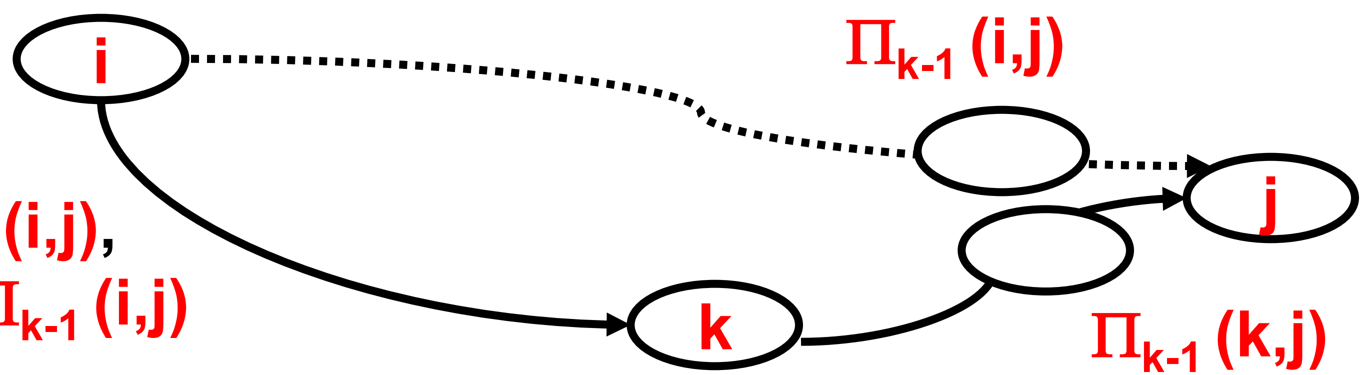
Floyd - Warshall algorithm

```
public void floydWarshall()
{ for ( int i = 1; i <= n; i++ )
    for ( int j = 1; j <= n; j++ )
      if ( i==j )
        D_0[i][j] = 0;
      else if ( (v_i,v_j)∈ E )
        D_0[i][j] = wt((v_i,v_j));
      else
        D_0[i][j] = Integer.MAX_VALUE;
  for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
      for (int j = 1; j<= n; j++)
        D_k[i][j] = Math.min(D_k-1[i][j], D_k-1[i][k] + D_k-1[k][j]);
}
```

- 3 nested loops - $O(n^3)$ time complexity
- nontrivial fact (not proved) – can drop the subscripts on the D_k matrices so that the algorithm uses $O(n^2)$ space

Floyd - Warshall algorithm – computing actual shortest paths

- $\Pi_k(i,j)$ is the predecessor of v_j on a shortest path from v_i to v_j whose intermediate vertices belong to $v_1, v_2, \ldots, v_k$
  - $\Pi_0(i,j) = \begin{cases} \text{null, if } i=j \\ i, \text{ if } (v_i, v_j) \in E \\ \text{null, otherwise} \end{cases}$
- Constructing $\Pi_k(i,j)$ for $k \geq 1$:



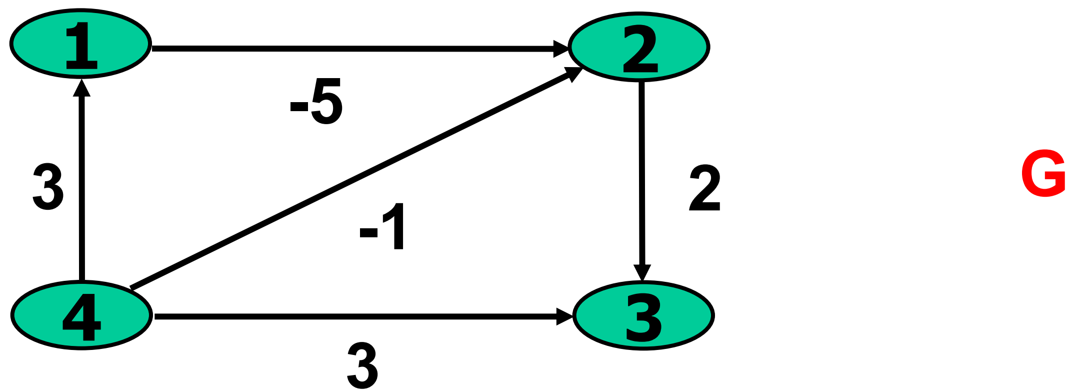
  - if $D_k(i,j) = D_{k-1}(i,j)$,
    $\Pi_k(i,j) = \Pi_{k-1}(i,j)$
  - else $\Pi_k(i,j) = \Pi_{k-1}(k,j)$

