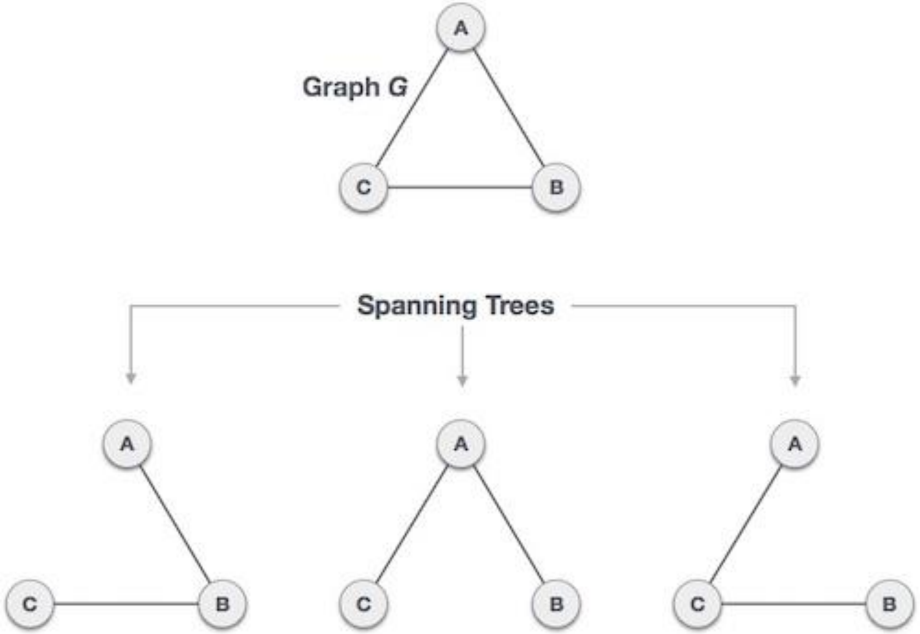
Summary II

SWE2016-44

Minimum Spanning Tree

Definitions

1. **Spanning Tree**
   - **Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree connects all the vertices together.**



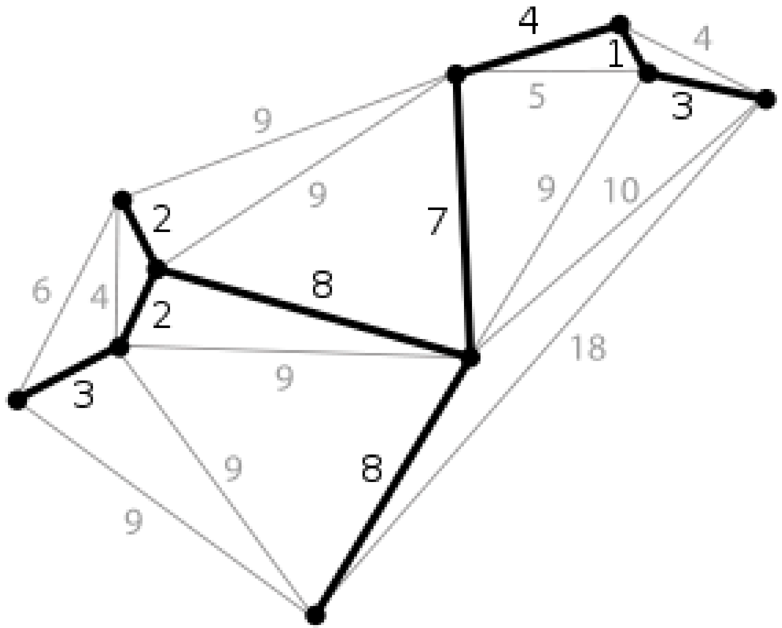
Definitions

1. **Spanning Tree**
   - **Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree connects all the vertices together.**

   - **Properties**
     - **The spanning tree does not have any cycle**
     - **Spanning tree has n-1 edges, where n is the number of vertices**
     - **From a complete graph, by removing maximum e - n + 1 edges, we can construct a spanning tree.**

Definitions

2. **Minimum Spanning Tree (MST)**
   - **The spanning tree of the graph whose sum of weights of edges is minimum.**
     - ***A graph may have more than 1 minimum spanning tree.***



Definitions

2. **Minimum Spanning Tree (MST)**

    - **The spanning tree of the graph whose sum of weights of edges is minimum.**
        - ***A graph may have more than 1 minimum spanning tree.***

    - **Two most important MST**
        - **Kruskal's Algorithm**
        - **Prim's Algorithm**

        ※ **Both are greedy algorithms.**

Kruskal’s Algorithm

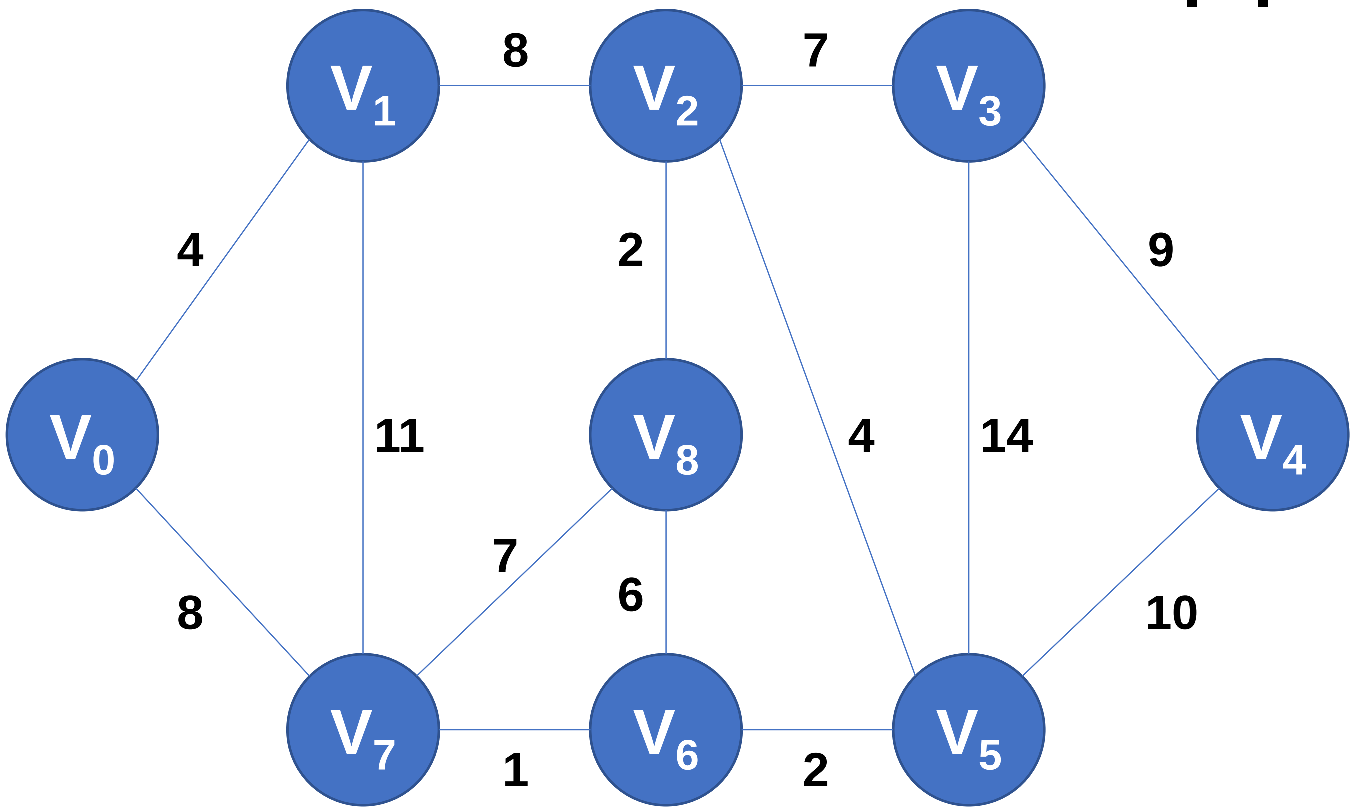
1. Sort all the edges in non-decreasing order of their weight.

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.

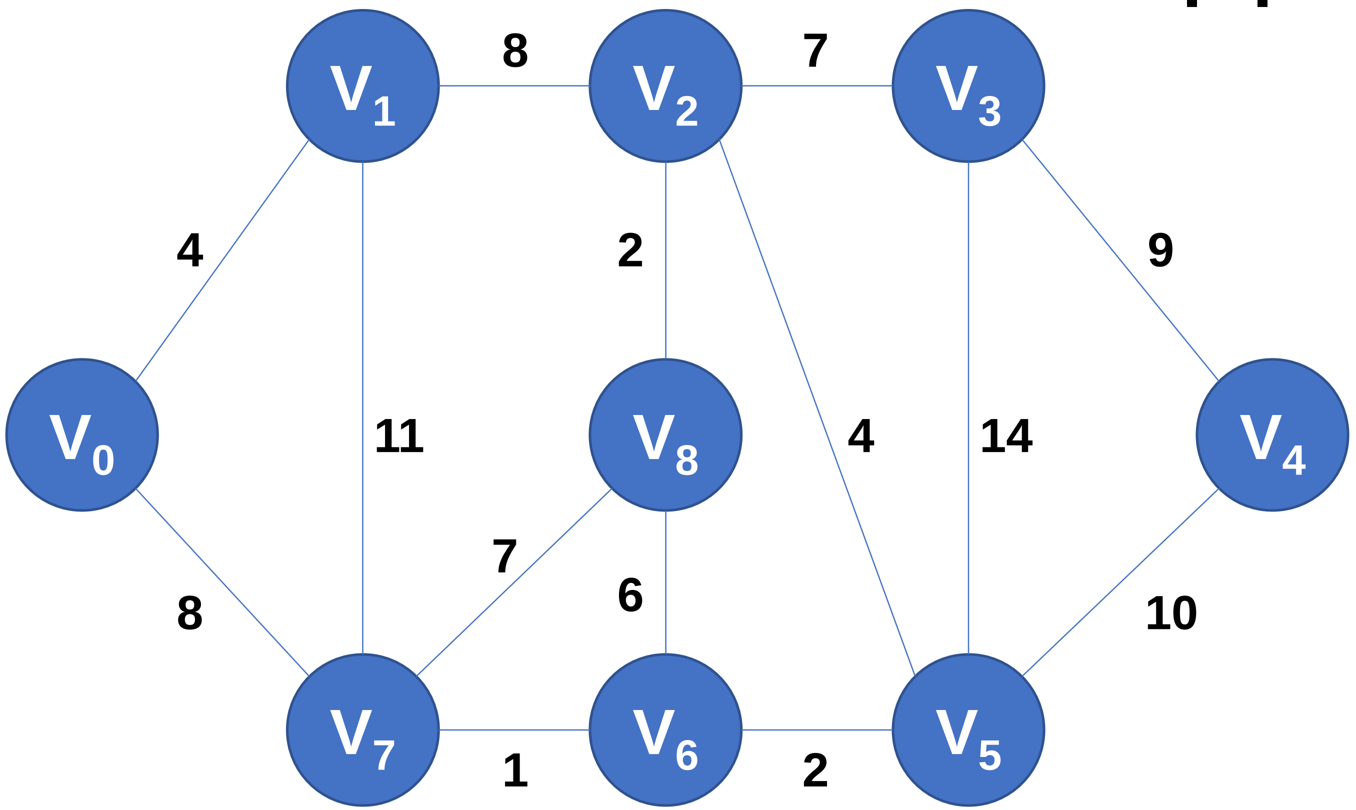
Examples

|V|=9



1. **Sort all the edges in non-decreasing 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.**

Examples

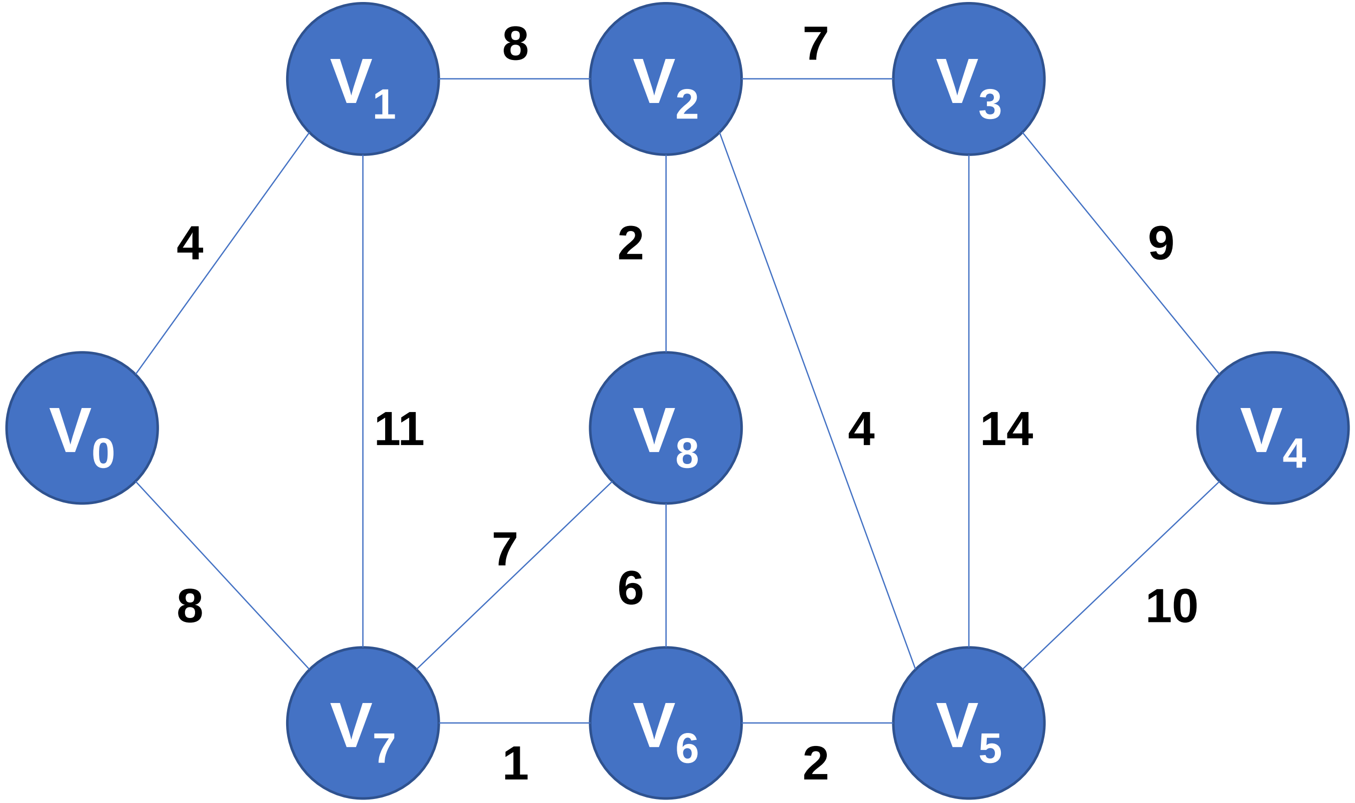


1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	V_8,V_6	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9

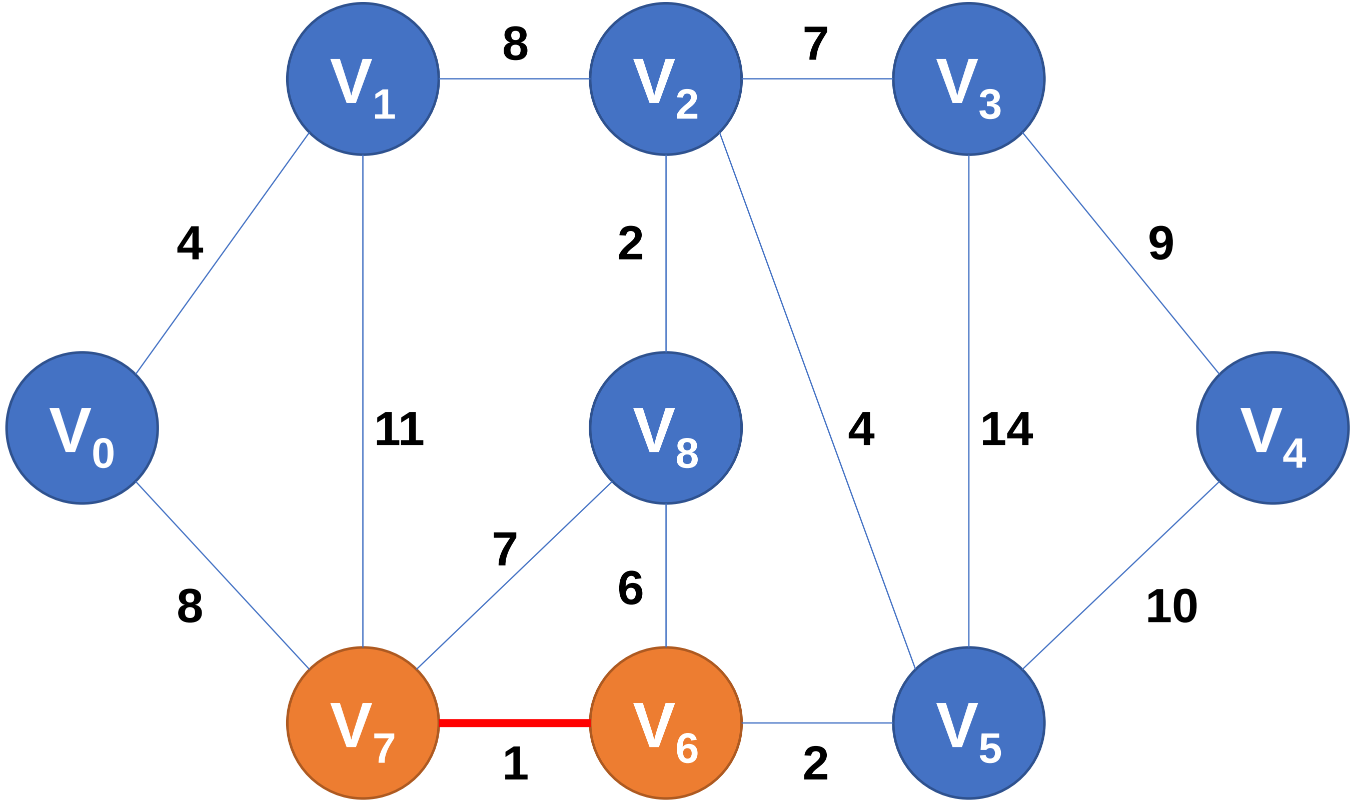


1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	V_8,V_6	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



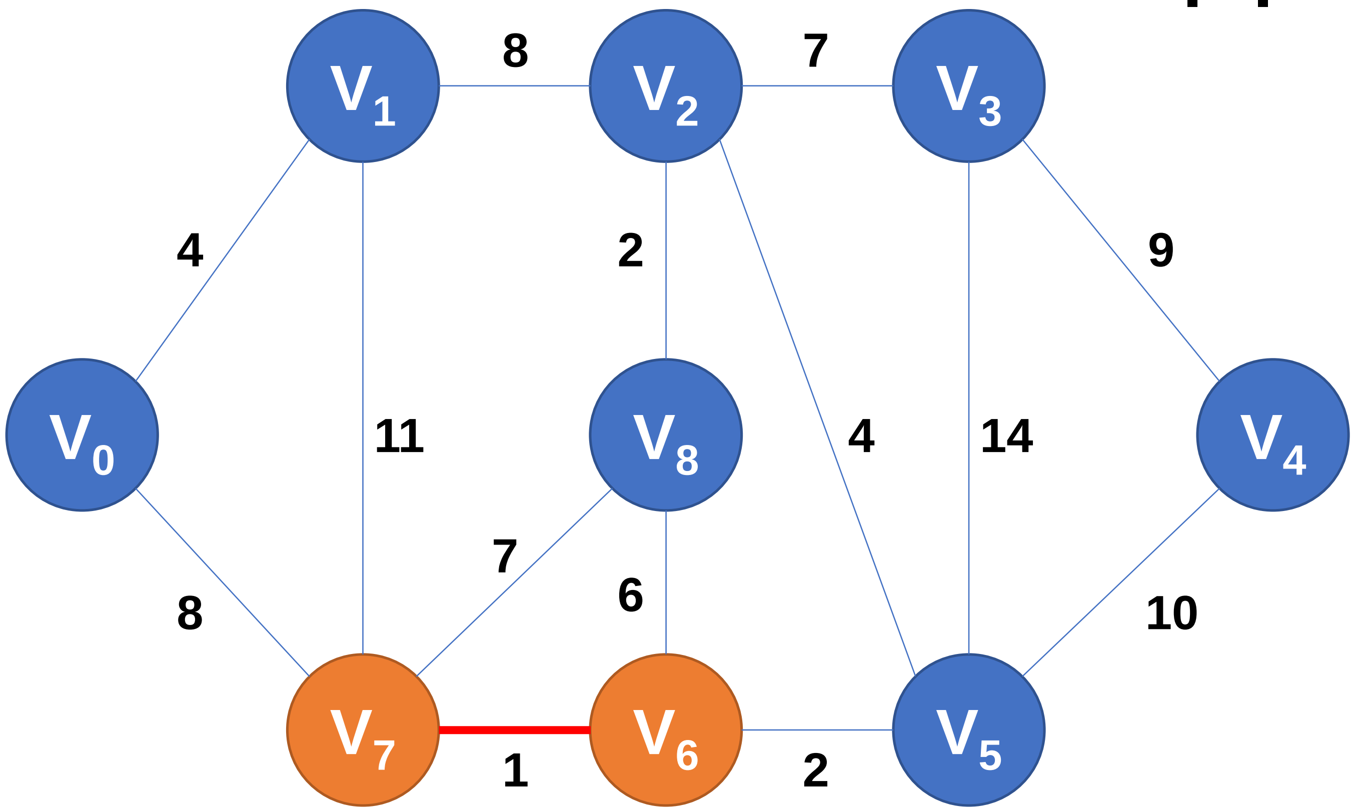
# of Edges in MST = 1

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	V_8,V_6	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



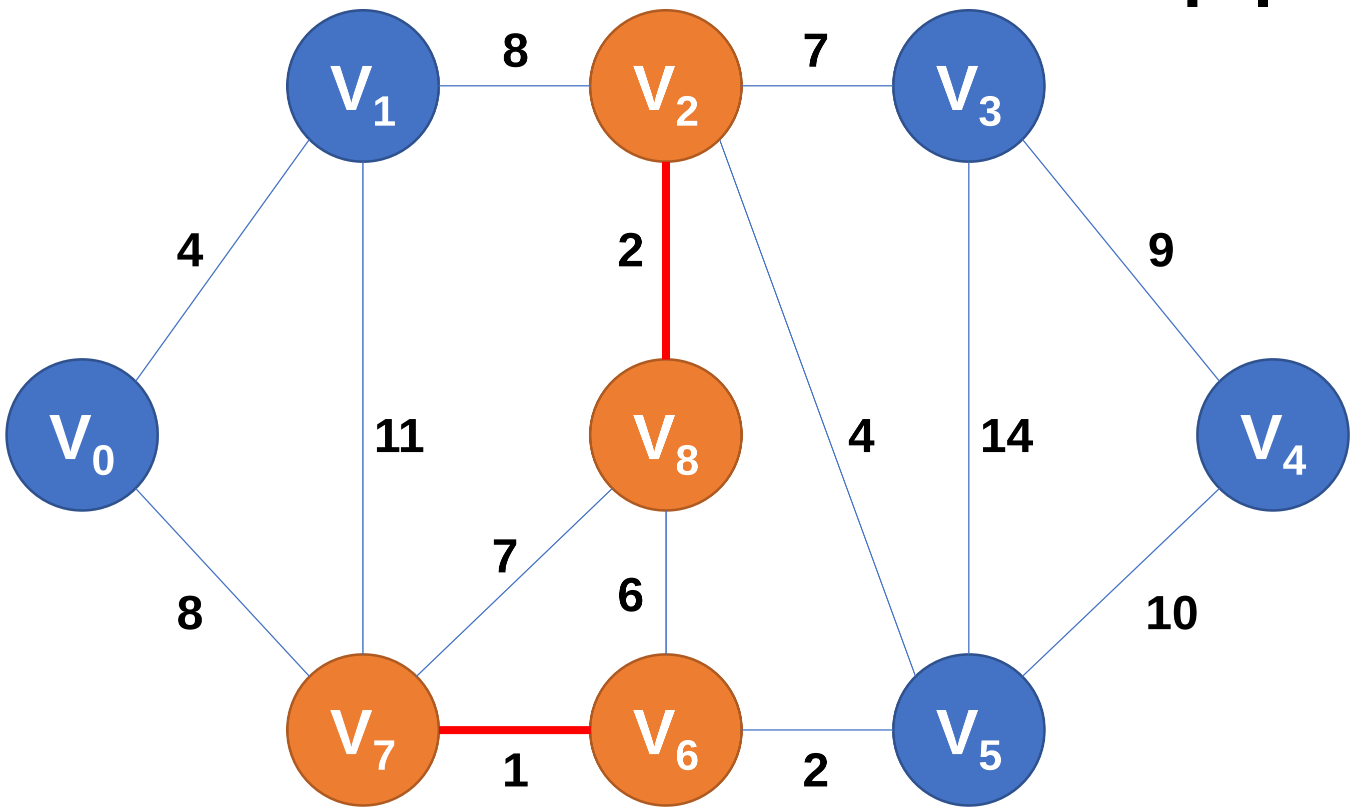
# of Edges in MST = 1

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	V_8,V_6	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

$|V|=9$



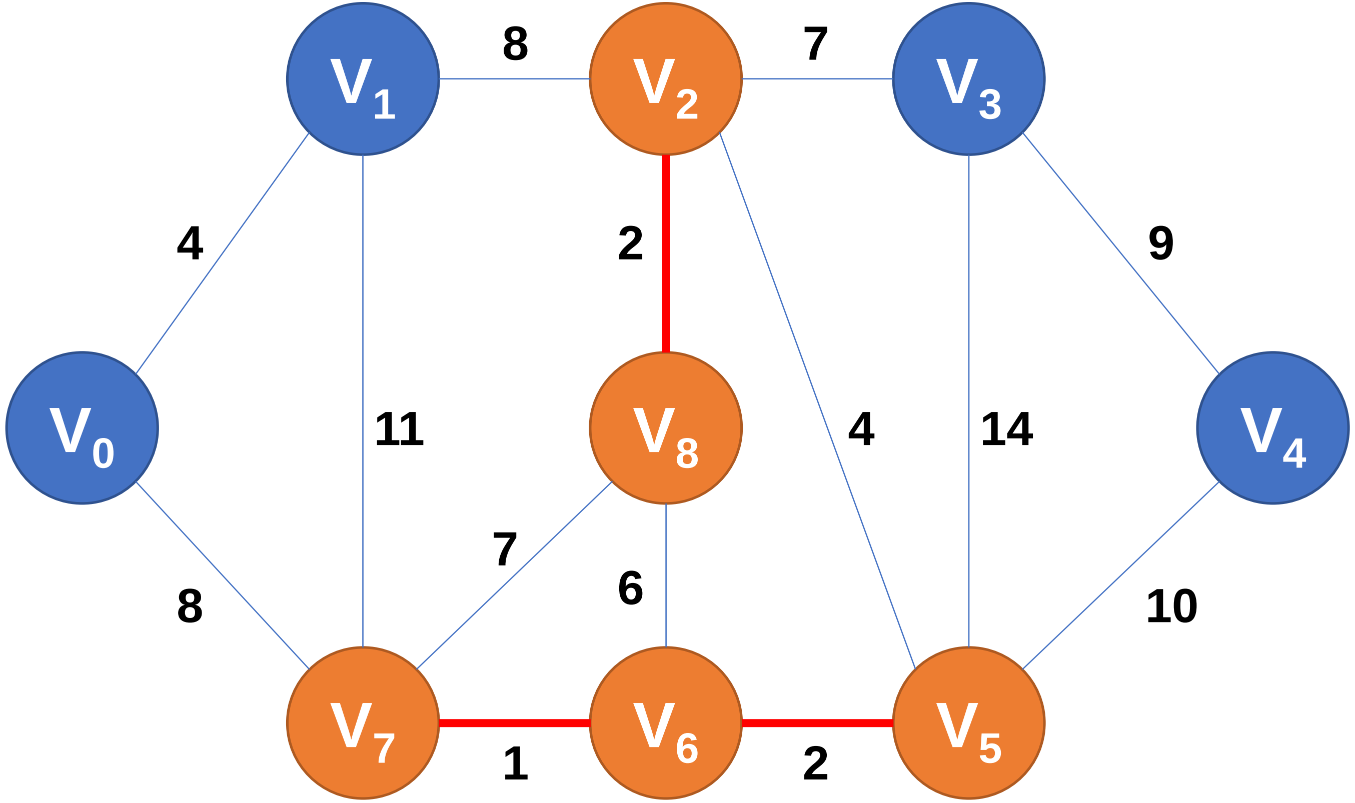
# of Edges in MST = 2

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	V_8,V_6	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



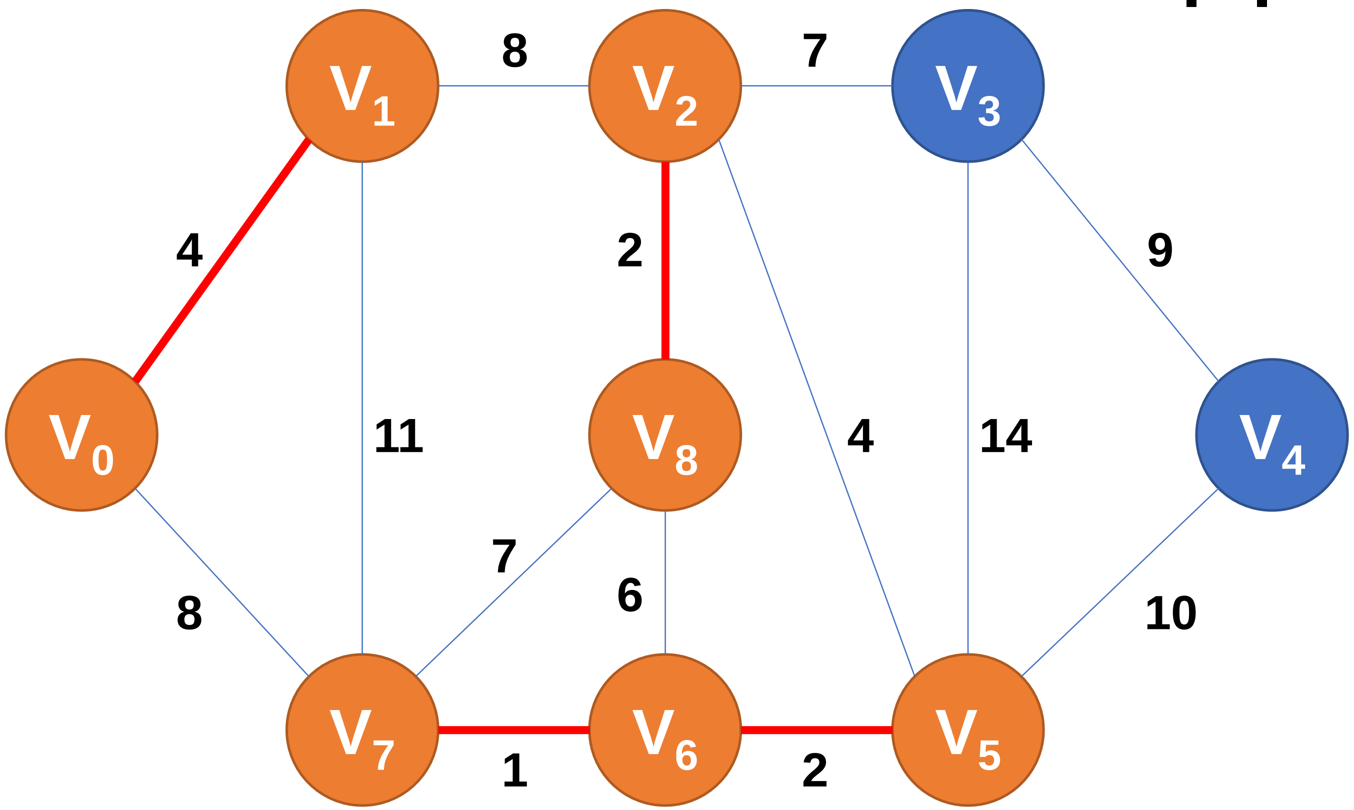
# of Edges in MST = 3

1. **Sort all the edges in non-decreasing 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.**

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	V_8,V_6	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



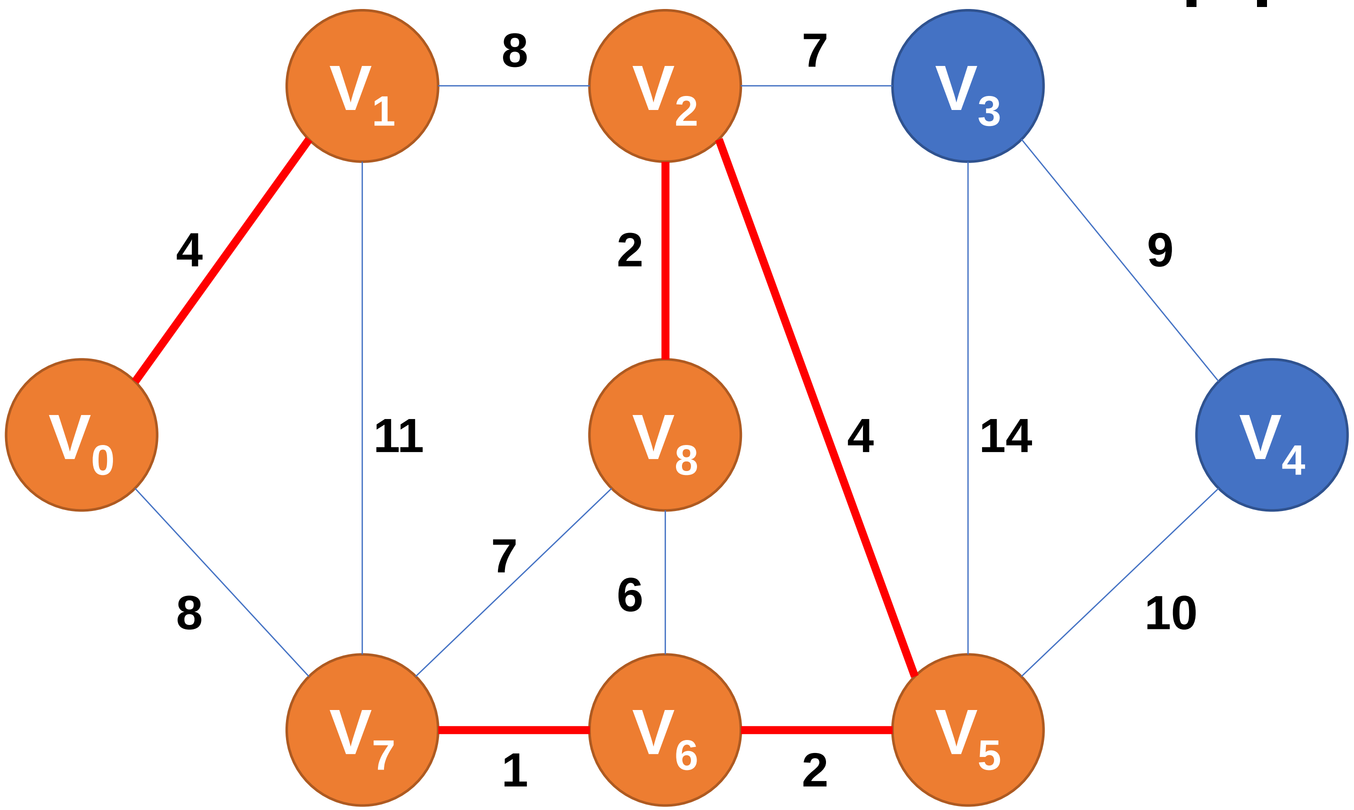
# of Edges in MST = 4

1. **Sort all the edges in non-decreasing 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.**

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	V_8,V_6	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



