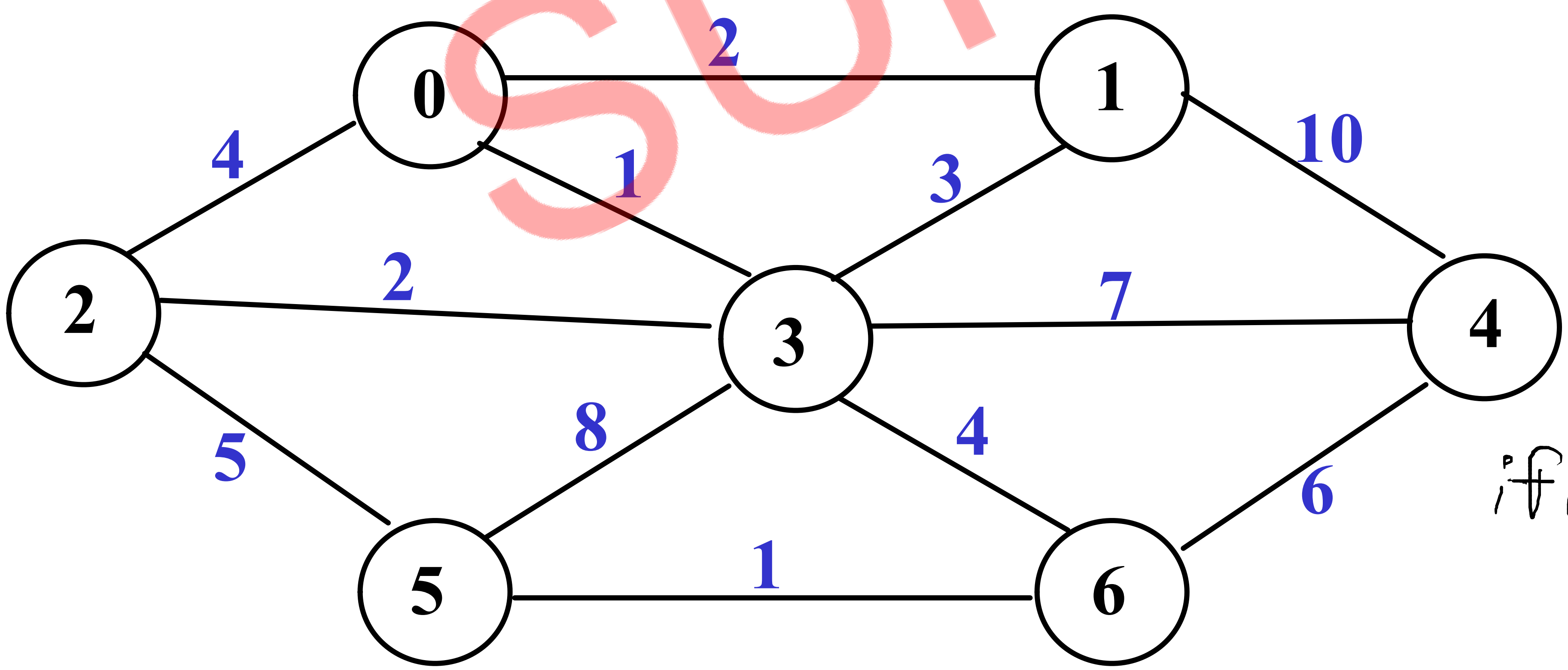
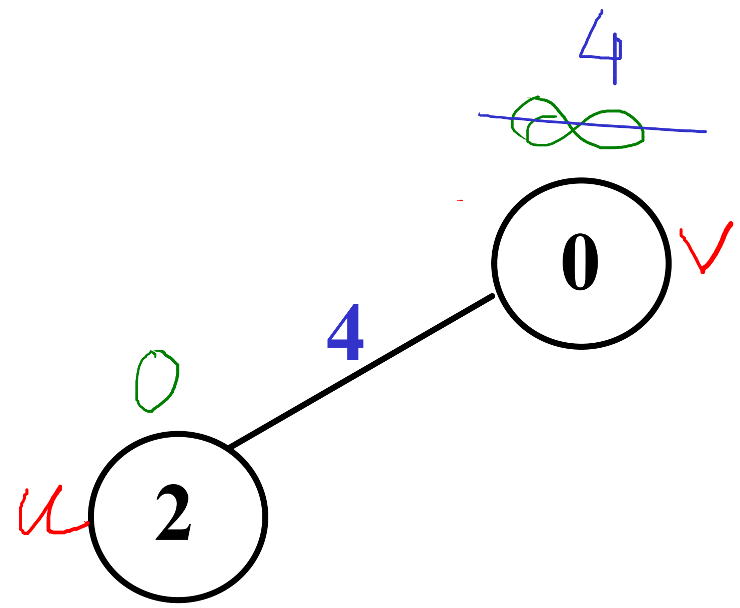
Prim's MST

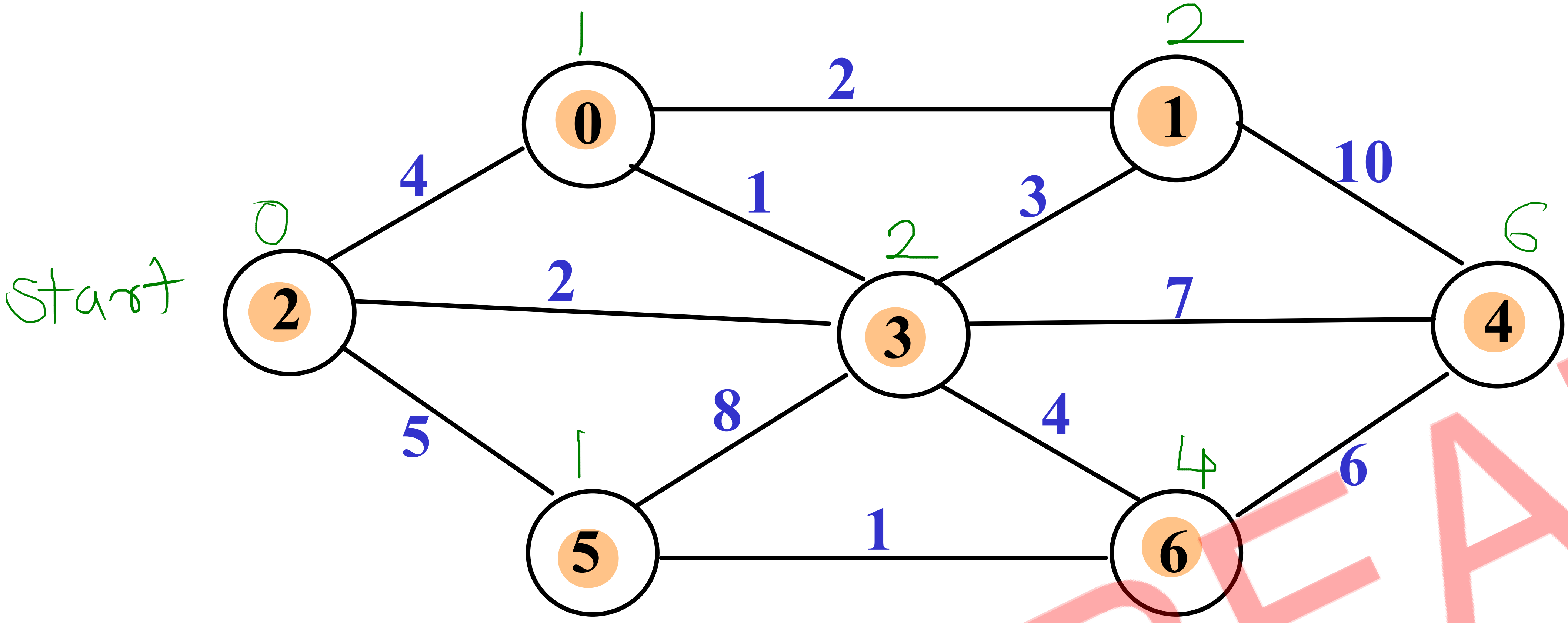
1. Create a set mst to keep track of vertices included in MST.
2. Also keep track of parent of each vertex. Initialize parent of each vertex -1.
3. Assign a key to all vertices in the input graph. Key for all vertices should be initialized to INF. The start vertex key should be 0.