- $\Pi^*(i,j)$ is the predecessor of v_j on a shortest path from v_i to v_j
- $\Pi^* = \Pi_n$

Floyd - Warshall algorithm – computing actual shortest paths

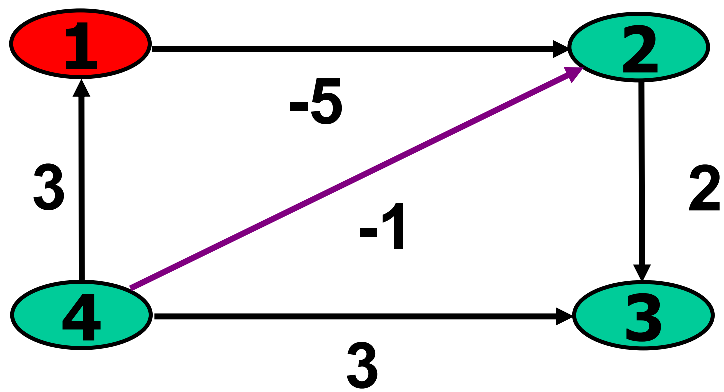
```
public void floydWarshall()
{ for ( int i = 1; i <= n; i++ )
    for ( int j = 1; j <= n; j++ )
      if ( i==j )
      { D0[i][j] = 0; Π0[i][j] = null;}
      else if ( (vi,vj)∈ E )
        D0[i][j] = wt((vi,vj)); Π0[i][j] = i;}
      else
        D0[i][j] = Integer.MAX_VALUE; Π0[i][j] = null;}
  for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
      for (int j = 1; j<= n; j++)
      { Dk[i][j] = Math.min(Dk-1[i][j],Dk-1[i][k]+Dk-1[k][j]);
        if (Dk[i][j] == Dk-1[i][j])
          Πk[i][j] = Πk-1[i][j];
        else
          Πk[i][j] = Πk-1[k][j];
      }
}
```

Example execution of the algorithm



$$D_0=\begin{pmatrix} 0 & -5 & \infty & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -1 & 3 & 0 \end{pmatrix} \qquad \Pi_0=\begin{pmatrix} - & 1 & - & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 4 & 4 & - \end{pmatrix}$$

Example execution of the algorithm

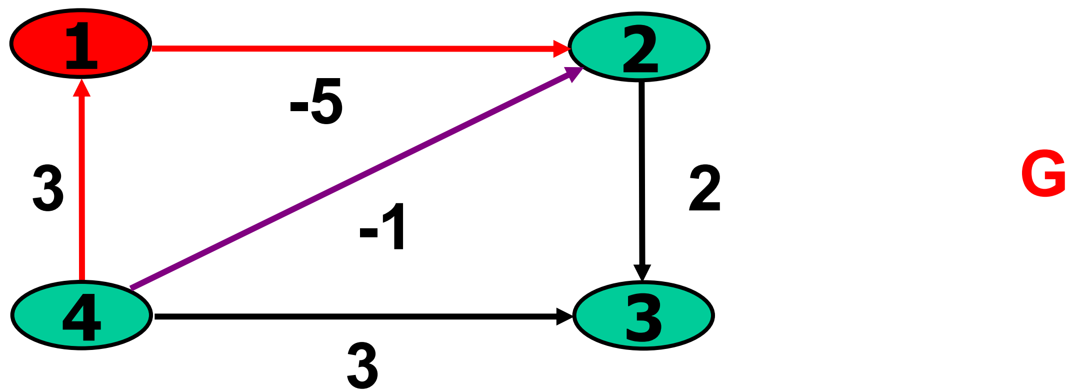


G

$$D_1=\begin{pmatrix} 0 & -5 & \infty & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -1 & 3 & 0 \end{pmatrix} \qquad \Pi_1=\begin{pmatrix} - & 1 & - & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 4 & 4 & - \end{pmatrix}$$

