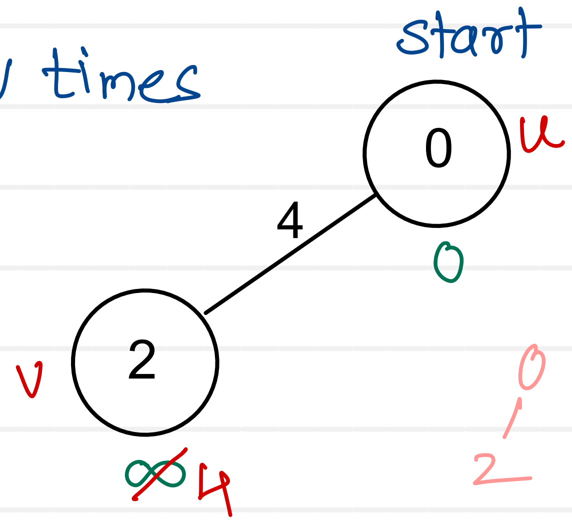
# Prim’s Algorithm

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 ← V times
    - i. Pick a vertex u which is not there in mst and has minimum key. ← V times
    - ii. Include vertex u to mst.
    - iii. Update key and parent of all adjacent vertices of u. ← V times
        - 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.

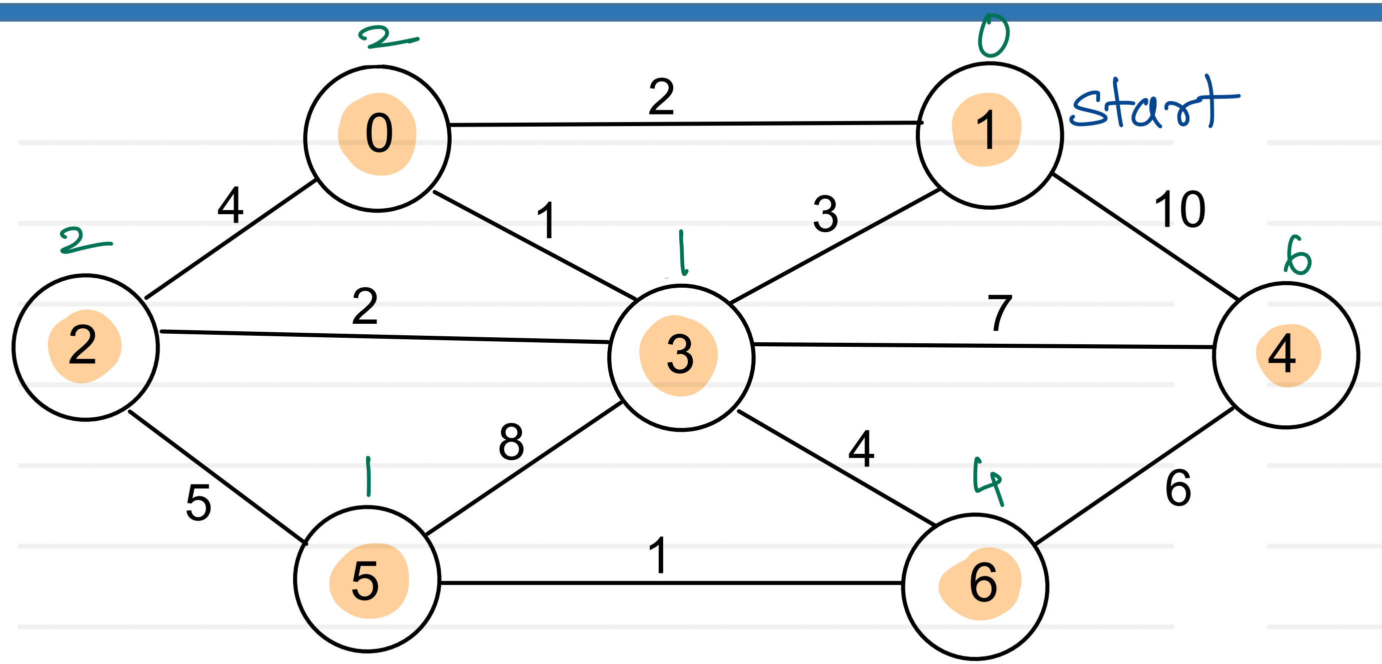



Time ∝ V * 2V

$\boxed{T(n) = O(V^2)}$ $\boxed{S(V) = O(V)}$

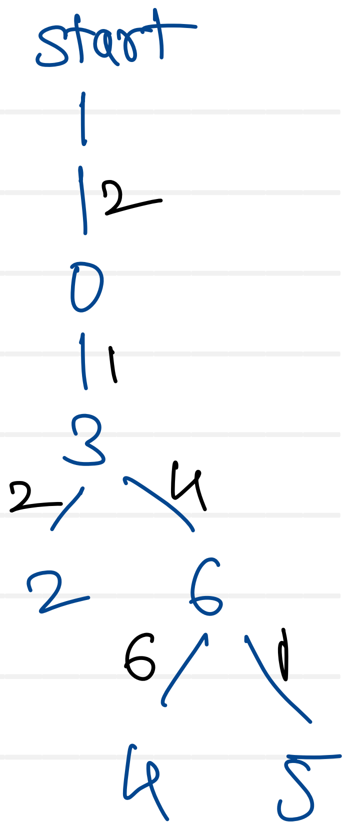
```
if(adjMat[u][v] < key[v])
    key[v] = adjMat[u][v];
```

# Prim's Algorithm




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

Wt = 16




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

| | K | P |
|---|---|---|
| 0 | 2 | 1 |
| 1 | 0 | -1 |
| 2 | ∞ | -1 |
| 3 | 3 | 1 |
| 4 | 10 | 1 |
| 5 | ∞ | -1 |
| 6 | ∞ | -1 |

| | K | P |
|---|---|---|
| 0 | 2 | 1 |
| 1 | 0 | -1 |
| 2 | 4 | 0 |
| 3 | 1 | 0 |