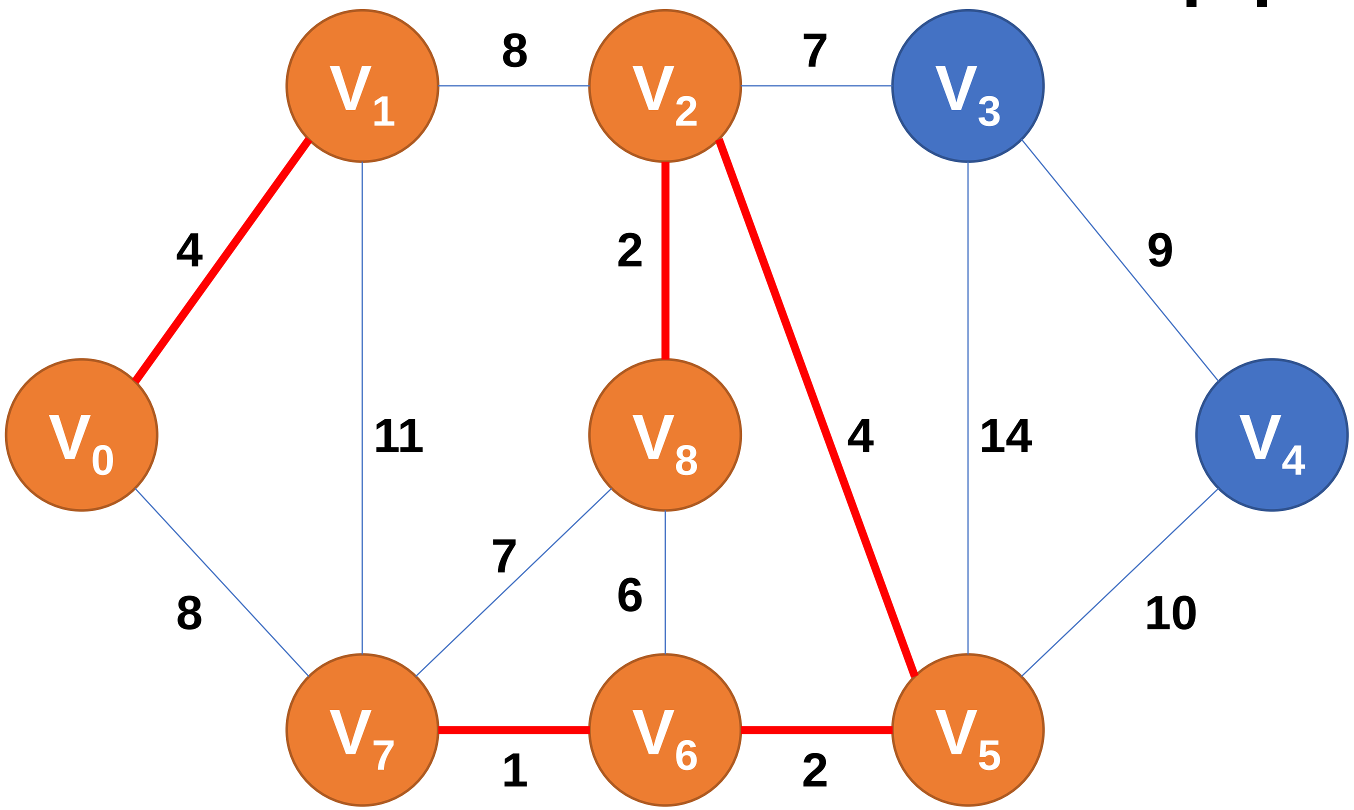
# of Edges in MST = 5

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	V_8,V_6	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



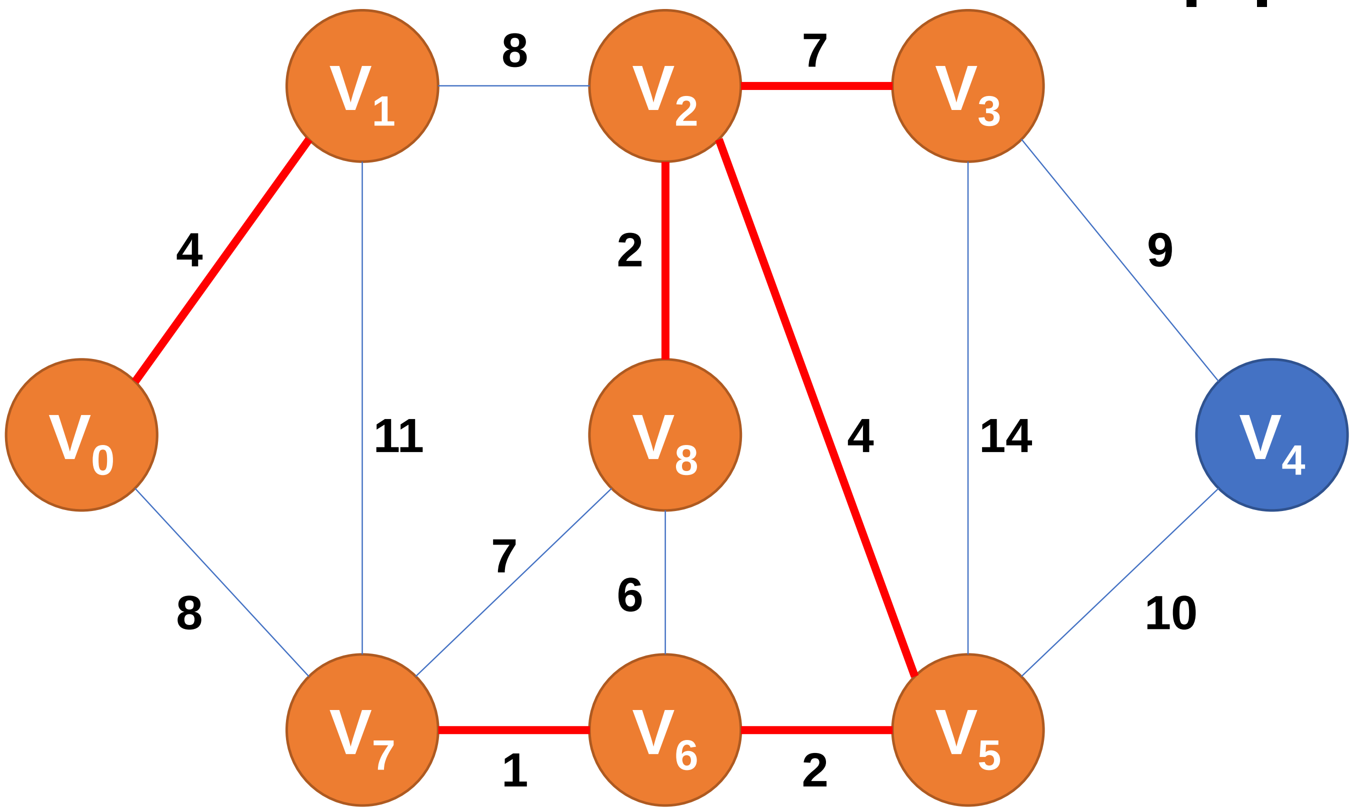
# of Edges in MST = 5

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	~~V_8,V_6~~	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	~~6~~	7	7	8	8	9	10	11	14

Examples

|V|=9



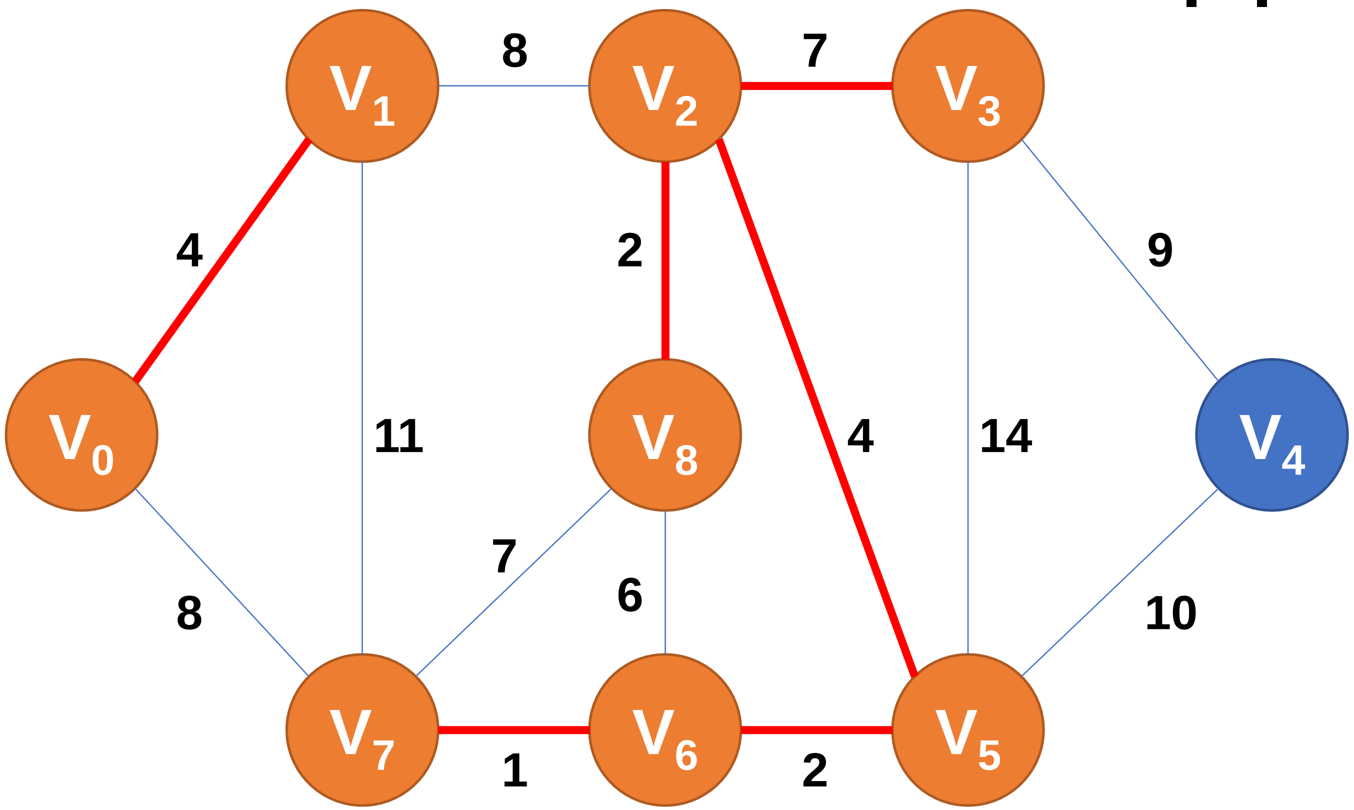
# of Edges in MST = 6

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	~~V_8,V_6~~	V_2,V_3	V_7,V_8	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	~~6~~	7	7	8	8	9	10	11	14

Examples

|V|=9



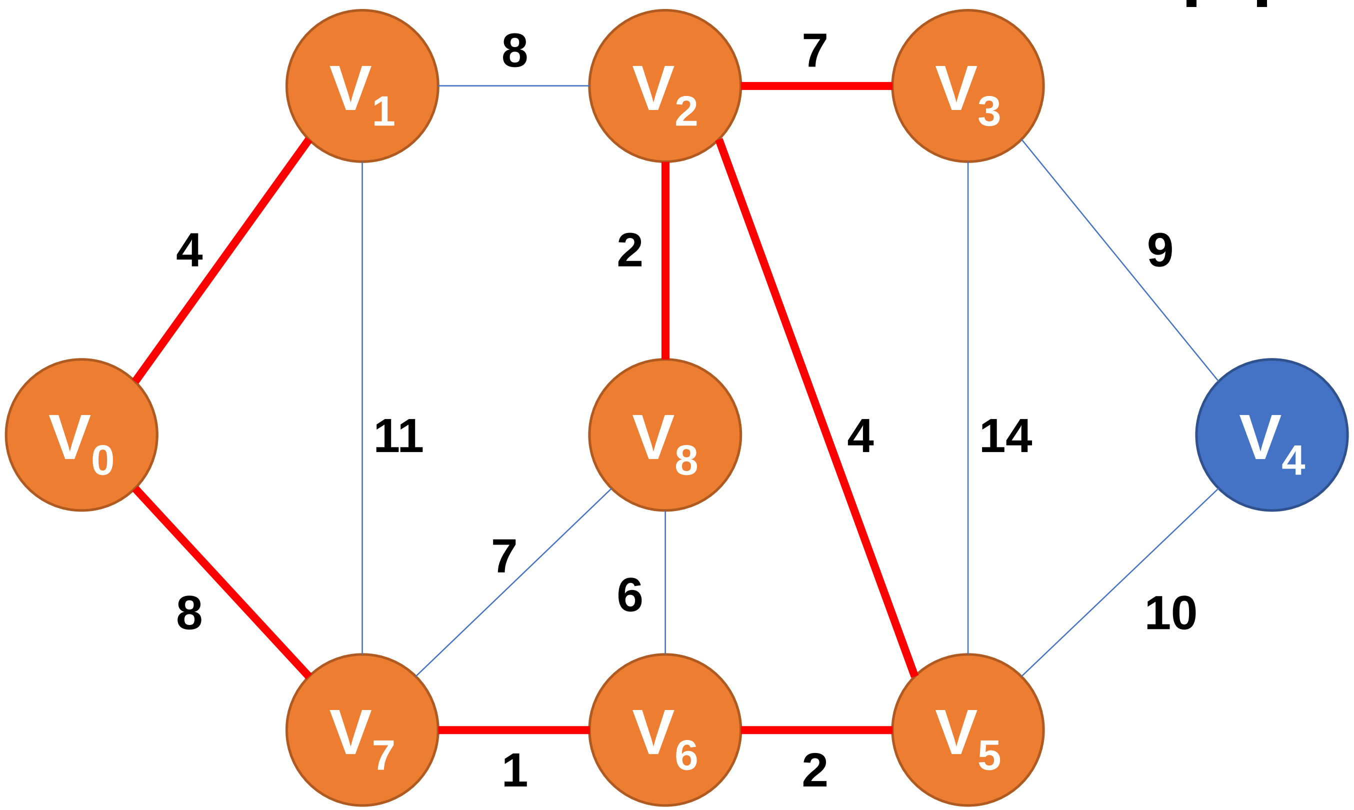
# of Edges in MST = 6

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	~~V_8,V_6~~	V_2,V_3	~~V_7,V_8~~	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	~~6~~	7	~~7~~	8	8	9	10	11	14

Examples

|V|=9



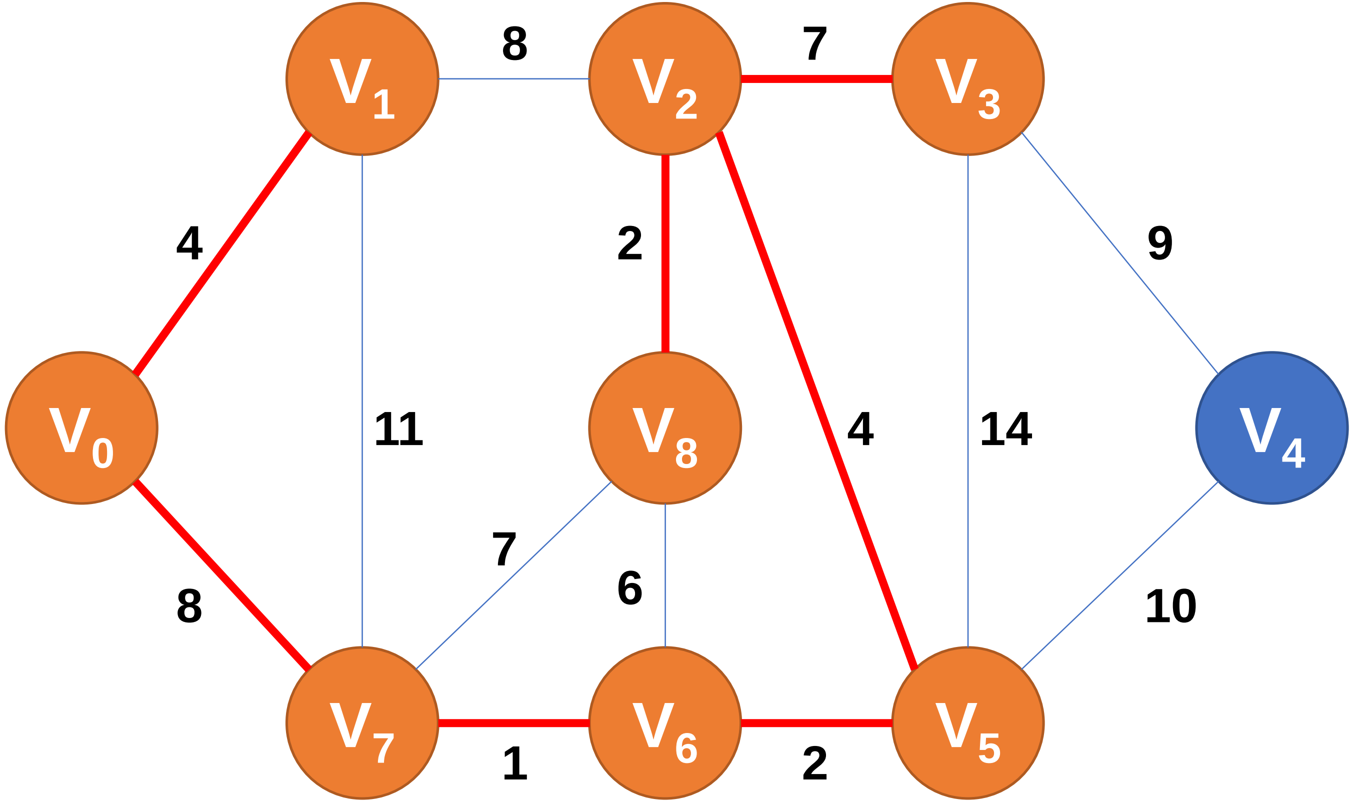
# of Edges in MST = 7

1. **Sort all the edges in non-decreasing 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.**

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	~~V_8,V_6~~	V_2,V_3	~~V_7,V_8~~	V_0,V_7	V_1,V_2	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	~~6~~	7	~~7~~	8	8	9	10	11	14

Examples

|V|=9



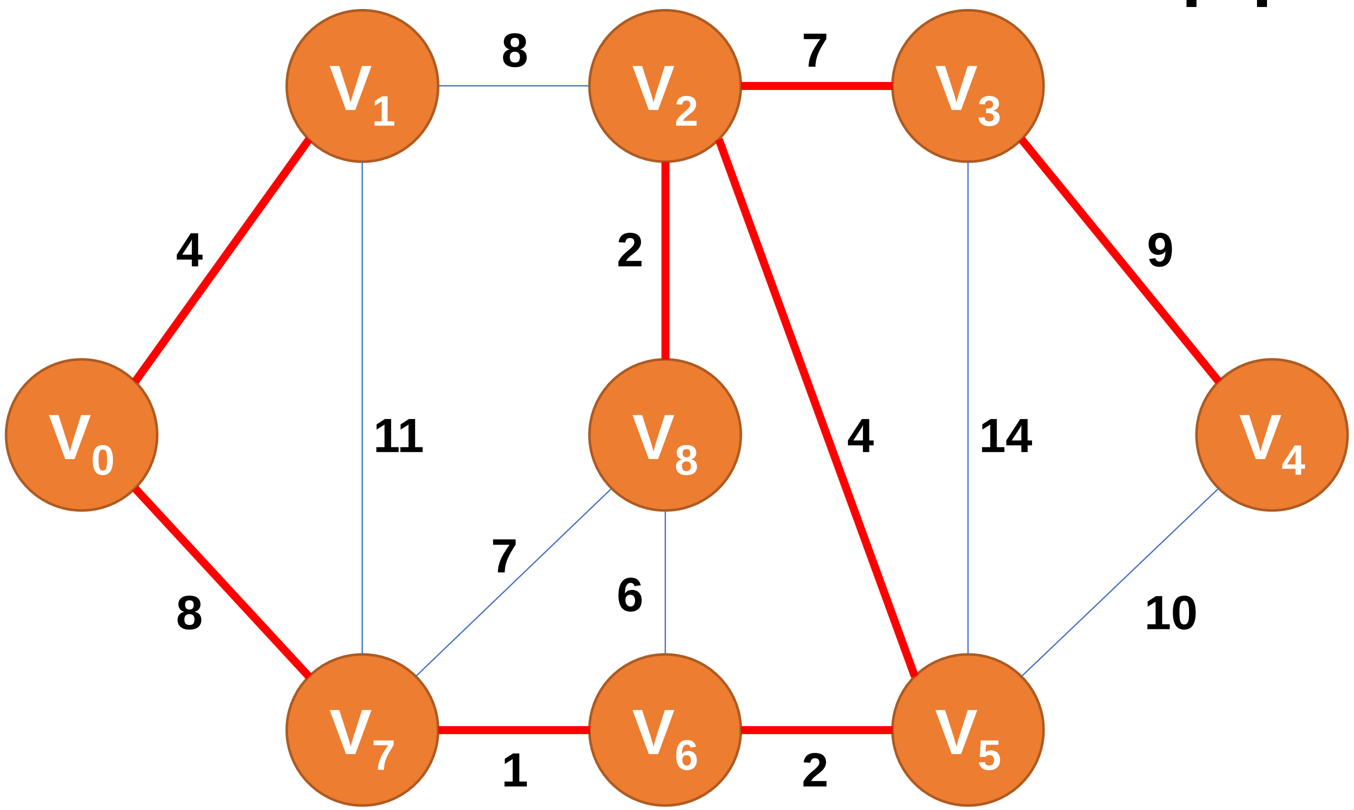
# of Edges in MST = 7

1. **Sort all the edges in non-decreasing 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.**

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	~~V_8,V_6~~	V_2,V_3	~~V_7,V_8~~	V_0,V_7	~~V_1,V_2~~	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Examples

|V|=9



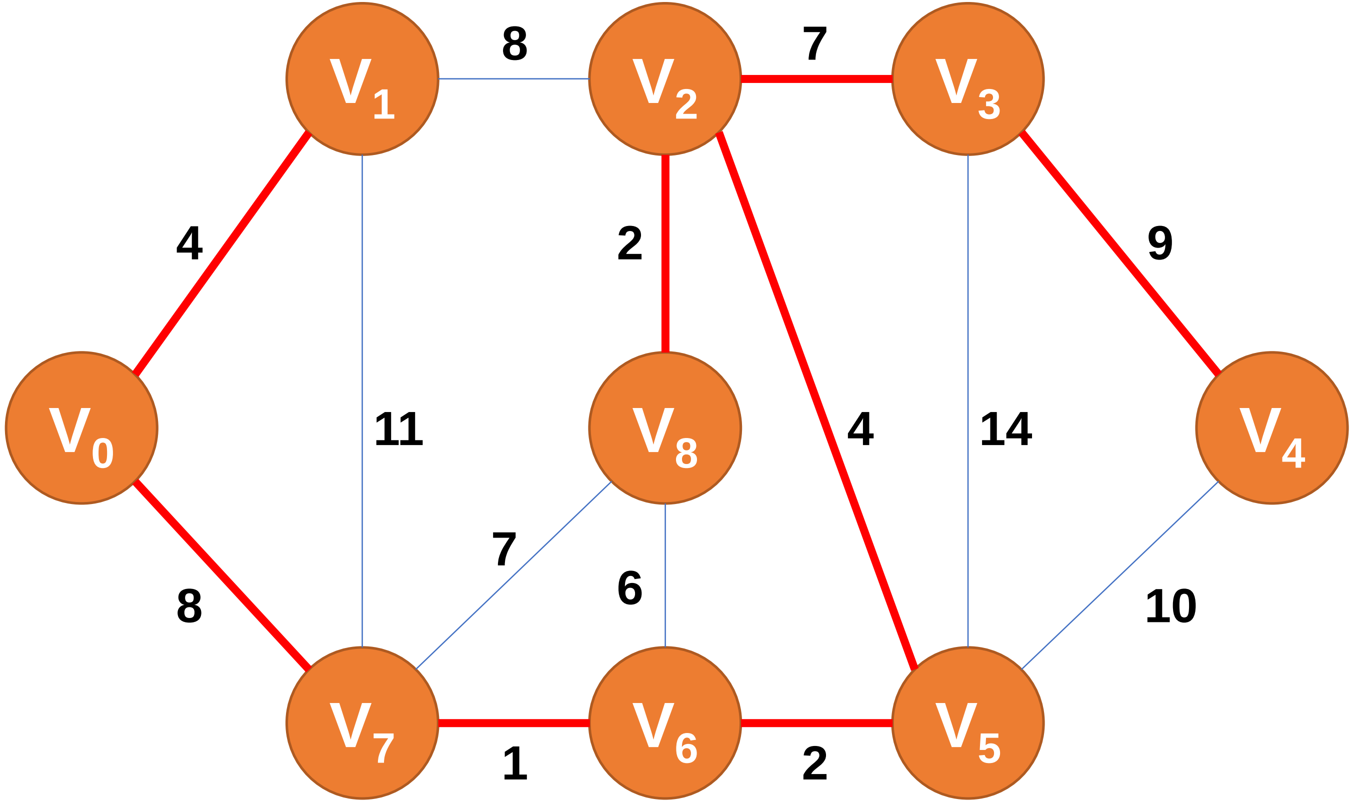
# of Edges in MST = 8

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	~~V_8,V_6~~	V_2,V_3	~~V_7,V_8~~	V_0,V_7	~~V_1,V_2~~	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Examples

|V|=9

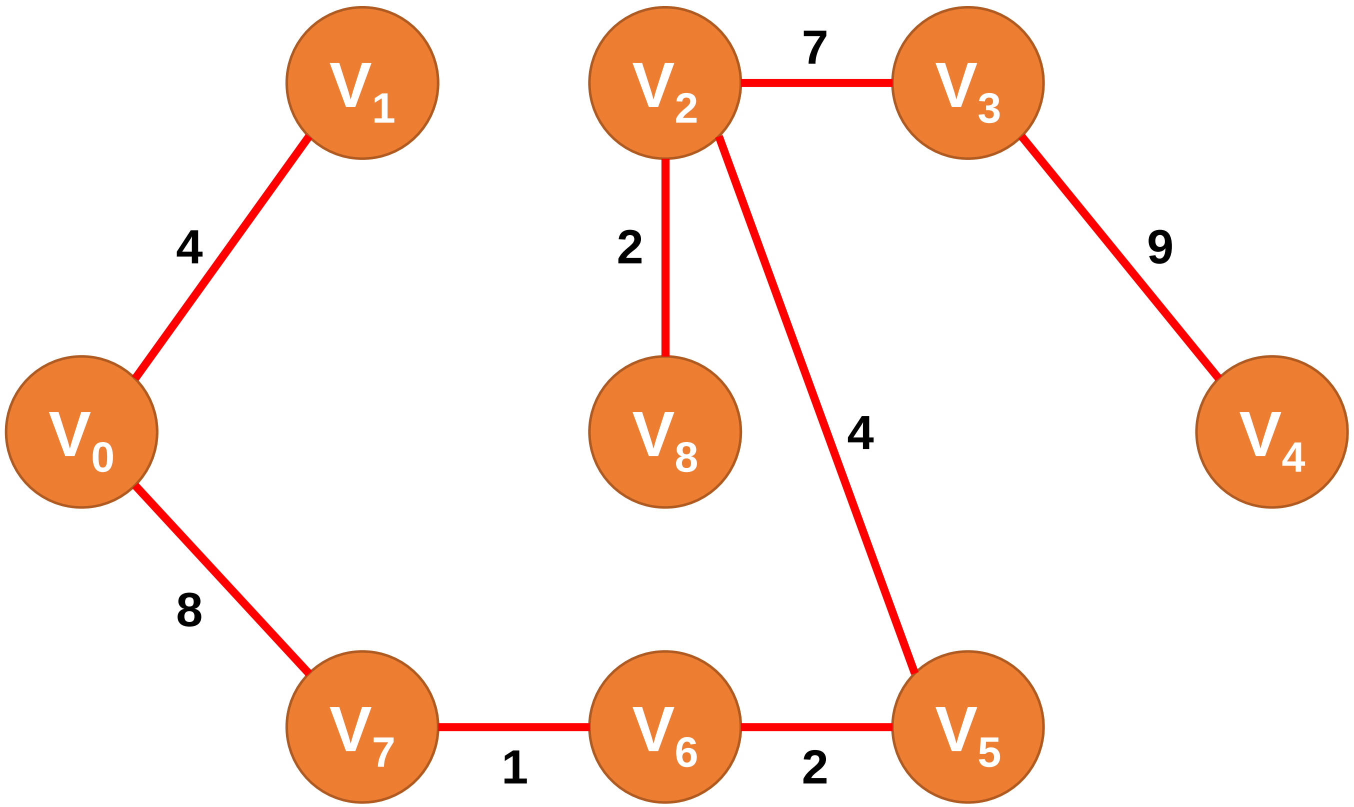


Terminate!!!

1. **Sort all the edges in non-decreasing 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.**

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	~~V_8,V_6~~	V_2,V_3	~~V_7,V_8~~	V_0,V_7	~~V_1,V_2~~	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Examples



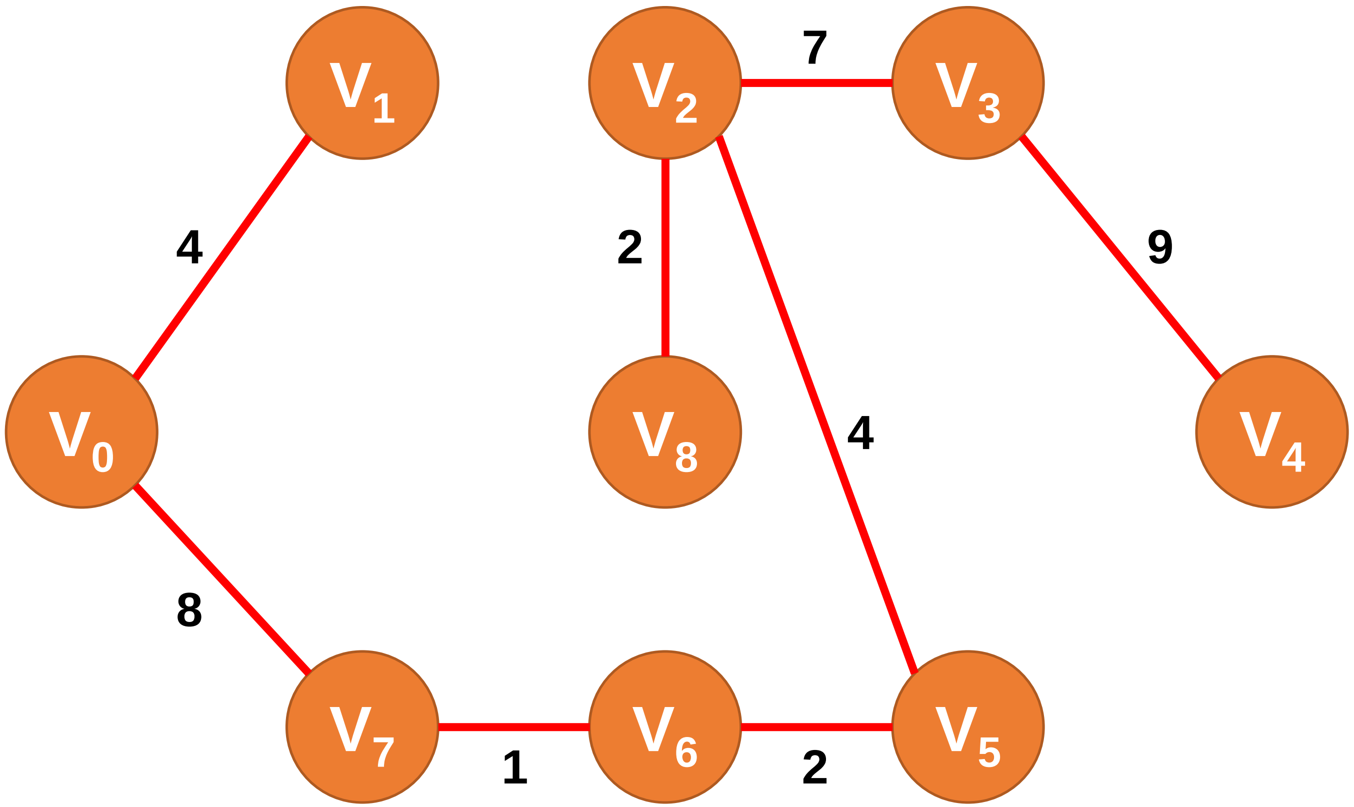
Now we have our MST!

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	~~V_8,V_6~~	V_2,V_3	~~V_7,V_8~~	V_0,V_7	~~V_1,V_2~~	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Examples

Weights of MST = 37

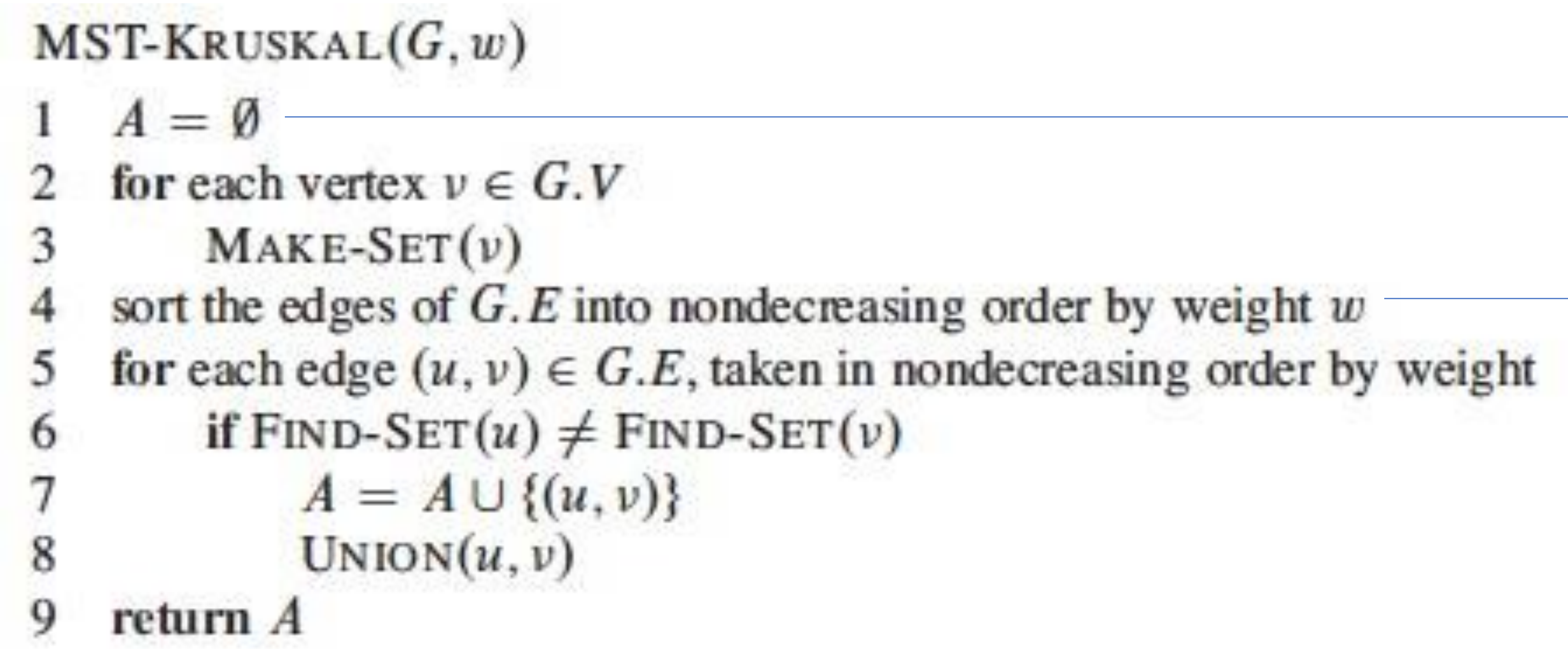


Now we have our MST!

1. Sort all the edges in non-decreasing 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.

V_7,V_6	V_8,V_2	V_6,V_5	V_0,V_1	V_2,V_5	~~V_8,V_6~~	V_2,V_3	~~V_7,V_8~~	V_0,V_7	~~V_1,V_2~~	V_3,V_4	V_5,V_4	V_1,V_7	V_3,V_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Pseudocode



```
MST-KRUSKAL$(G, w)$
$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET$(v)$
sort the edges of $G.E$ into nondecreasing order by weight $w$
for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
    if FIND-SET$(u) \neq$ FIND-SET$(v)$
        $A = A \cup \{(u, v)\}$
        UNION$(u, v)$
return $A$
```

O(1)

O(E log E)

O(E log V)

Overall Complexity = O(E Log E + E Log V)

→ $V-1 \leq E \leq (V^2-V)/2$ → O(Log E) ≈ O(Log V)

→ Therefore, time complexity is O(E Log E) or O(E Log V)

Prim’s Algorithm (1)

1. Create a set mstSet that keeps track of vertices already included in MST.

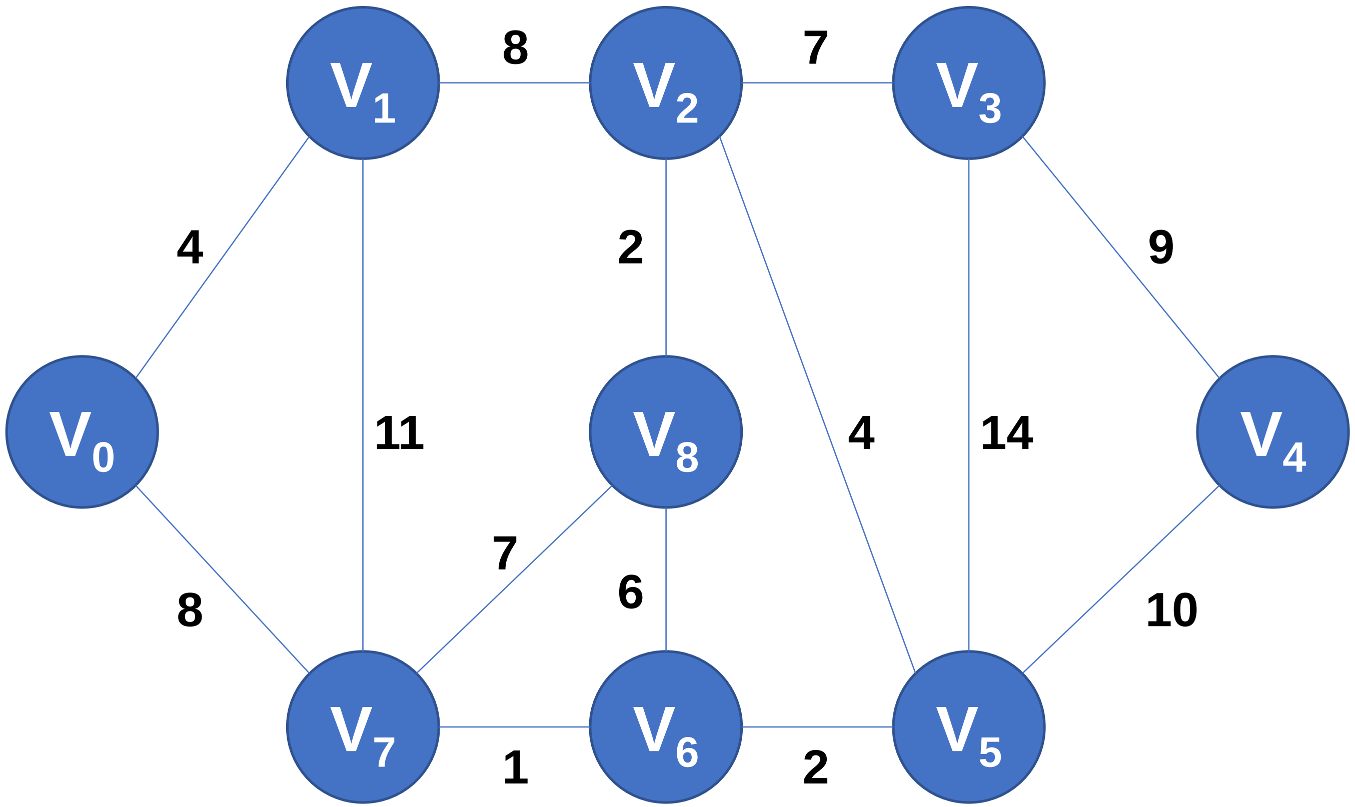
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

Prim’s Algorithm (2)

3. While mstSet doesn’t include all vertices
    a. Pick a vertex u which is not there in mstSet and has minimum key value.
    b. Include u to mstSet.
    c. Update key value of all adjacent vertices of u.