k=1

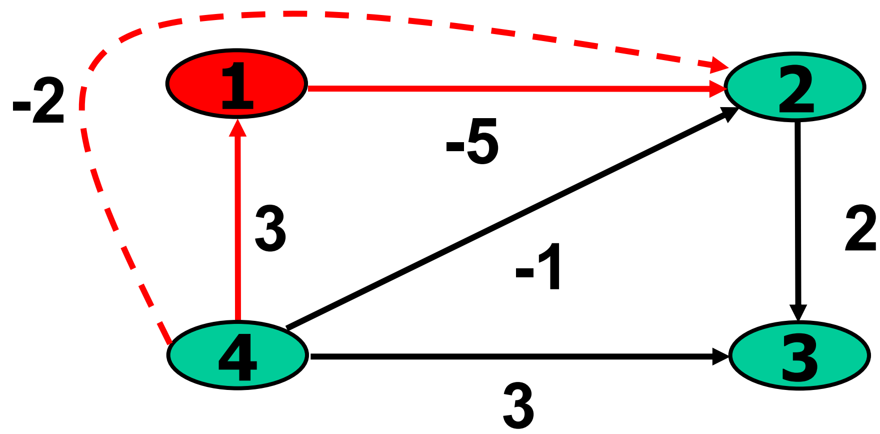
Example execution of the algorithm



$$D_1 = \begin{pmatrix} 0 & -5 & \infty & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 3 & 0 \end{pmatrix} \qquad \Pi_1 = \begin{pmatrix} - & 1 & - & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 4 & 4 & - \end{pmatrix}$$

k=1

Example execution of the algorithm

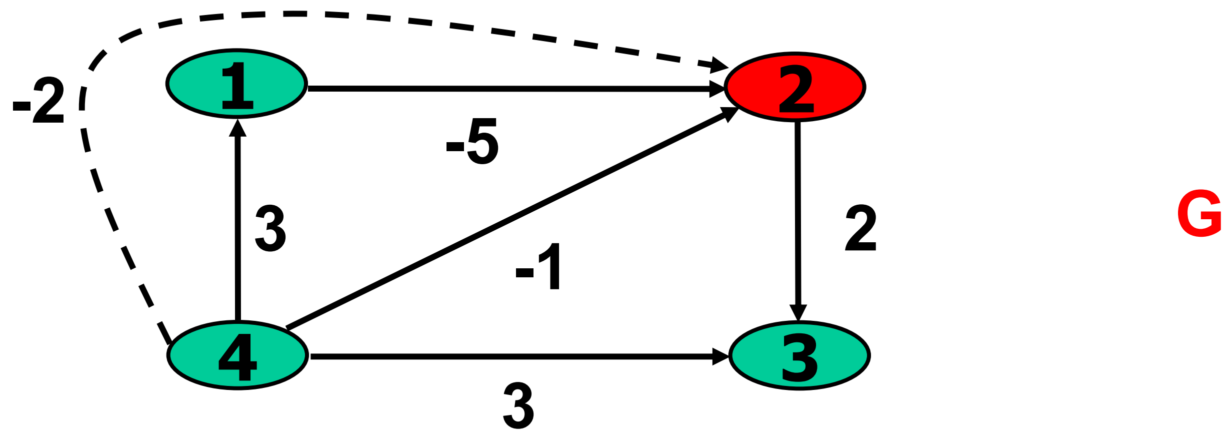


G

$$D_1 = \begin{pmatrix} 0 & -5 & \infty & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 3 & 0 \end{pmatrix} \qquad \Pi_1 = \begin{pmatrix} - & 1 & - & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 4 & - \end{pmatrix}$$