4. While mst doesn't 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.






```
if(adjMat[u][v] < keys[v])
    keys[v] = adjMat[u][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

wt → 16

	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

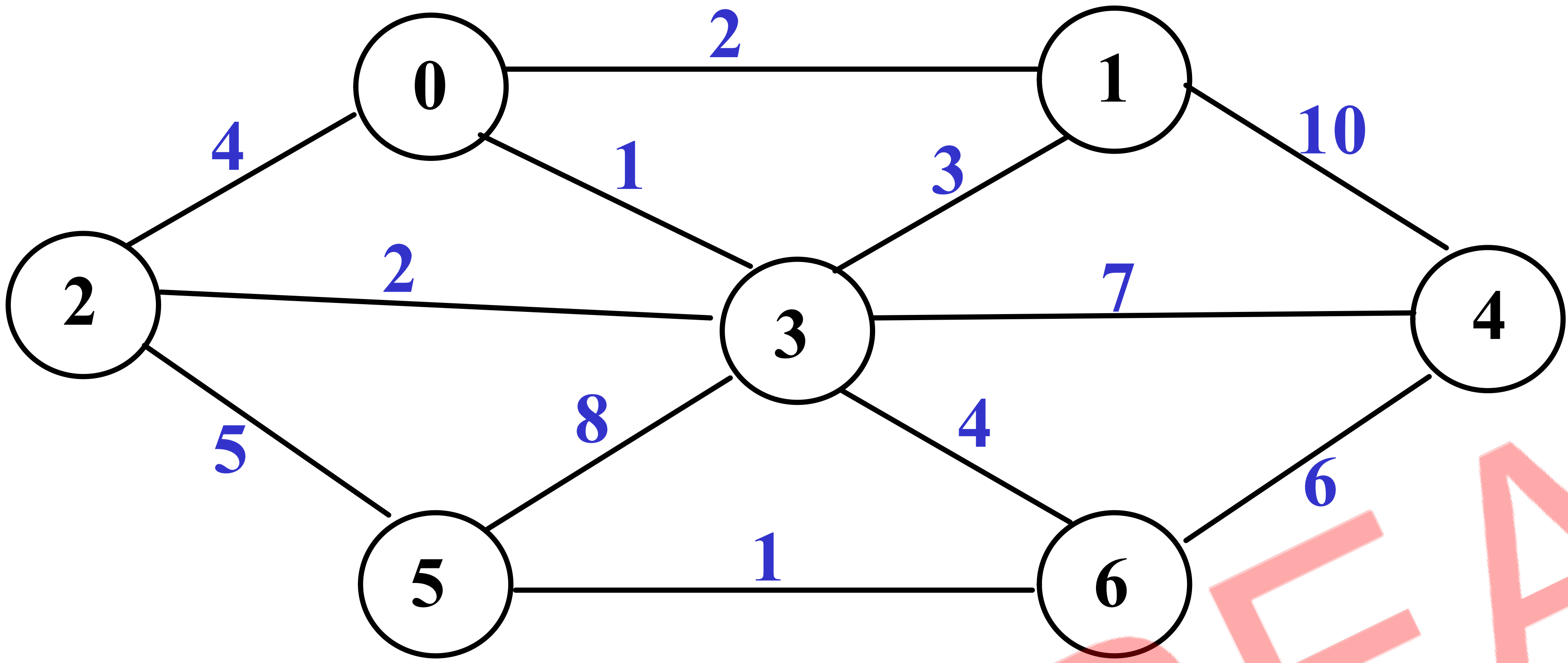
	K	P
0	1	3
1	2	0
2	0	-1
3	2	2
4	7	3
5	5	2
6	4	3

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

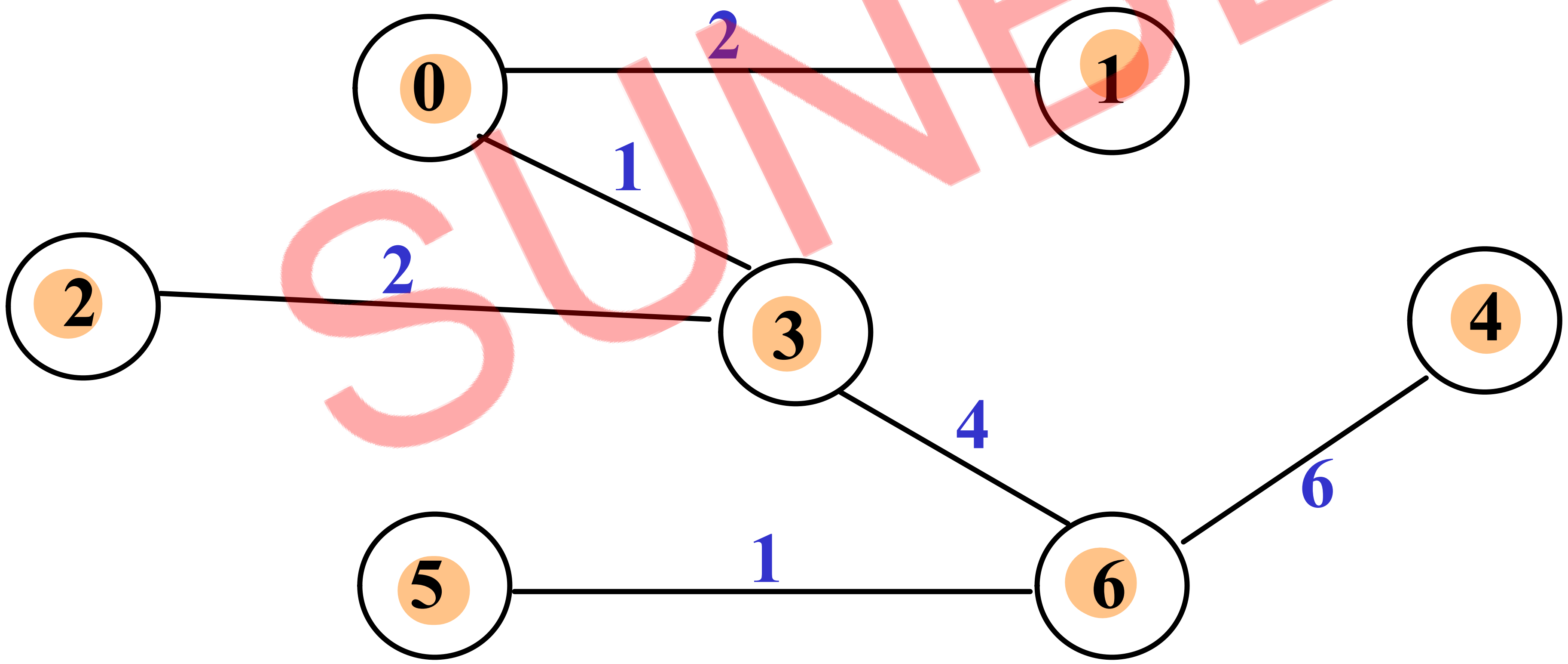
	K	P
0	1	3
1	2	0
2	0	-1
3	2	2
4	6	6
5	1	6
6	4	3

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

Graph



