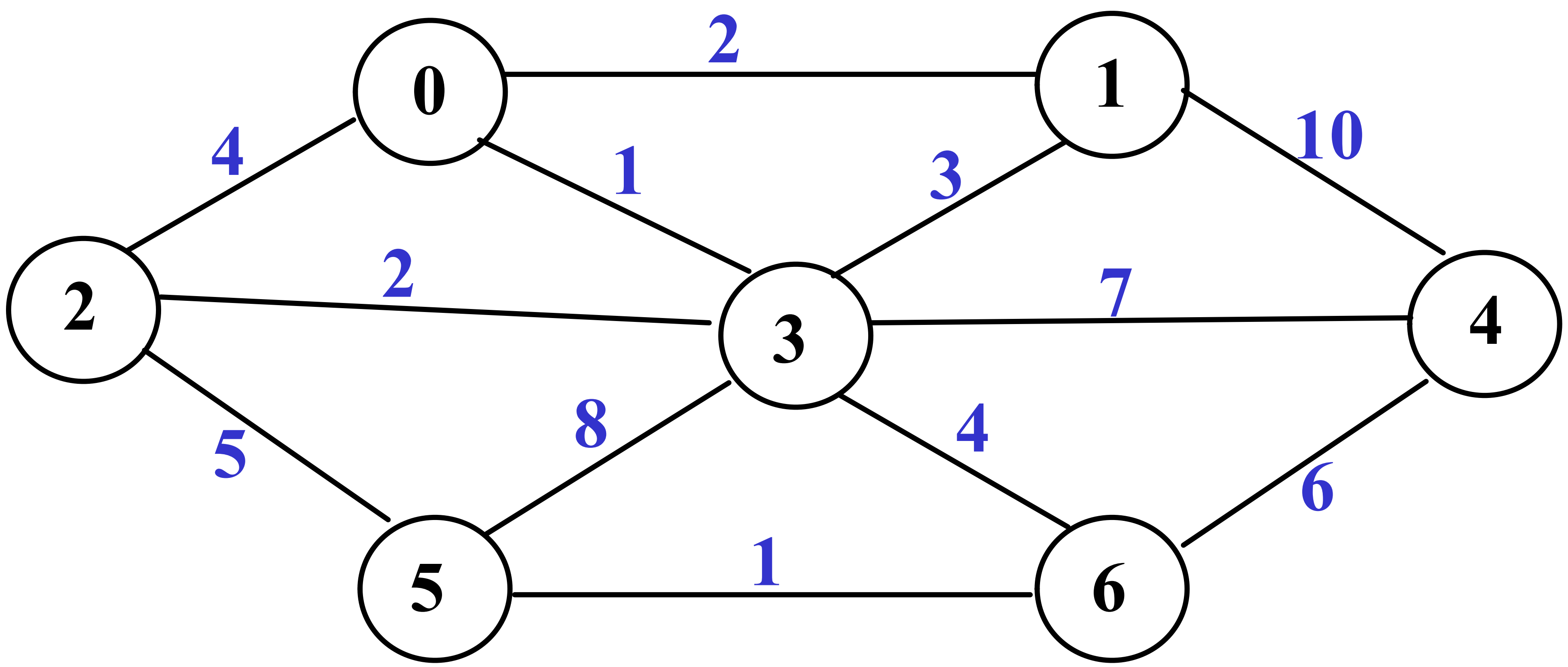
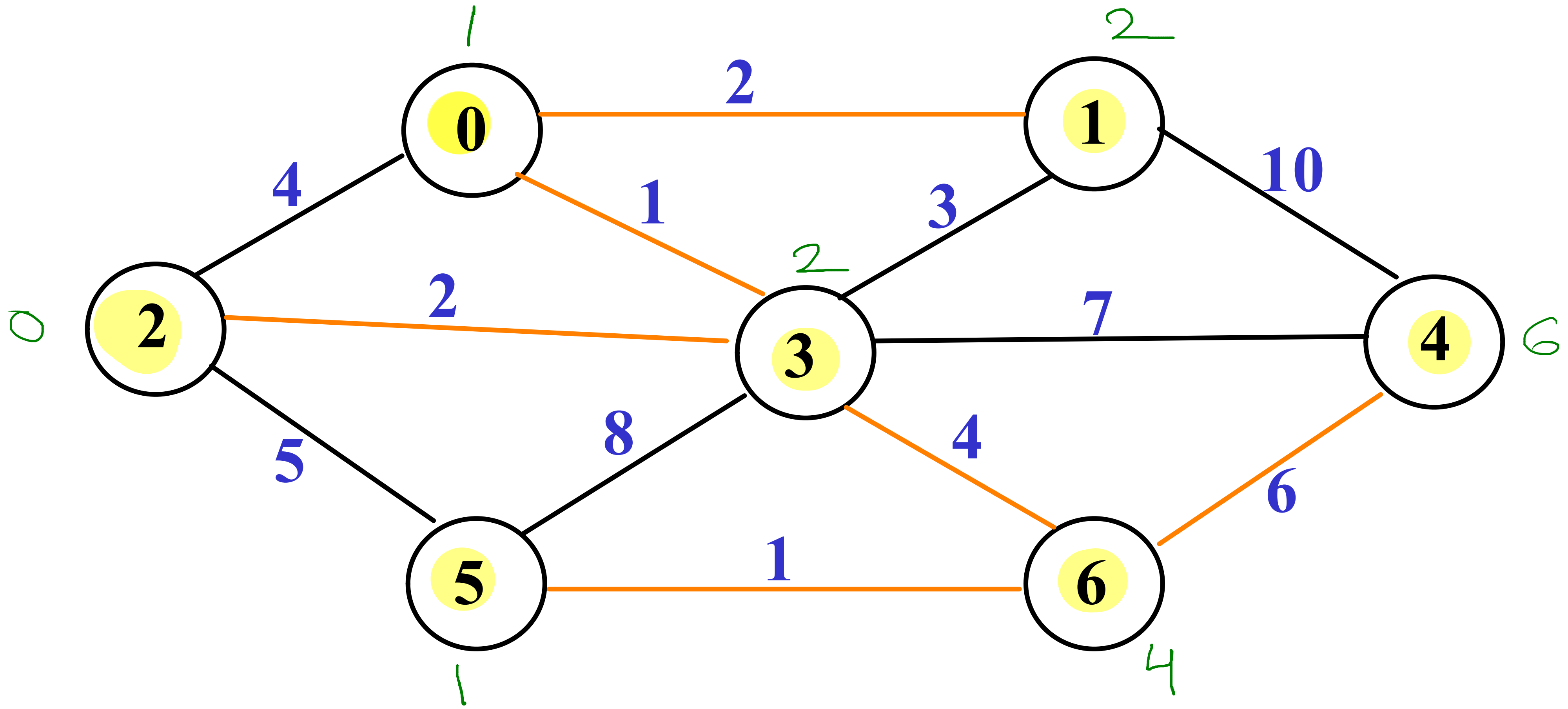
Prim's MST

1. Create a set mst to keep track of vertices included in MST.
2. Also keep track of parent of each vertex. Initialize parent of each vertex -1.
3. Assign a key to all vertices in the input graph. Key for all vertices should be initialized to INF. The start vertex key should be 0.
4. While mst doesn't include all the vertices
    - i. Pick a vertex u which is not there in mst and has minimum key.
    - ii. Include vertex u to mst.
    - iii. Update key and parent of all adjacent vertices of u.
        - a. For each adjacent vertex v,
            if weight of edge u-v is less than the current key of v,
            then update the key as weight of u-v.
        - b. Record u as parent of v.