k=1

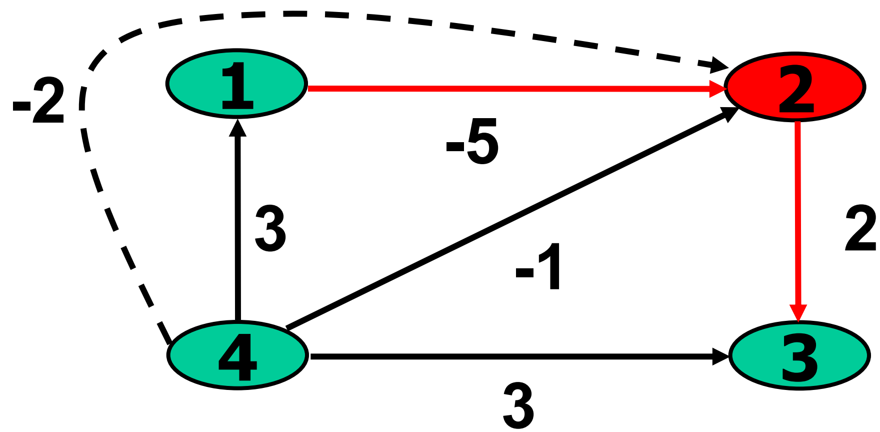
Example execution of the algorithm



$$D_2=\begin{pmatrix} 0 & -5 & \infty & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 3 & 0 \end{pmatrix} \qquad \Pi_2=\begin{pmatrix} - & 1 & - & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 4 & - \end{pmatrix}$$

k=2

Example execution of the algorithm

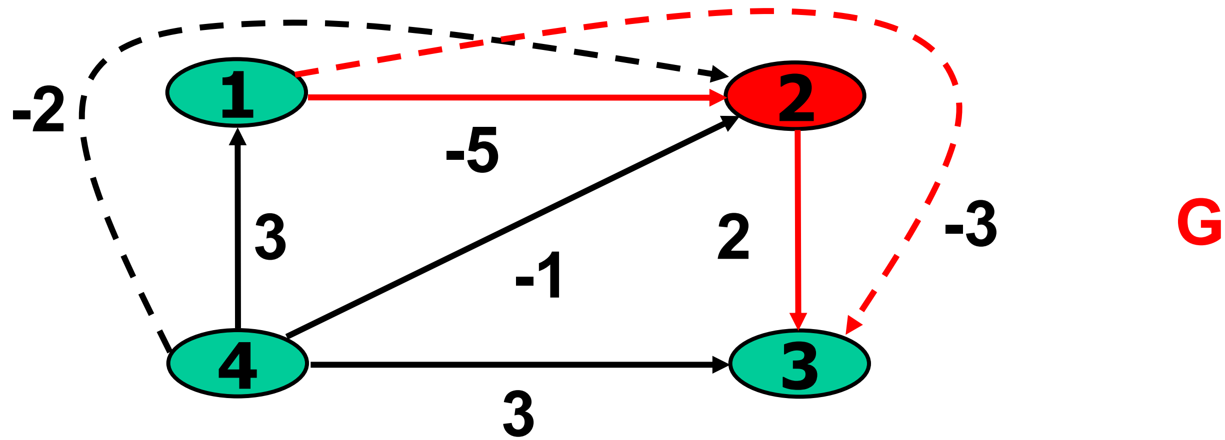


G

$$D_2=\begin{pmatrix} 0 & -5 & -3 & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 3 & 0 \end{pmatrix} \qquad \Pi_2=\begin{pmatrix} - & 1 & - & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 4 & - \end{pmatrix}$$