To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

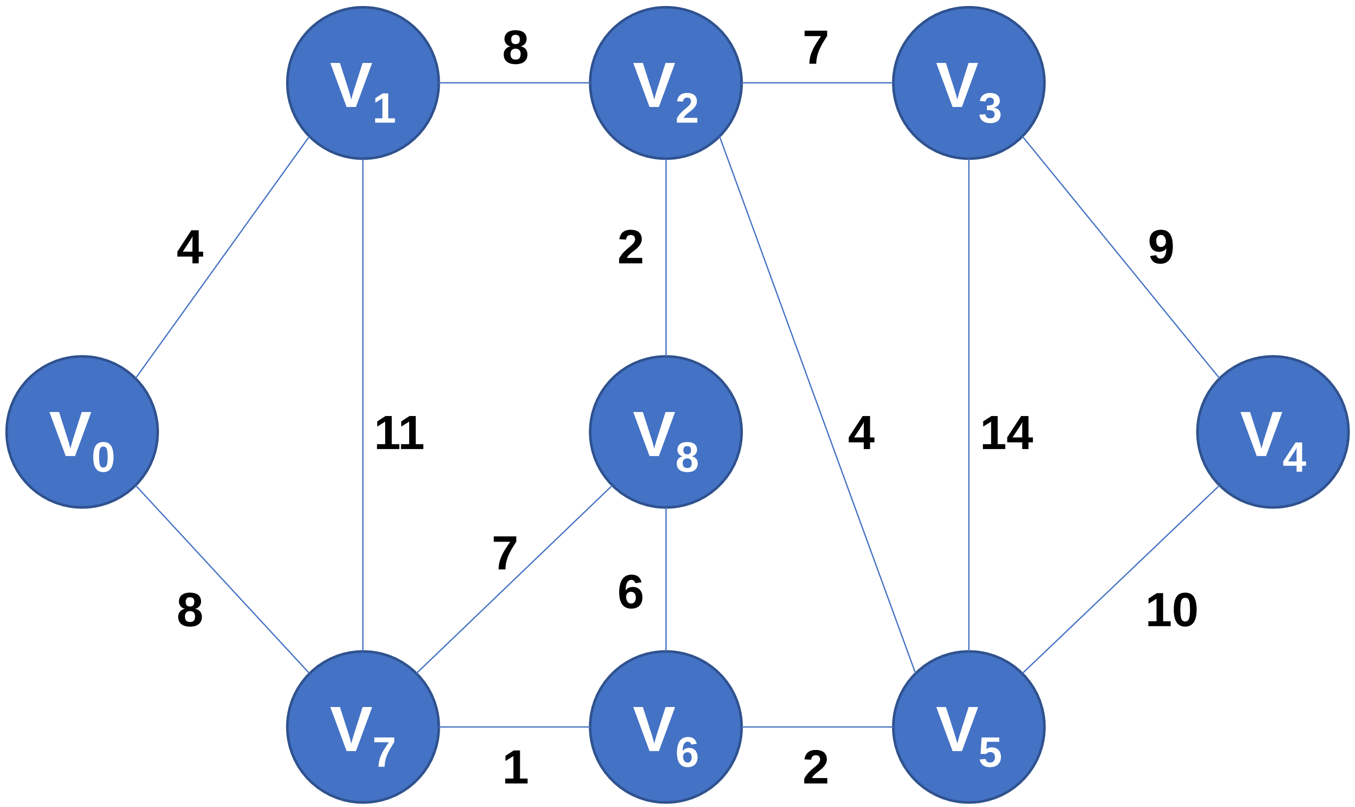
Examples



1. **Create a set mstSet that keeps track of vertices already included in MST.**
2. **Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.**
3. **While mstSet doesn’t include all vertices**
   a. **Pick a vertex u which is not there in mstSet and has minimum key value.**
   b. **Include u to mstSet.**
   c. **Update key value of all adjacent vertices of u.**

mstSet={}

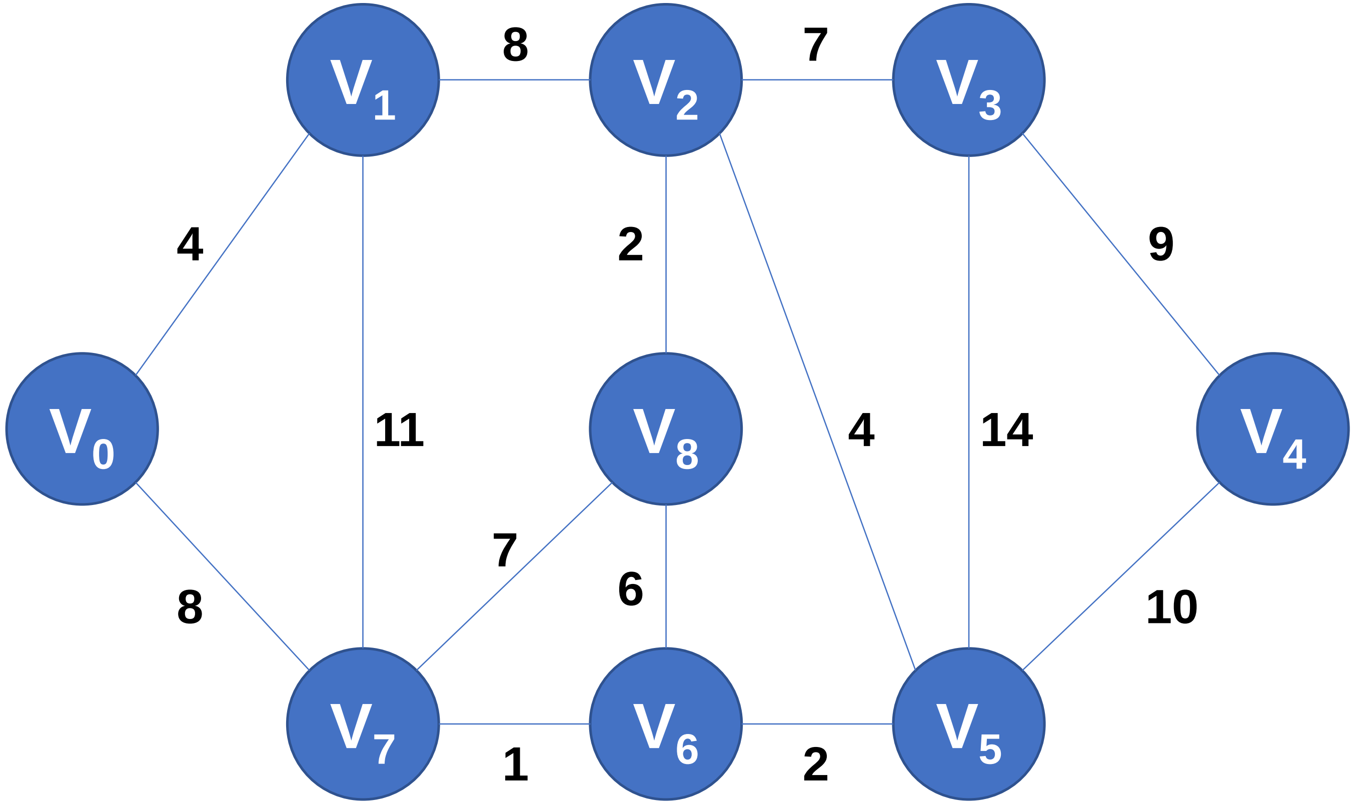
Examples



1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={}

Examples

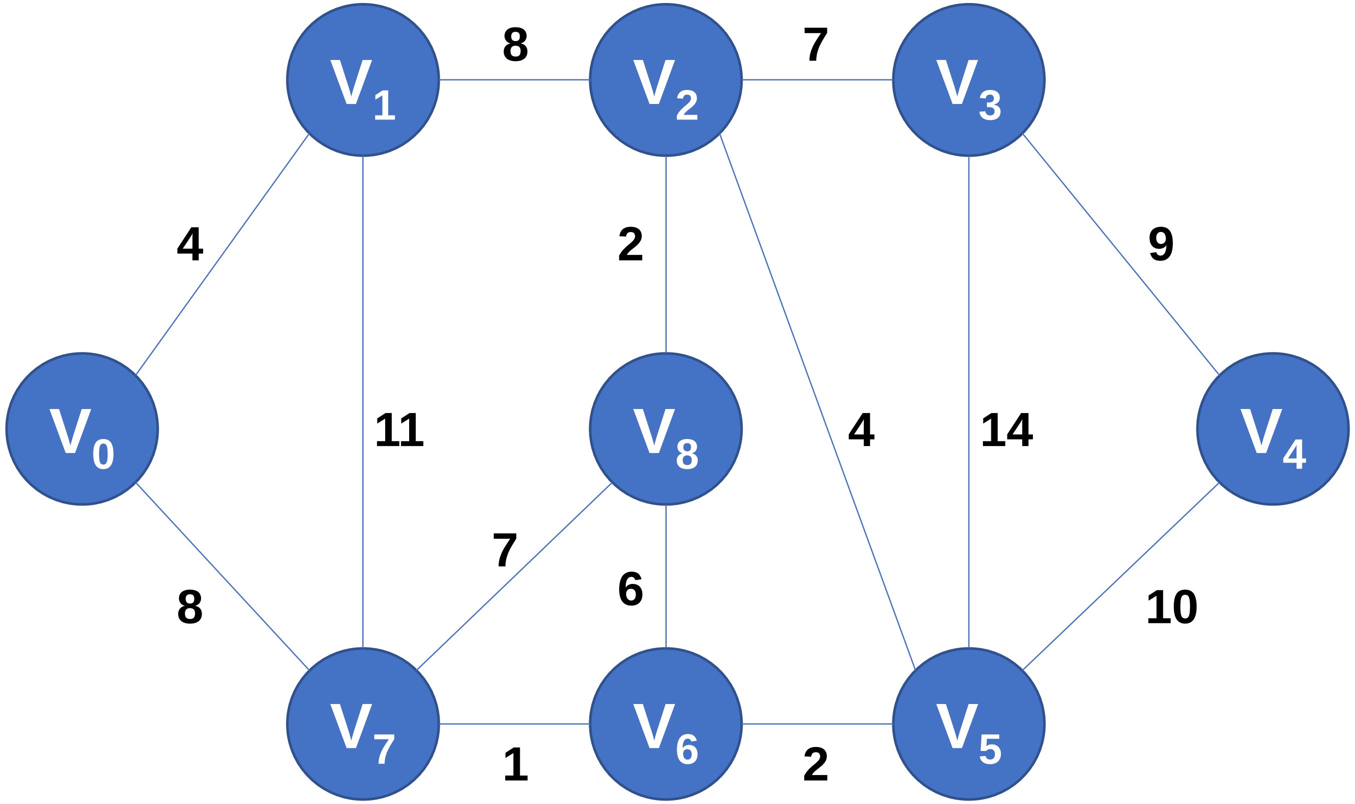


Ver.	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
Key	0	∞	∞	∞	∞	∞	∞	∞	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={}

Examples

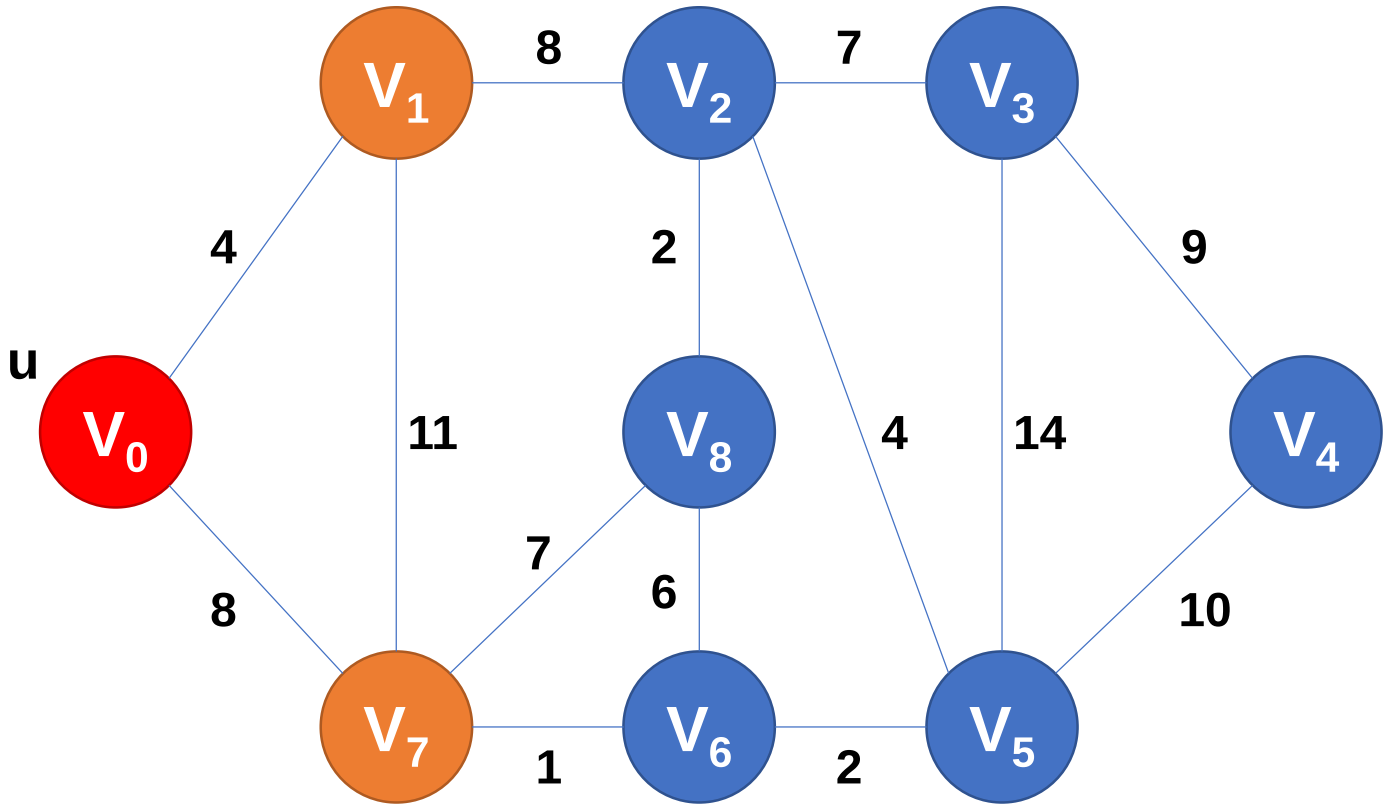


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	∞	∞	∞	∞	∞	∞	∞	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0}



If w(u,v)<v.key, v.key=w(u,v)

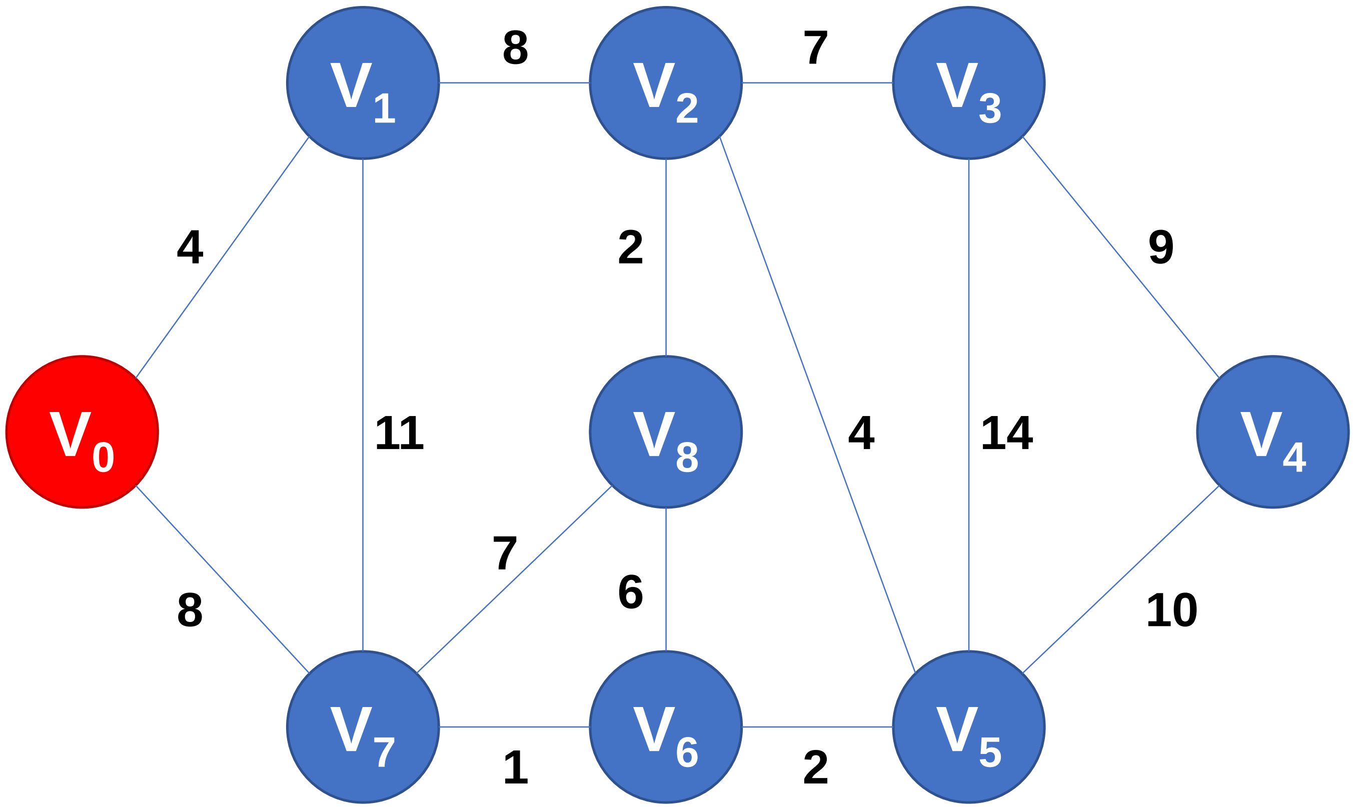
u

Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	∞	∞	∞	∞	∞	∞	∞	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

$\text{mstSet}=\{v_0\}$

Examples

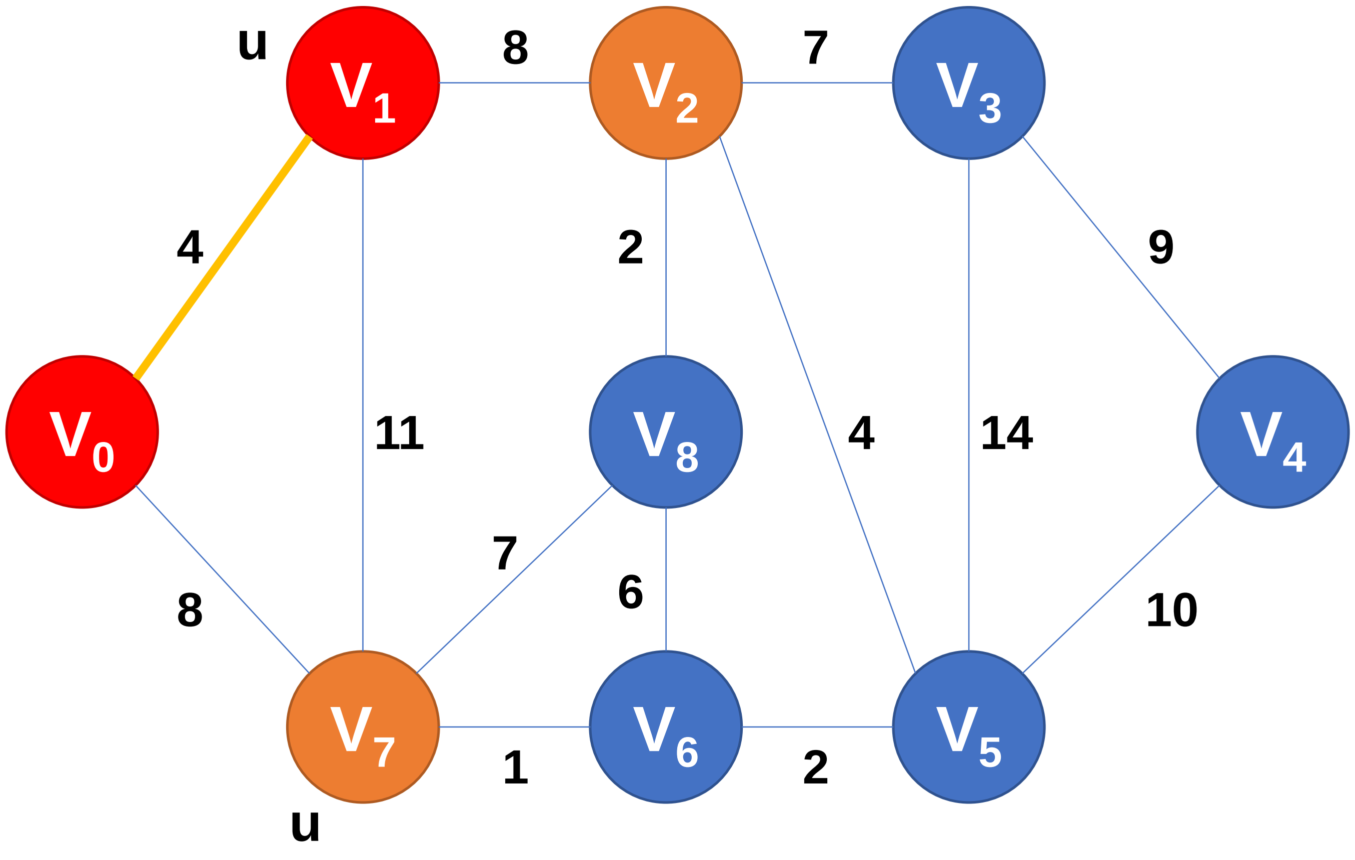


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	∞	∞	∞	∞	∞	8	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0, v_1}

Examples

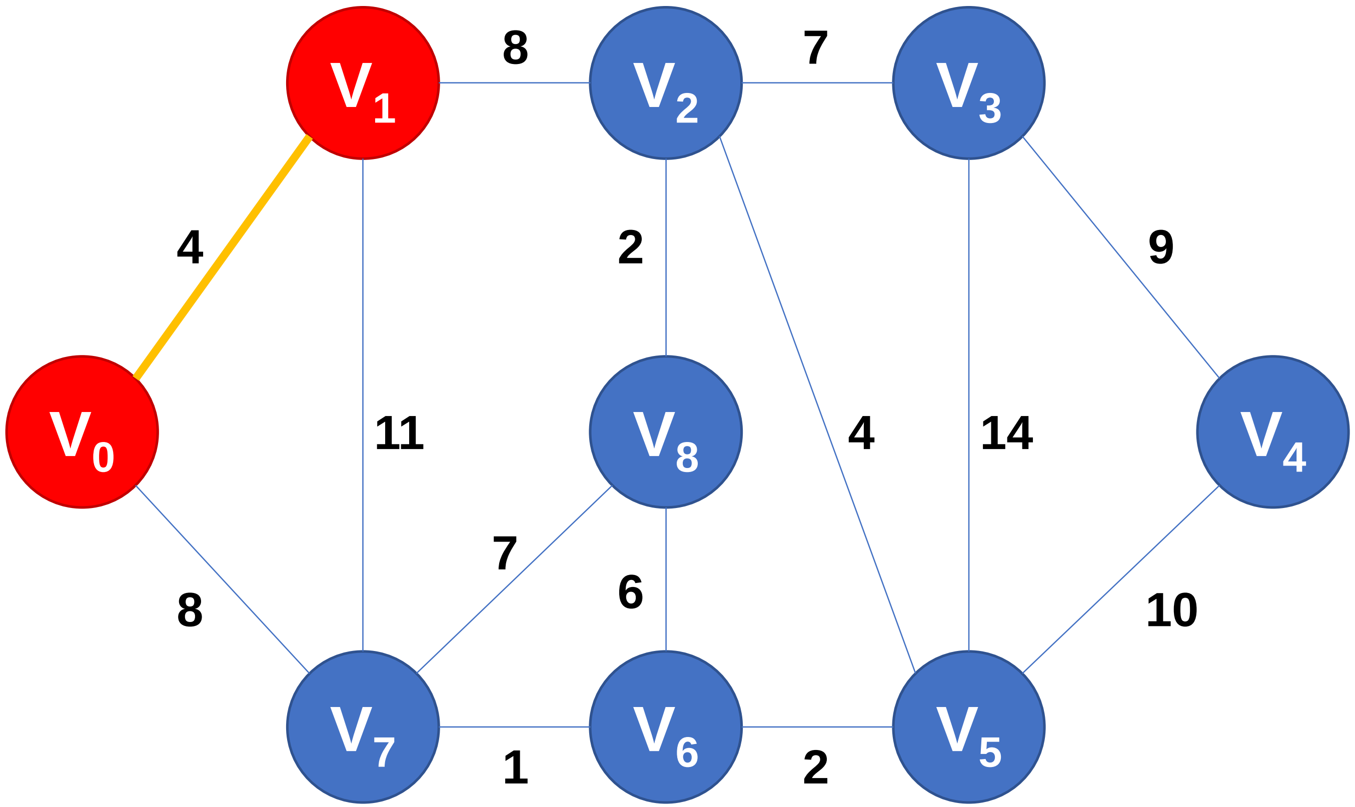


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	∞	∞	∞	∞	∞	8	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0, v_1}

Examples

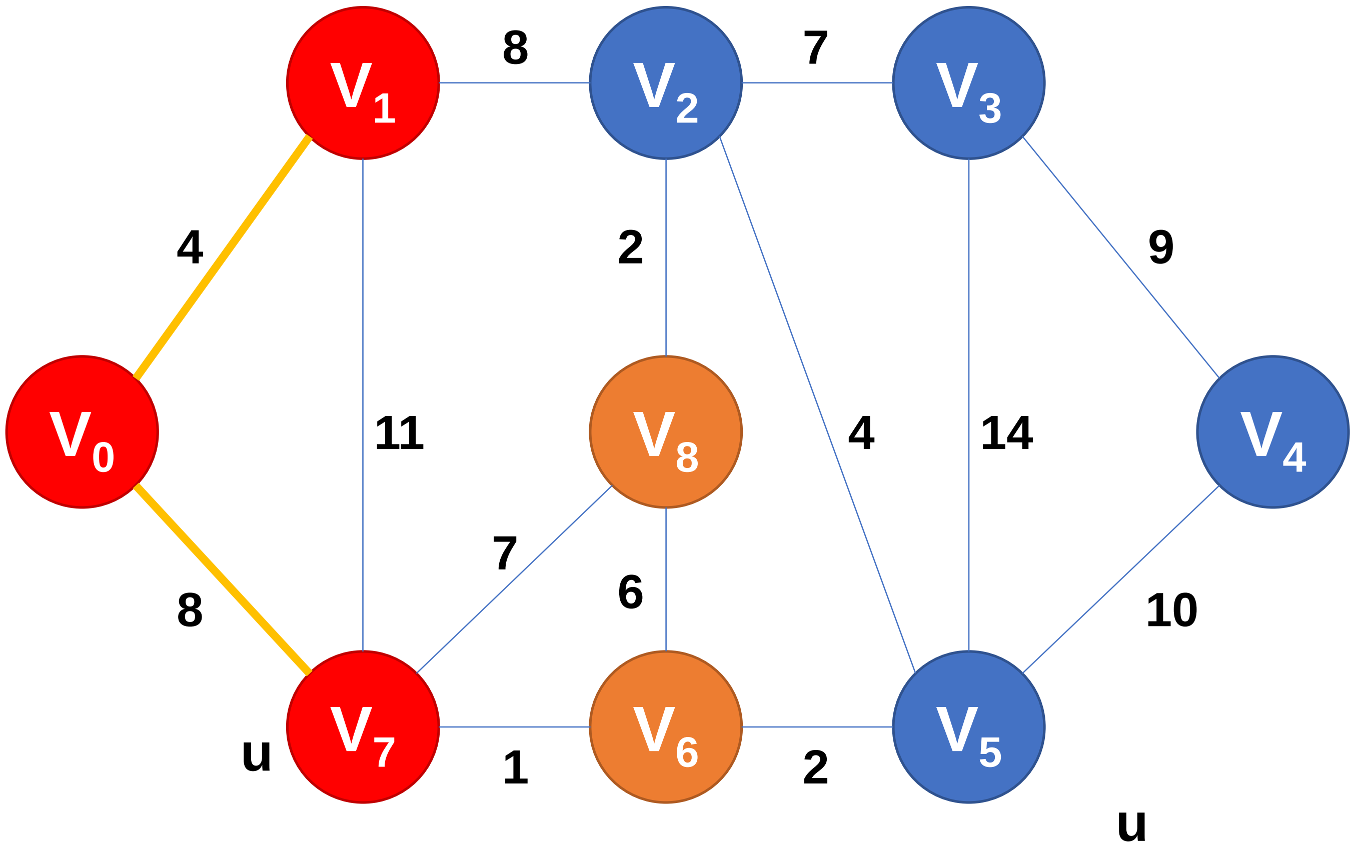


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	∞	∞	8	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0,v_1,v_7}

Examples

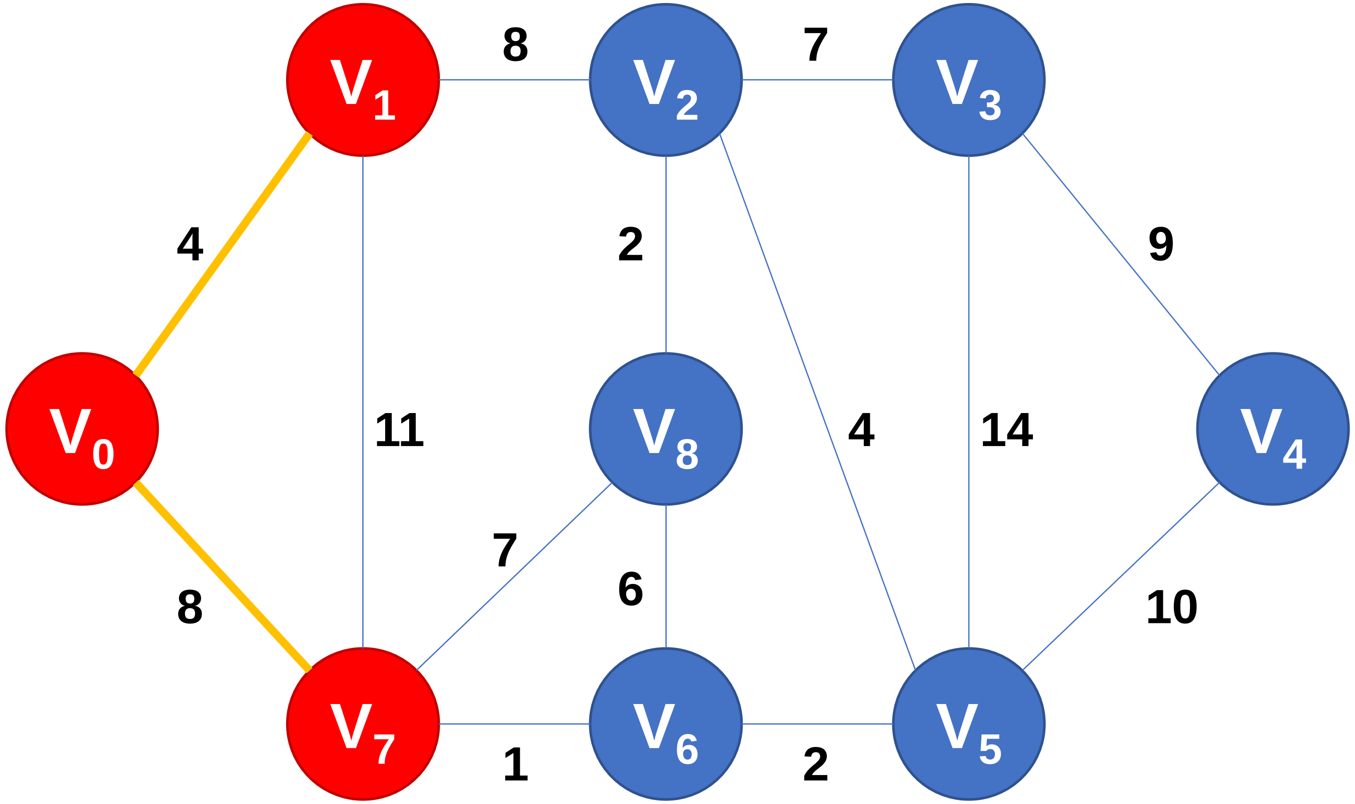


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	∞	∞	8	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

$\text{mstSet}=\{v_0, v_1, v_7\}$

Examples



Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	∞	1	8	7

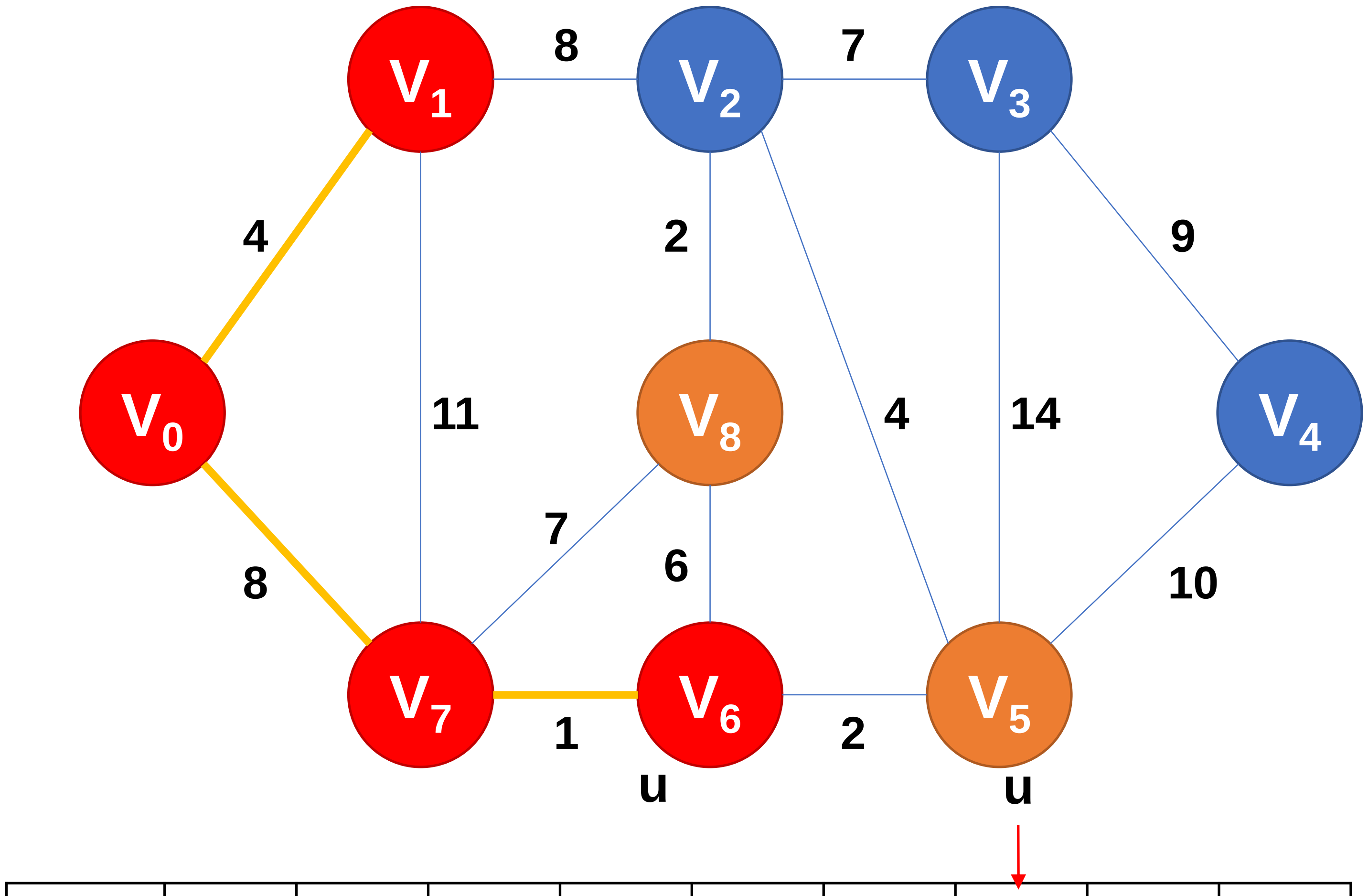
1. Create a set mstSet that keeps track of vertices already included in MST.