Prim's MST



	K	P
0	1	3
1	2	0
2	0	-1
3	2	2
4	6	6
5	1	6
6	4	3

	K	P
0	4	2
1	∞	-1
2	0	-1
3	2	2
4	∞	-1
5	5	2
6	∞	-1

	K	P
0	1	3
1	3	3
2	0	-1
3	2	2
4	7	3
5	5	2
6	4	3

	K	P
0	1	3
1	2	0
2	0	-1
3	2	2
4	7	3
5	5	2
6	4	3

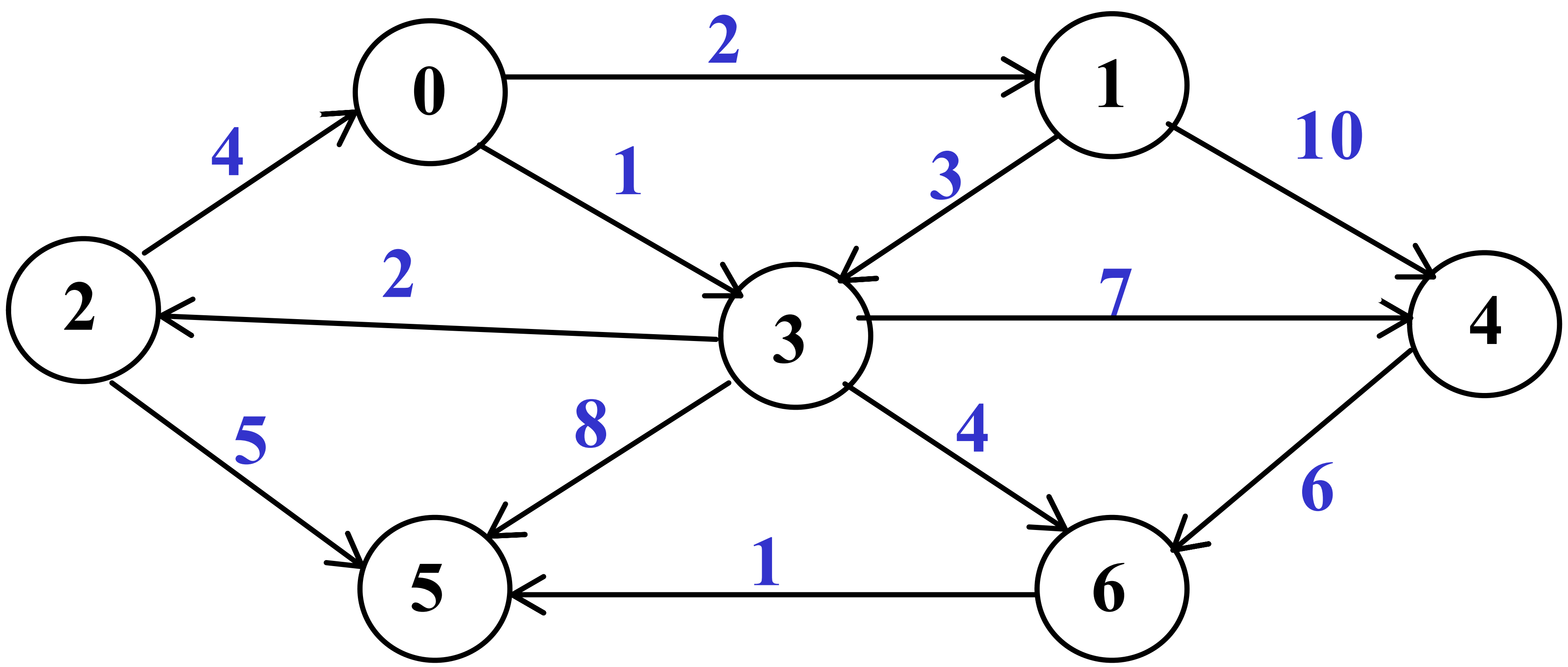
	K	P
0	1	3
1	2	0
2	0	-1
3	2	2
4	7	3
5	5	2
6	4	3

	K	P
0	1	3
1	2	0
2	0	-1
3	2	2
4	6	6
5	1	6
6	4	3

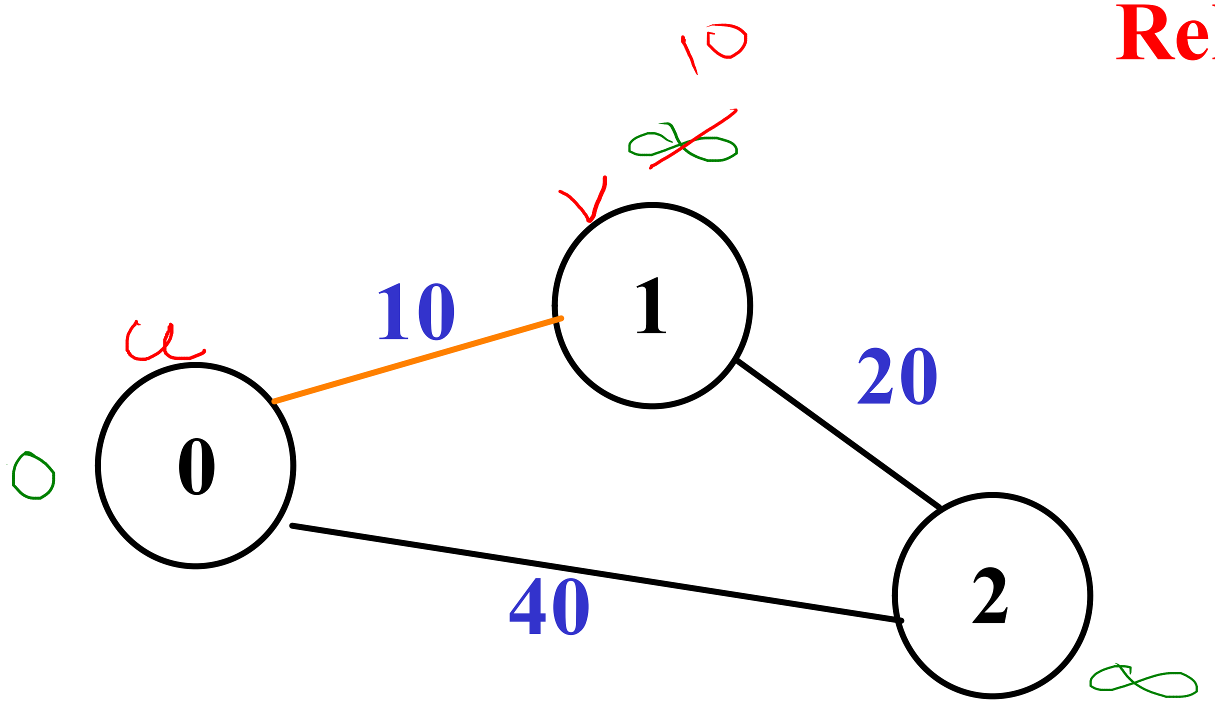
	K	P
0	1	3
1	2	0
2	0	-1
3	2	2
4	6	6
5	1	6
6	4	3

Dijkstra's Algorithm

1. Create a set spt to keep track of vertices included in shortest path tree.
2. Track distance of all vertices in the input graph. Distance for all vertices should be initialized to INF. The start vertex distance should be 0.
3. While spt doesn't include all the vertices
   i. Pick a vertex u which is not there in spt and has minimum distance.
   ii. Include vertex u to spt.
   iii. Update distances of all adjacent vertices of u.
   For each adjacent vertex v,
      if distance of u + weight of edge u-v is less than the current distance of v, then update its distance as distance of u + weight of edge u-v.