| 4 | 10 | 1 |
| 5 | ∞ | -1 |
| 6 | ∞ | -1 |

| | K | P |
|---|---|---|
| 0 | 2 | 1 |
| 1 | 0 | -1 |
| 2 | 2 | 3 |
| 3 | 1 | 0 |
| 4 | 7 | 3 |
| 5 | 8 | 3 |
| 6 | 4 | 3 |

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

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

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

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Key | 2 | 0 | ∞ | 3 | 10 | ∞ | ∞ |

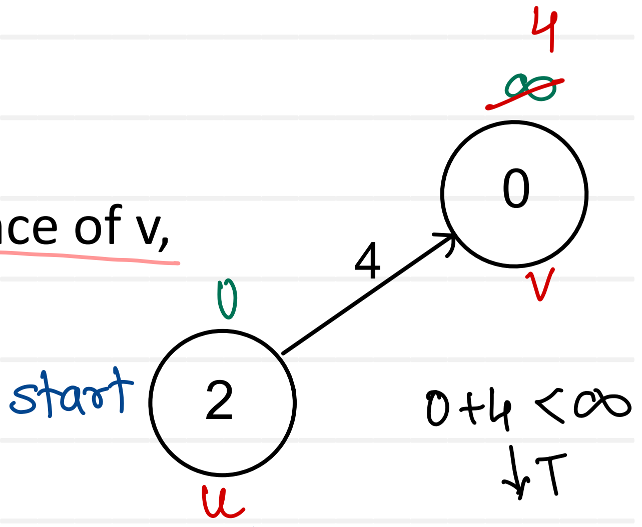
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| mst | F | T | F | F | F | F | F |

| minkey | minkeyVertex | i | condition |
|---|---|---|---|
| ∞ | -1 | 0 | T |
| 2 | 0 | 1 | F |
| | | 2 | F |
| | | 3 | F |
| | | 4 | F |
| | | 5 | F |
| | | 6 | F |
| | | ~~7~~ | |

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

# Dijkstra’s Algorithm

1. Create a set spt to keep track of vertices included in shortest path tree. (SPT)
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.