k=2

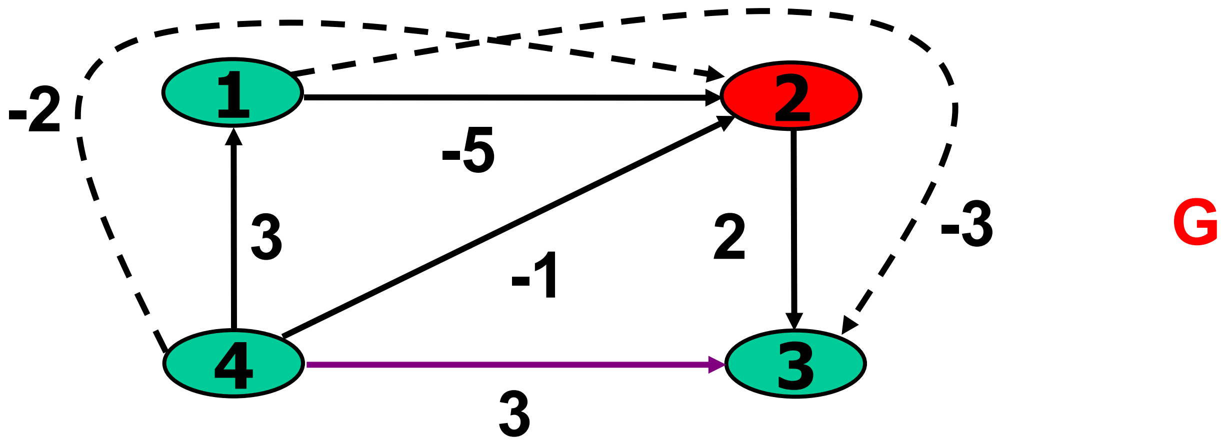
Example execution of the algorithm



$$D_2=\begin{pmatrix} 0 & -5 & -3 & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 3 & 0 \end{pmatrix} \qquad \Pi_2=\begin{pmatrix} - & 1 & 2 & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 4 & - \end{pmatrix}$$

k=2

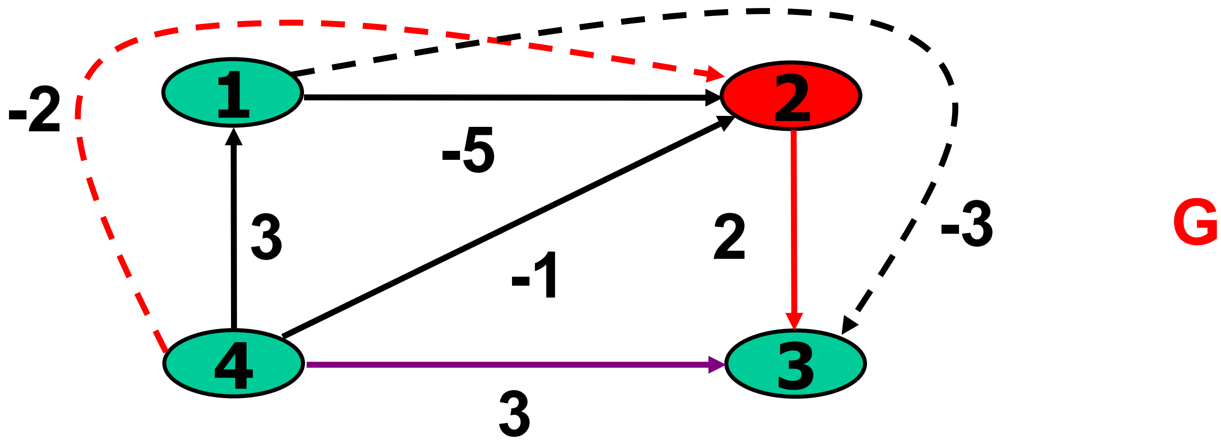
Example execution of the algorithm



$$D_2=\begin{pmatrix} 0 & -5 & -3 & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 3 & 0 \end{pmatrix} \qquad \Pi_2=\begin{pmatrix} - & 1 & 2 & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 4 & - \end{pmatrix}$$