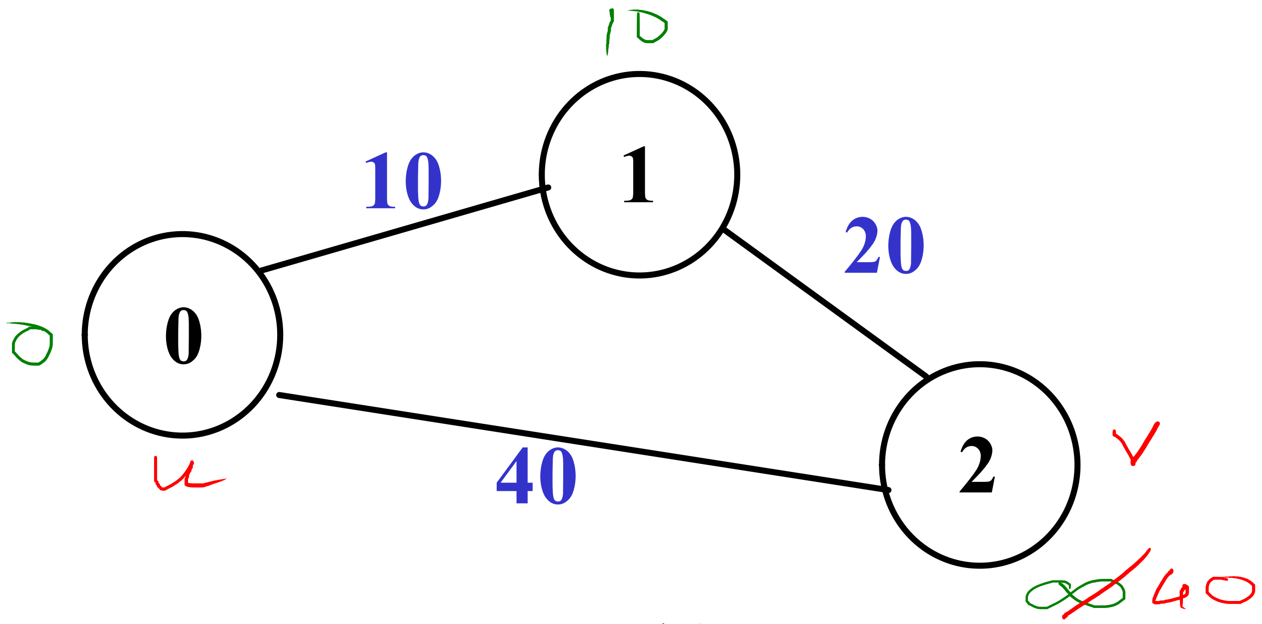


Relaxation

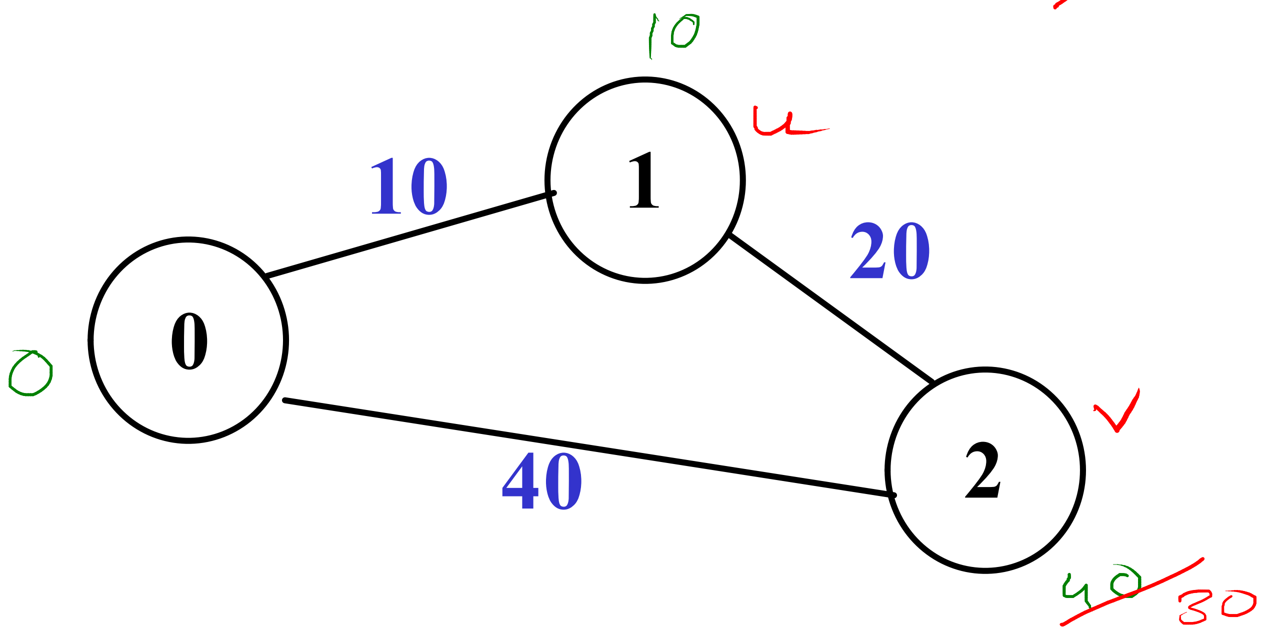



```
if(dist[u]+wt(u,v) < dist[v])
    dist[v]=dist[u]+wt(u,v)

if(0+10 < ∞)
    dist[1] = 10;
```

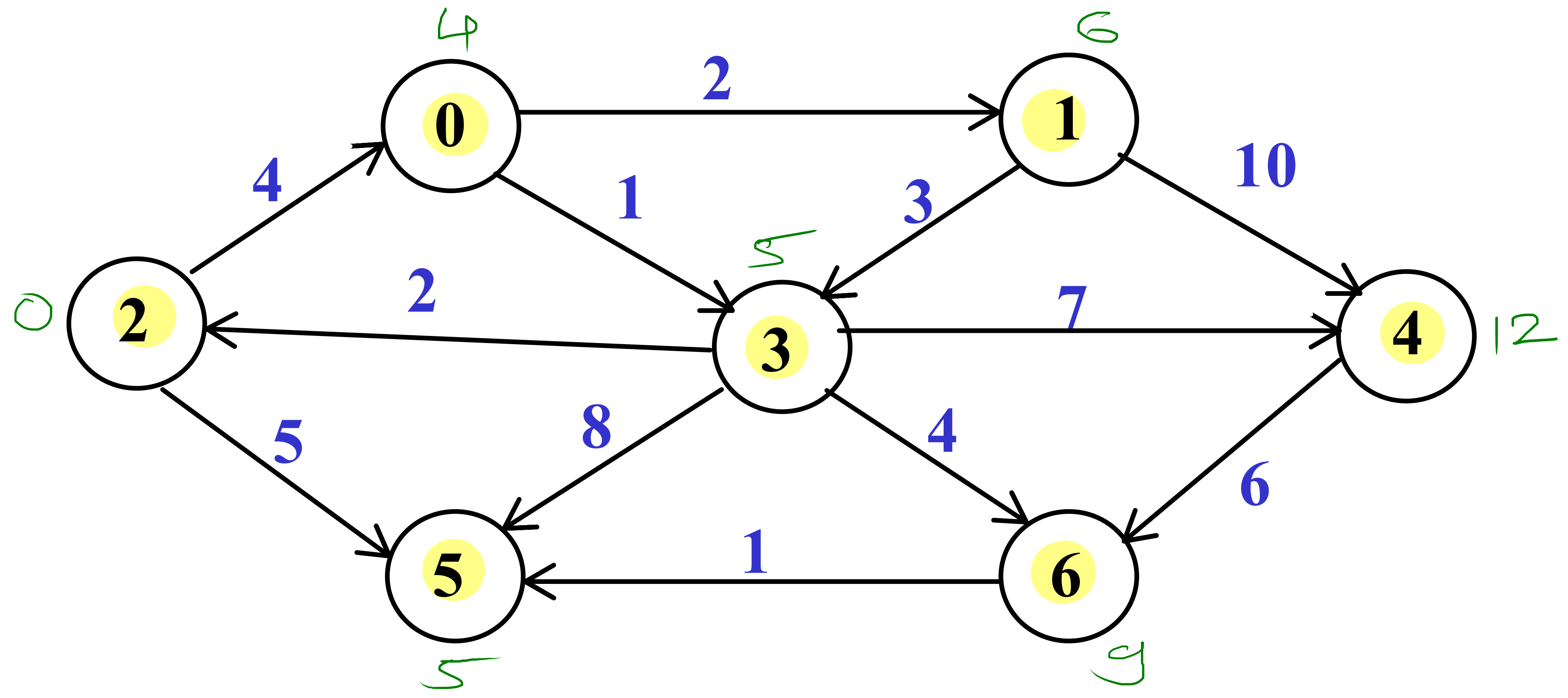



```
if(dist[0]+wt(0,2) < dist[2])
    dist[2] = dist[0]+wt(0,2)
if(0+40 < ∞)
    dist[2] = 40;
```




```
if( 10+20 < 40)
    dist[2] = 30
```

Dijkstra's Algorithm



	D	P
0	4	2
1	6	0
2	0	-1
3	5	0
4	12	3
5	5	2
6	9	3

	D	P
0	4	2
1	∞	-1
2	0	-1
3	∞	-1
4	∞	-1
5	5	2
6	∞	-1

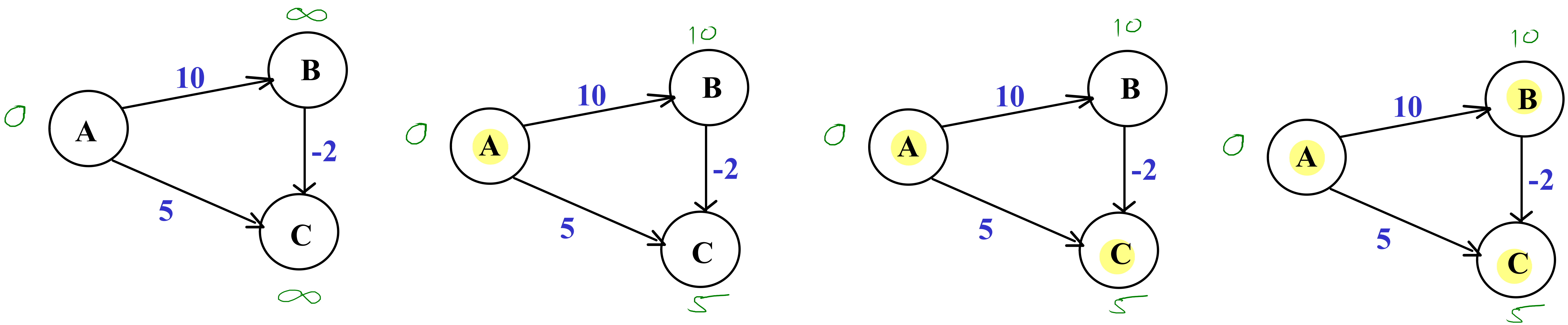
	D	P
0	4	2
1	6	0
2	0	-1
3	5	0
4	∞	-1
5	5	2
6	∞	-1

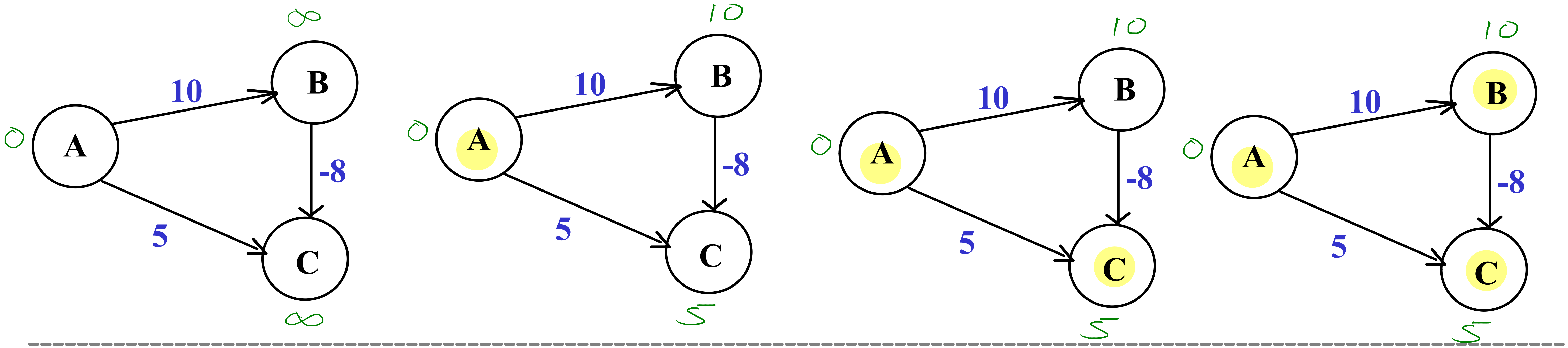
	D	P
0	4	2
1	6	0
2	0	-1
3	5	0
4	12	3
5	5	2
6	9	3