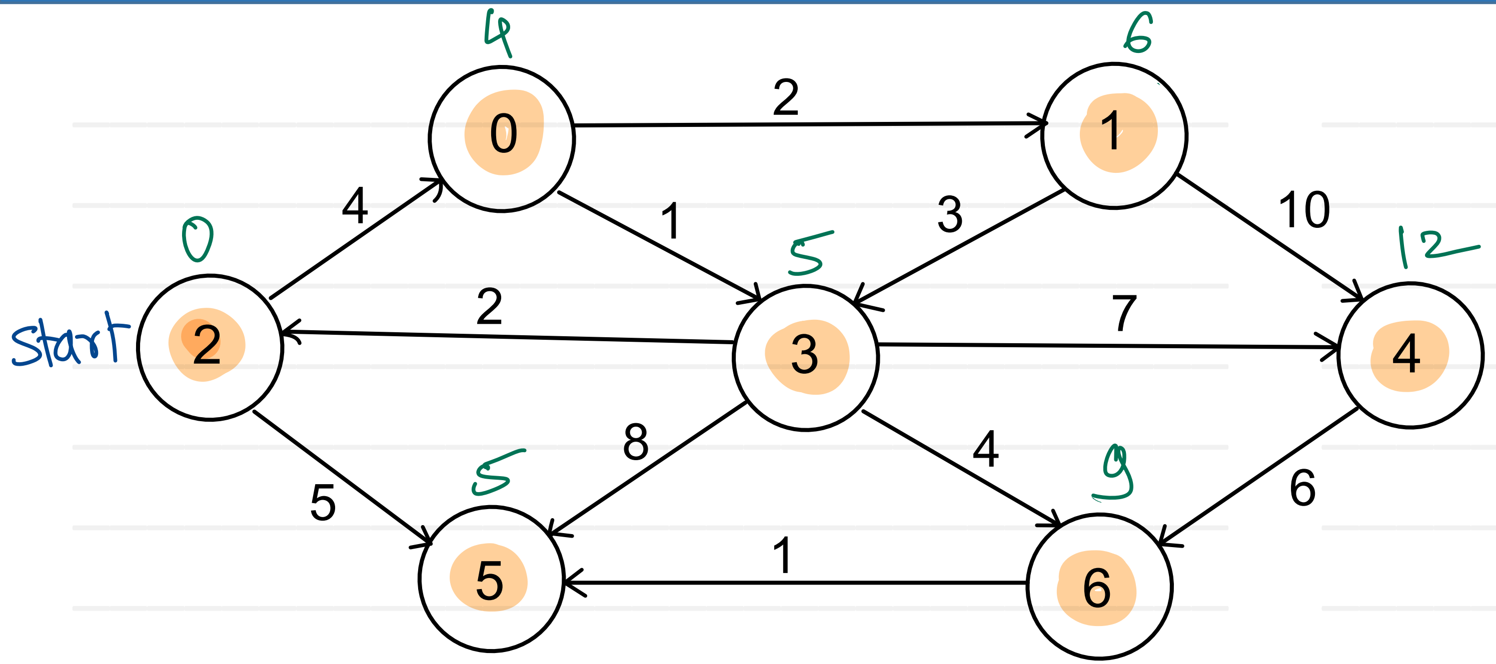



$$\boxed{T(V) = O(V^2)}$$

$$\boxed{S(V) = O(V)}$$

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

# Dijkstra’s Algorithm




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

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