k=2

Example execution of the algorithm

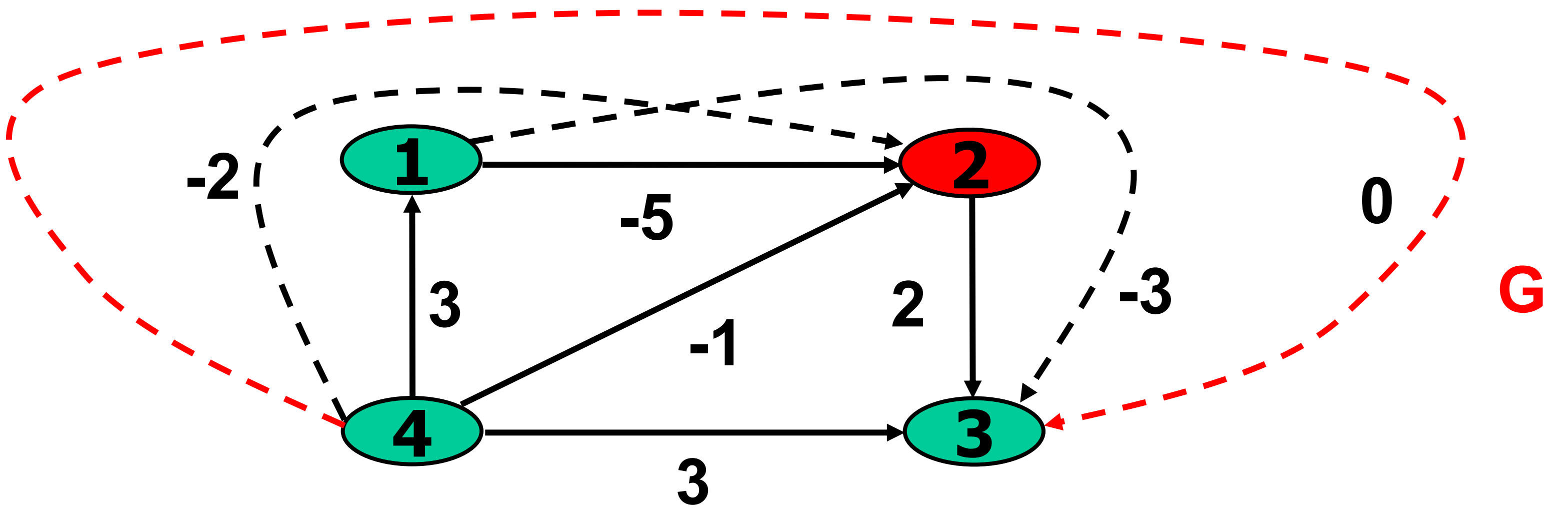


$$D_2=\begin{pmatrix} 0 & -5 & -3 & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 0 & 0 \end{pmatrix}$$

$$\Pi_2=\begin{pmatrix} - & 1 & 2 & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 4 & - \end{pmatrix}$$