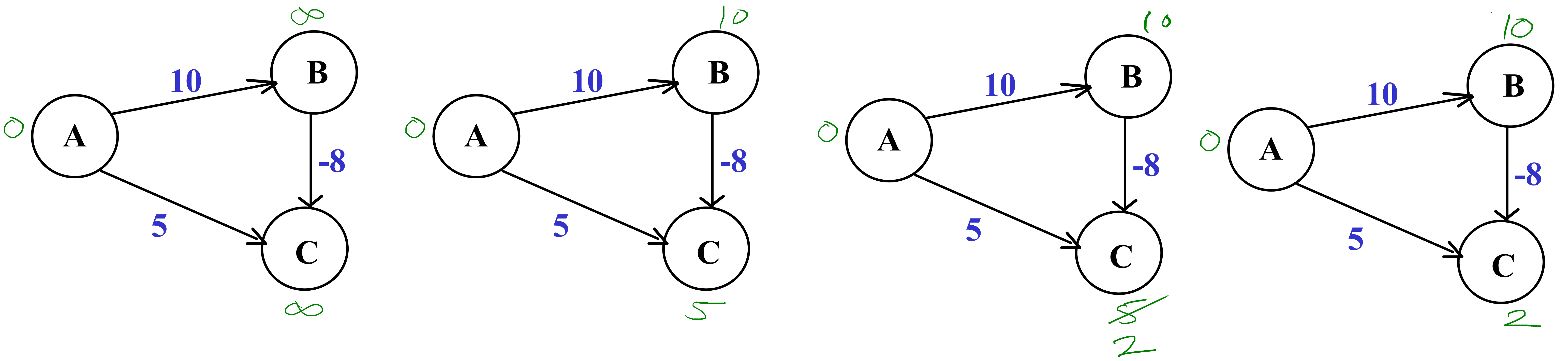
	D	P
0	4	2
1	6	0
2	0	-1
3	5	0
4	12	3
5	5	2
6	9	3