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

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

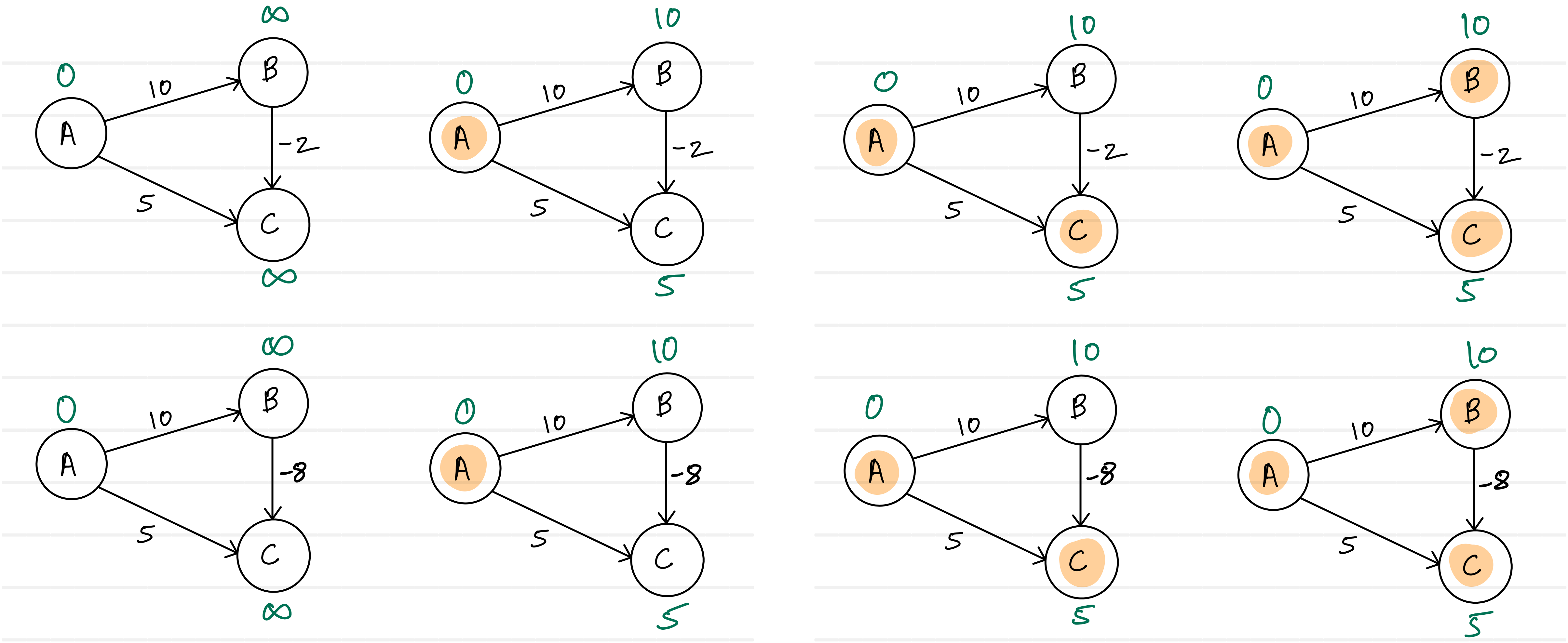
| | D |
|---|---|
| 0 | 4 |
| 1 | 6 |
| 2 | 0 |
| 3 | 5 |
| 4 | 12 |
| 5 | 5 |
| 6 | 9 |