2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

$mstSet=\{v_0,v_1,v_7,v_6\}$

Examples

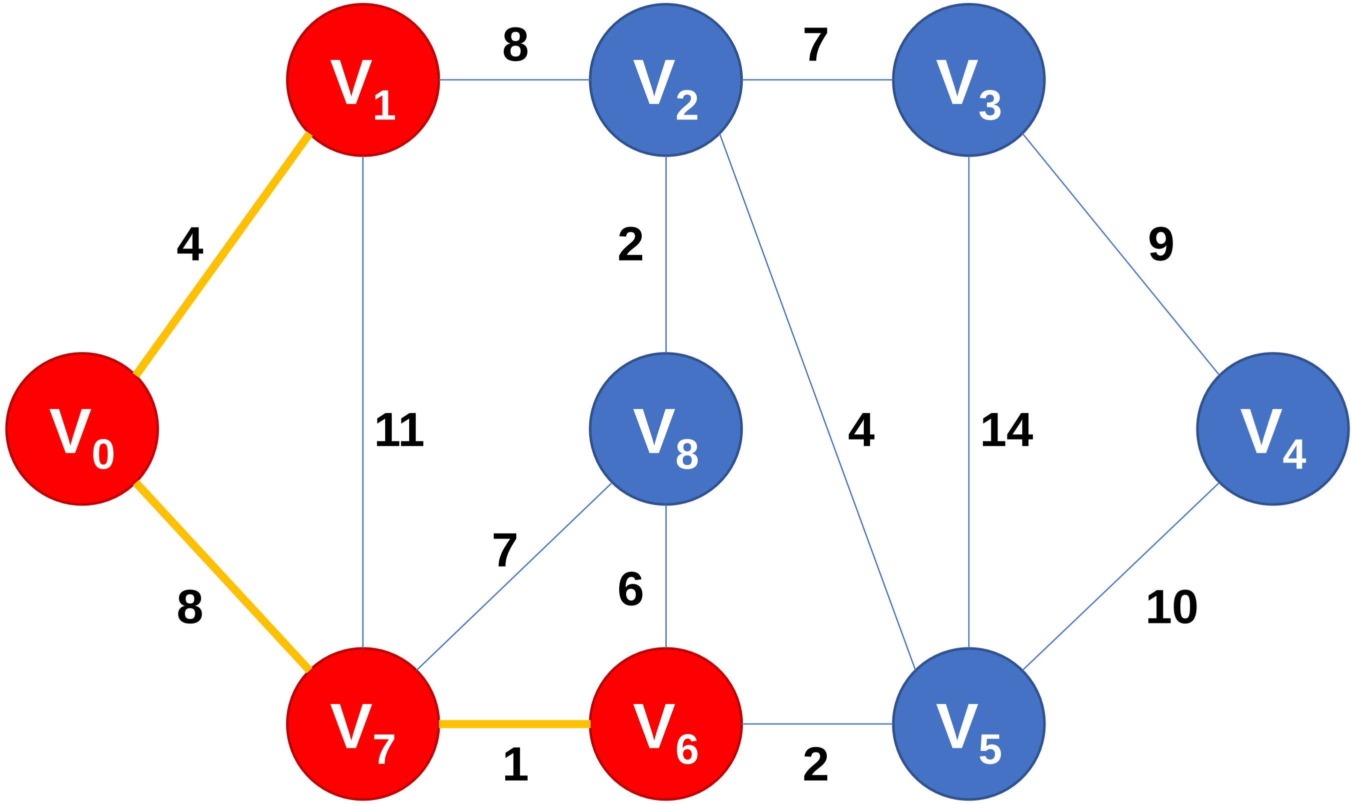


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	∞	1	8	7

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

$\text{mstSet}=\{v_0,v_1,v_7,v_6\}$

Examples

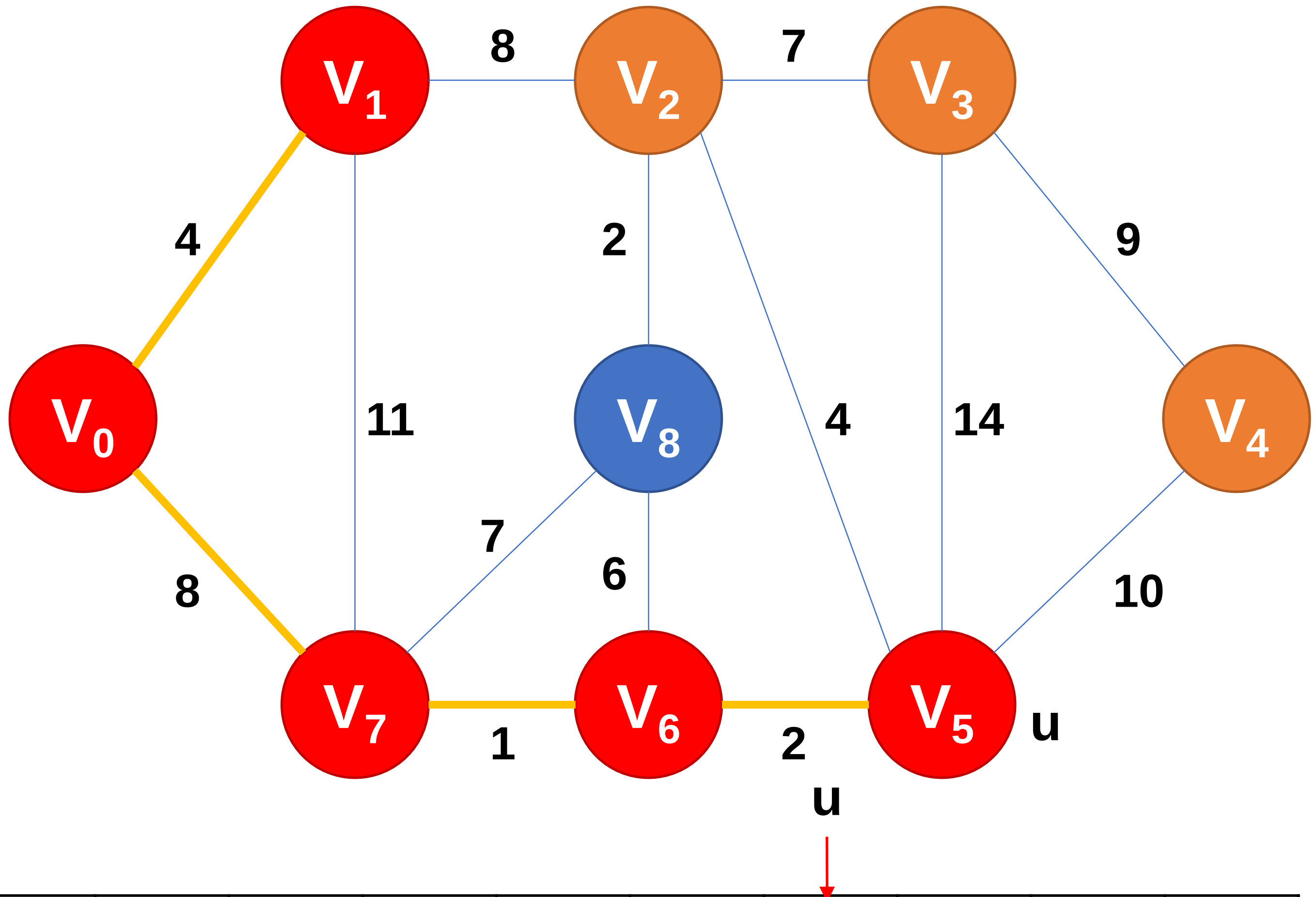


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	2	1	8	6

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0,v_1,v_7,v_6,v_5}

Examples

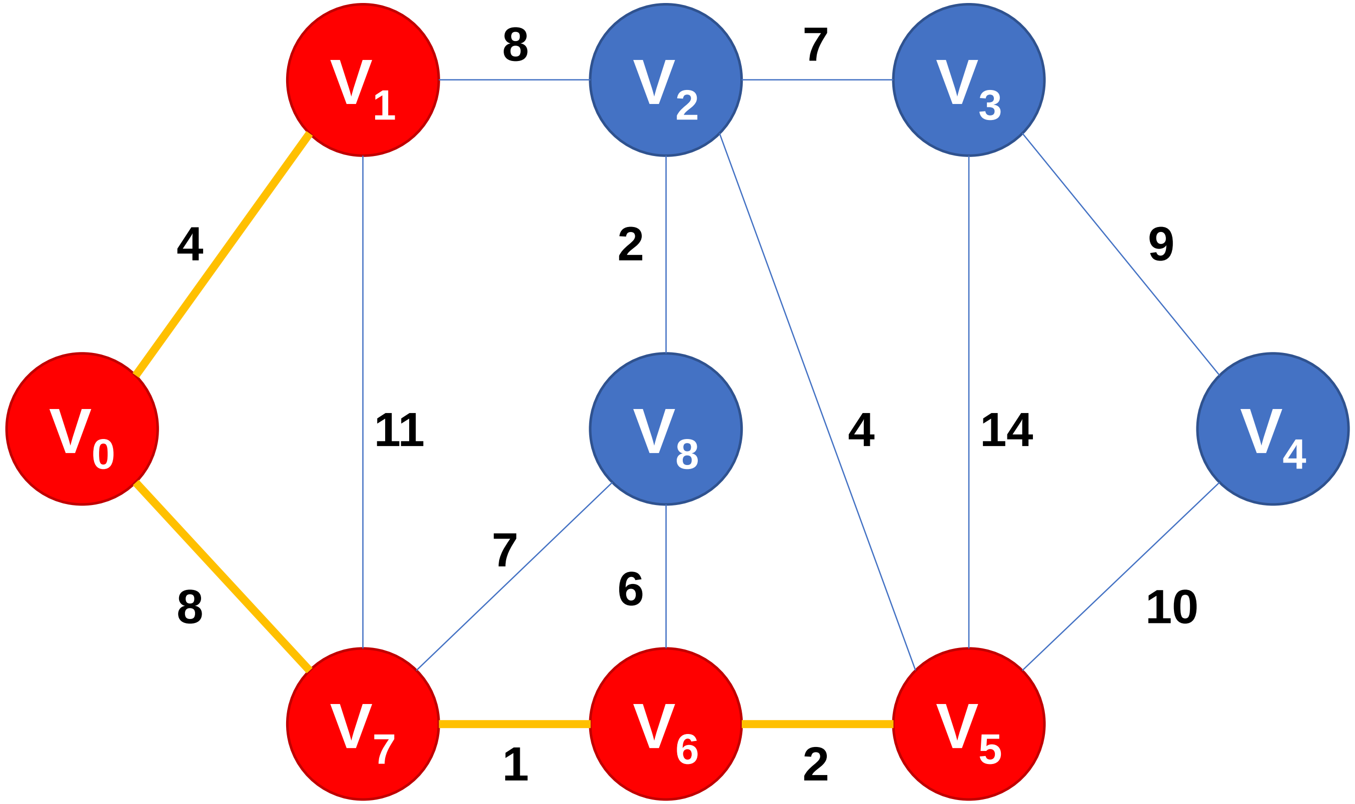


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	2	1	8	6

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

$mstSet = \{v_0, v_1, v_7, v_6, v_5\}$

Examples

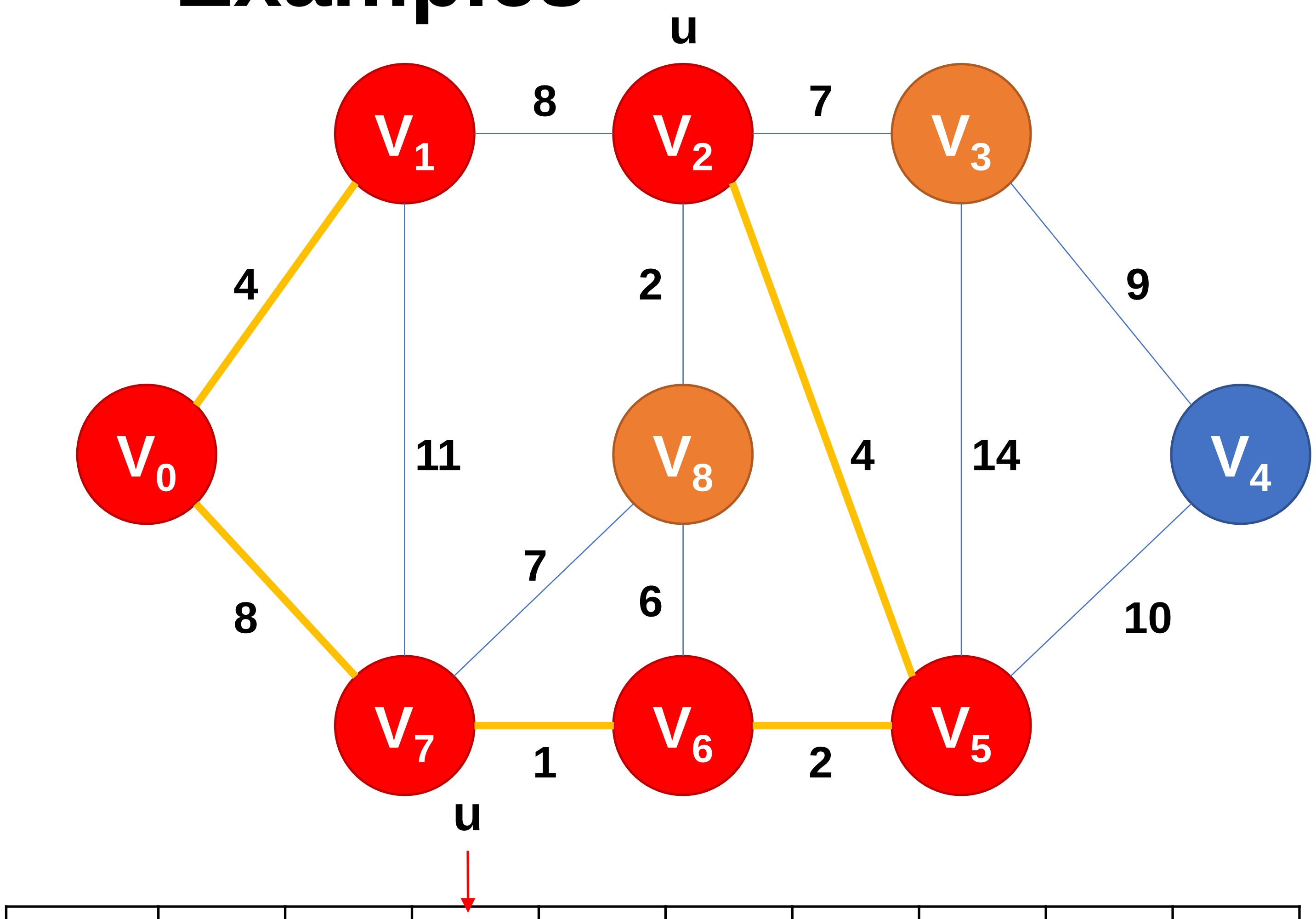


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	14	10	2	1	8	6

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0,v_1,v_7,v_6,v_5,v_2}

Examples

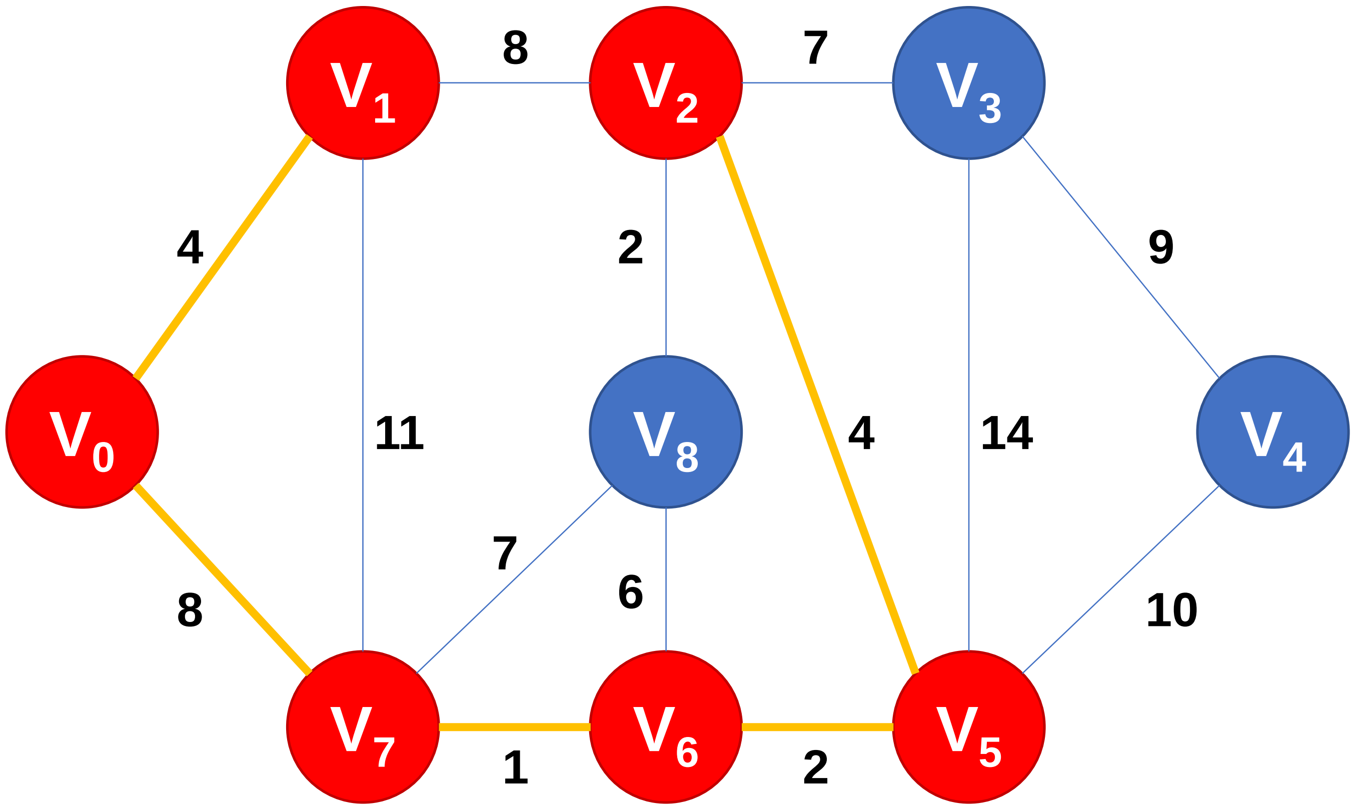


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	14	10	2	1	8	6

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0,v_1,v_7,v_6,v_5,v_2}

Examples

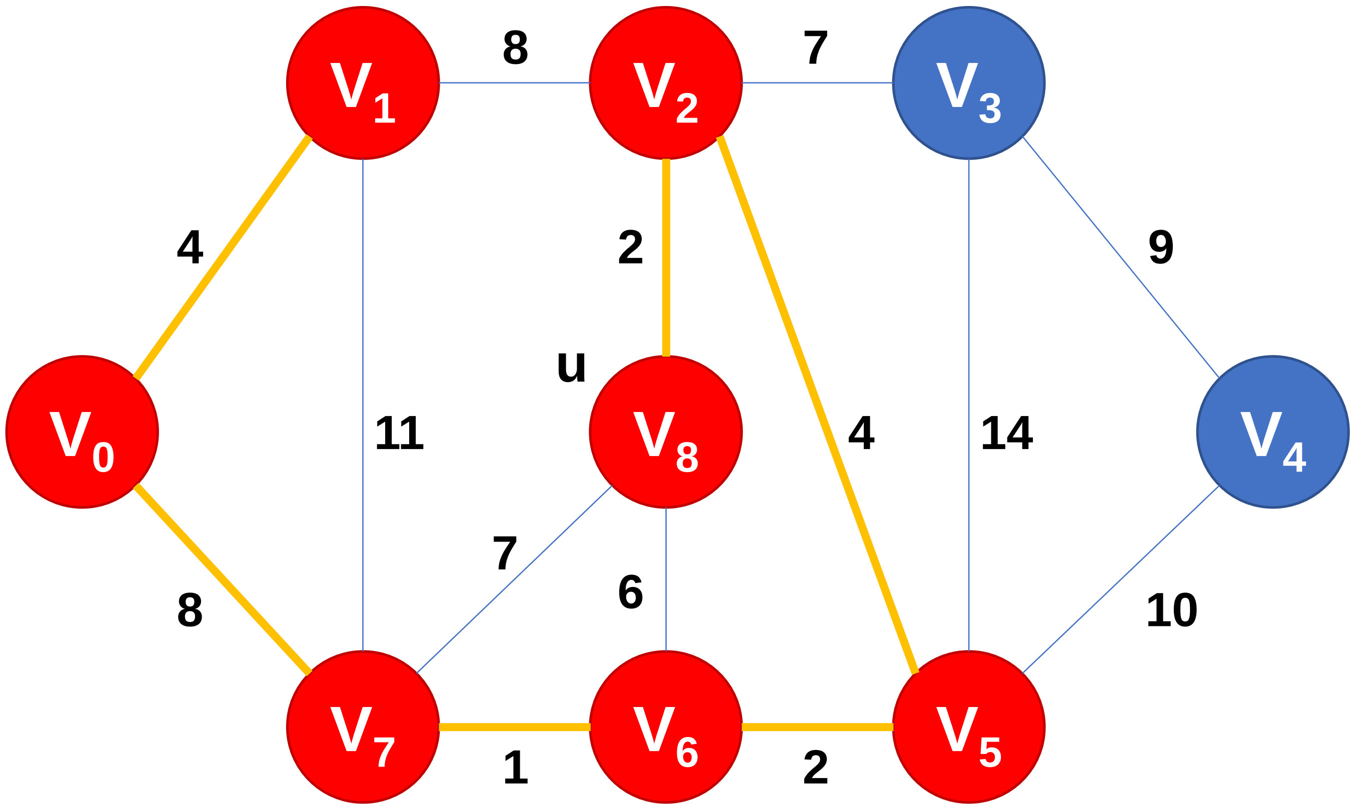


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	10	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

$mstSet=\{v_0,v_1,v_7,v_6,v_5,v_2,v_8\}$

Examples



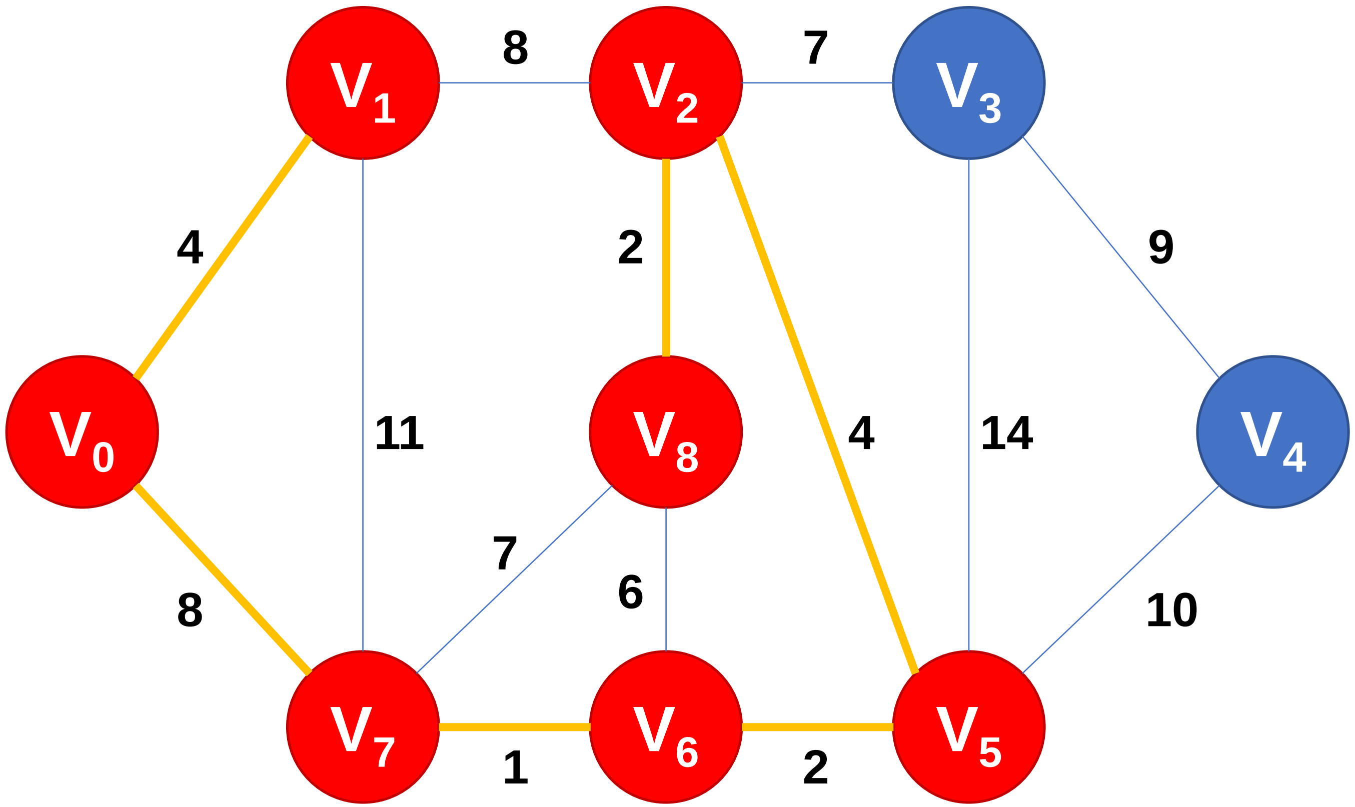
u

Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	10	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0,v_1,v_7,v_6,v_5,v_2,v_8}

Examples

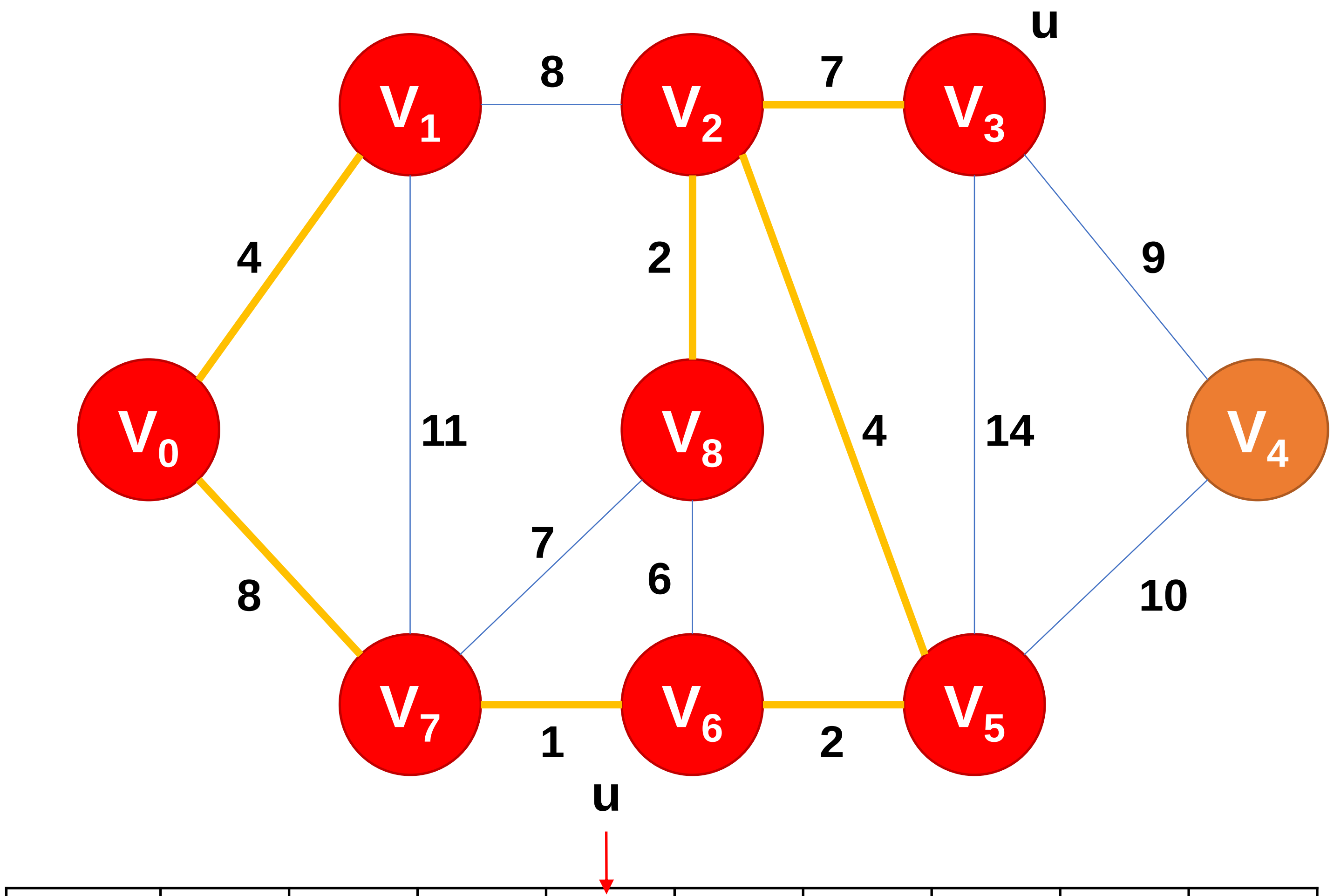


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	10	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0,v_1,v_7,v_6,v_5,v_2,v_8,v_3}

Examples

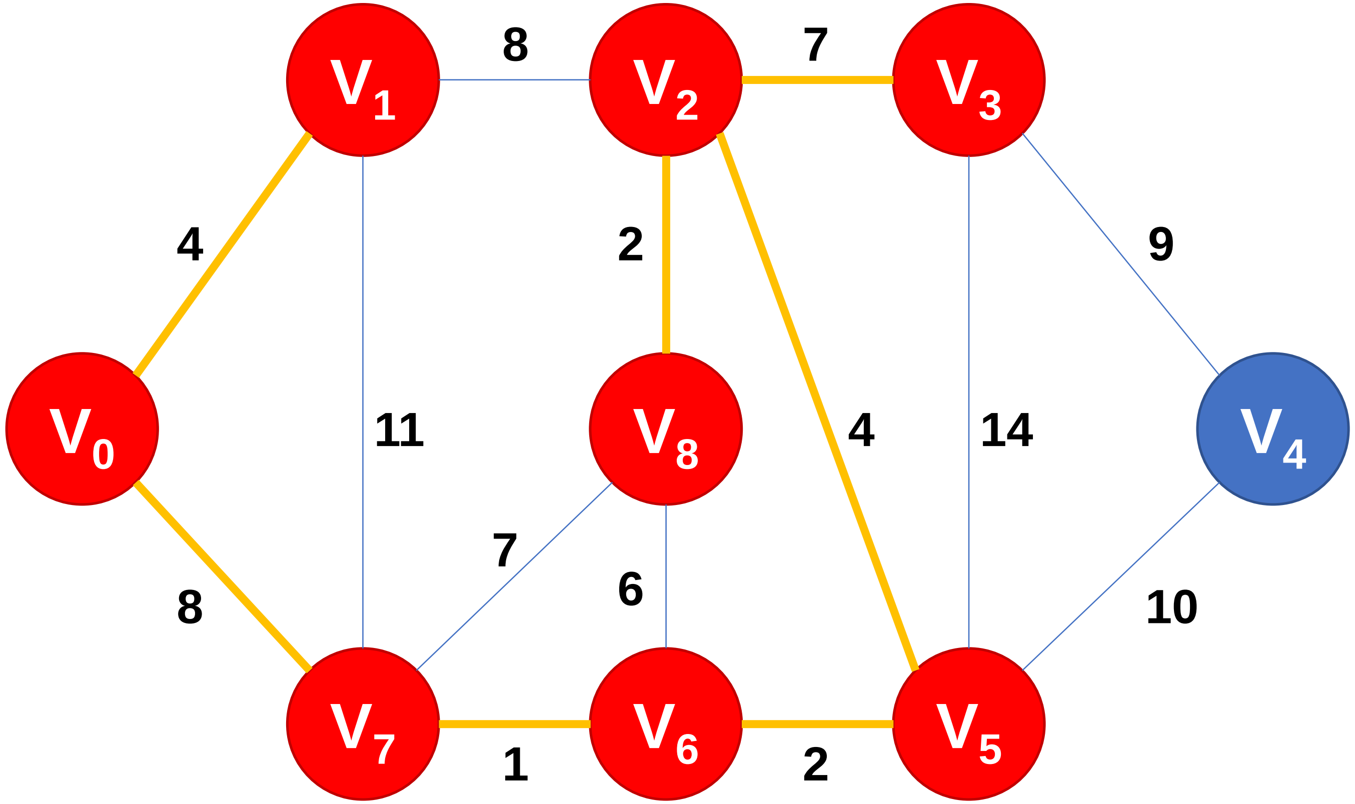


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	10	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0,v_1,v_7,v_6,v_5,v_2,v_8,v_3}

Examples

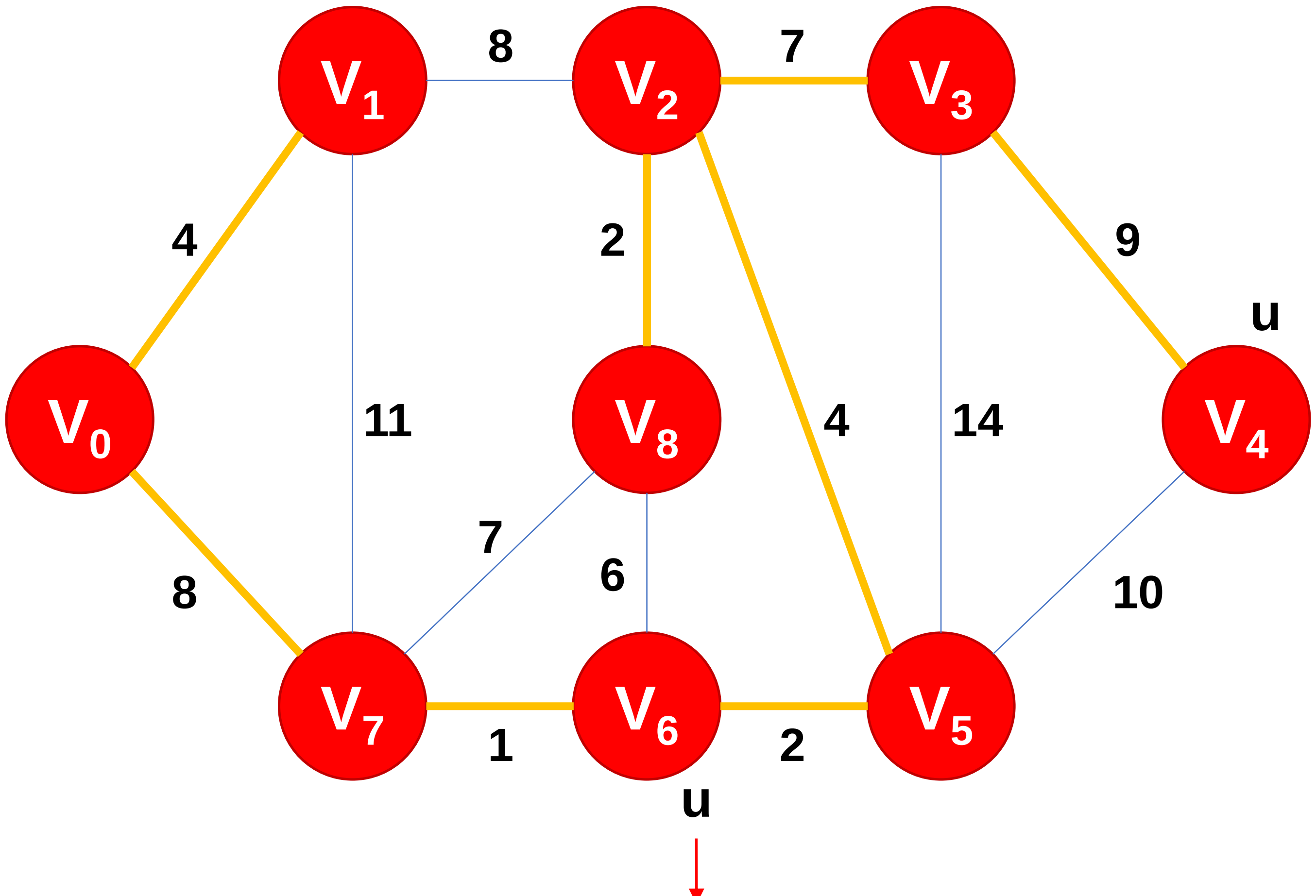


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	9	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0,v_1,v_7,v_6,v_5,v_2,v_8,v_3,v_4}