	D	P
0	4	2
1	6	0
2	0	-1
3	5	0
4	12	3
5	5	2
6	9	3

	D	P
0	4	2
1	6	0
2	0	-1
3	5	0
4	12	3
5	5	2
6	9	3

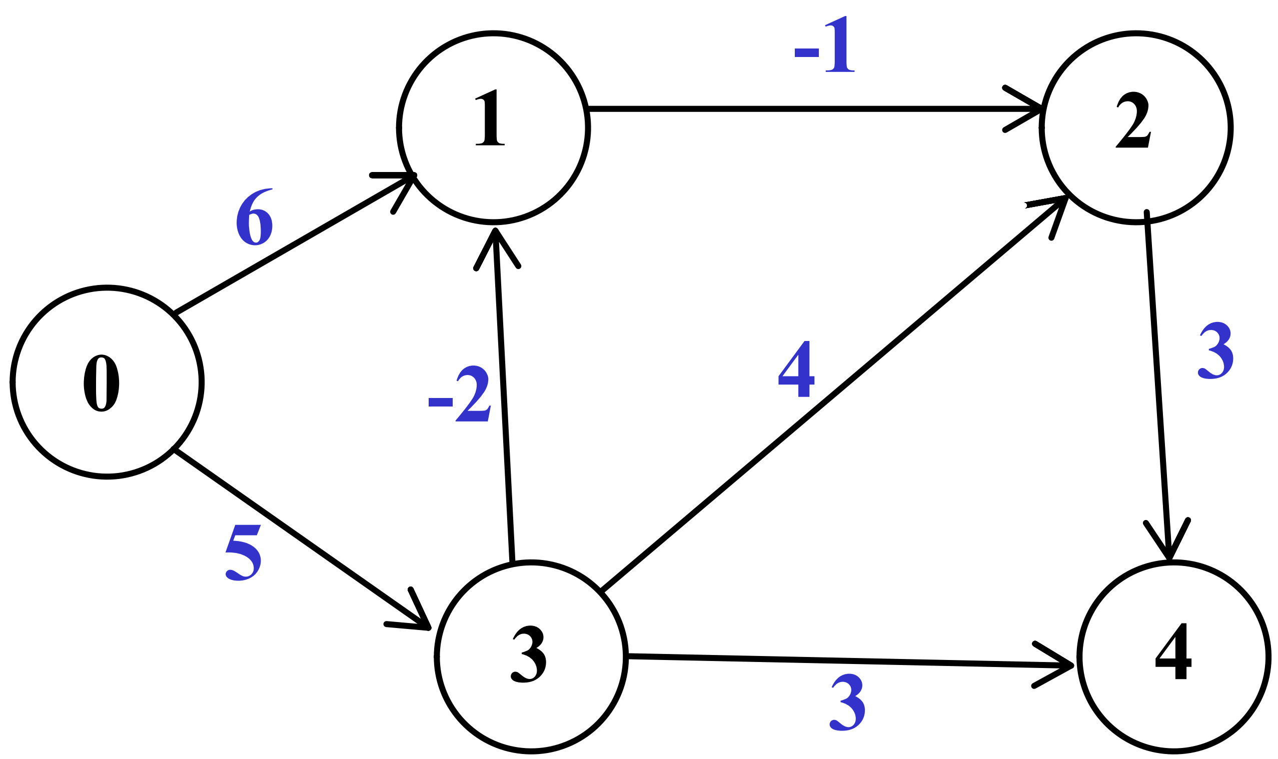

A
B
C
10
5
-2
0
∞
∞
0
10
5

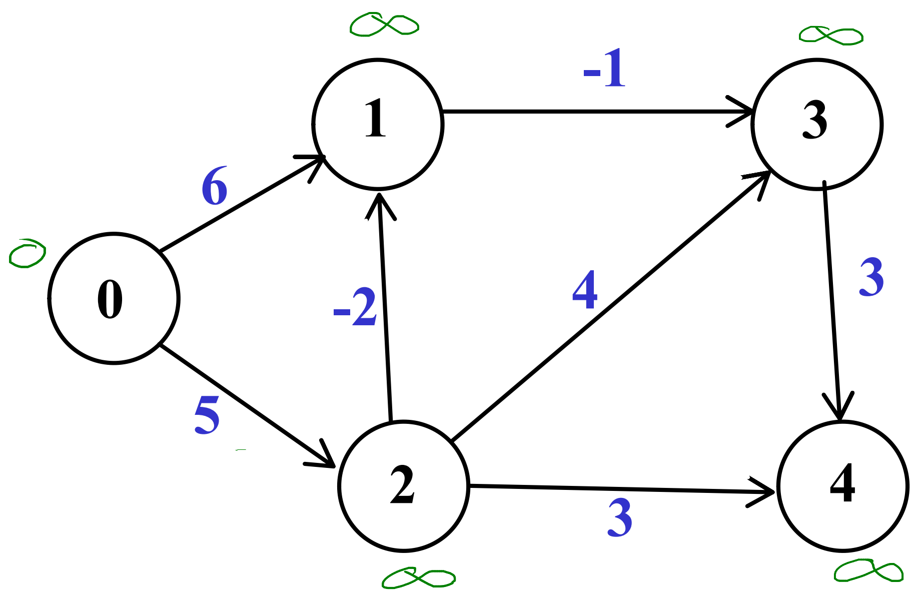

A
B
C
10
5
-8
0
∞
∞
0
10
5

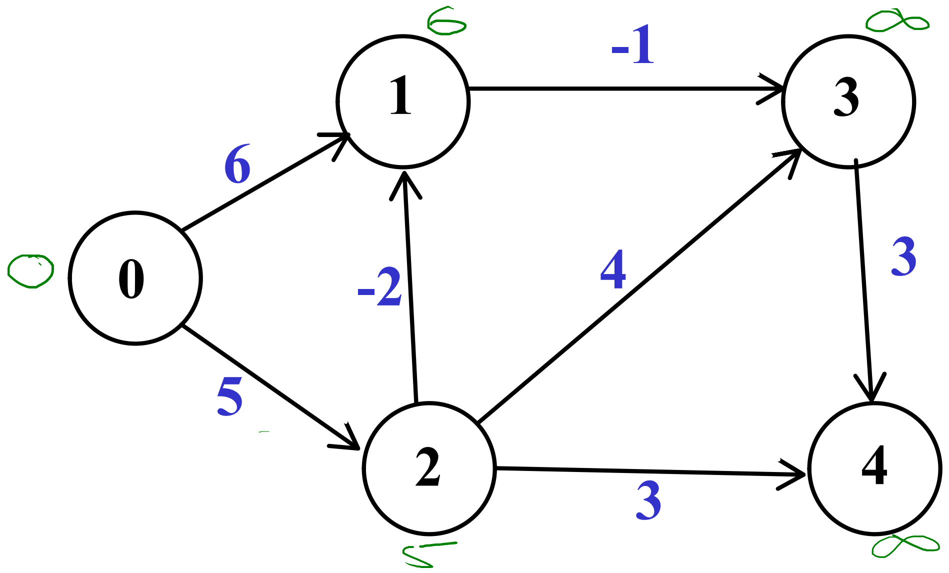

A
B
C
10
5
-8
0
∞
∞
0
10
5
2

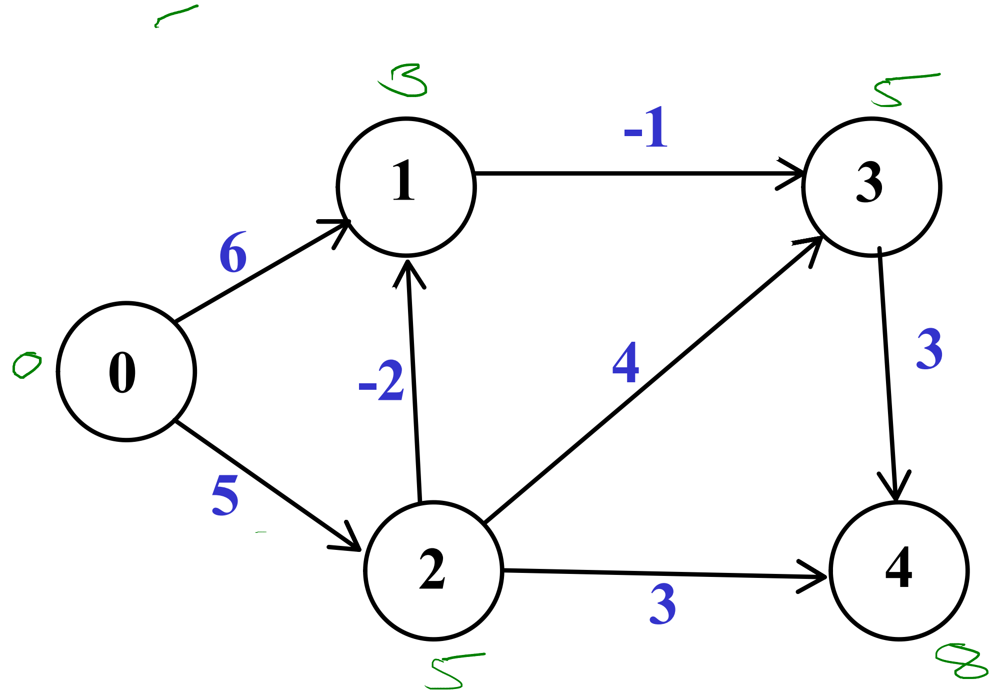
Bellman Ford Algorithm

1. Initializes distances from the source to all vertices as infinite and
   distance to the source itself as 0.
2. Calculates shortest distance V-1 times:
   For each edge u-v,
   if dist[v] > dist[u] + weight of edge u-v,
   then update dist[v], so that
   dist[v] = dist[u] + weight of edge u-v.
3. Check if negative edge in the graph:
   For each edge u-v,
   if dist[v] > dist[u] + weight of edge (u,v),
   then graph has -ve weight cycle.