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

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

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

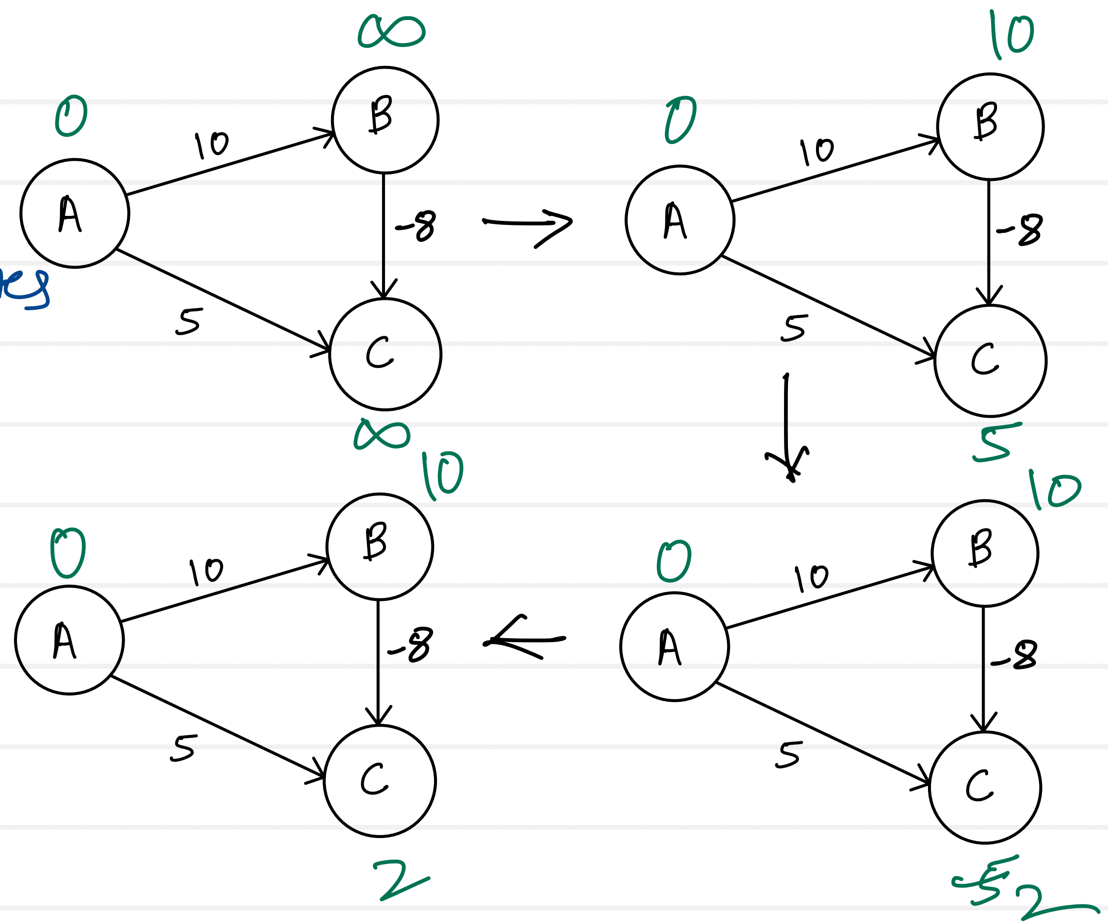
# Dijkstra's Algorithm




if graph contains negative edges, Dijkstra's algo may fail

# 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: ≈ V-1 times
   For each edge u-v, → E times
   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 cycle in the graph: -1 time
   For each edge u-v, → E times
   if dist[v] > dist[u] + weight of edge (u,v),
   then graph has -ve weight cycle.