Examples

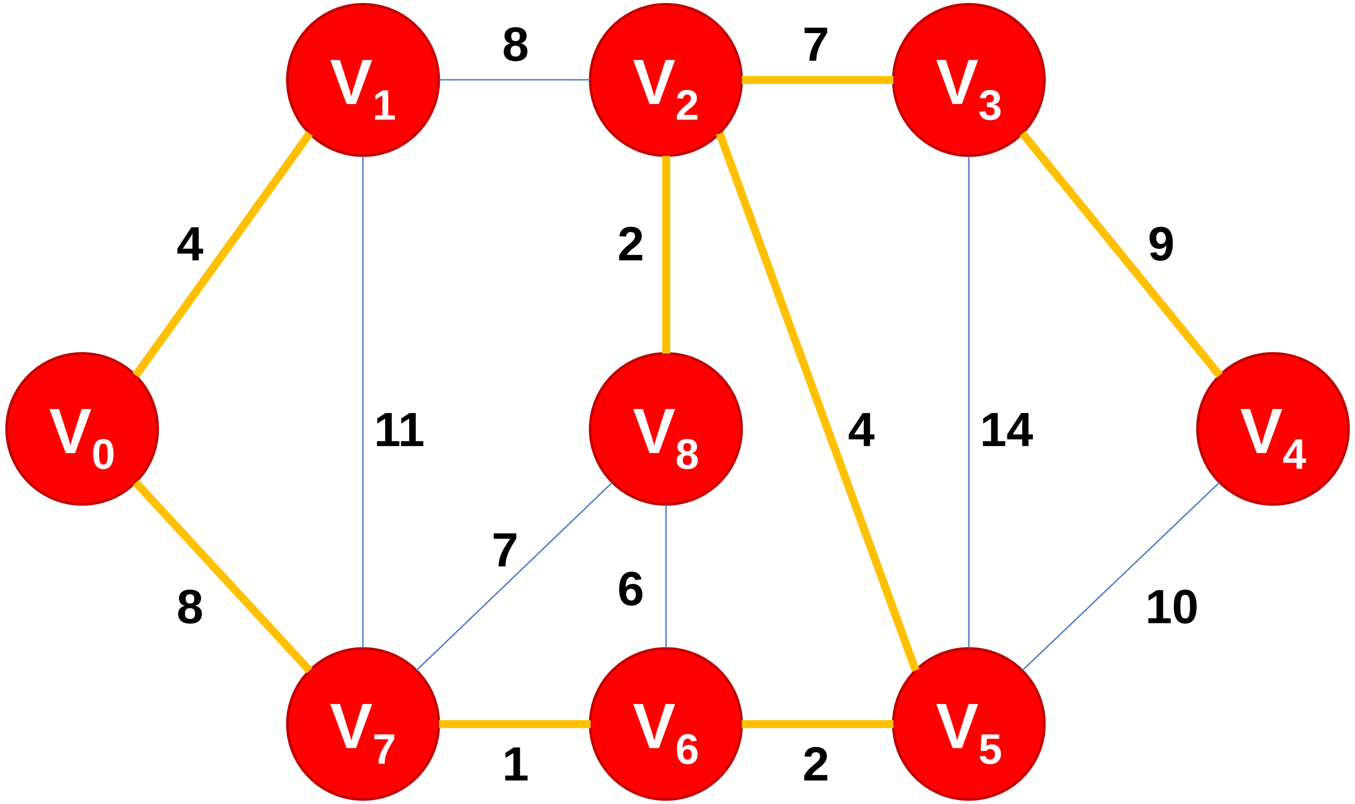


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	9	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={v_0,v_1,v_7,v_6,v_5,v_2,v_8,v_3,v_4}

Examples



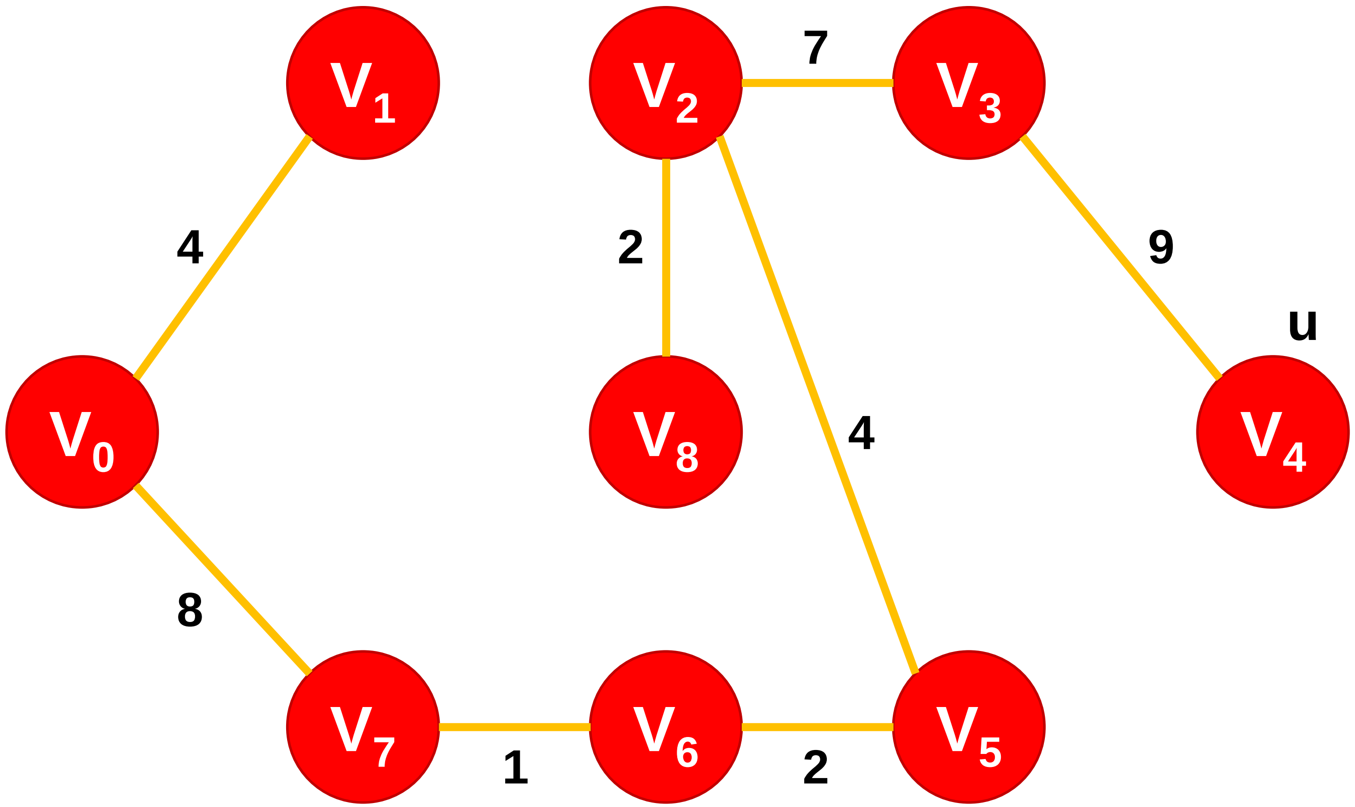
Terminate!!

Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	9	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

Weights of MST = 37



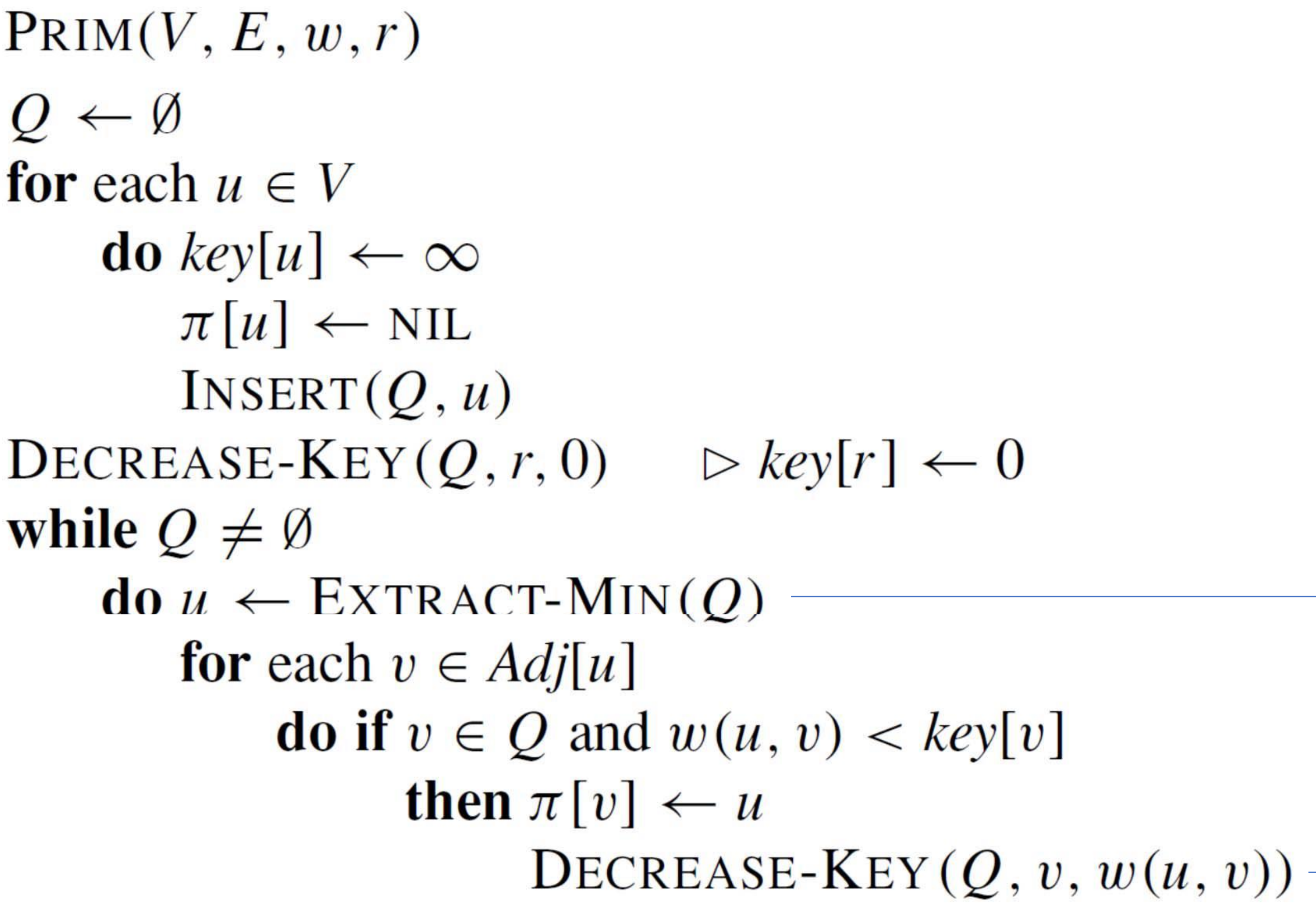
Ver.	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
Key	0	4	4	7	9	2	1	8	2

Time Complexity

- Time Complexity of Prim's Algorithm is $O(V^2)$ if adjacency matrix is used.

- If adjacency list is used, then the time complexity of Prim's algorithm can be reduced to O(E Log V) with the help of binary heap and O(E + V Log V) with the help of Fibonacci heap.

Pseudocode (when we use Binary Heap)



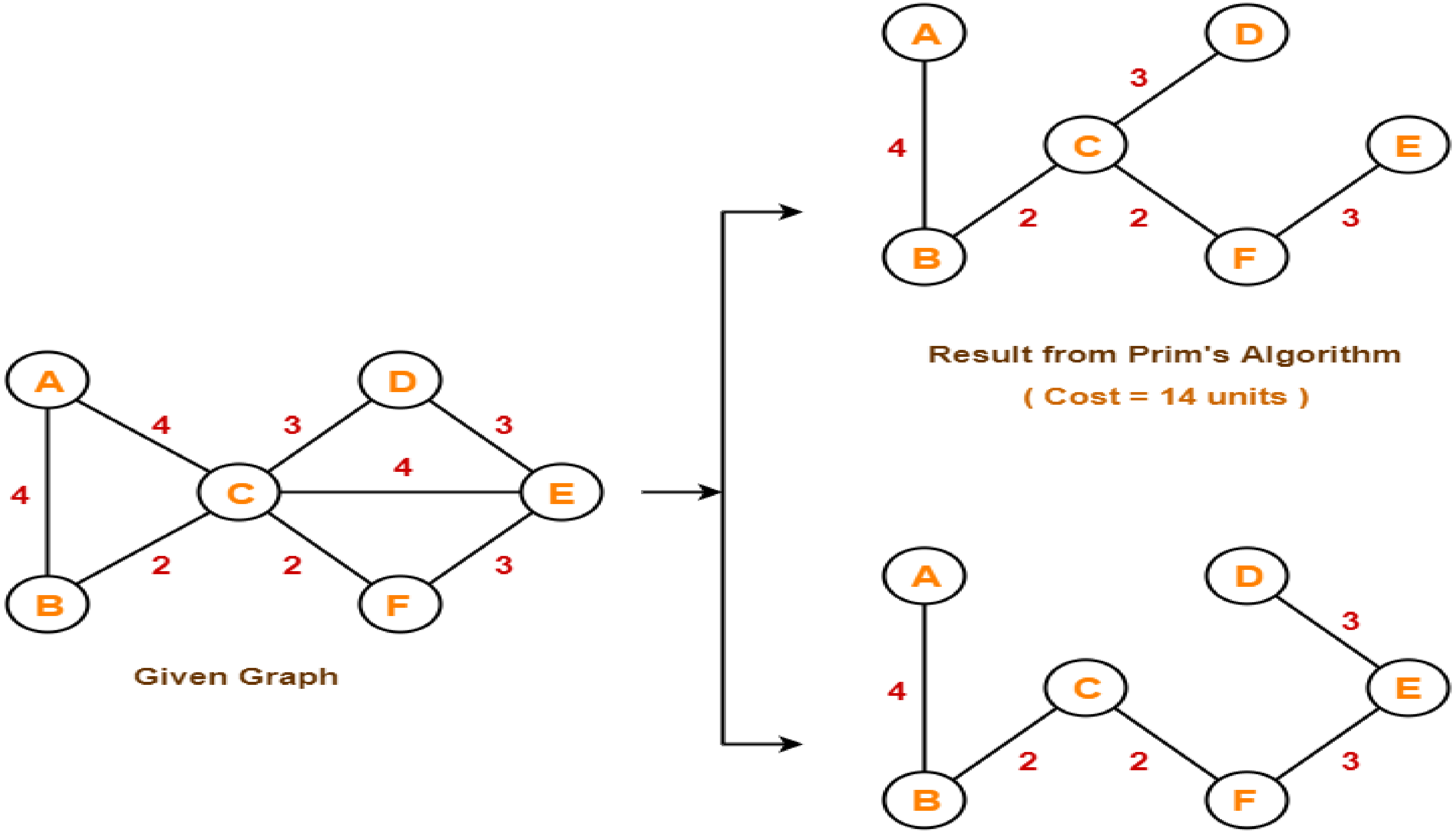
```
PRIM($V, E, w, r$)
$Q \leftarrow \emptyset$
for each $u \in V$
    do $key[u] \leftarrow \infty$
       $\pi[u] \leftarrow$ NIL
       INSERT($Q, u$)
DECREASE-KEY($Q, r, 0$)    $\triangleright$ $key[r] \leftarrow 0$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN($Q$)
       for each $v \in Adj[u]$
           do if $v \in Q$ and $w(u, v) < key[v]$
                then $\pi[v] \leftarrow u$
                     DECREASE-KEY($Q, v, w(u, v)$)
```

O(log V)

O(log V)

→ O(E log V + V log V) = O(E log V)

Kruskal’s Algorithm vs Prim’s Algorithm



Given Graph

Result from Prim's Algorithm

(Cost = 14 units)

Result from Kruskal's Algorithm

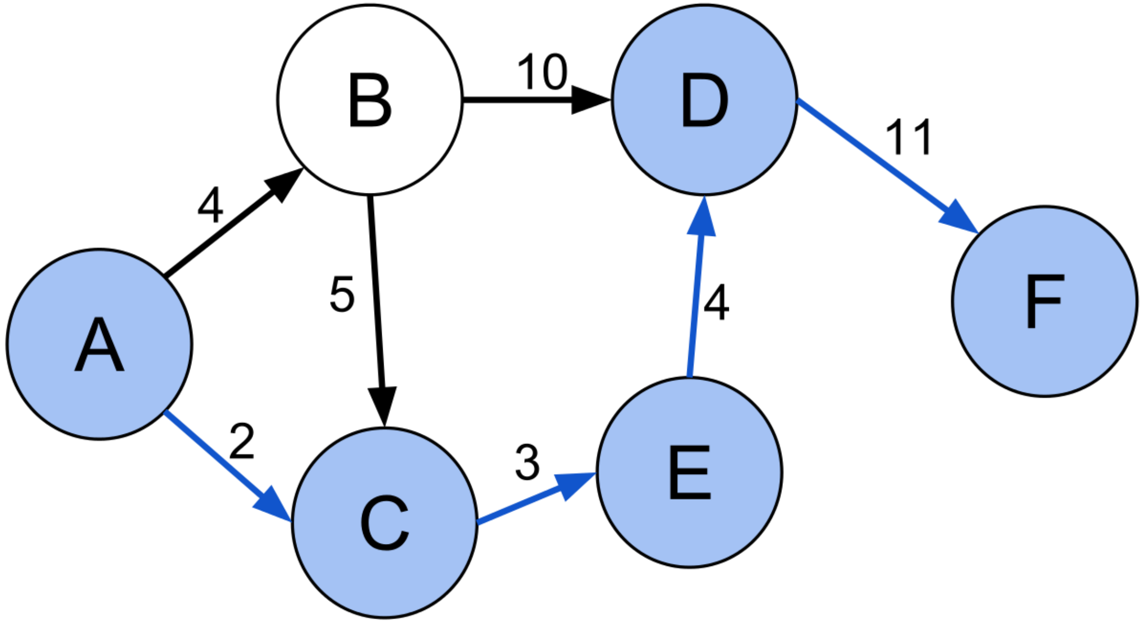
(Cost = 14 units)

Shortest Path Problem

Problem Statement

Find a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimized.

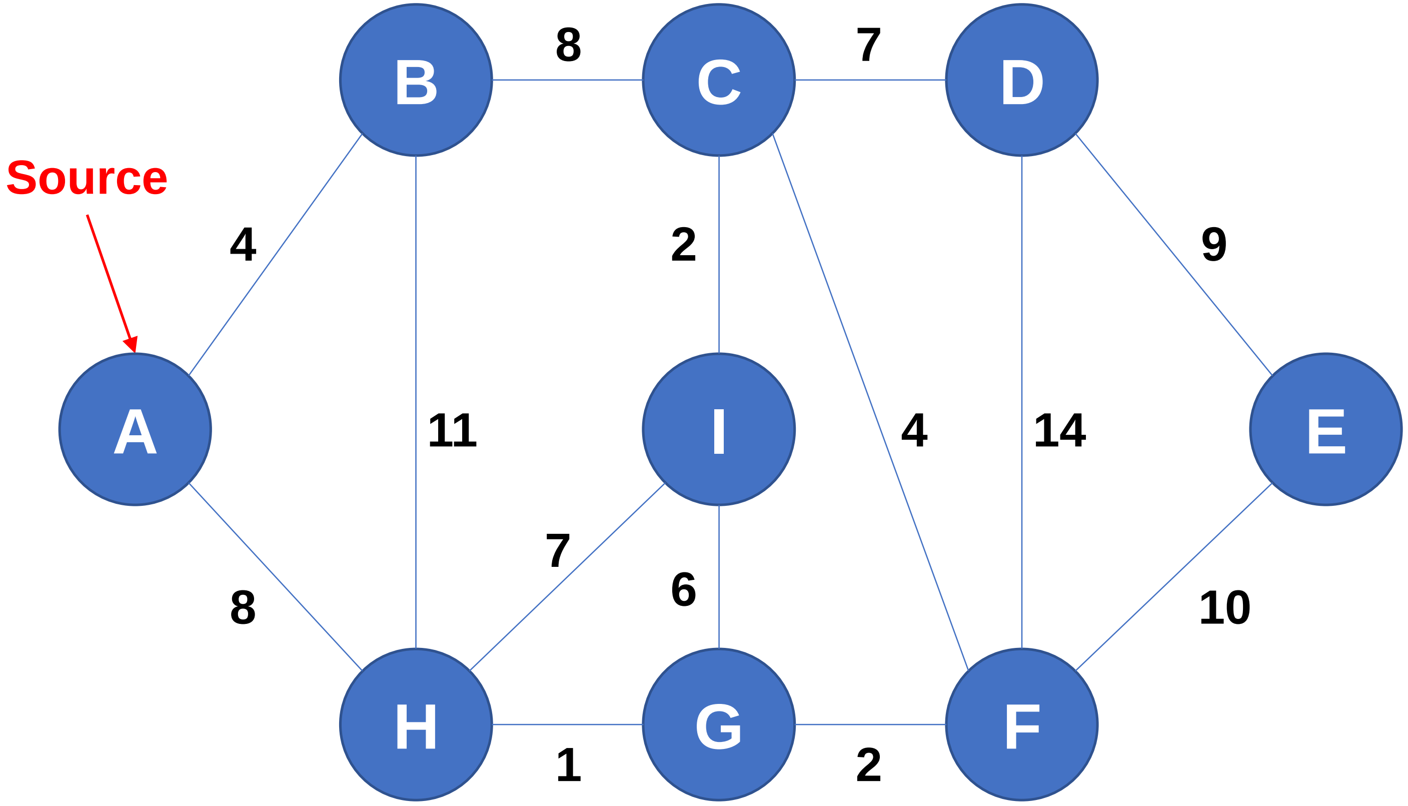
A graph is a series of nodes connected by edges. Graphs can be weighted and directional.



Shortest Path Algorithms

- **Dijkstra's algorithm: solves the single-source shortest path problem with non-negative edge weight.**
- **Bellman–Ford algorithm: solves the single-source problem if edge weights may be negative.**
- **A* search algorithm: solves for single pair shortest path using heuristics to try to speed up the search.**
- **Floyd–Warshall algorithm: solves all pairs shortest paths.**
- **Johnson's algorithm: solves all pairs shortest paths, and may be faster than Floyd–Warshall on sparse graphs.**
- **Viterbi algorithm: solves the shortest stochastic path problem with an additional probabilistic weight on each node.**

Dijkstra's algorithm

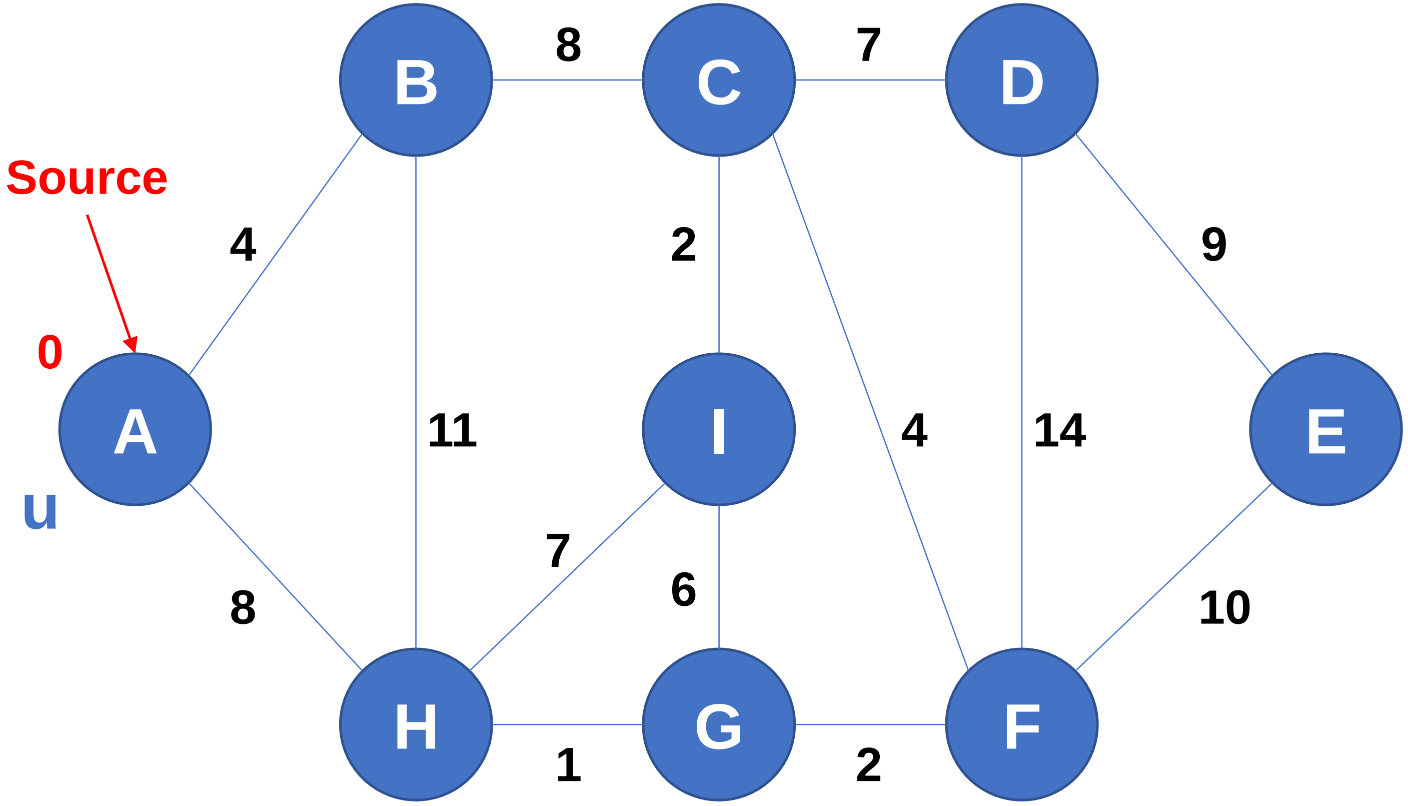


distance

parent

Vertex	d	π
A	0	NIL
B	∞	NIL
C	∞	NIL
D	∞	NIL
E	∞	NIL
F	∞	NIL
G	∞	NIL
H	∞	NIL
I	∞	NIL

Dijkstra's algorithm



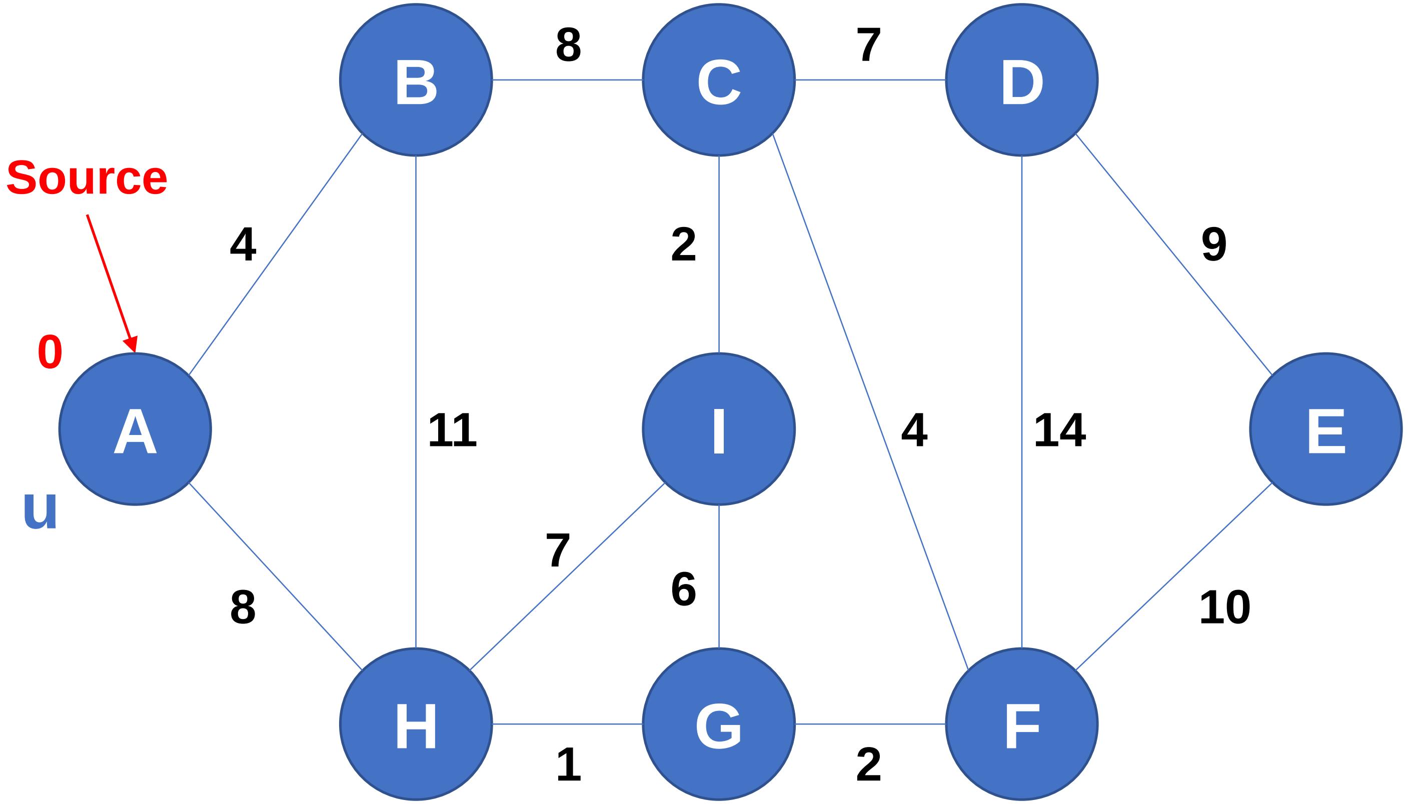
Update B: d[u]+4=4 < ∞

Update H: d[u]+8=8 < ∞

distance parent

Vertex	d	π
✗ A	0	NIL
B	∞	NIL
C	∞	NIL
D	∞	NIL
E	∞	NIL
F	∞	NIL
G	∞	NIL
H	∞	NIL
I	∞	NIL

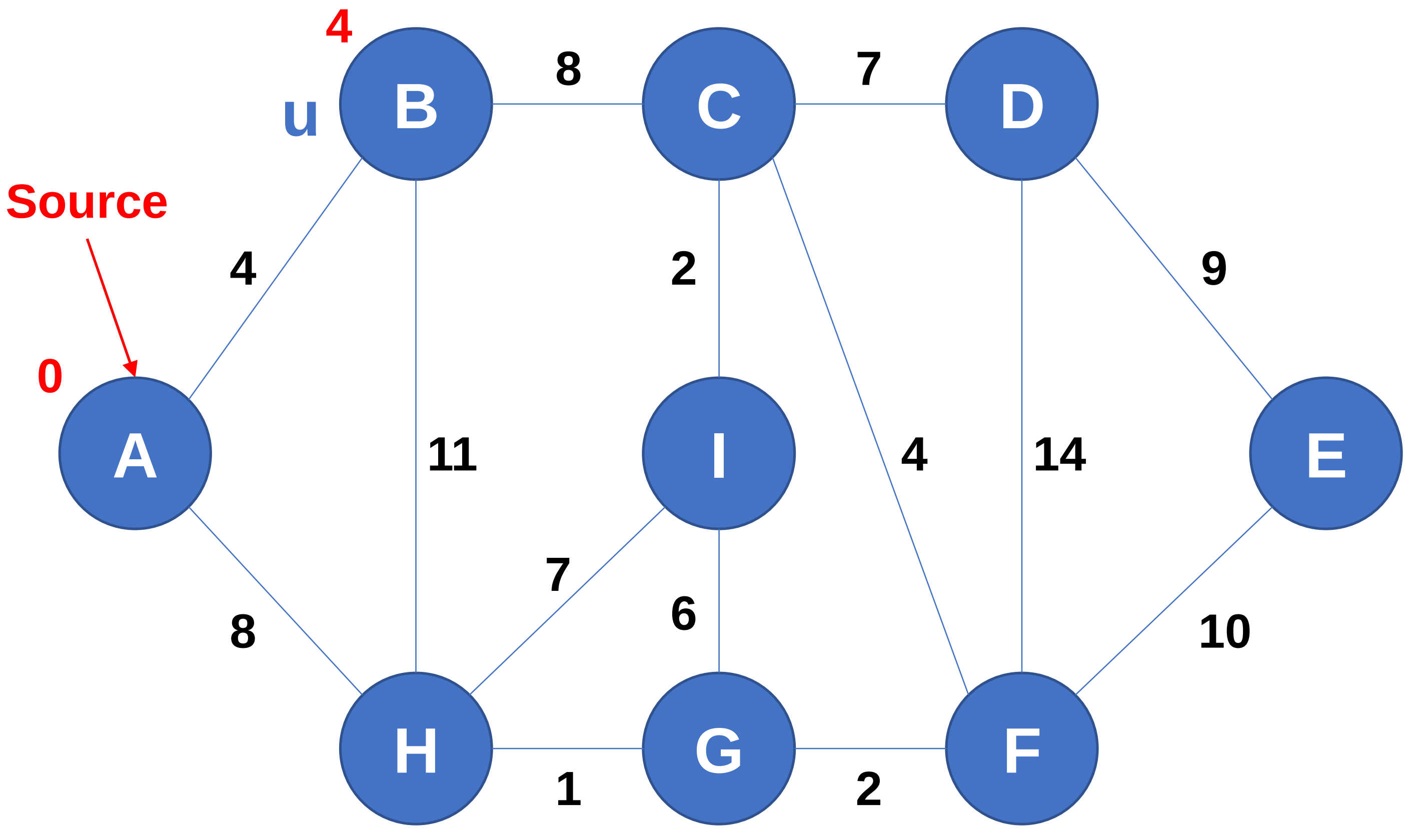
Dijkstra's algorithm



Update B: d[u]+4=4 < ∞
Update H: d[u]+8=8 < ∞

Vertex	d (distance)	π (parent)
~~A~~	0	NIL
B	4	A
C	∞	NIL
D	∞	NIL
E	∞	NIL
F	∞	NIL
G	∞	NIL
H	8	A
I	∞	NIL

Dijkstra's algorithm

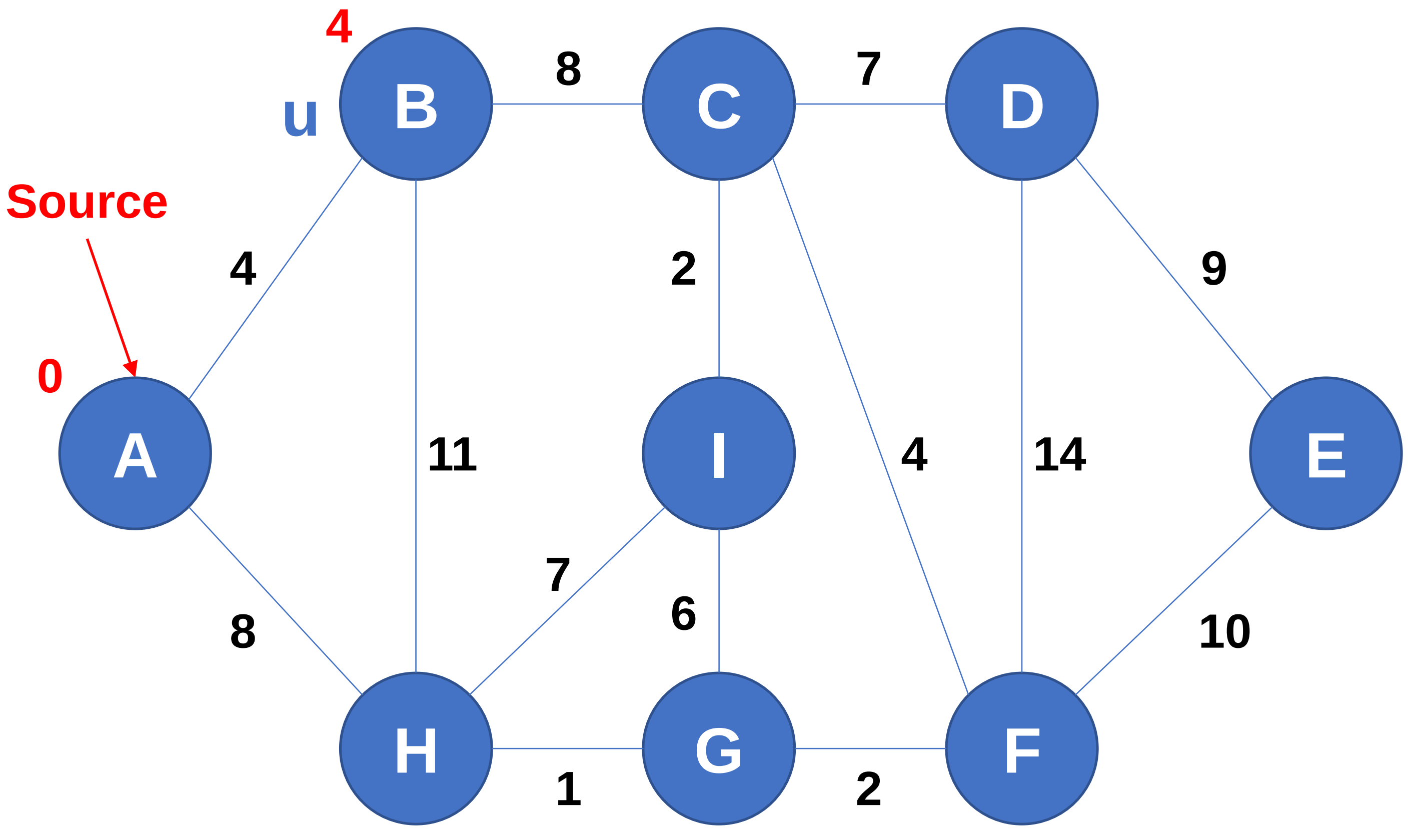


	distance	parent
Vertex	d	π
✗ A	0	NIL
✗ B	4	A
C	∞	NIL
D	∞	NIL
E	∞	NIL
F	∞	NIL
G	∞	NIL
H	8	A
I	∞	NIL