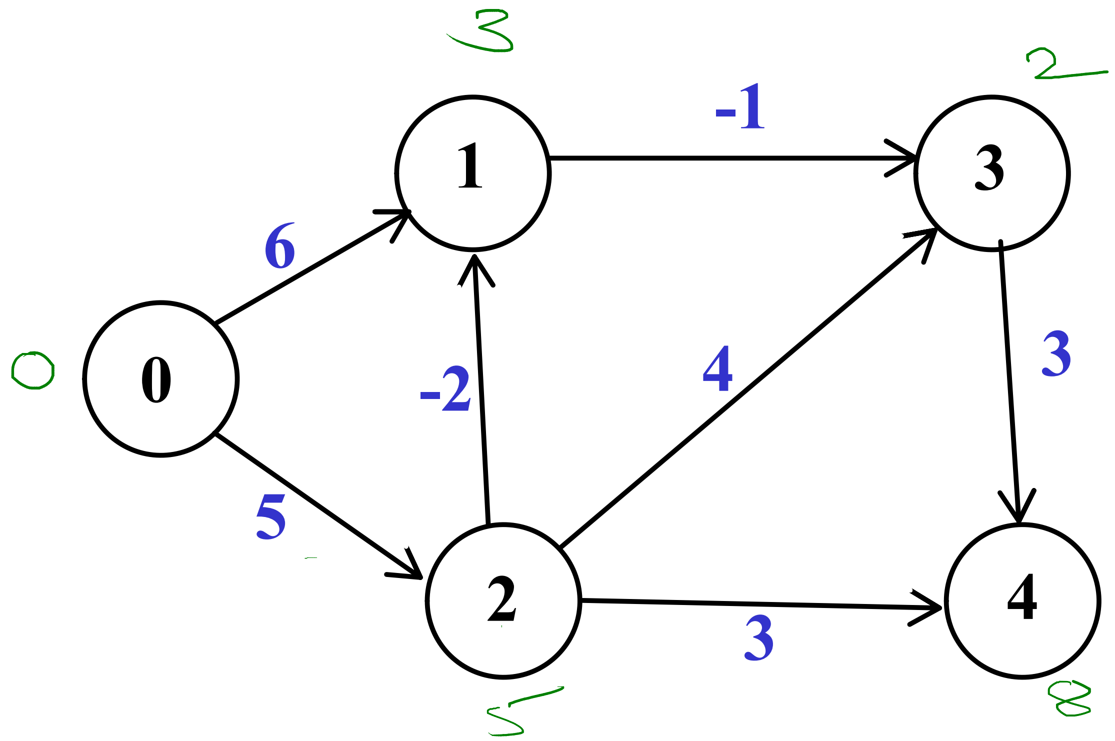


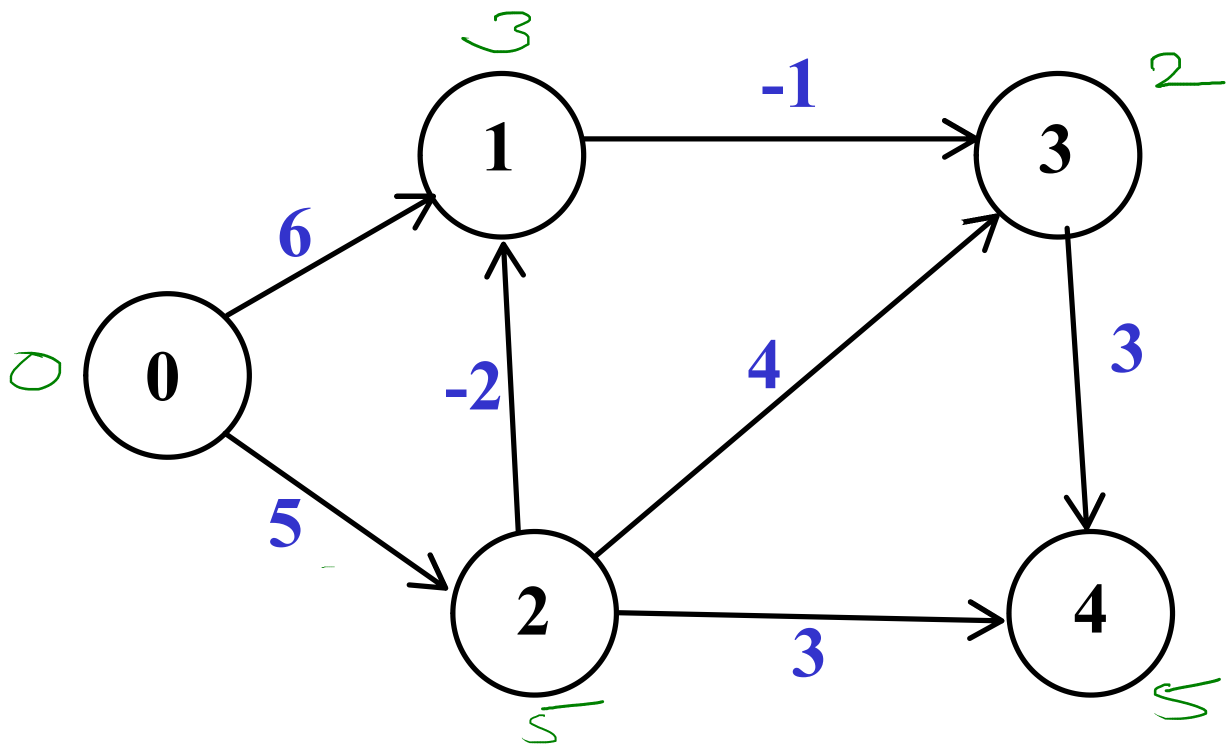


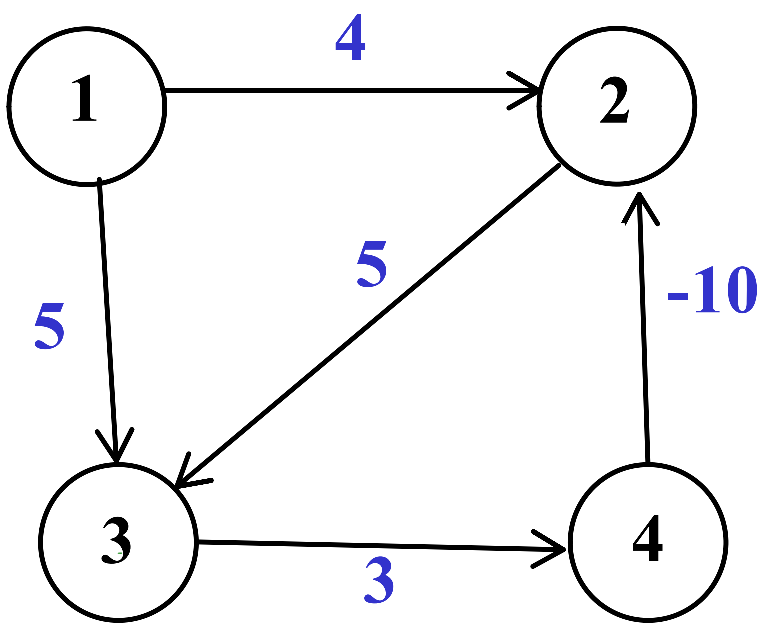


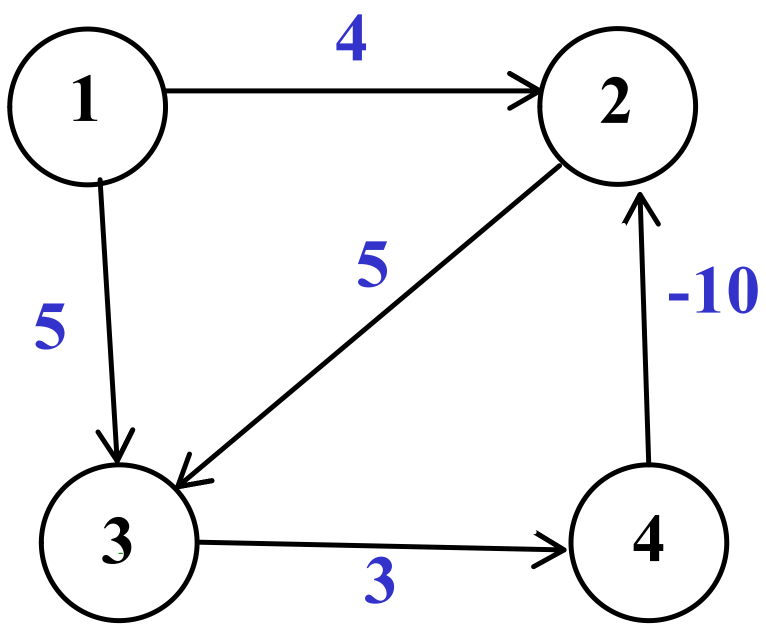


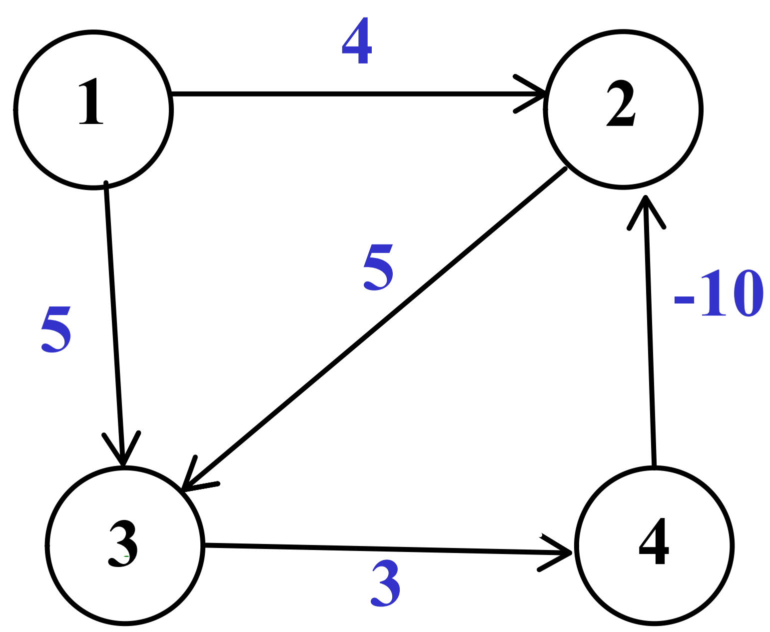
	0	1	2	3	4
	0	∞	∞	∞	∞
Pass 1	0	6	5	∞	∞
Pass 2	0	3	5	5	8
Pass 3	0	3	5	2	8
Pass 4	0	3	5	2	5