k=2

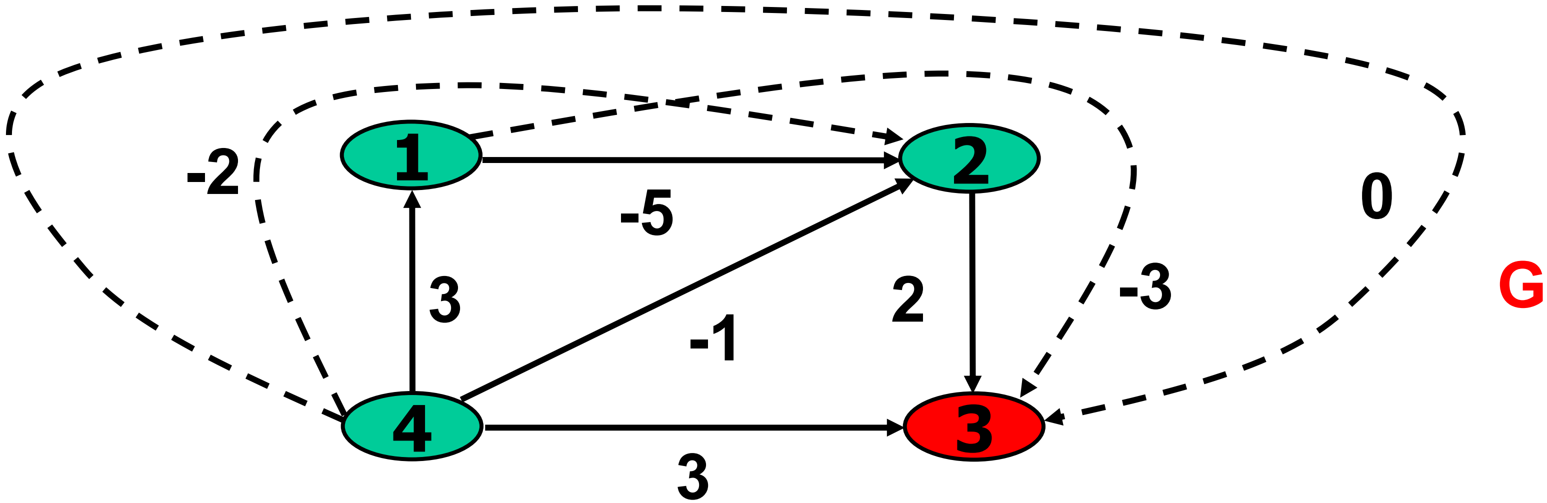
Example execution of the algorithm



$$D_2 = \begin{pmatrix} 0 & -5 & -3 & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 0 & 0 \end{pmatrix} \qquad \Pi_2 = \begin{pmatrix} - & 1 & 2 & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 2 & - \end{pmatrix}$$