Update C: d[u]+8=12 < ∞

Update H: d[u]+11=15 > 8

Dijkstra's algorithm

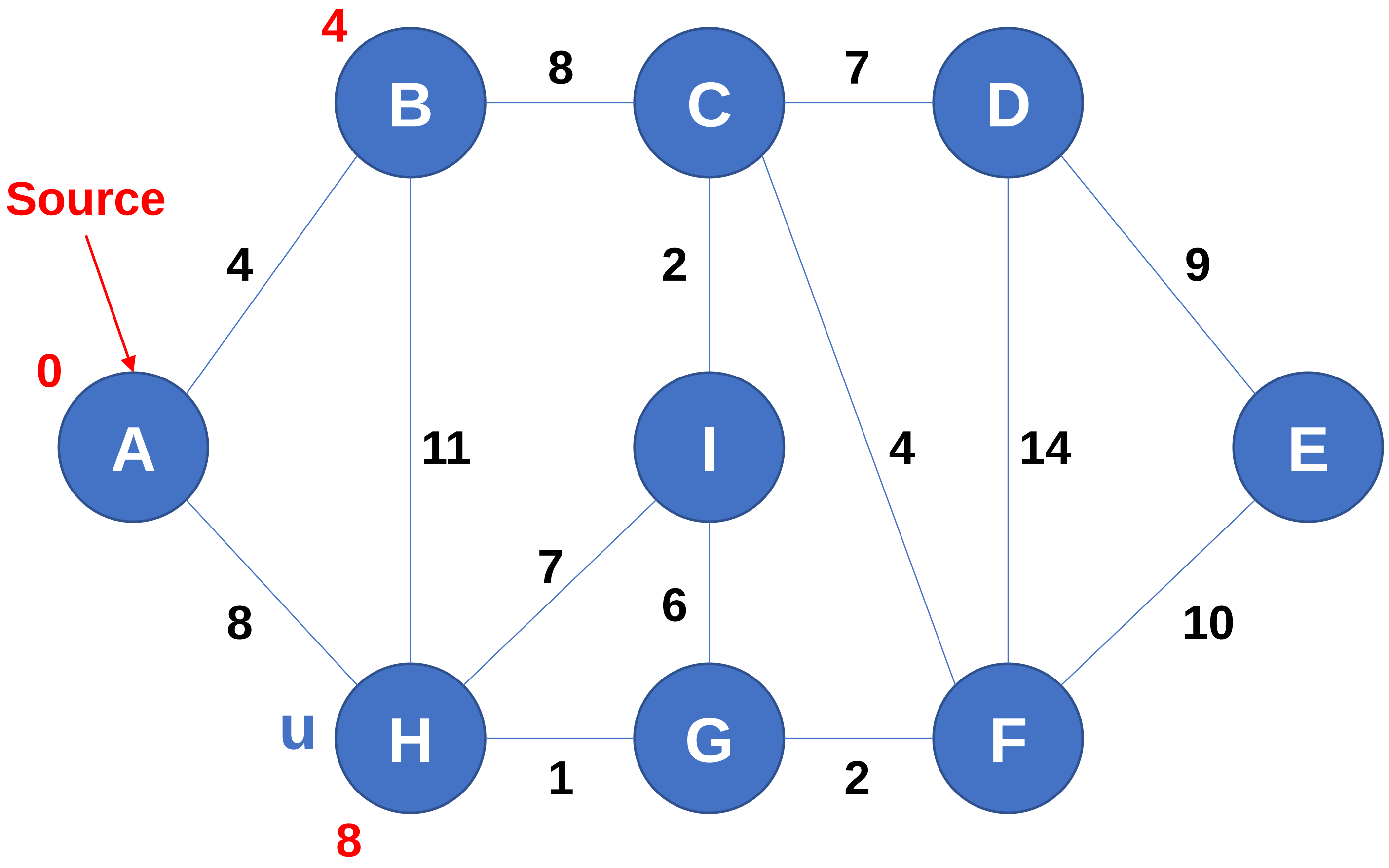


Update C: d[u]+8=12 < ∞

Update H: d[u]+11=15 > 8

Vertex	d distance	π parent
✗ A	0	NIL
✗ B	4	A
C	12	B
D	∞	NIL
E	∞	NIL
F	∞	NIL
G	∞	NIL
H	8	A
I	∞	NIL

Dijkstra's algorithm



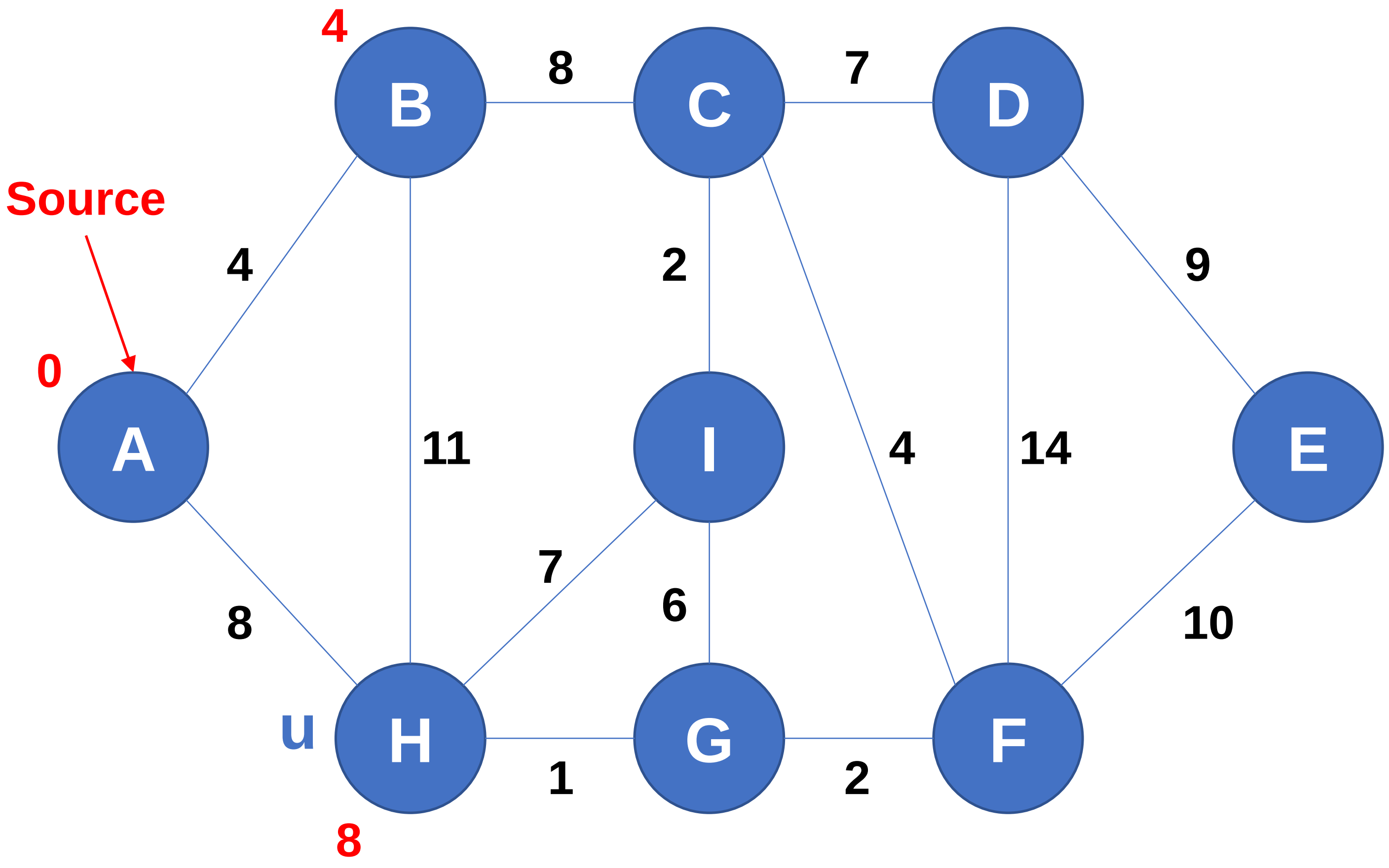
Update G: d[u]+1=9 < ∞

Update I: d[u]+7=15 < ∞

distance   parent

Vertex	d	π
✗ A	0	NIL
✗ B	4	A
C	12	B
D	∞	NIL
E	∞	NIL
F	∞	NIL
G	∞	NIL
✗ H	8	A
I	∞	NIL

Dijkstra's algorithm

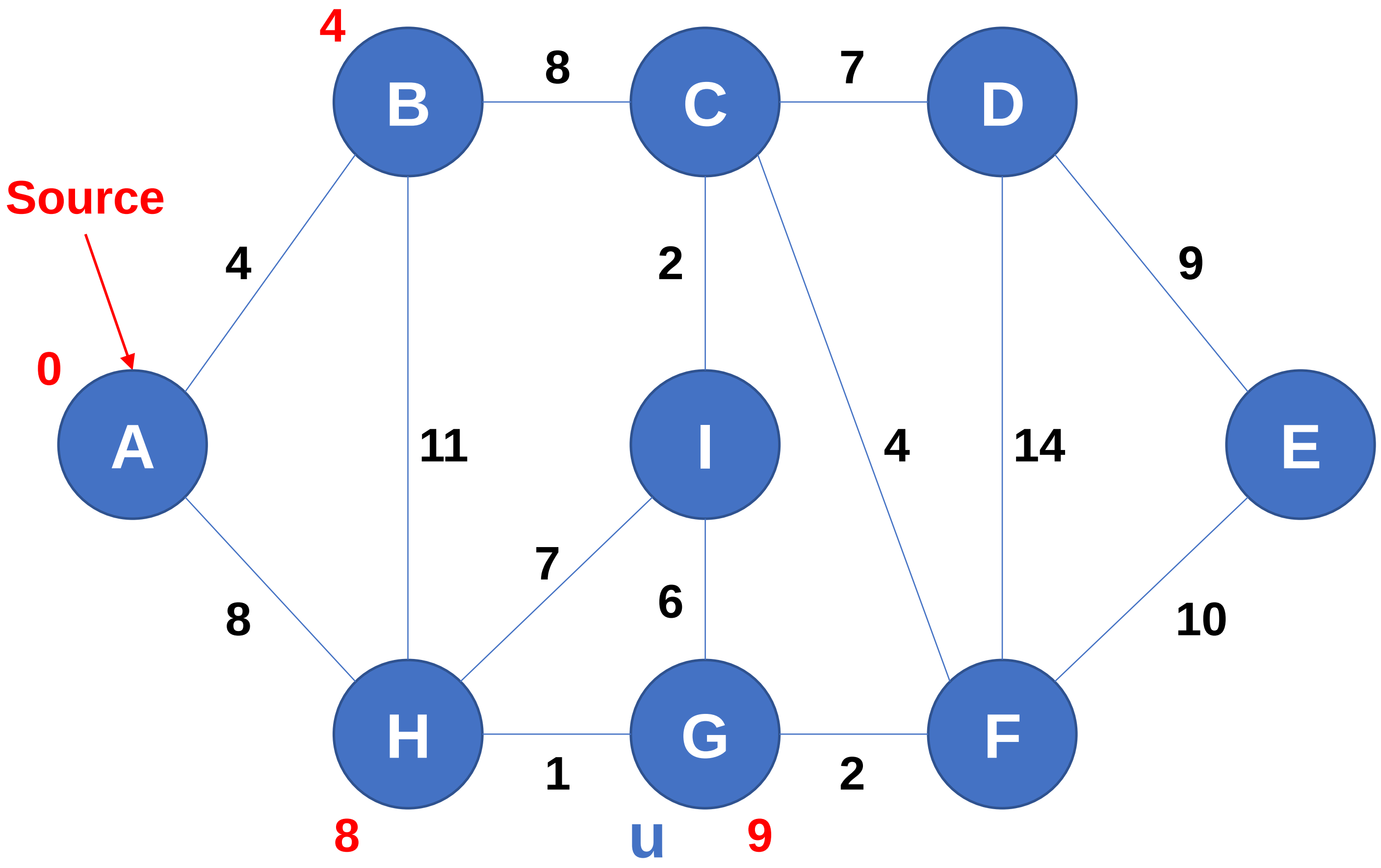


Update G: d[u]+1=9 < ∞

Update I: d[u]+7=15 < ∞

	distance	parent
Vertex	d	π
✗ A	0	NIL
✗ B	4	A
C	12	B
D	∞	NIL
E	∞	NIL
F	∞	NIL
G	9	H
✗ H	8	A
I	15	H

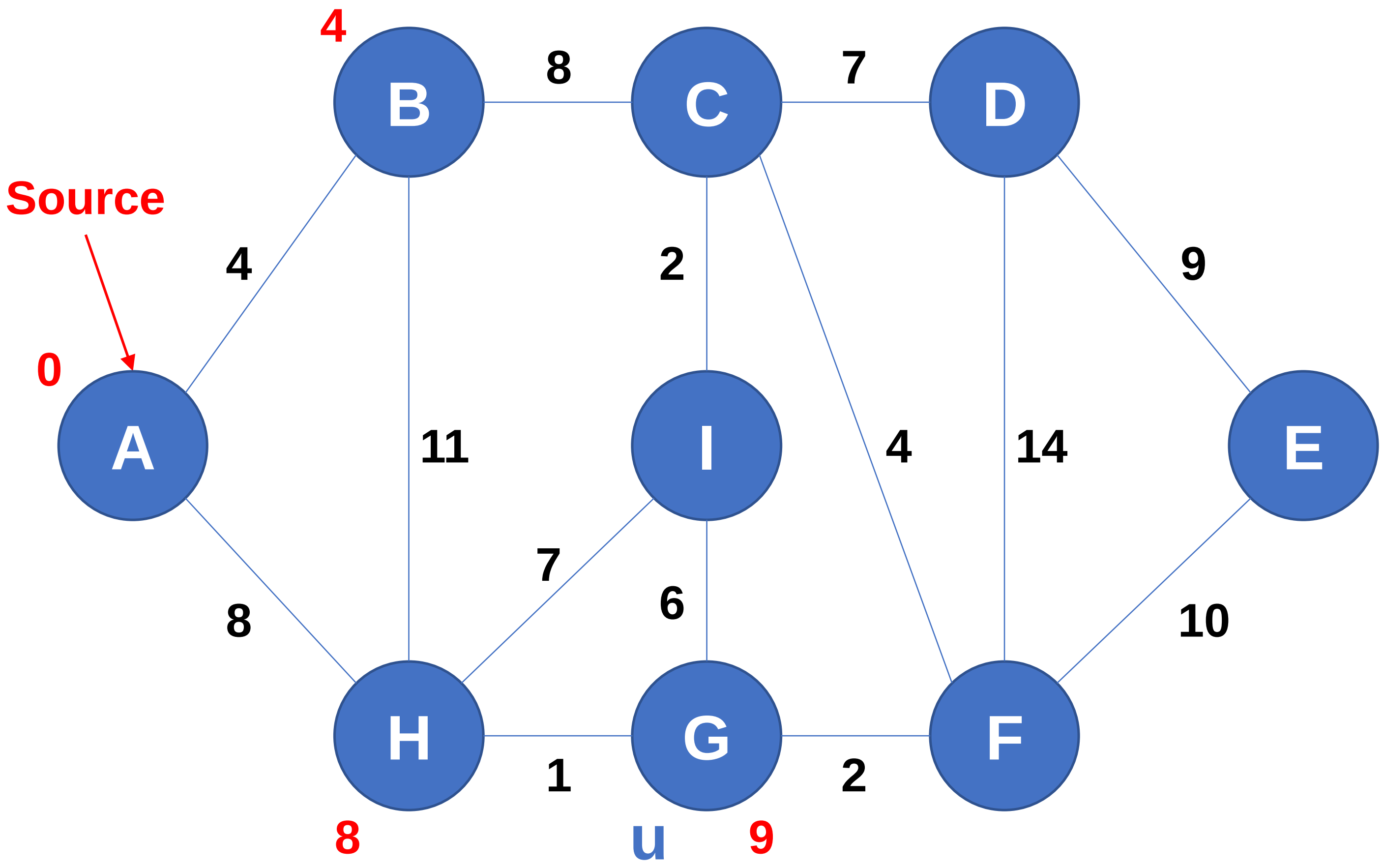
Dijkstra's algorithm



Update F: d[u]+2=11 < ∞
Update I: d[u]+6=15 = 15

	distance	parent
Vertex	d	π
✗ A	0	NIL
✗ B	4	A
C	12	B
D	∞	NIL
E	∞	NIL
F	∞	NIL
✗ G	9	H
✗ H	8	A
I	15	H

Dijkstra's algorithm



Update F: d[u]+2=11 < ∞

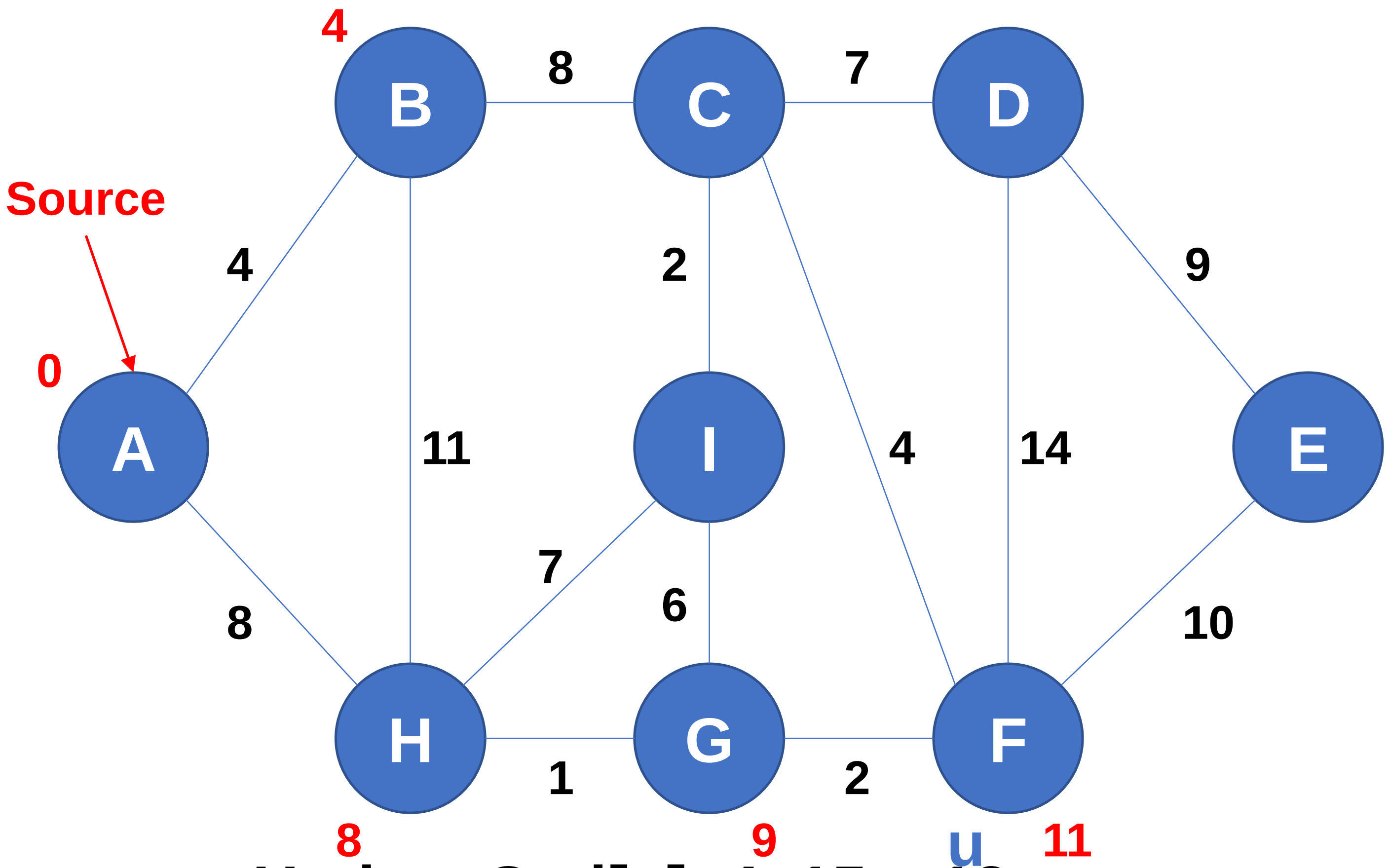
Update I: d[u]+6=15 = 15

distance

parent

Vertex	d	π
✗ A	0	NIL
✗ B	4	A
C	12	B
D	∞	NIL
E	∞	NIL
F	11	G
✗ G	9	H
✗ H	8	A
I	15	H

Dijkstra's algorithm

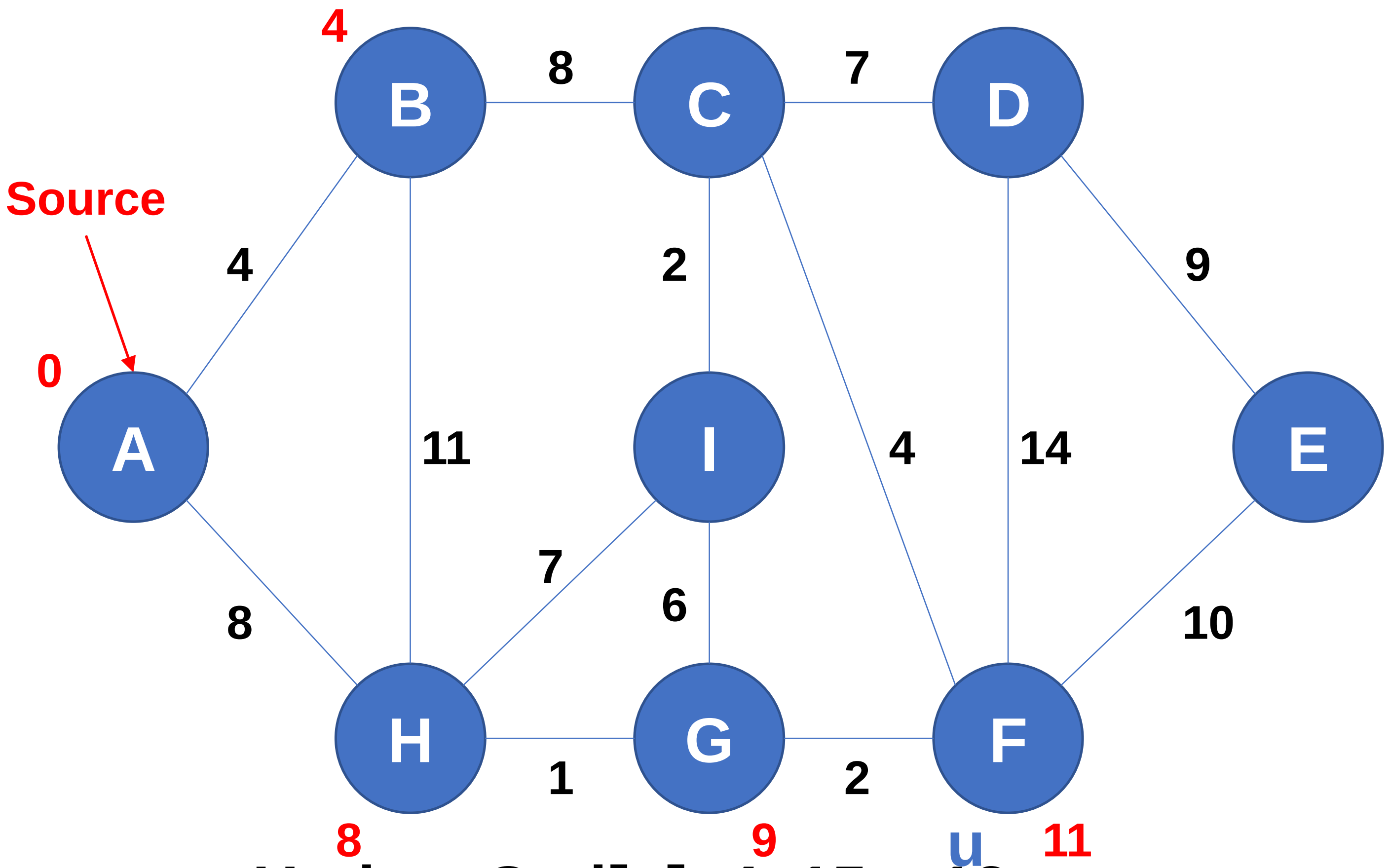


Update C: d[u]+4=15 > 12
Update D: d[u]+14=25 < ∞
Update E: d[u]+10=21 < ∞

distance parent

Vertex	d	π
✗ A	0	NIL
✗ B	4	A
C	12	B
D	∞	NIL
E	∞	NIL
✗ F	11	G
✗ G	9	H
✗ H	8	A
I	15	H

Dijkstra's algorithm

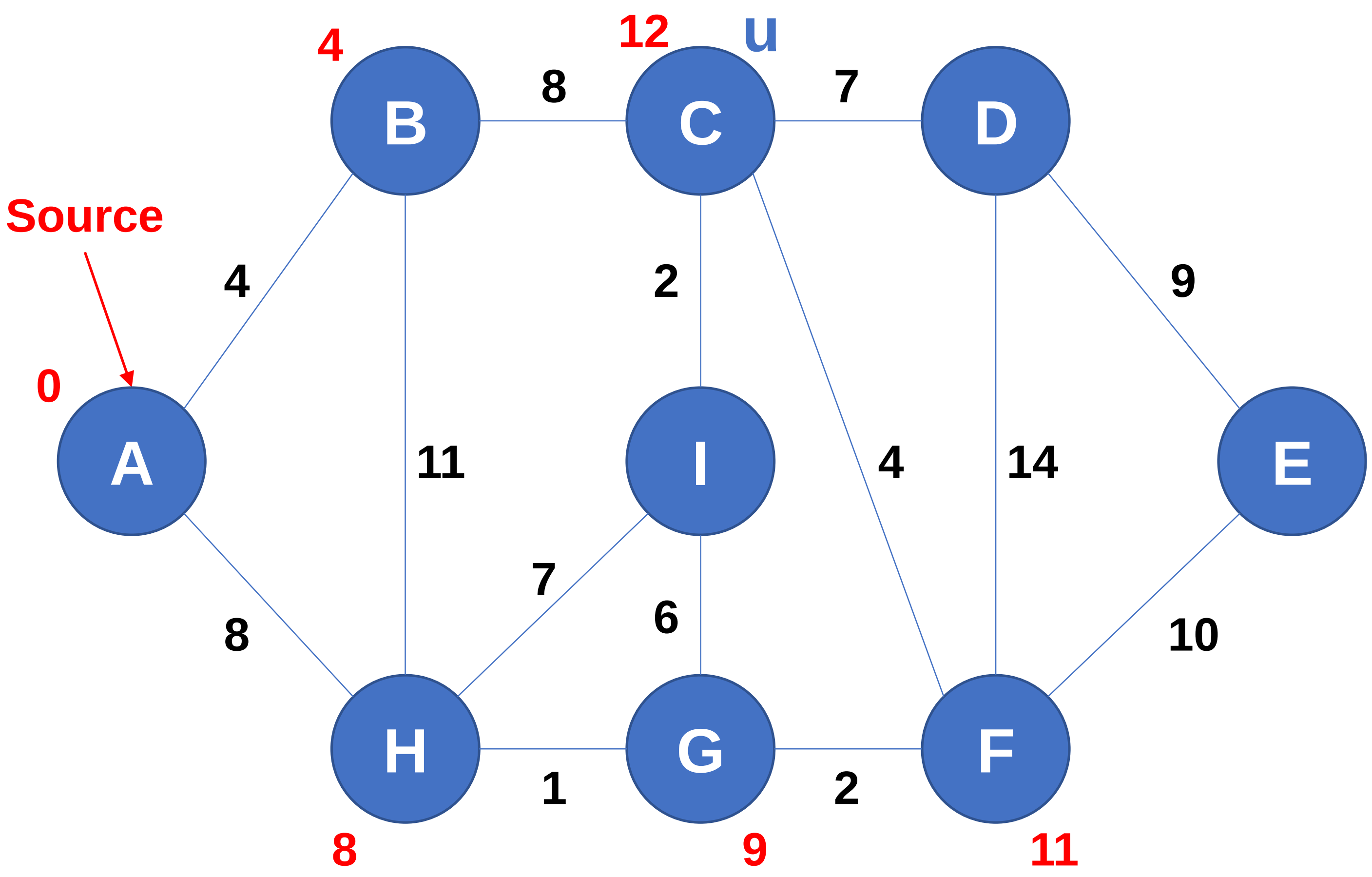


Update C: d[u]+4=15 > 12
Update D: d[u]+14=25 < ∞
Update E: d[u]+10=21 < ∞

distance parent

Vertex	d	π
✗ A	0	NIL
✗ B	4	A
C	12	B
D	25	F
E	21	F
✗ F	11	G
✗ G	9	H
✗ H	8	A
I	15	H

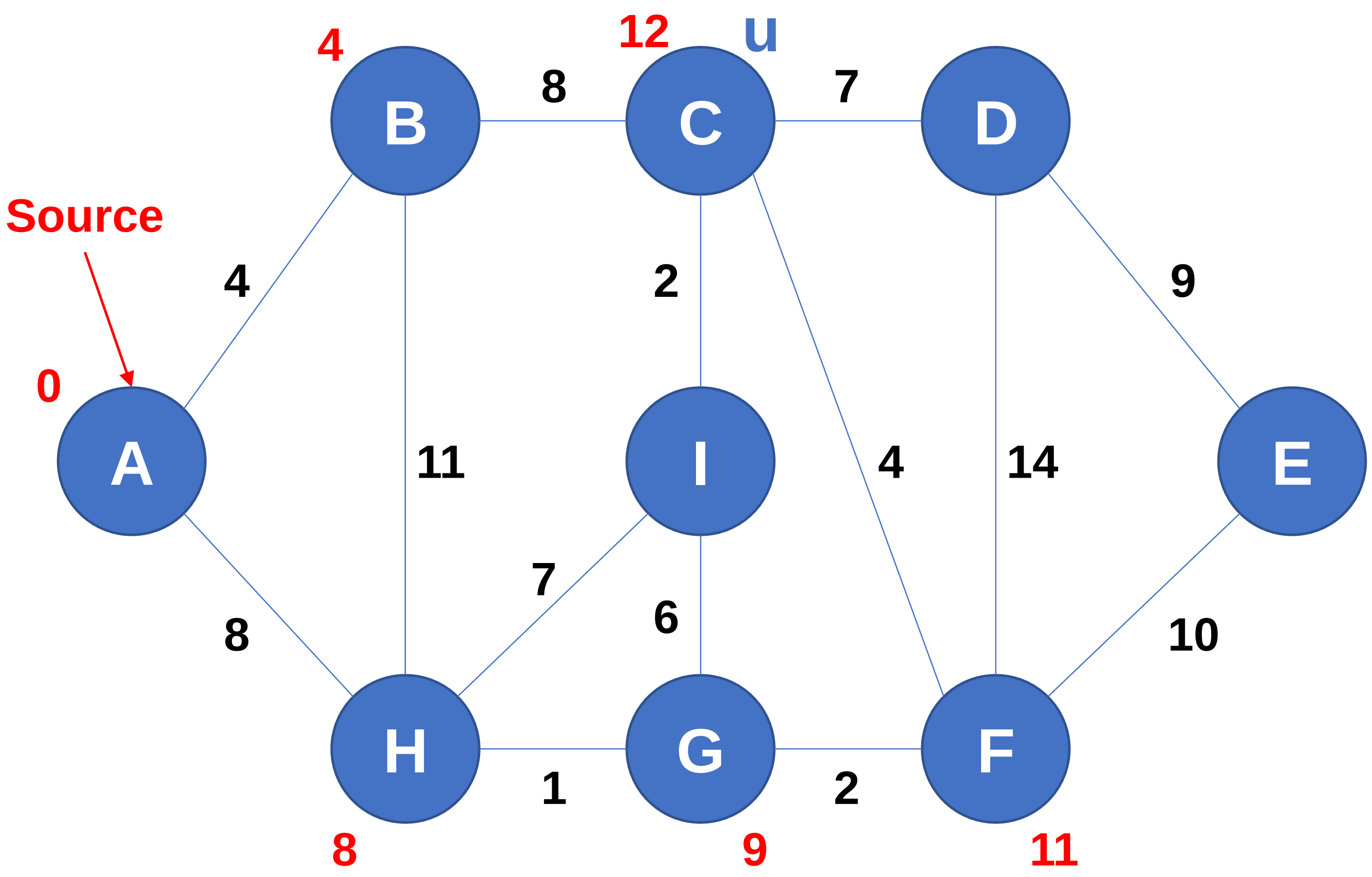
Dijkstra's algorithm



Update D: d[u]+7=19 < 25
Update I: d[u]+2=14 < 15

	distance	parent
Vertex	d	π
✗ A	0	NIL
✗ B	4	A
✗ C	12	B
D	25	F
E	21	F
✗ F	11	G
✗ G	9	H
✗ H	8	A
I	15	H

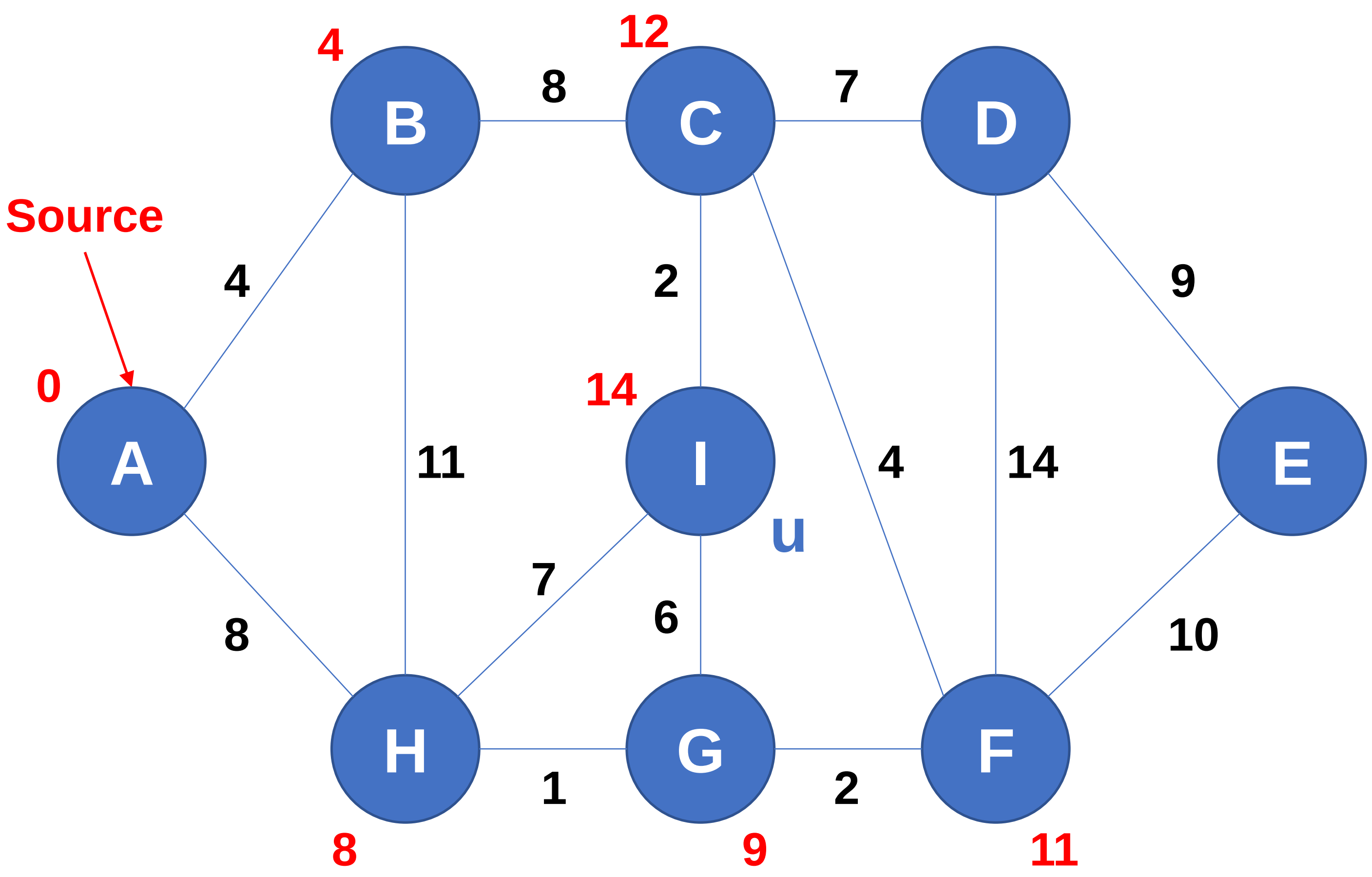
Dijkstra's algorithm



Update D: d[u]+7=19 < 25
Update I: d[u]+2=14 < 15

	distance	parent
Vertex	**d**	**π**
✗ A	0	NIL
✗ B	4	A
✗ C	12	B
D	19	C
E	21	F
✗ F	11	G
✗ G	9	H
✗ H	8	A
I	14	C