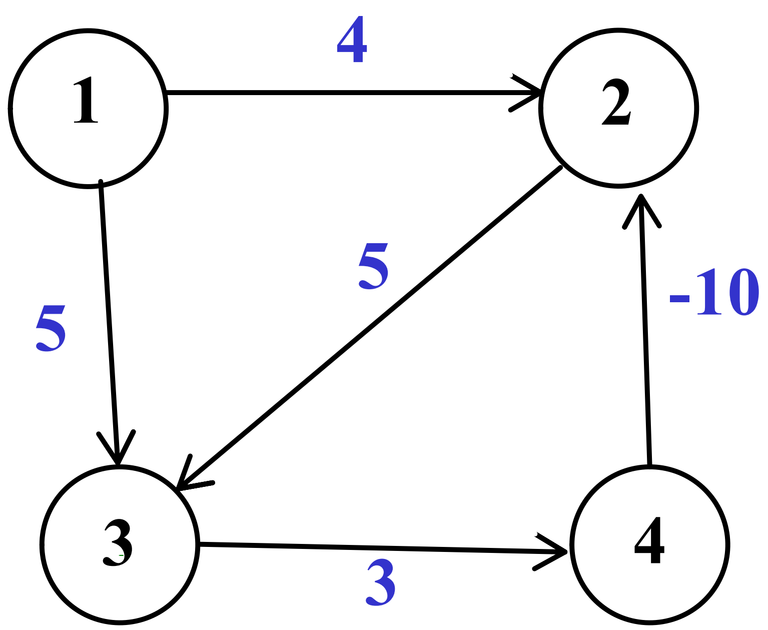


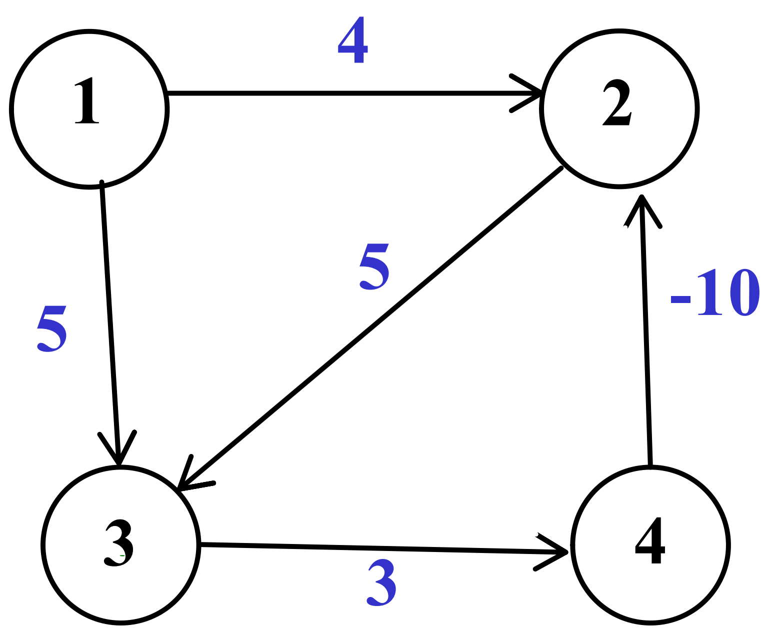


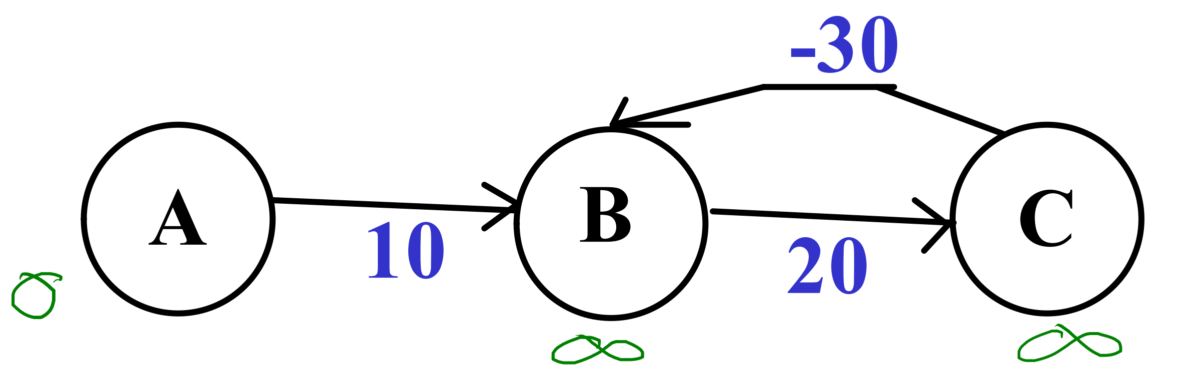

1
4
2
5
5
-10
3
3
4

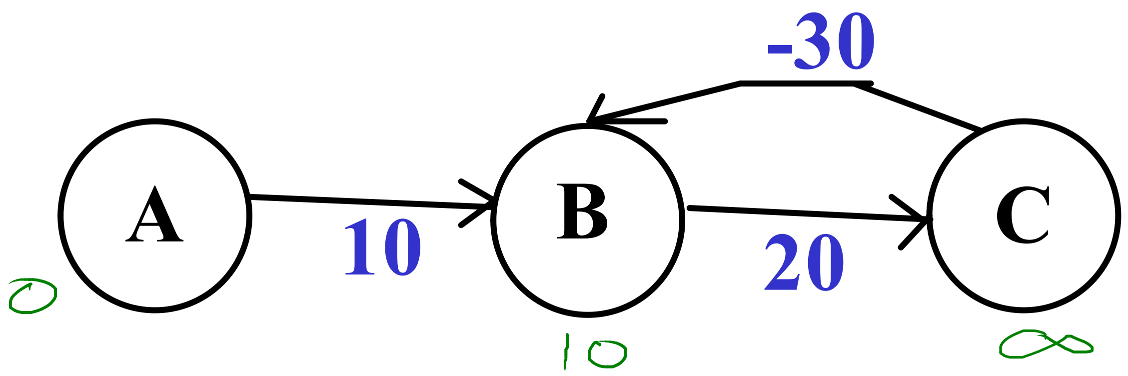

1
4
2
5
5
-10
3
3
4

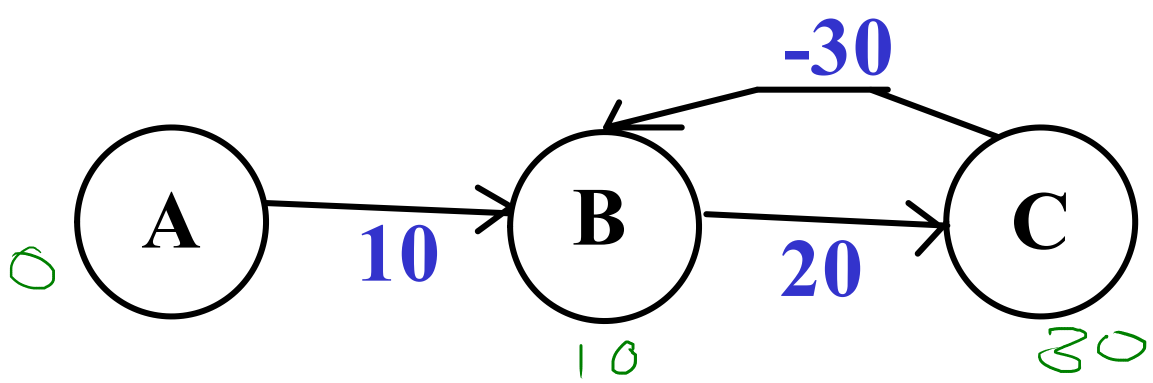

1
4
2
5
5
-10
3
3
4

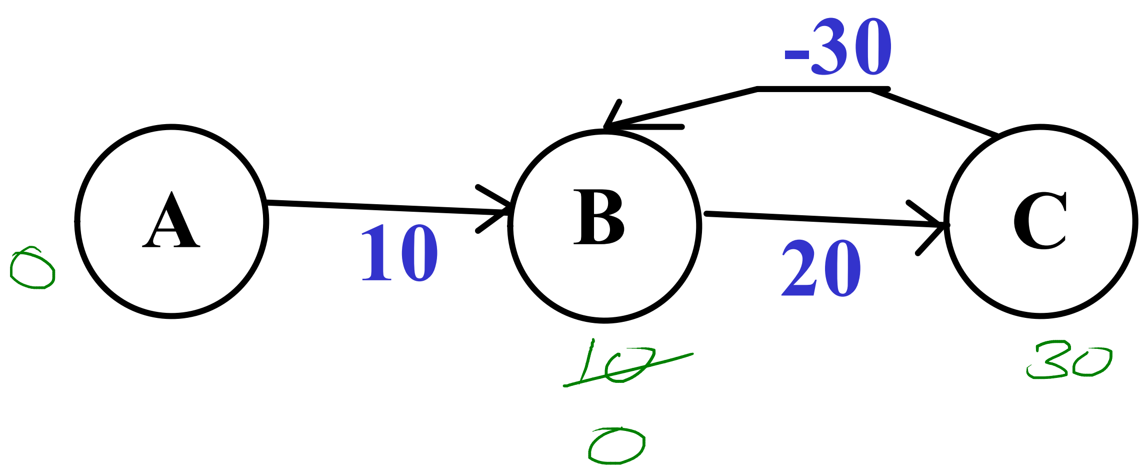

1
4
2
5
5
-10
3
3
4


1
4
2
5
5
-10
3
3
4


-30
A
B
C
10
20
0
∞
∞


-30
A
B
C
10
20
0
10
∞

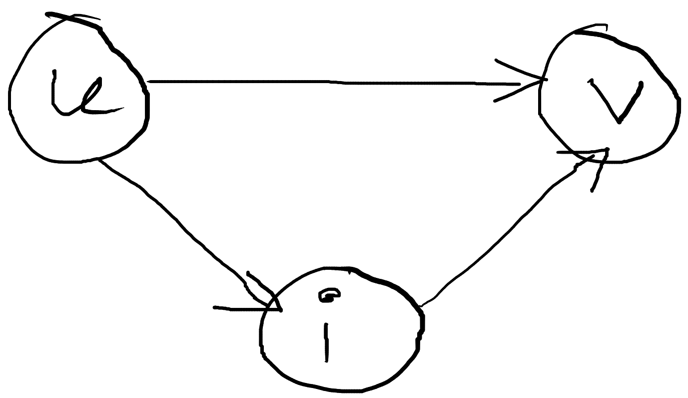

-30
A
B
C
10
20
0
10
30


-30
A
B
C
10
20
0
~~10~~
0
30

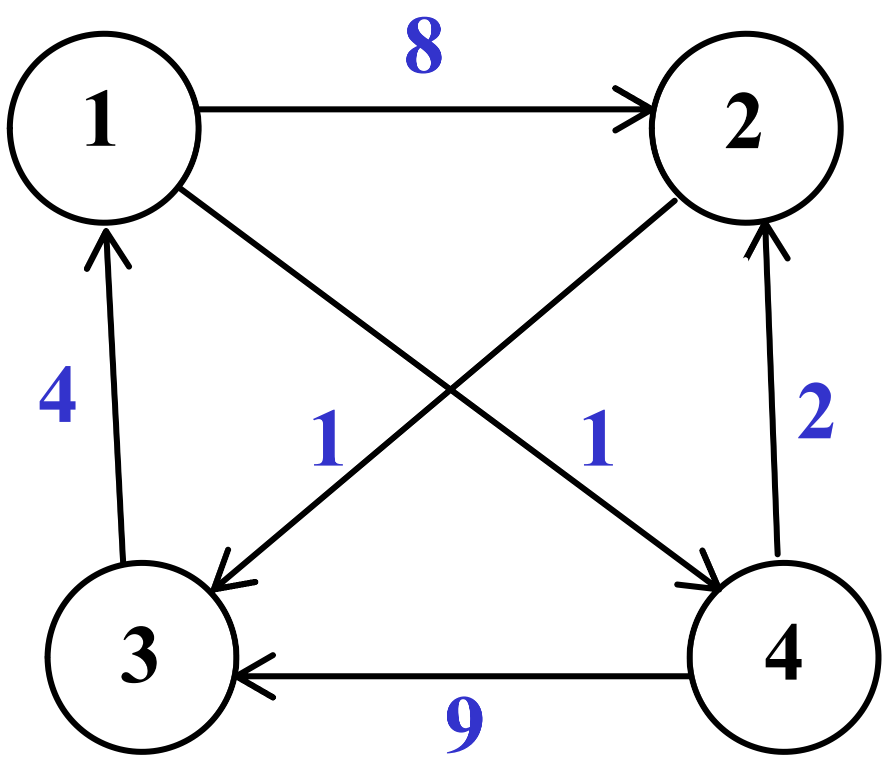
Warshall Floyd Algorithm

1. Create distance matrix to keep distance of every vertex from each vertex.
   Initially assign it with weights of all edges among vertices
   (i.e. adjacency matrix).

2. Consider each vertex (i) in between pair of any two vertices (s, d) and
   find the optimal distance between s & d considering intermediate vertex
   i.e. dist(s,d) = dist(s,i) + dist(i,d),
   if dist(s,i) + dist(i,d) $<$ dist(s,d).



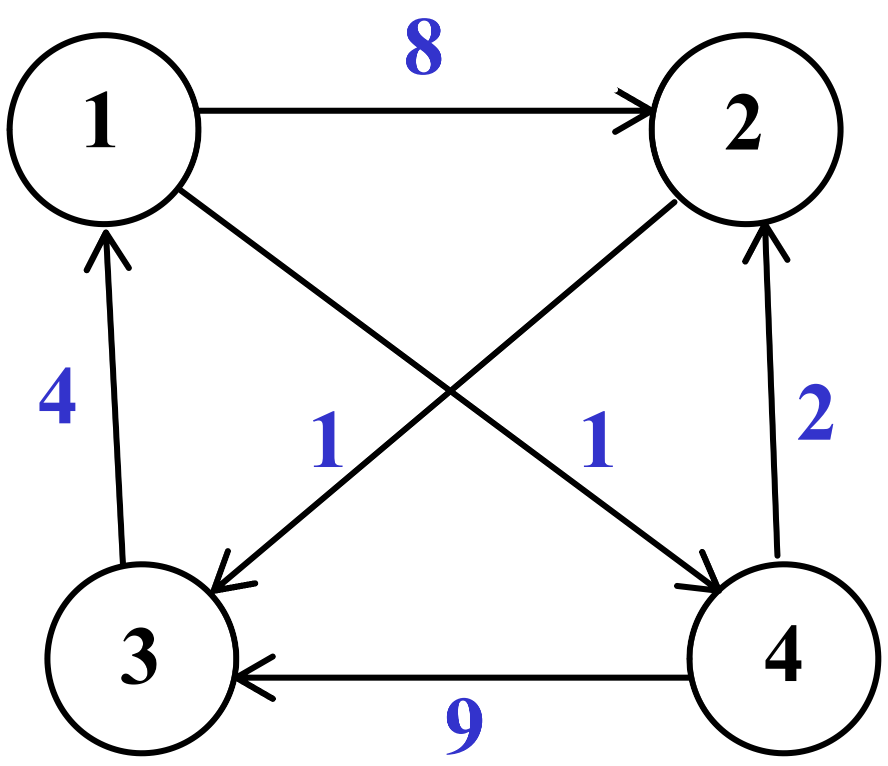
$$\text{if}(\text{dist}(u,i) + \text{dist}(i,v) < \text{dist}(u,v))$$
$$\text{dist}[v] = \text{dist}(u,i) + \text{dist}(i,v)$$