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



Weight =16

```
int findMinKey(){
    int minKey = INF;
    for(int i = 0 ; i < vertexCount ; i++){
        if(keys[i] < minKey){
            minKey = keys[i];
        }
    }
    return minKey;
}
```

	K
0	∞
1	∞
2	0
3	∞
4	∞
5	∞
6	∞

vertexCount=7

minkey	i	keys[i]<minkey
∞	0	F
	1	F
0	2	T
	3	F
	4	F
	5	F
	6	F

```
int findMinKeyVertex(){
    int minKey = INF, minKeyVertex = -1;
    for(int i = 0 ; i < vertexCount ; i++){
        if(!mst[i] && keys[i] < minKey){
            minKey = keys[i];
            minKeyVertex = i;
        }
    }
    return minKeyVertex;
}
```

	K
0	∞
1	∞
2	0
3	∞
4	∞
5	∞
6	∞

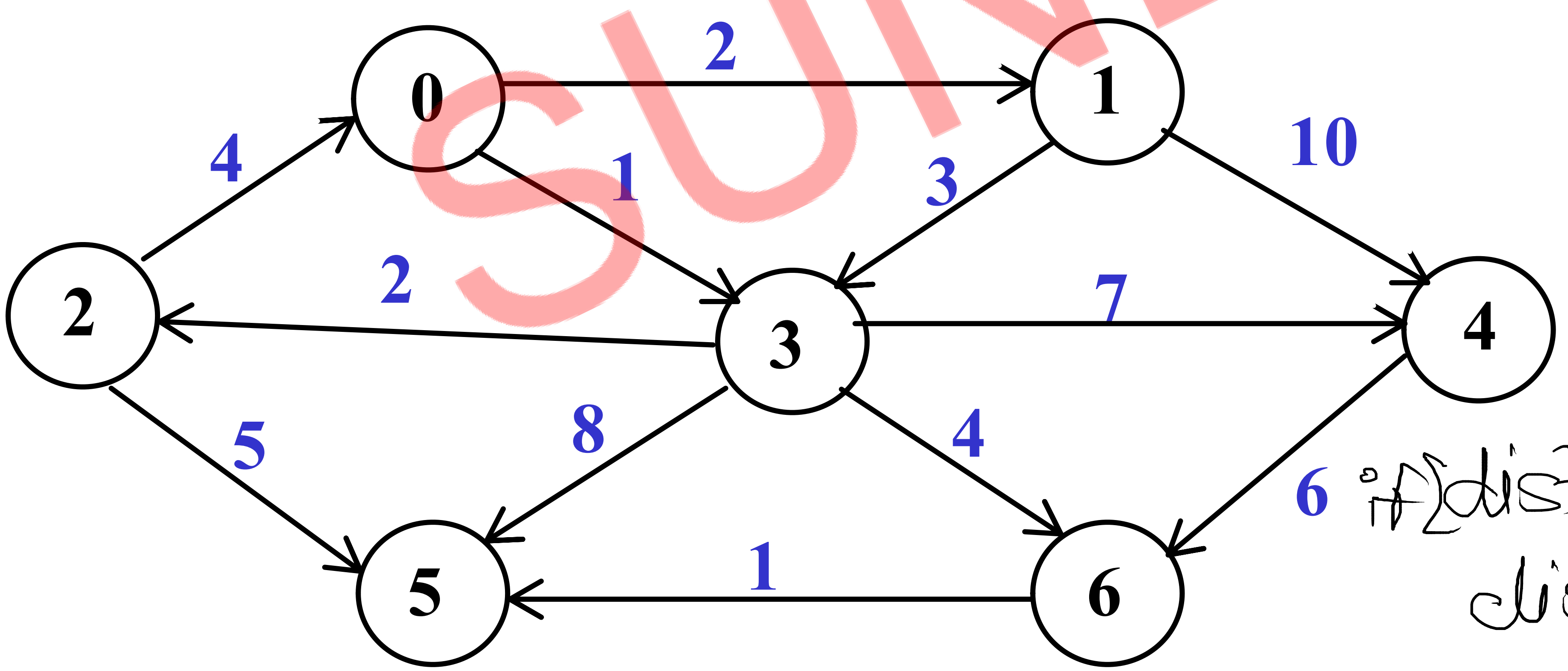
vertexCount=7

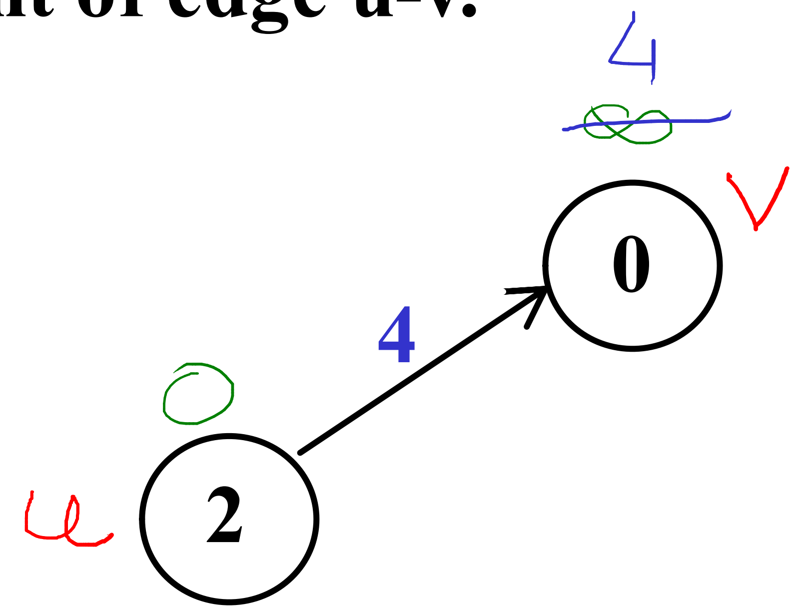
minkey	minkey vertex	i	keys[i]<minkey
∞	-1	0	F
		1	F
0	2	2	T
		3	F
		4	F
		5	F
		6	F

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.**

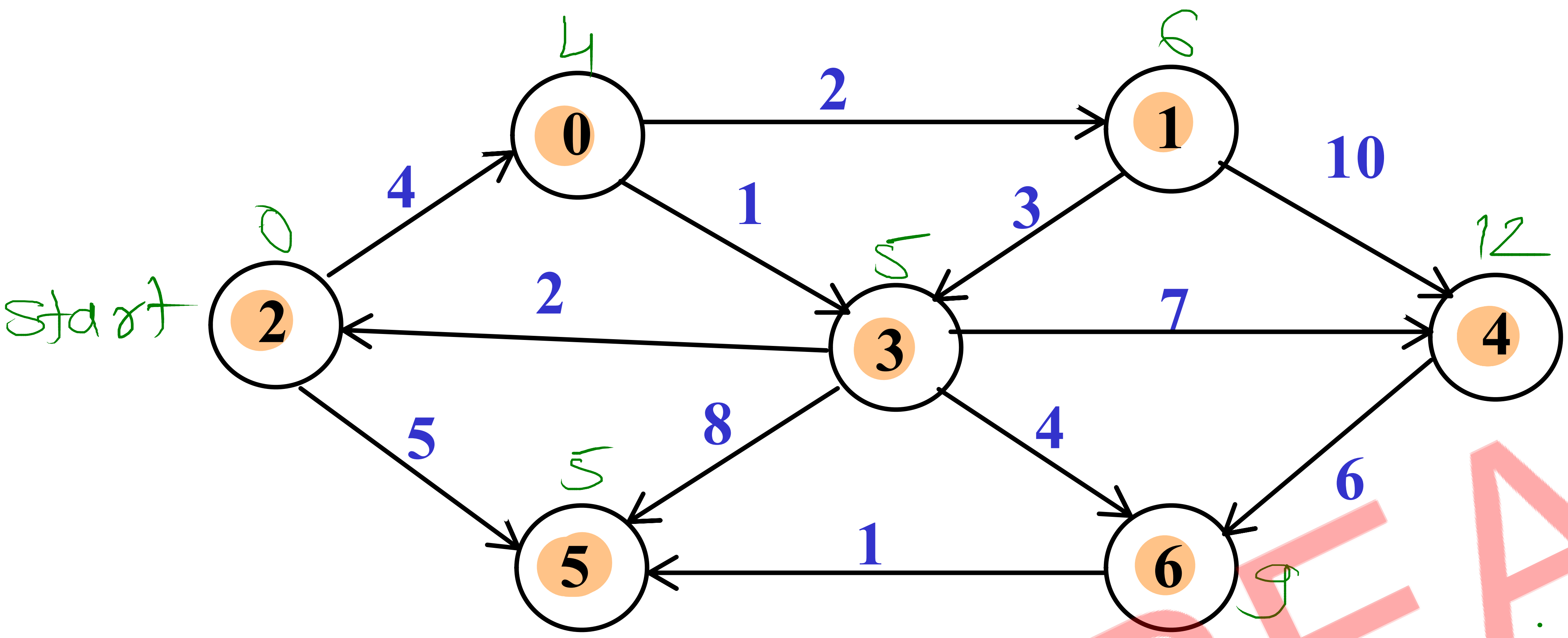
For each adjacent vertex v,
- **if distance of u + weight of edge u-v is less than the current distance of v, then update its distance as distance of u + weight of edge u-v.**