k=2

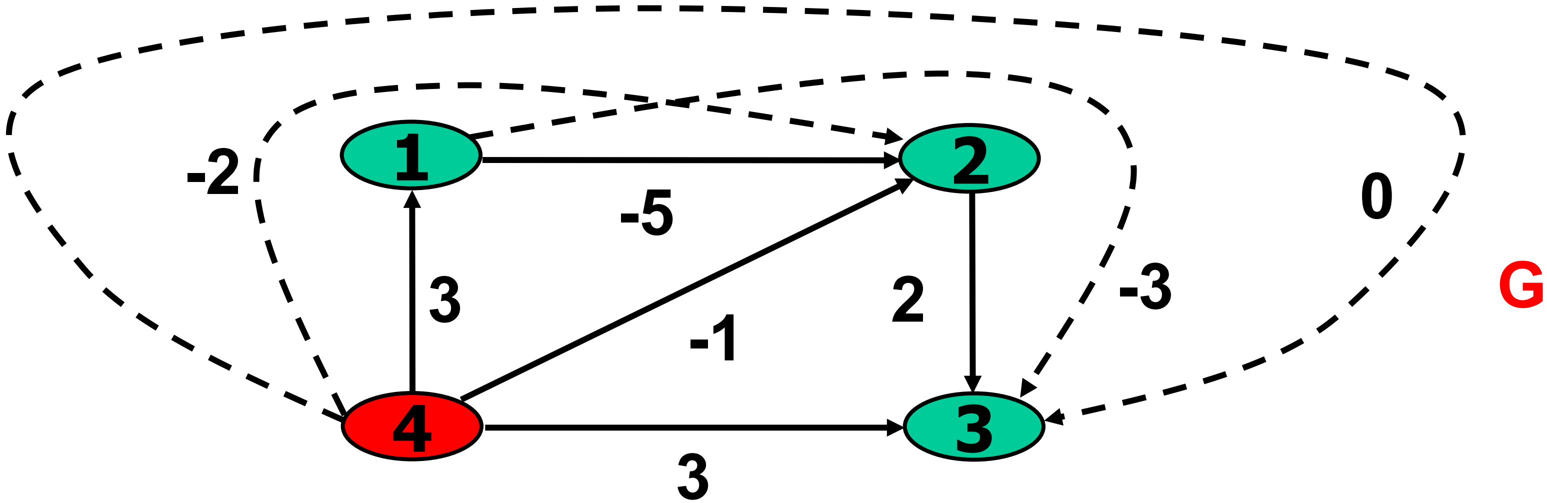
Example execution of the algorithm



$$D_3=\begin{pmatrix} 0 & -5 & -3 & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 0 & 0 \end{pmatrix} \qquad \Pi_3=\begin{pmatrix} - & 1 & 2 & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 2 & - \end{pmatrix}$$

k=3

Example execution of the algorithm

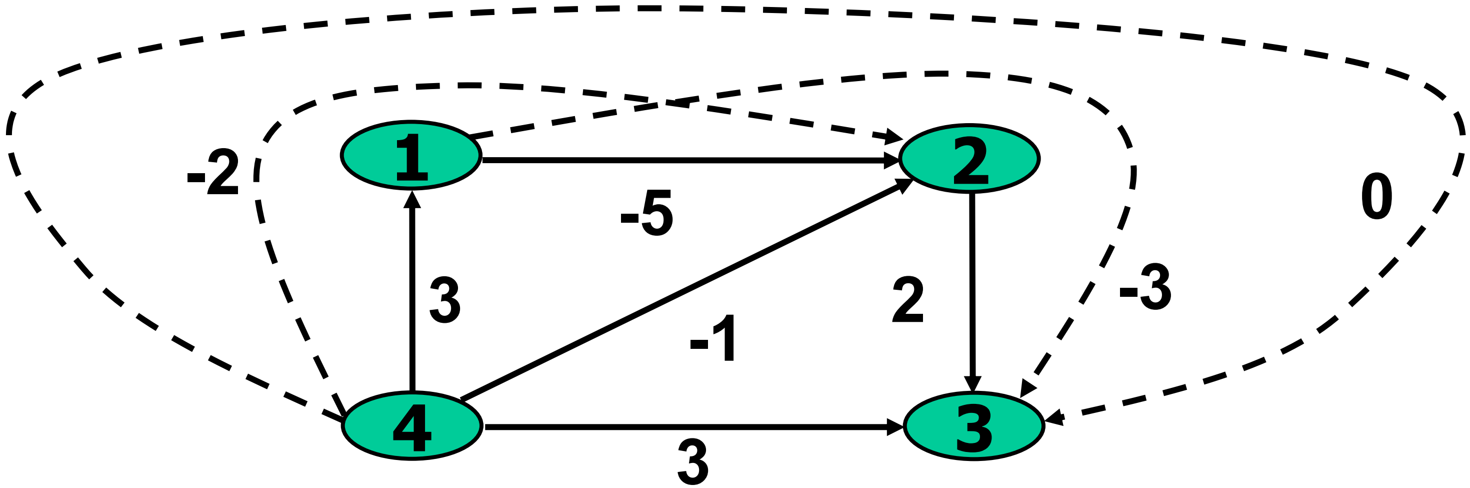


$$D_4=\begin{pmatrix} 0 & -5 & -3 & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 0 & 0 \end{pmatrix}$$

$$\Pi_4=\begin{pmatrix} - & 1 & 2 & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 2 & - \end{pmatrix}$$

k=4

Example execution of the algorithm



G

$$D^* = \begin{pmatrix} 0 & -5 & -3 & \infty \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & -2 & 0 & 0 \end{pmatrix} \qquad \Pi^* = \begin{pmatrix} - & 1 & 2 & - \\ - & - & 2 & - \\ - & - & - & - \\ 4 & 1 & 2 & - \end{pmatrix}$$

$D^*(i,j)$ contains the shortest path distance from v_i to v_j

$\Pi^*(i,j)$ contains the predecessor of v_j on a shortest path from from v_i to v_j