$$\begin{array}{cc} & \begin{array}{cccc}1 & 2 & 3 & 4\end{array} \\ \begin{array}{c}1\\2\\3\\4\end{array} & \begin{bmatrix}\infty & 8 & \infty & 1\\ \infty & \infty & 1 & \infty\\ 4 & \infty & \infty & \infty\\ \infty & 2 & 9 & \infty\end{bmatrix}\end{array}$$

$$A^0 = \begin{array}{cc} & \begin{array}{cccc}1 & 2 & 3 & 4\end{array} \\ \begin{array}{c}1\\2\\3\\4\end{array} & \begin{bmatrix}0 & 8 & \infty & 1\\ \infty & 0 & 1 & \infty\\ 4 & \infty & 0 & \infty\\ \infty & 2 & 9 & 0\end{bmatrix}\end{array}$$

$$A^1 = \begin{array}{cc} & \begin{array}{cccc}1 & 2 & 3 & 4\end{array} \\ \begin{array}{c}1\\2\\3\\4\end{array} & \begin{bmatrix}0 & 8 & \infty & 1\\ \infty & 0 & 1 & \infty\\ 4 & 12 & 0 & 5\\ \infty & 2 & 9 & 0\end{bmatrix}\end{array}$$



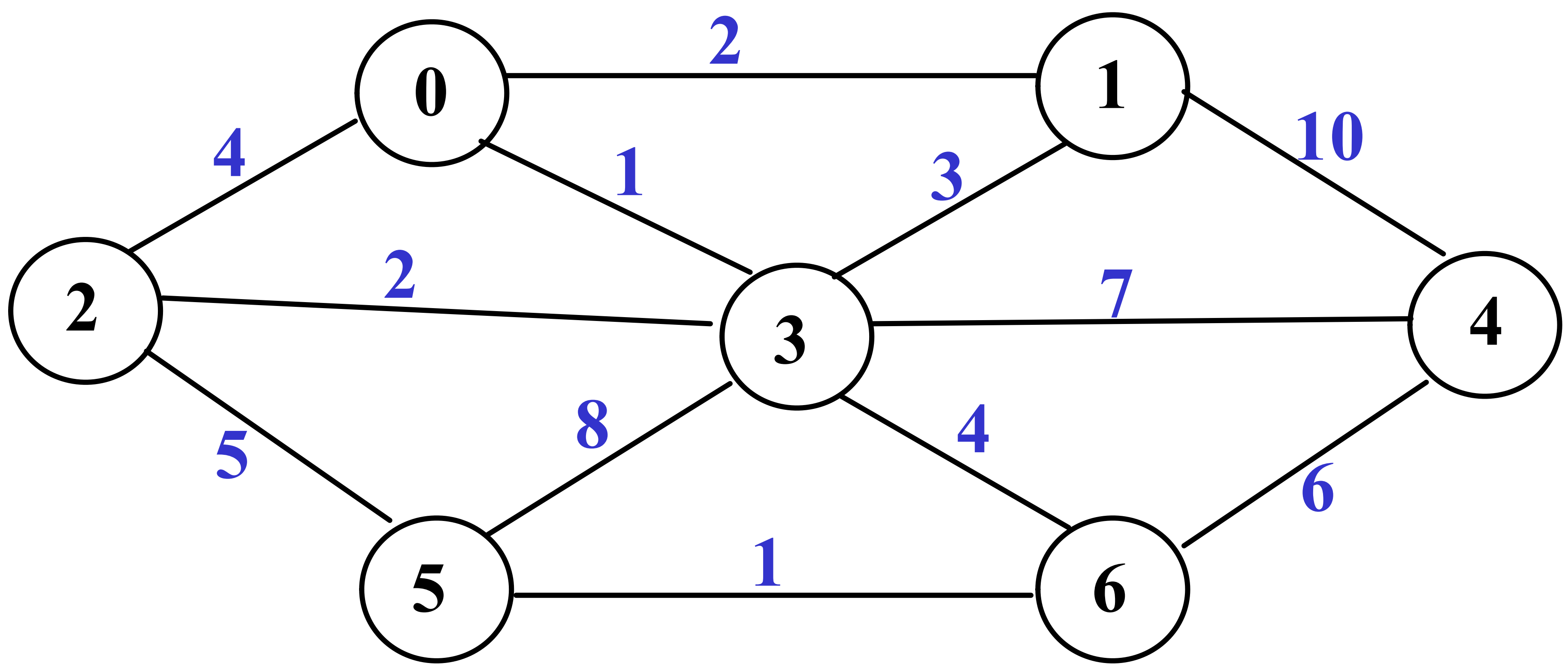
$$A^2 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix} \end{array}$$

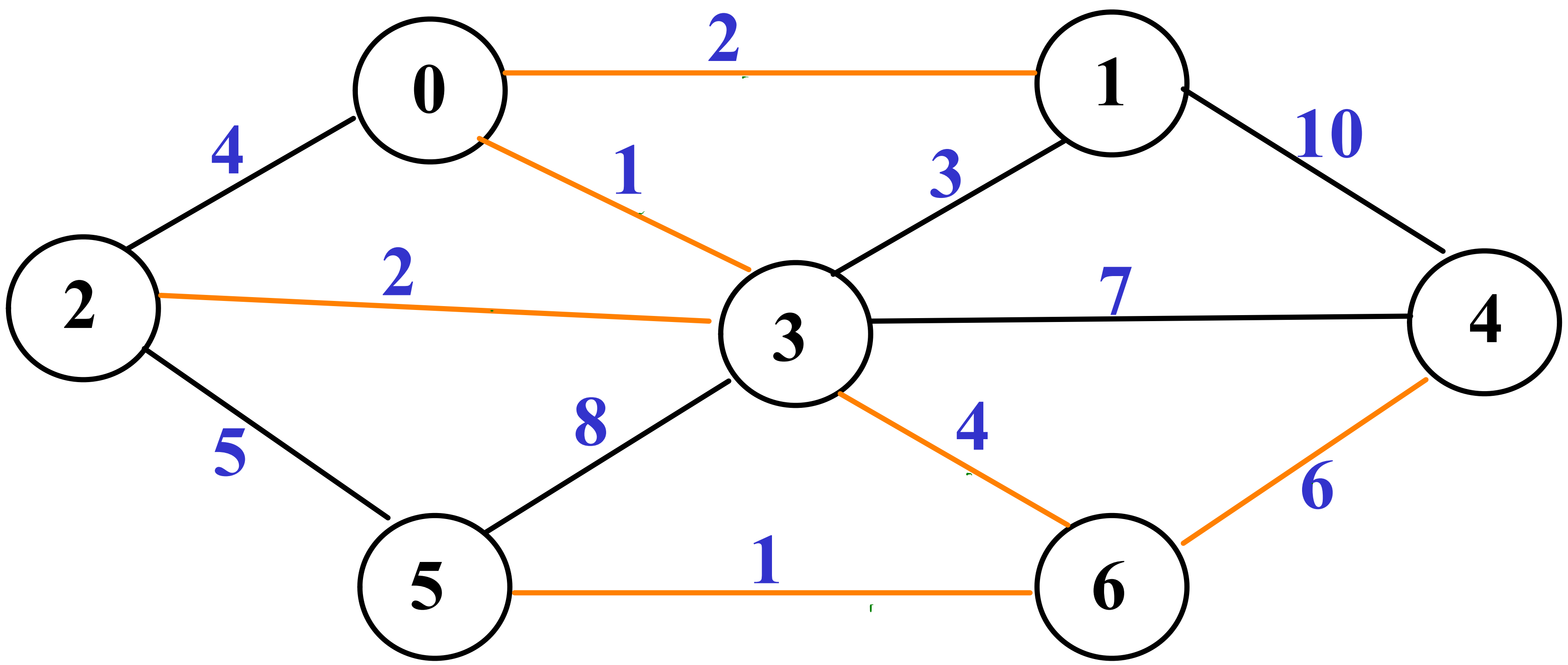
$$A^3 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

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

Kruskal's MST

1. Sort all the edges in ascending order of their weight.
2. Pick the smallest edge.
   Check if it forms a cycle with the spanning tree formed so far.
   If cycle is not formed, include this edge.
   Else, discard it.
3. Repeat step 2 until there are (V-1) edges in the spanning tree.





Vertex count = 7

Edge count = 6

Weight = 16

0 3 - 1

6 5 - 1

0 1 - 2

3 2 - 2

1 3 - 3

2 0 - 4

3 6 - 4

2 5 - 5

4 6 - 6

3 4 - 7

3 5 - 8

1 4 - 10