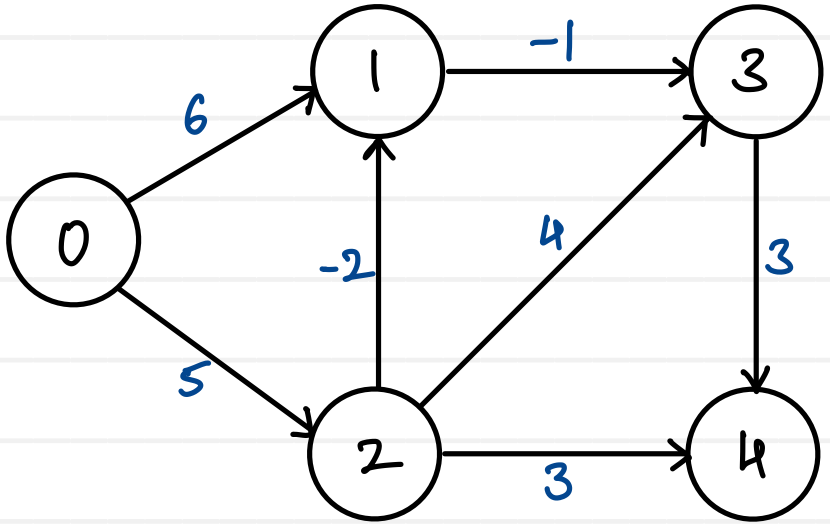

$\text{Time} \propto (V-1)E + E$

$\text{Time} \propto VE - \cancel{E} + \cancel{E}$

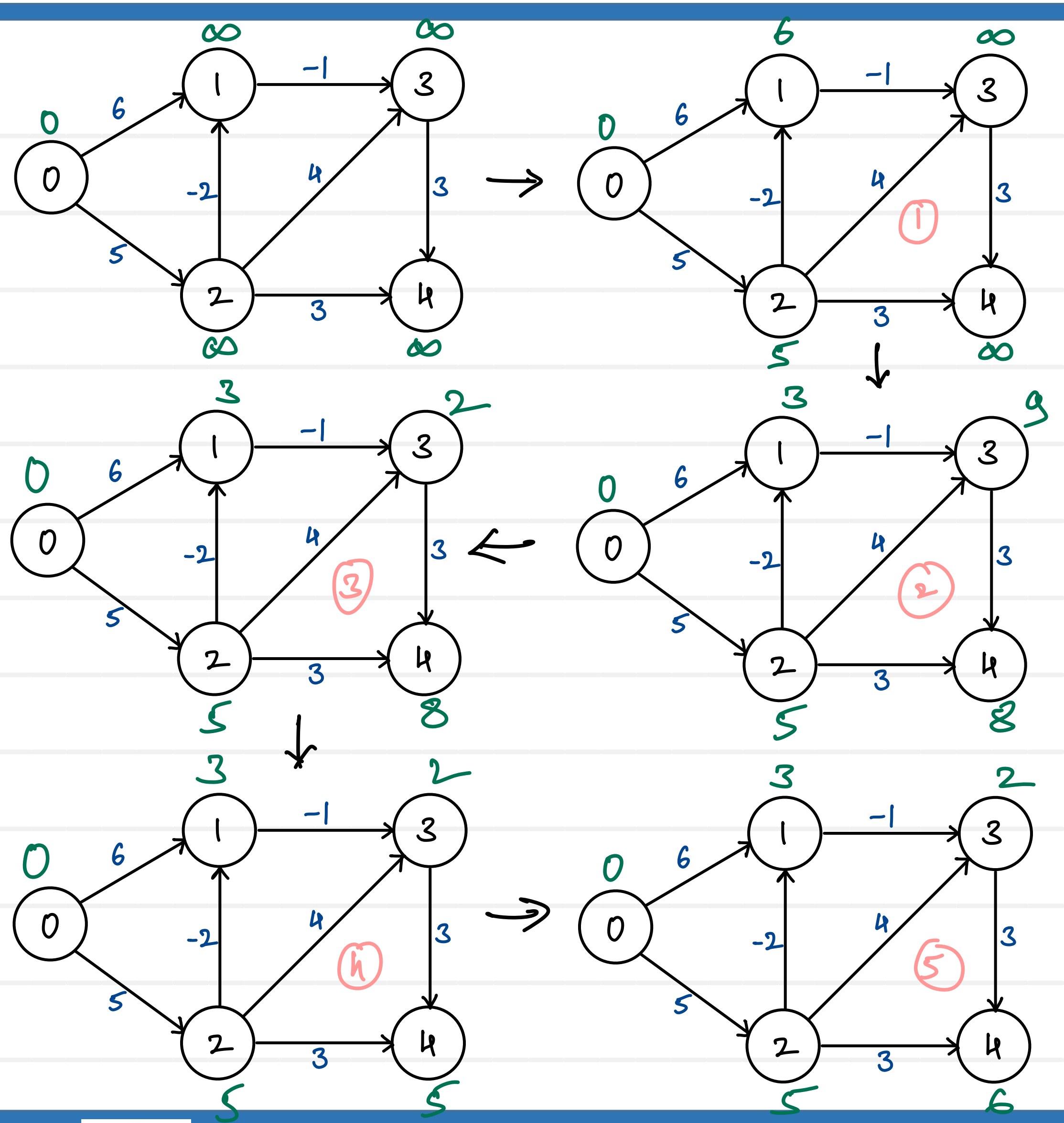
$\boxed{T(V,E) = O(VE)}$

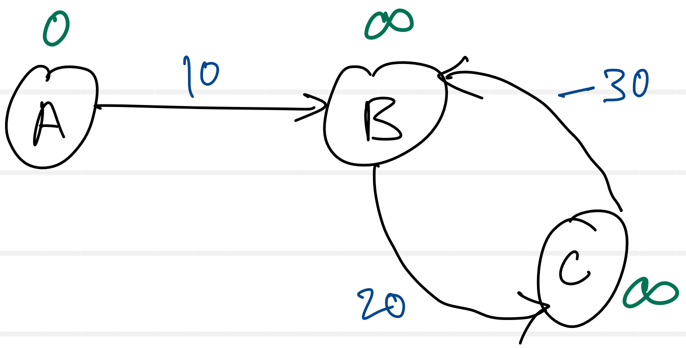
$\boxed{S(V) = O(V)}$

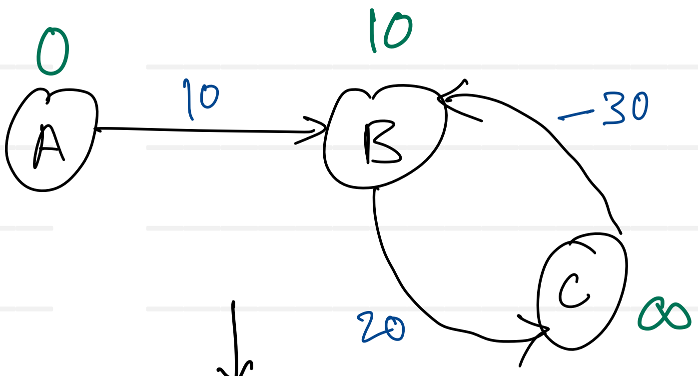
# Bellaman Ford Algorithm



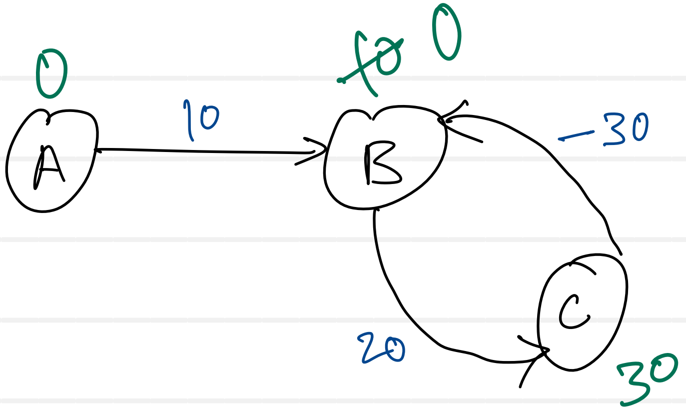
| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| Pass 1 | 0 | 6 | 5 | ∞ | ∞ |
| Pass 2 | 0 | 3 | 5 | 9 | 8 |
| Pass 3 | 0 | 3 | 5 | 2 | 8 |
| Pass 4 | 0 | 3 | 5 | 2 | 5 |

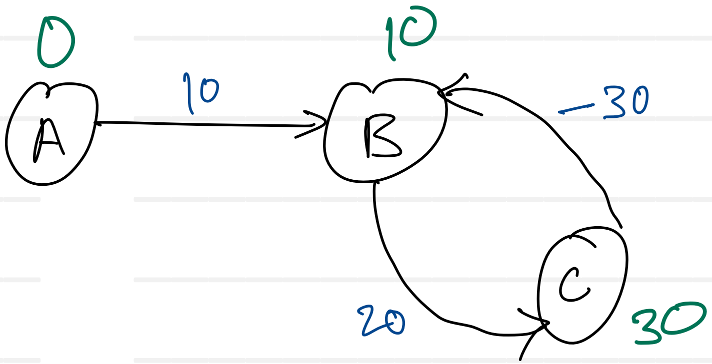





-ve edge cycle : addition of wts of edges in cycle is -ve




if graph contains -ve edge cycle, Bellaman Ford fail

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com