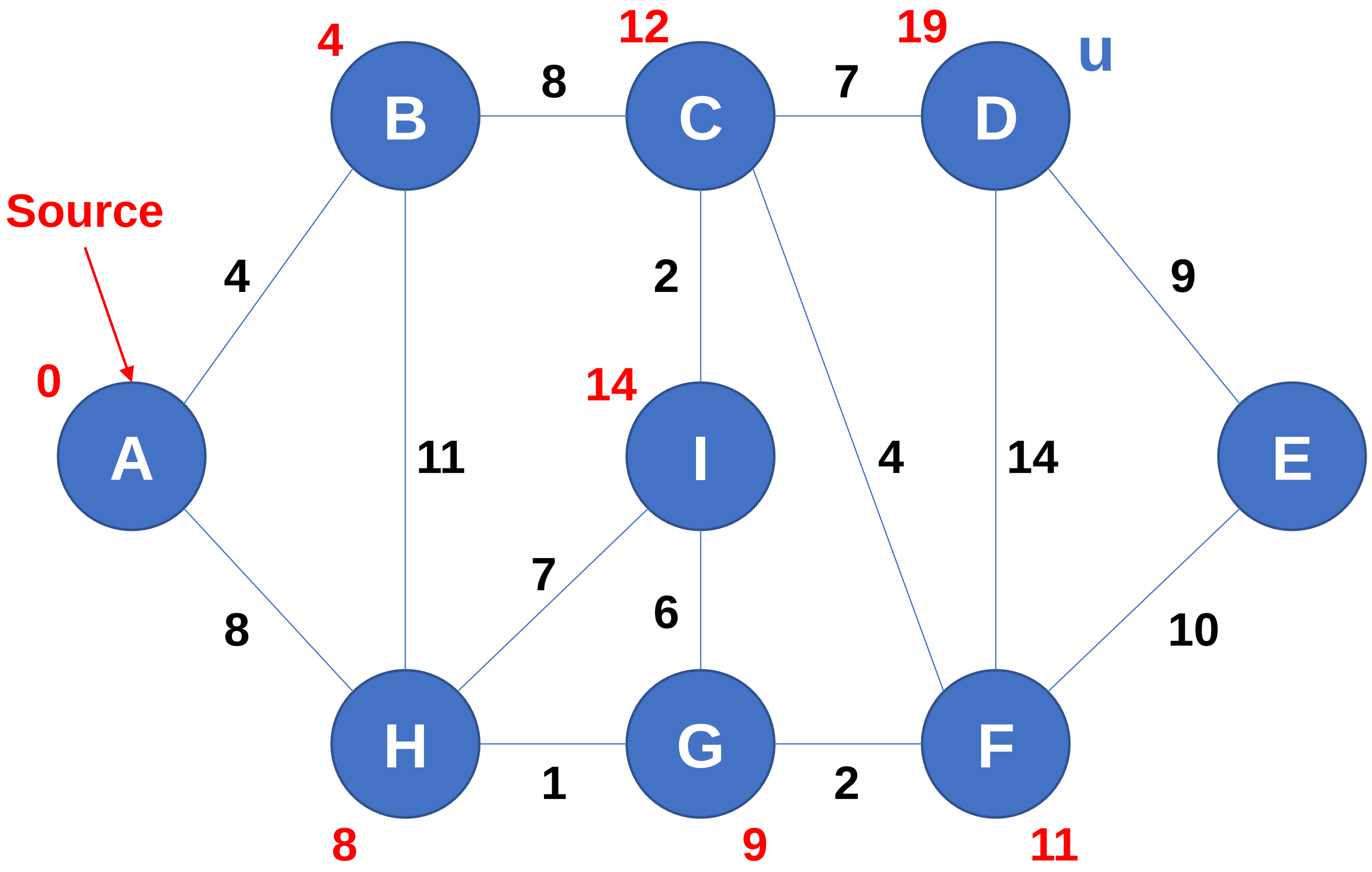
Dijkstra's algorithm



distance parent

Vertex	d	π
✗ A	0	NIL
✗ B	4	A
✗ C	12	B
D	19	C
E	21	F
✗ F	11	G
✗ G	9	H
✗ H	8	A
✗ I	14	C

Dijkstra's algorithm

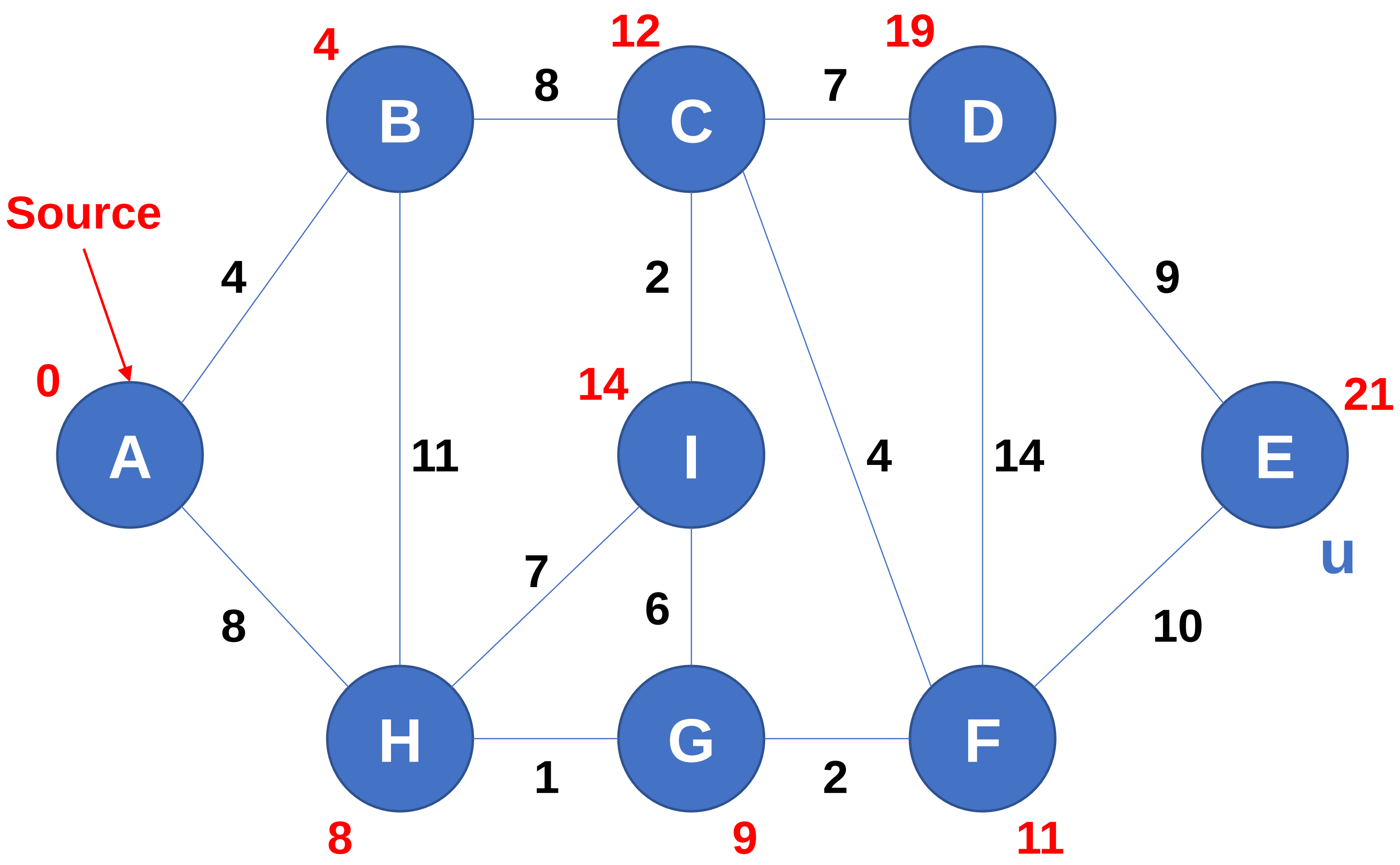


distance parent

Vertex	d	π
✗ A	0	NIL
✗ B	4	A
✗ C	12	B
✗ D	19	C
E	21	F
✗ F	11	G
✗ G	9	H
✗ H	8	A
✗ I	14	C

Update E: d[u]+9=28 > 21

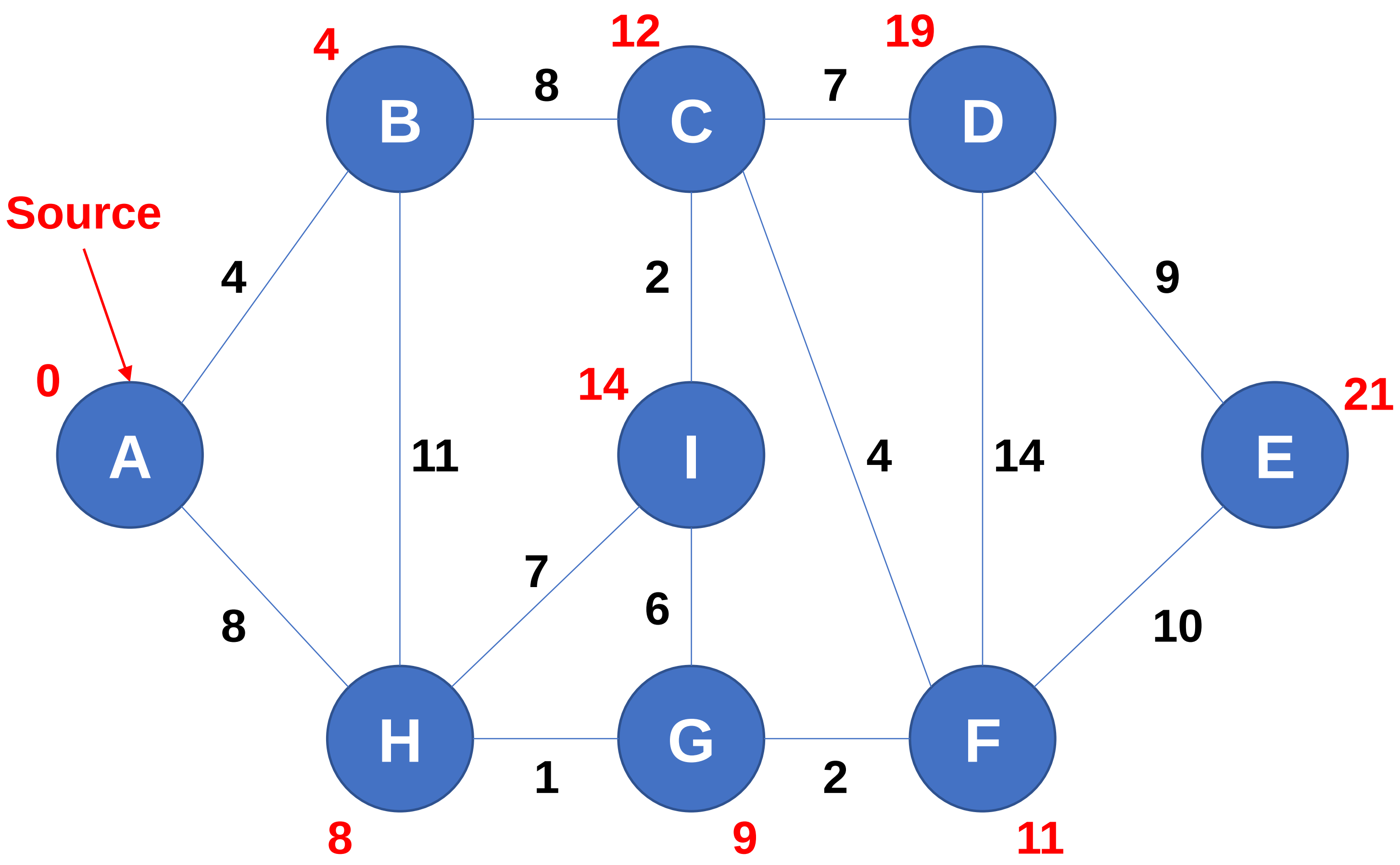
Dijkstra's algorithm



distance ↑ parent ↑

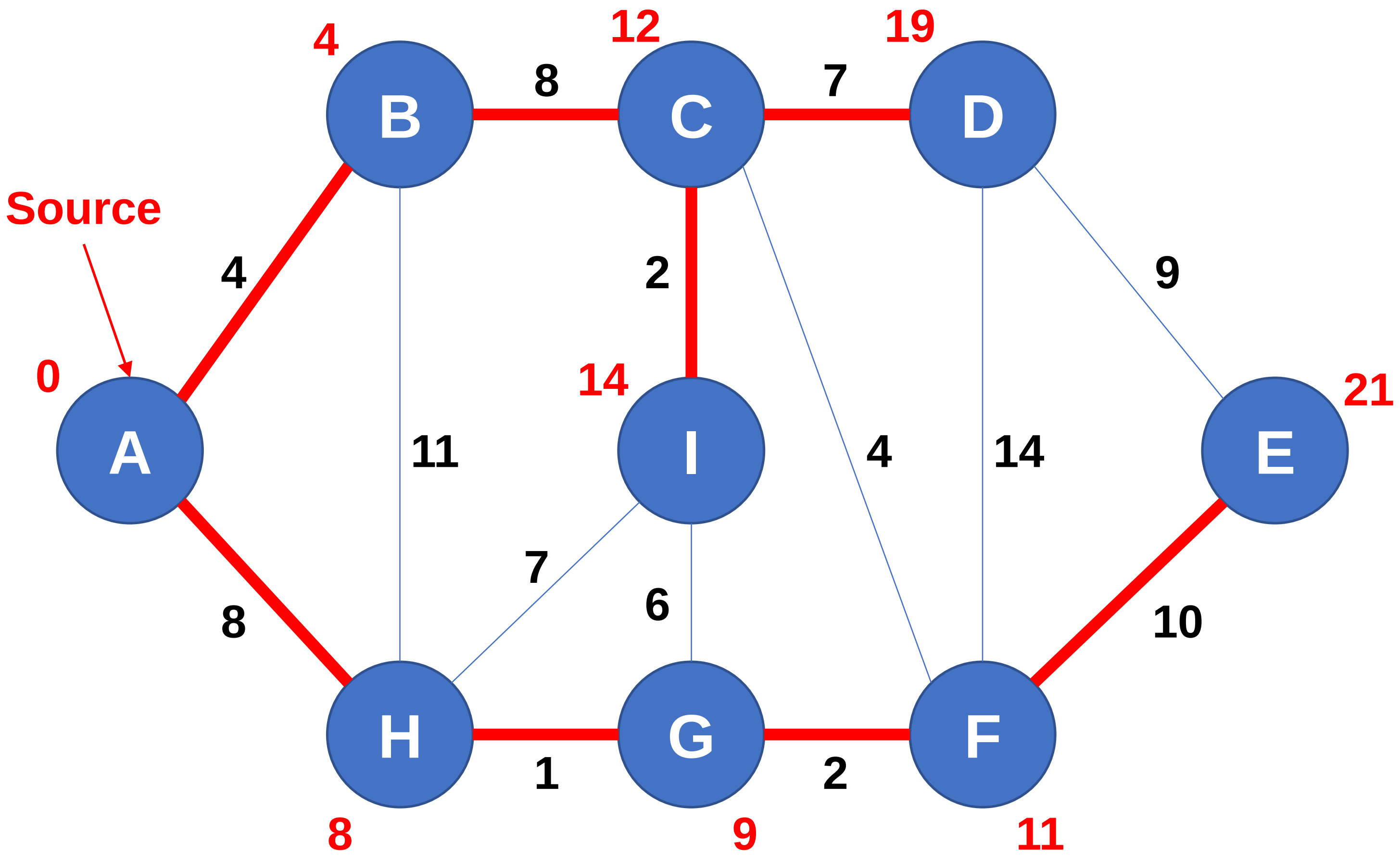
Vertex	d	π
✗ A	0	NIL
✗ B	4	A
✗ C	12	B
✗ D	19	C
✗ E	21	F
✗ F	11	G
✗ G	9	H
✗ H	8	A
✗ I	14	C

Dijkstra's algorithm



	distance	parent
Vertex	d	π
✗ A	0	NIL
✗ B	4	A
✗ C	12	B
✗ D	19	C
✗ E	21	F
✗ F	11	G
✗ G	9	H
✗ H	8	A
✗ I	14	C

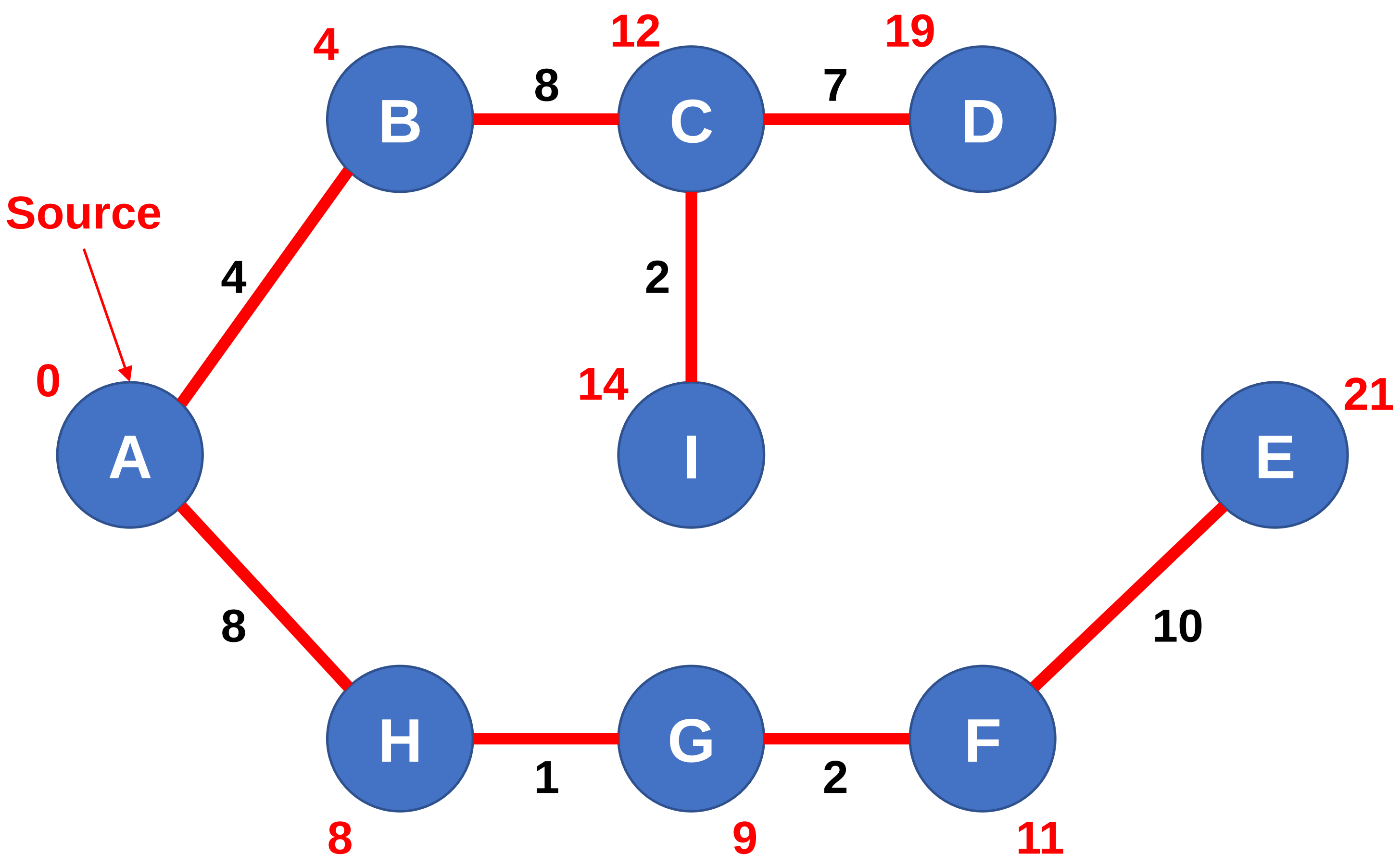
Dijkstra's algorithm



distance parent

Vertex	d	π
✗ A	0	NIL
✗ B	4	A
✗ C	12	B
✗ D	19	C
✗ E	21	F
✗ F	11	G
✗ G	9	H
✗ H	8	A
✗ I	14	C

Dijkstra's algorithm



distance

parent

Vertex	d	π
✗ A	0	NIL
✗ B	4	A
✗ C	12	B
✗ D	19	C
✗ E	21	F
✗ F	11	G
✗ G	9	H
✗ H	8	A
✗ I	14	C

Complexity

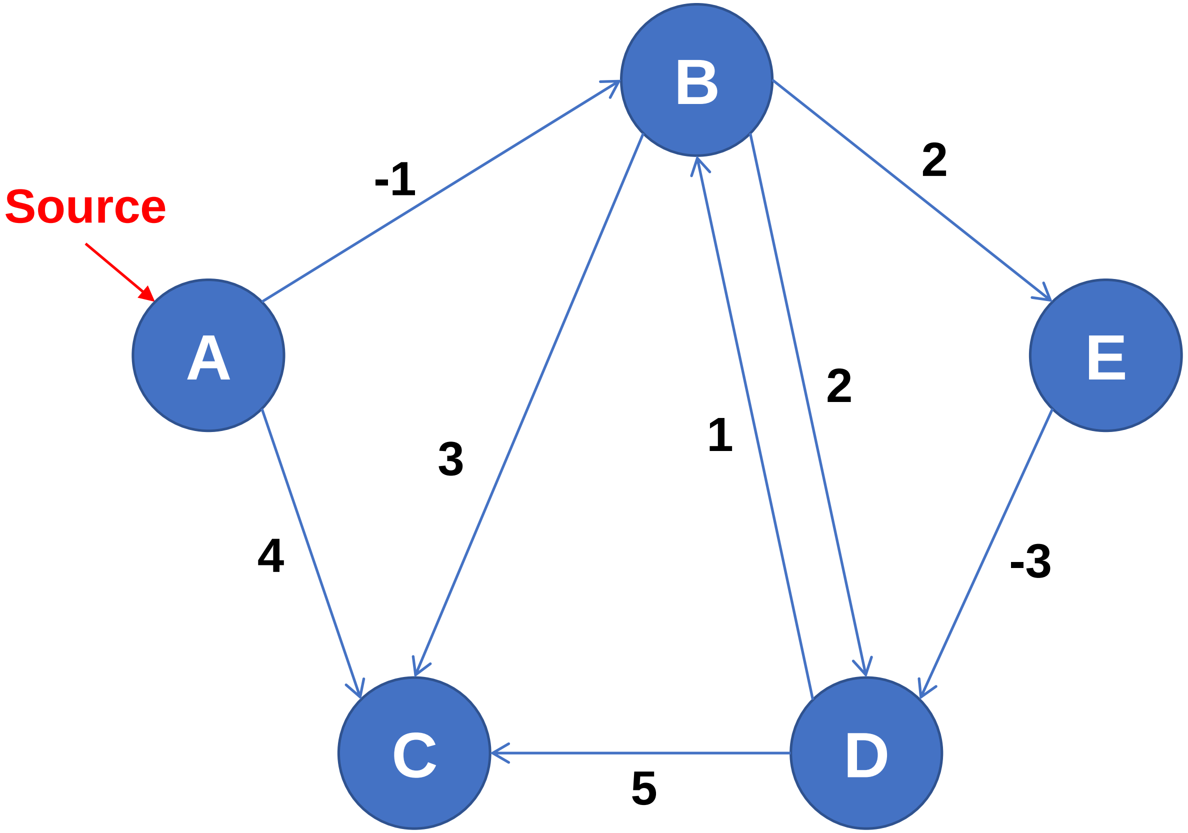
Time Complexity: O(V^2)

If the input graph is represented using adjacency list, it can be reduced to O(E log V) with the help of binary heap.

Pseudo-Code

```
$d[s] \leftarrow 0$
**for** each $v \in V - \{s\}$:
    **do** $d[v] \leftarrow \infty$
$S \leftarrow \emptyset$
$Q \leftarrow V$
**while** $Q \neq \emptyset$:
    **do** $u \leftarrow \mathrm{Extract} - \mathrm{Min}(Q)$:
        $S \leftarrow S \cup \{u\}$
        **for** each $v \in Adj[u]$:
            **if** $d[v] > d[u] + w(u, v)$:
                $d[v] = d[u] + w(u, v)$
```

Bellman Ford's algorithm

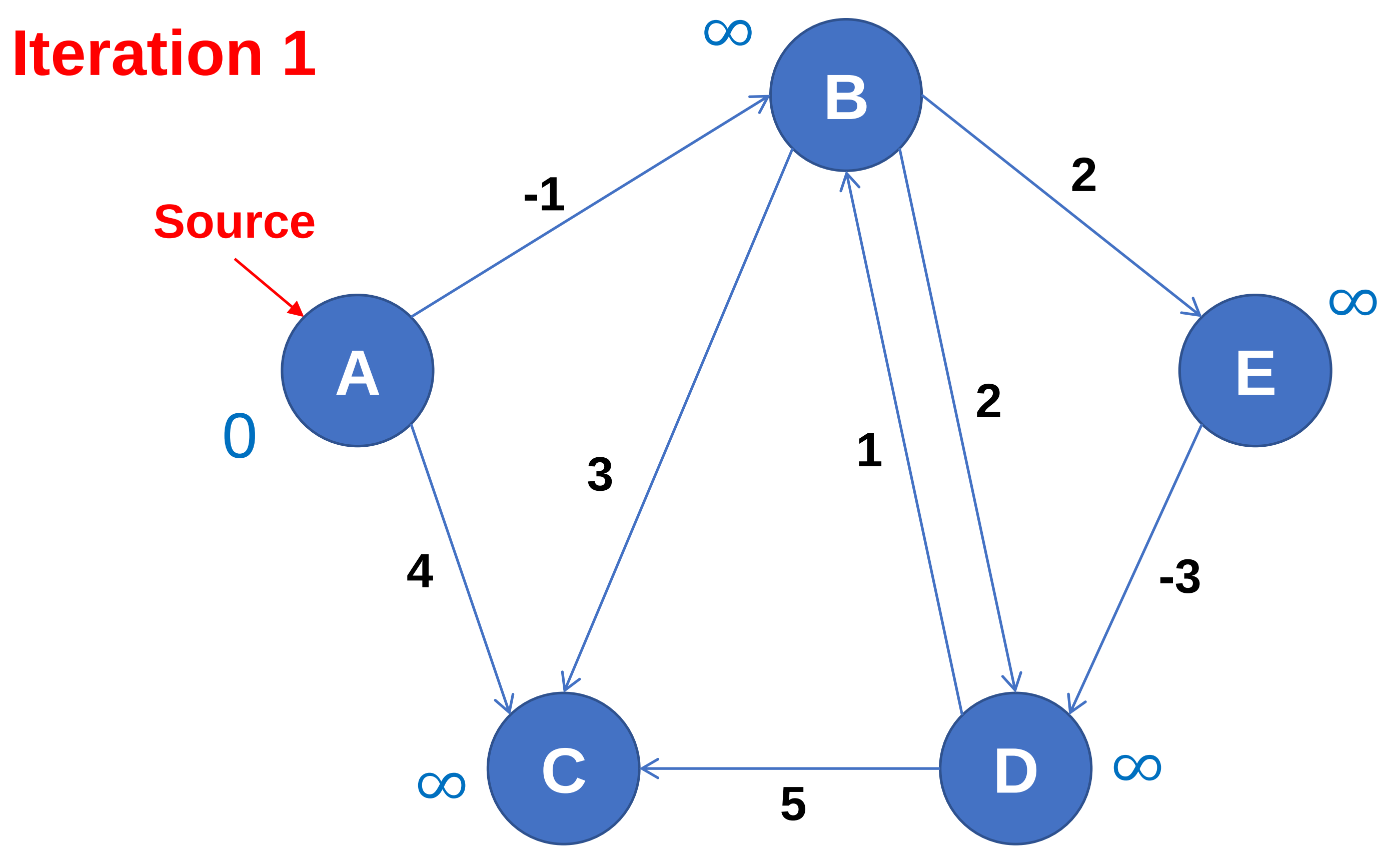


distance parent

Vertex	d	π
A	0	NIL
B	∞	NIL
C	∞	NIL
D	∞	NIL
E	∞	NIL

Let all edges are processed in following order:
(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

Bellman Ford's algorithm



Vertex	d	π
A	0	NIL
B	∞	NIL
C	∞	NIL
D	∞	NIL
E	∞	NIL

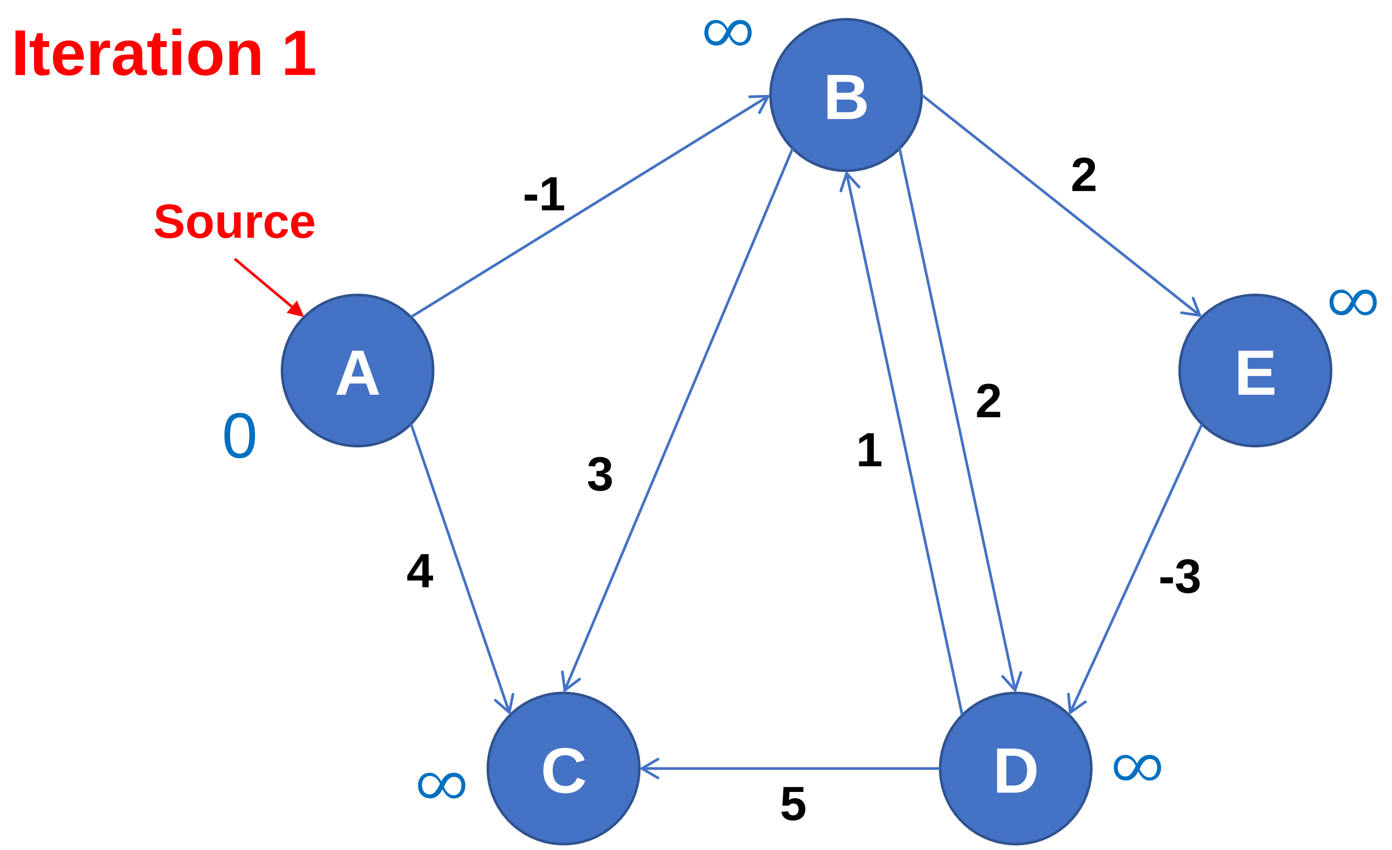
distance ↑ d

parent ↑ π

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, E), (D, B), (B, D): d[u]+edge(u,v)=∞ = ∞

Bellman Ford's algorithm



distance ↑ d

parent ↑ π

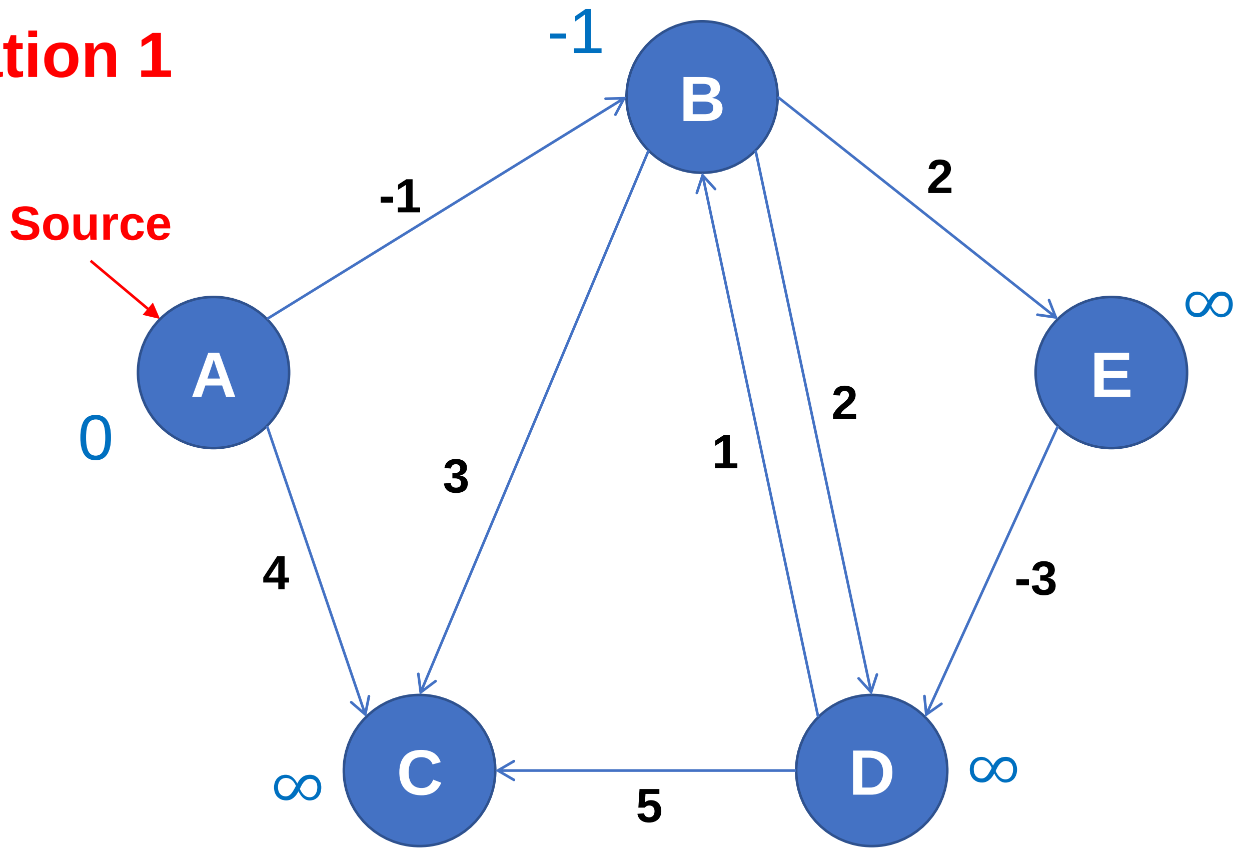
Vertex	d	π
A	0	NIL
B	∞	NIL
C	∞	NIL
D	∞	NIL
E	∞	NIL

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(A, B): d[u]+edge(u,v)=0+(-1) < ∞