if(dist[u] + adjMat[u][v] < dist[v])
dist[v] = dist[u] + adjMat[u][v];

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

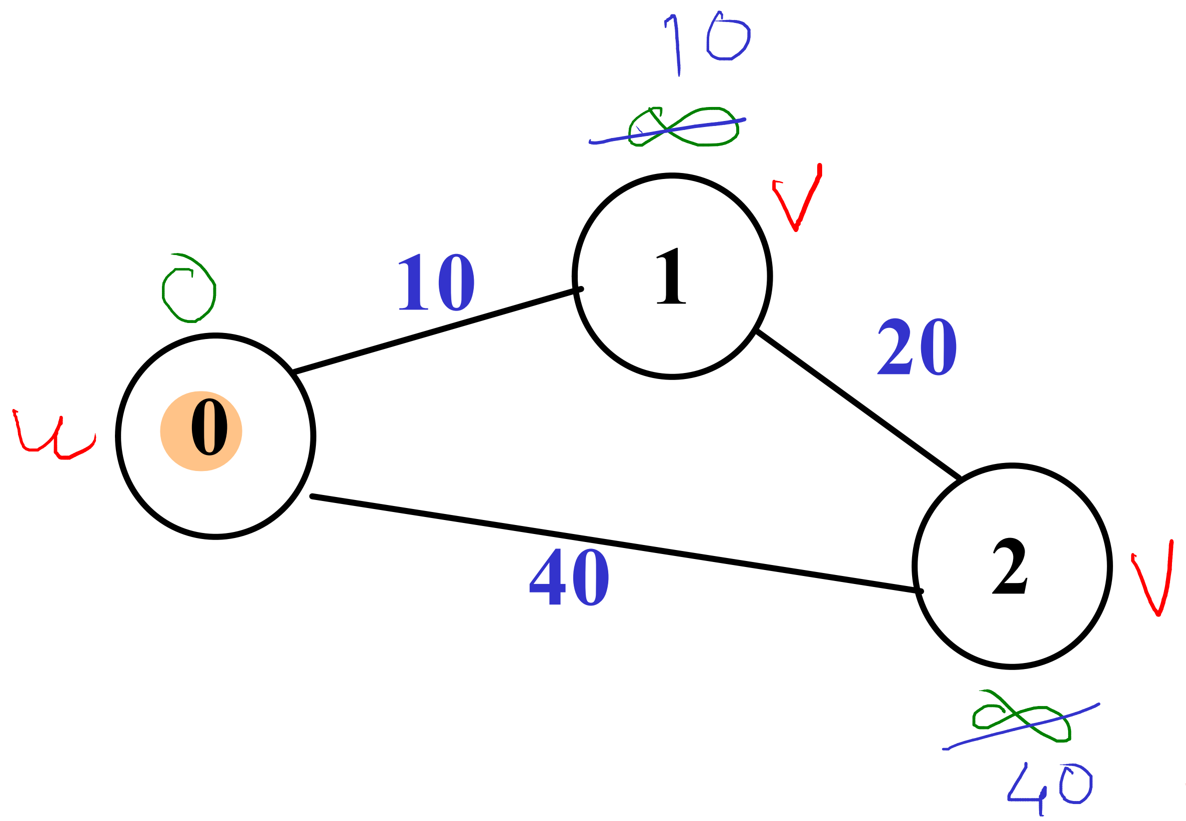
	D	P
0	4	2
1	6	0
2	0	-1
3	5	0
4	12	3
5	5	2
6	9	3

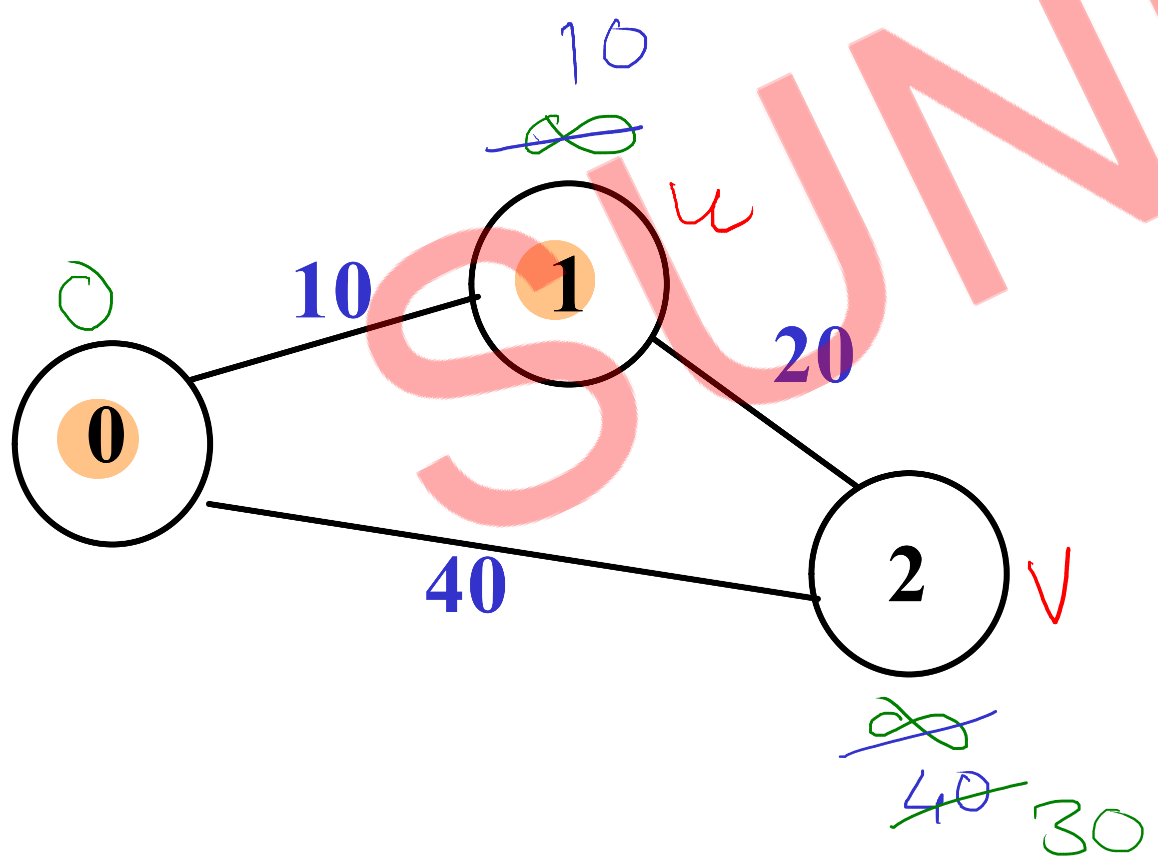
	D	P
0	4	2
1	6	0
2	0	-1
3	5	0
4	12	3
5	5	2
6	9	3

	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

Relaxation





if (dist[u] + adjMat[u][v] < dist[v])

dist[v] = dist[u] + adjMat[u][v]

dist[0] + wt[0][1] < dist[1]

0 + 10 < ∞

Algorithm Design/Problem solving technique: Divide and Conquor

- Bigger problem is divided into sub problems
- All sub problems are solved individually
- All solutions of sub problems are conquored(merged) to get final solution

- generally recursion is used to implement

e.g. Merge sort and Quick sort

pune → delhi

road   train   plane ← solutions

min fare → train ← optimal sol^n

min time → plane ← optimal sol^n

Problem solving technique: Greedy approach

- A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.
- We can make choice that seems best at the moment and then solve the sub-problems that arise later.
- The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the sub-problem.
- It iteratively makes one greedy choice after another, reducing each given problem into a smaller one.
- A greedy algorithm never reconsiders its choices.
- A greedy strategy may not always produce an optimal solution.

eg. Greedy algorithm decides minimum number of coins to give while making change.

coins available : 50, 20, 10, 5, 2, 1

36 → 20

16 → 20 10

6 → 20 10 5

1 → 20 10 5 1

e.g. Prim's MST

Dijkstra's SPT

Kruskal's MST