Bellman Ford's algorithm

Iteration 1



distance ↑ d
parent ↑ π

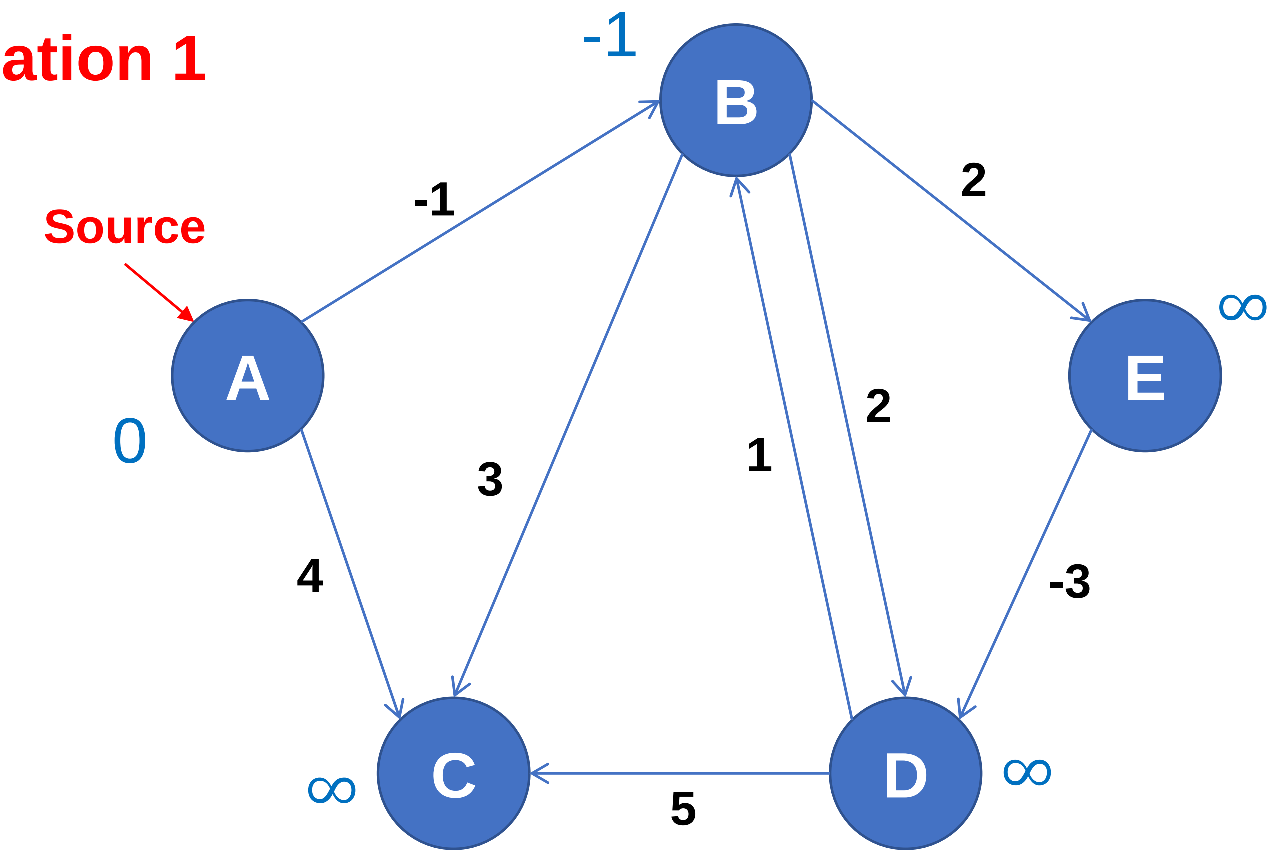
Vertex	d	π
A	0	NIL
B	-1	A
C	∞	NIL
D	∞	NIL
E	∞	NIL

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(A, B): $d[u]+edge(u,v)=0+(-1) < \infty$

Bellman Ford's algorithm

Iteration 1



distance

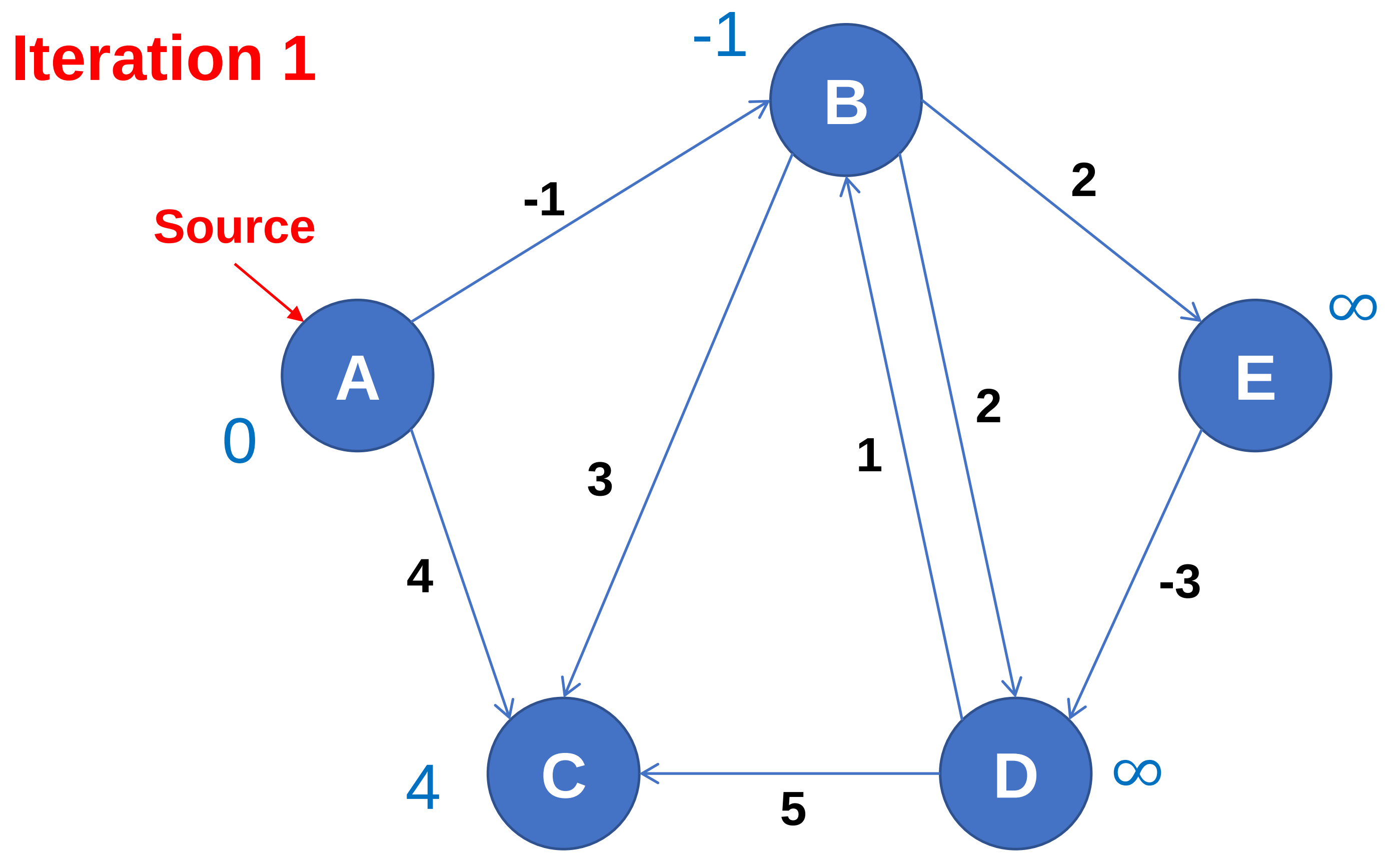
parent

Vertex	d	π
A	0	NIL
B	-1	A
C	∞	NIL
D	∞	NIL
E	∞	NIL

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(A, C): $d[u]+edge(u,v)=0+4 < \infty$

Bellman Ford's algorithm



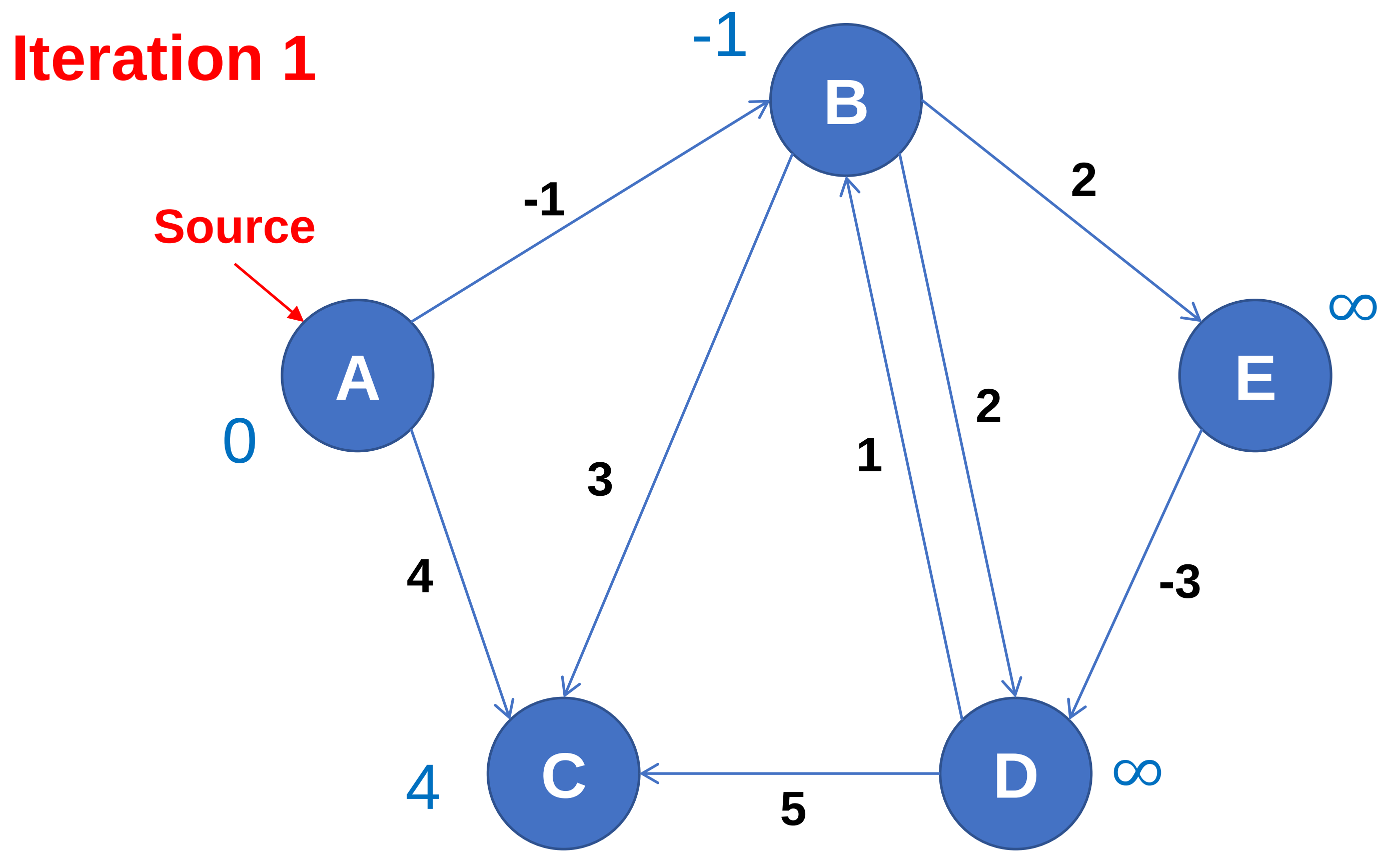
Vertex	d	π
A	0	NIL
B	-1	A
C	4	A
D	∞	NIL
E	∞	NIL

distance → d; parent → π

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(A, C): d[u]+edge(u,v)=0+4 < ∞

Bellman Ford's algorithm



distance ↑ d

parent ↑ π

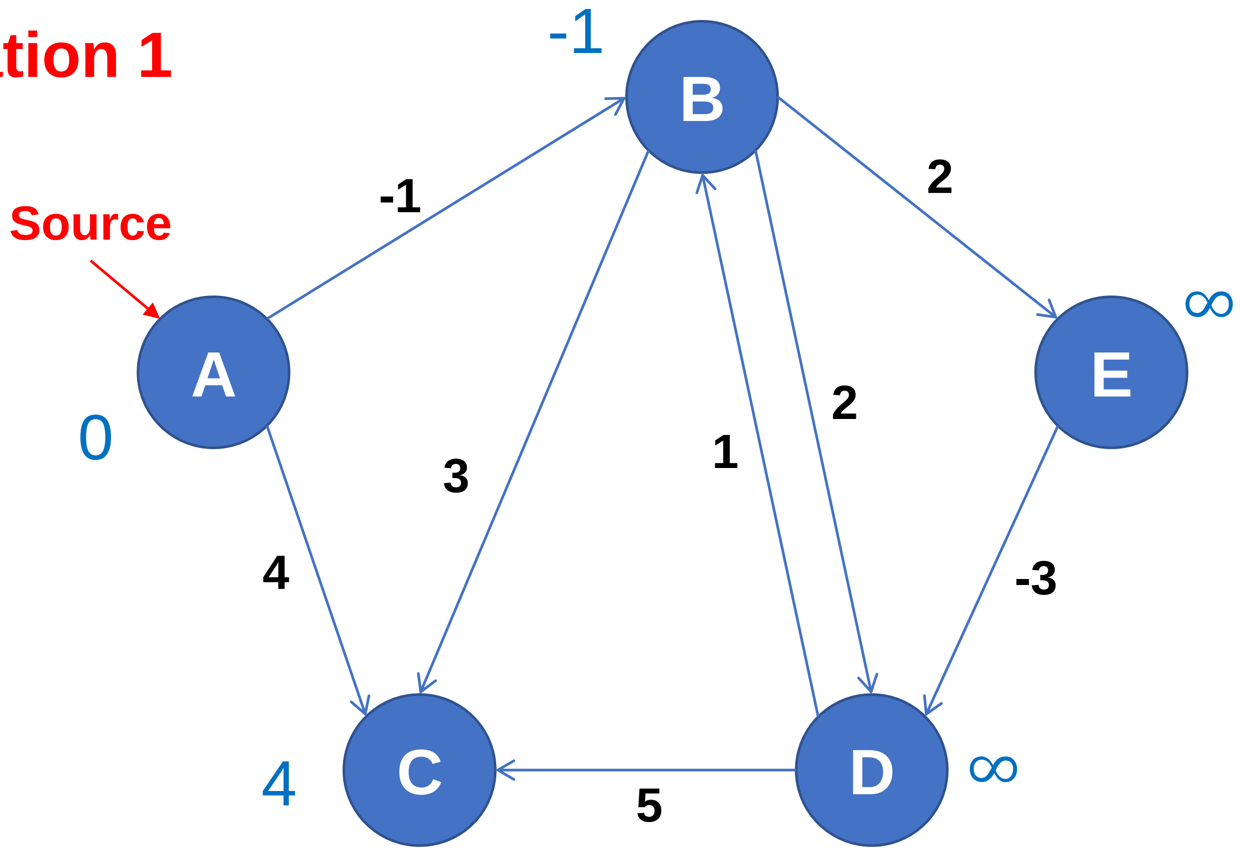
Vertex	d	π
A	0	NIL
B	-1	A
C	4	A
D	∞	NIL
E	∞	NIL

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(D, C): d[u]+edge(u,v)=∞ > 4

Bellman Ford's algorithm

Iteration 1



distance

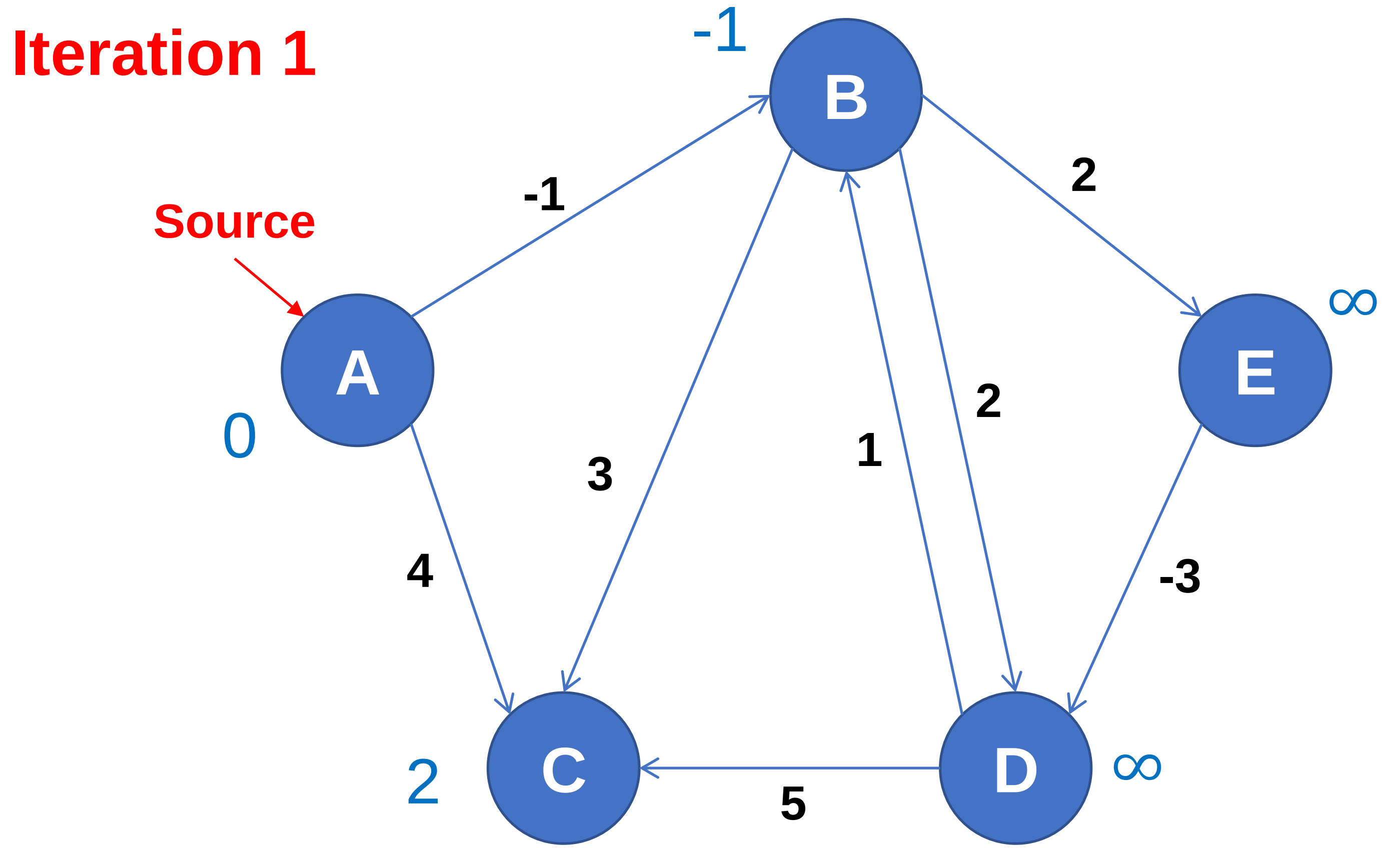
parent

Vertex	d	π
A	0	NIL
B	-1	A
C	4	A
D	∞	NIL
E	∞	NIL

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, C): d[u]+edge(u,v)=(-1)+3 < 4

Bellman Ford's algorithm



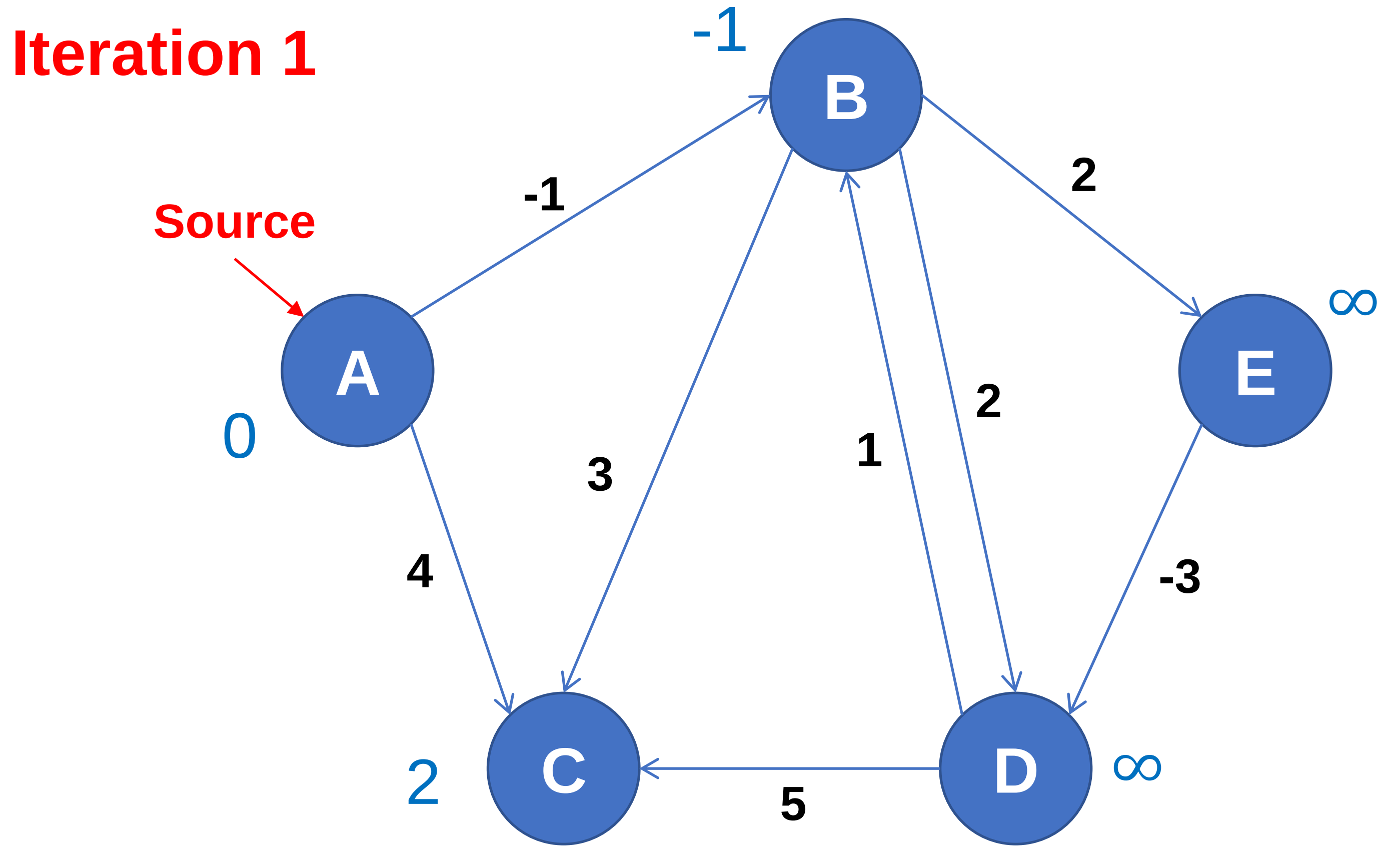
distance parent

Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	∞	NIL
E	∞	NIL

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, C): d[u]+edge(u,v)=(-1)+3=2 < 4

Bellman Ford's algorithm

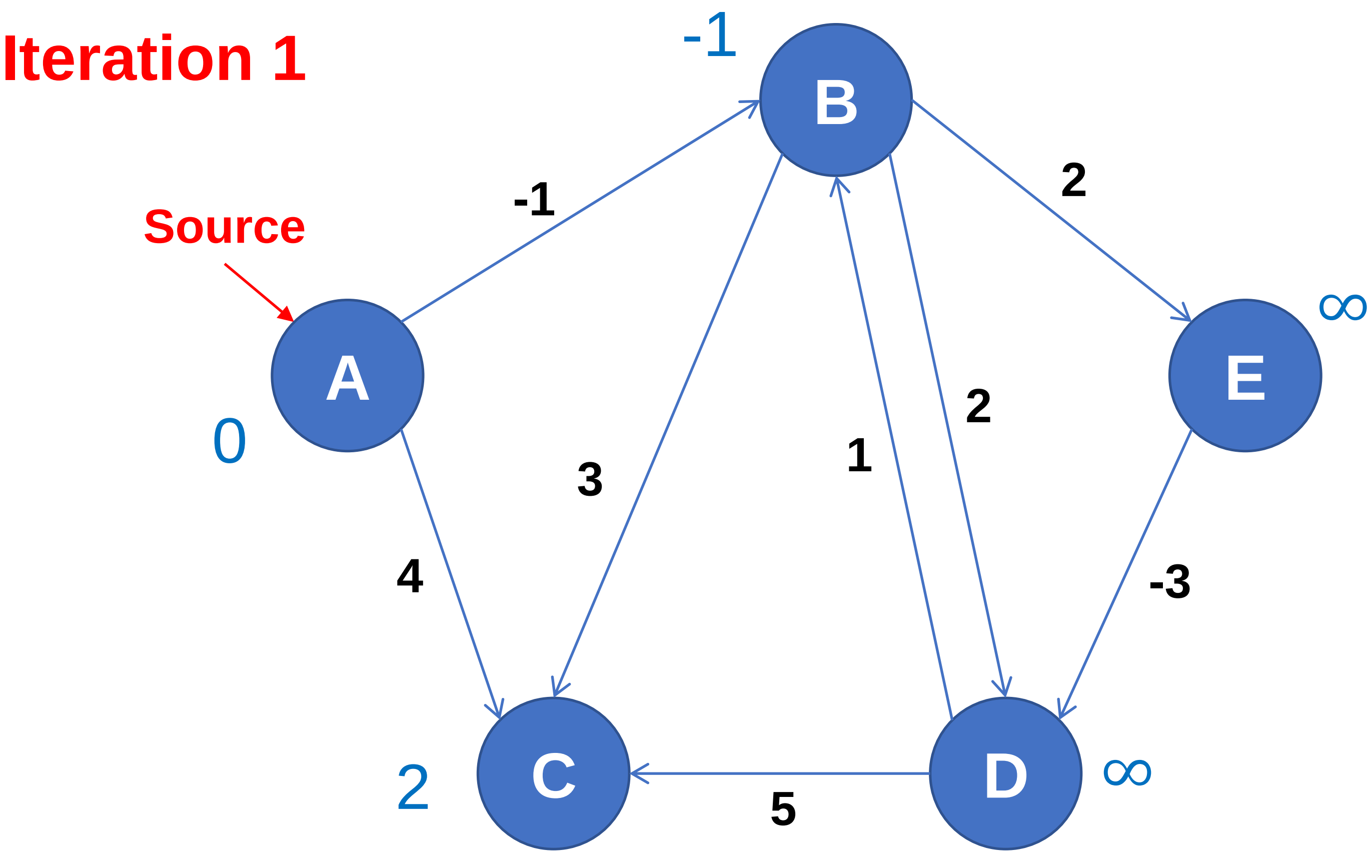


Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	∞	NIL
E	∞	NIL

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(E, D): d[u]+edge(u,v)= ∞ = ∞

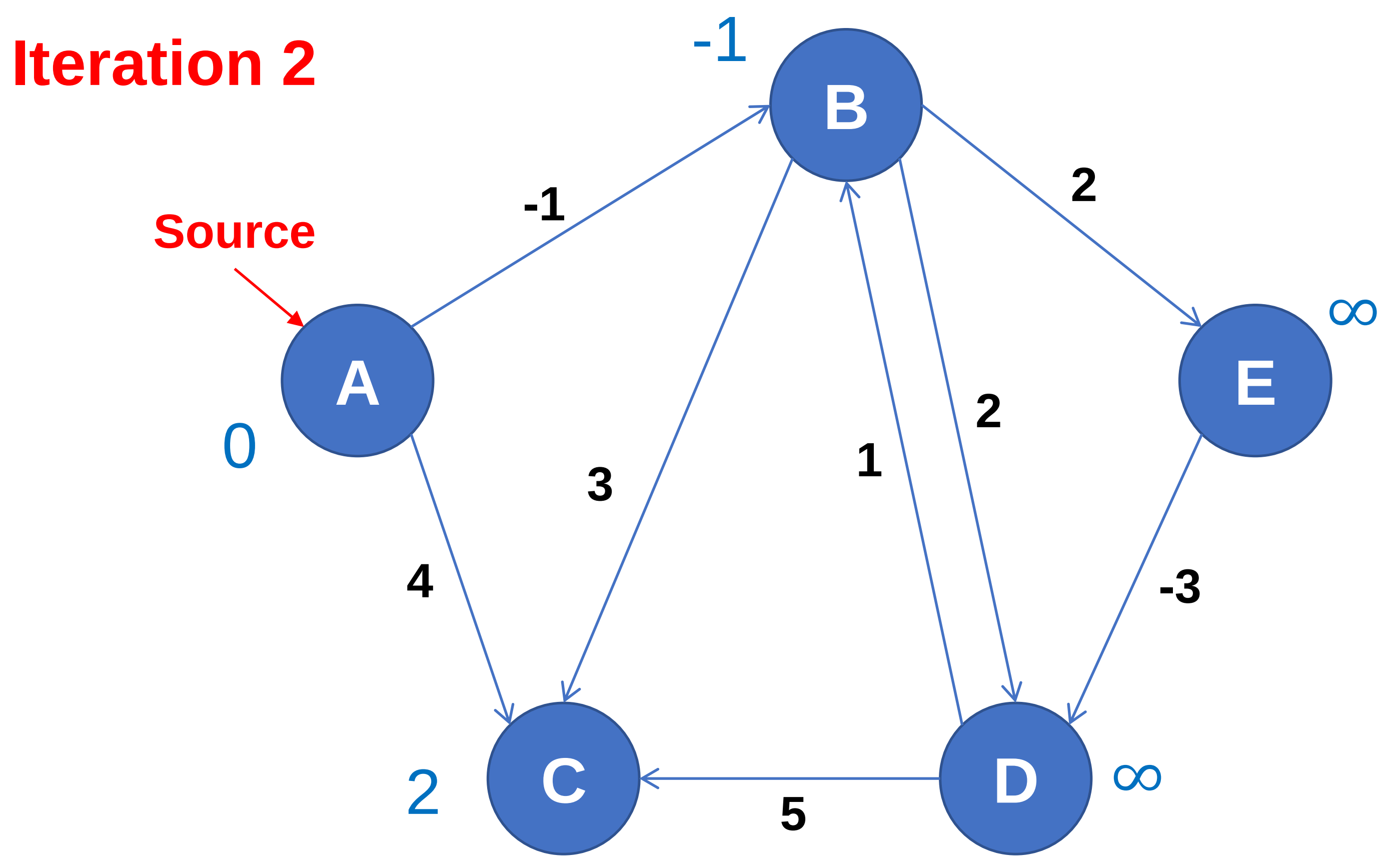
Bellman Ford's algorithm



Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	∞	NIL
E	∞	NIL

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

Bellman Ford's algorithm



Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	∞	NIL
E	∞	NIL

distance: d

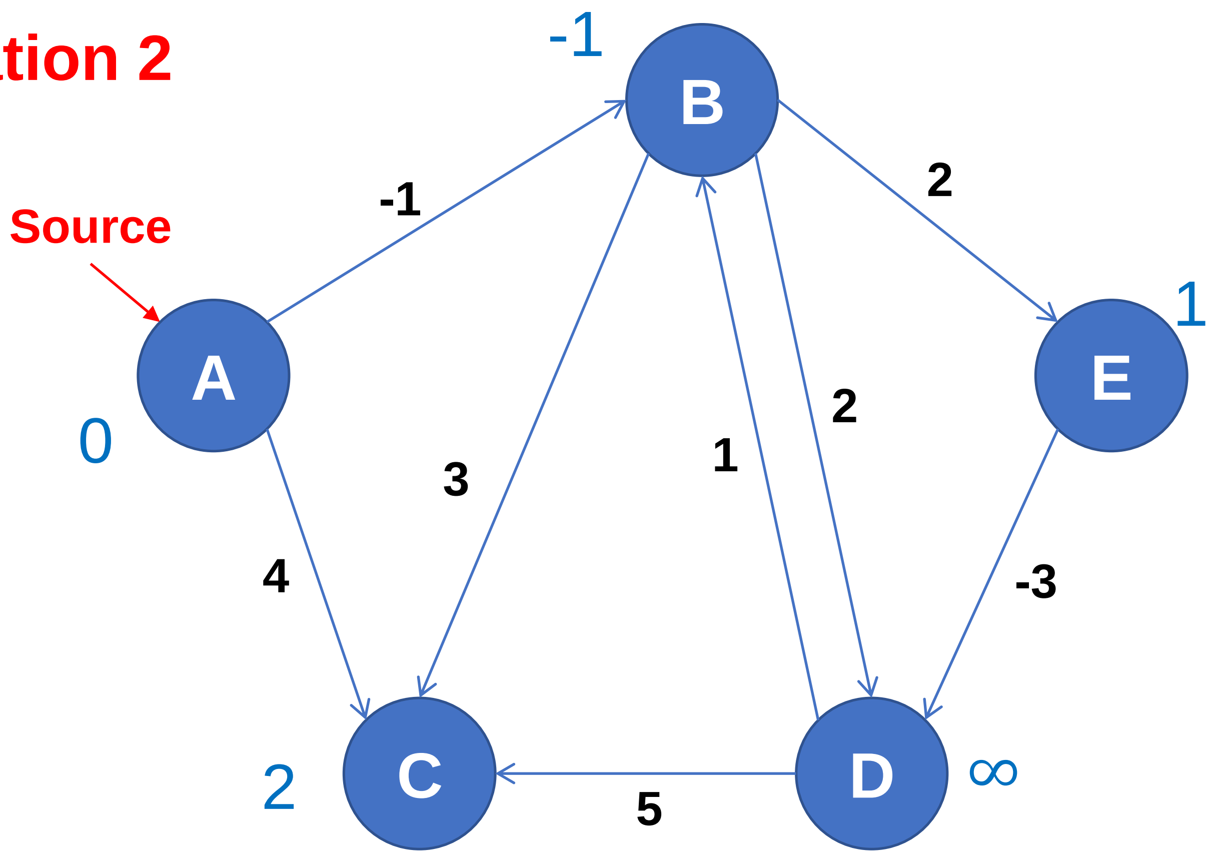
parent: π

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, E): d[u]+edge(u,v)=(-1)+2=1 < ∞

Bellman Ford's algorithm

Iteration 2



distance

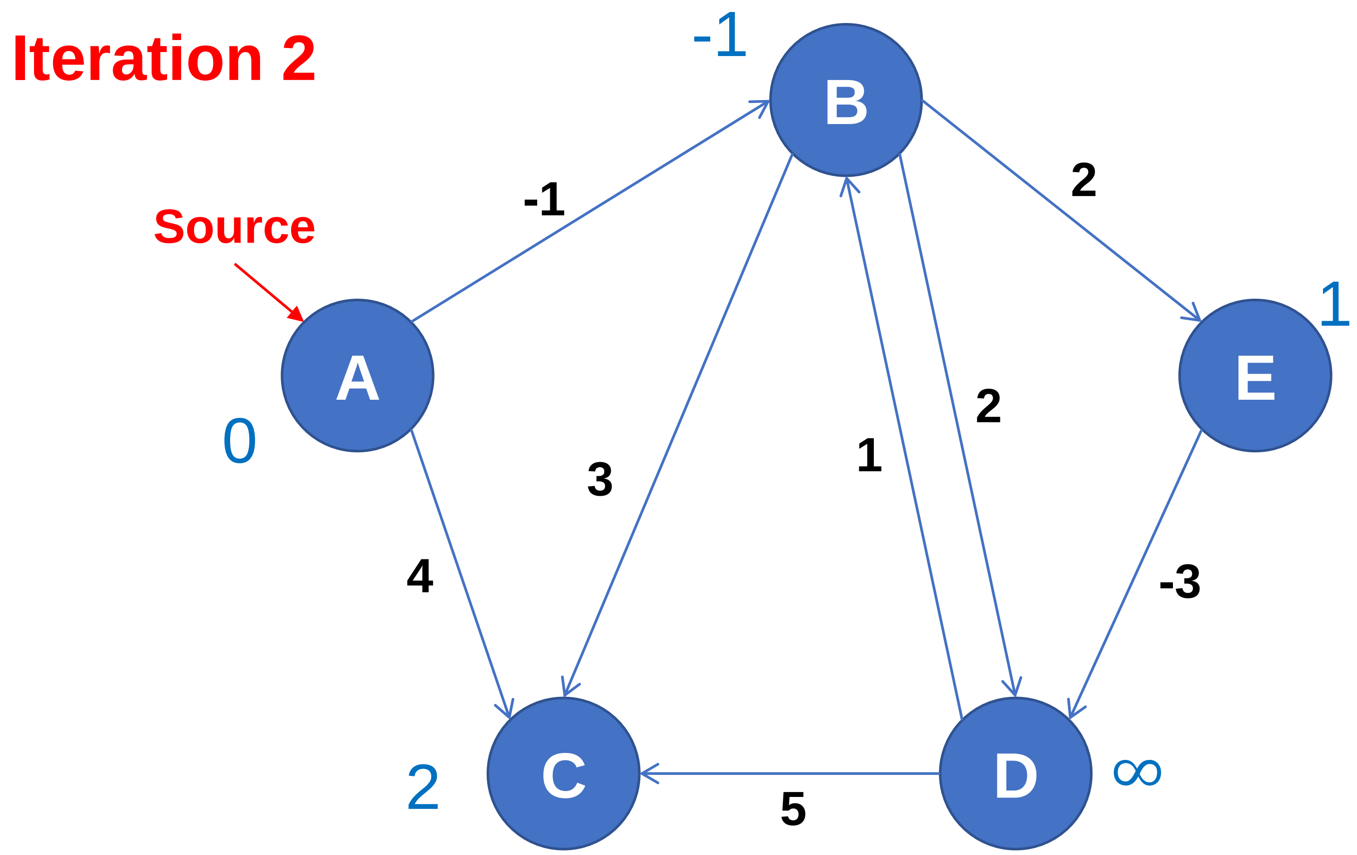
parent

Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	∞	NIL
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, E): d[u]+edge(u,v)=(-1)+2=1 < ∞

Bellman Ford's algorithm



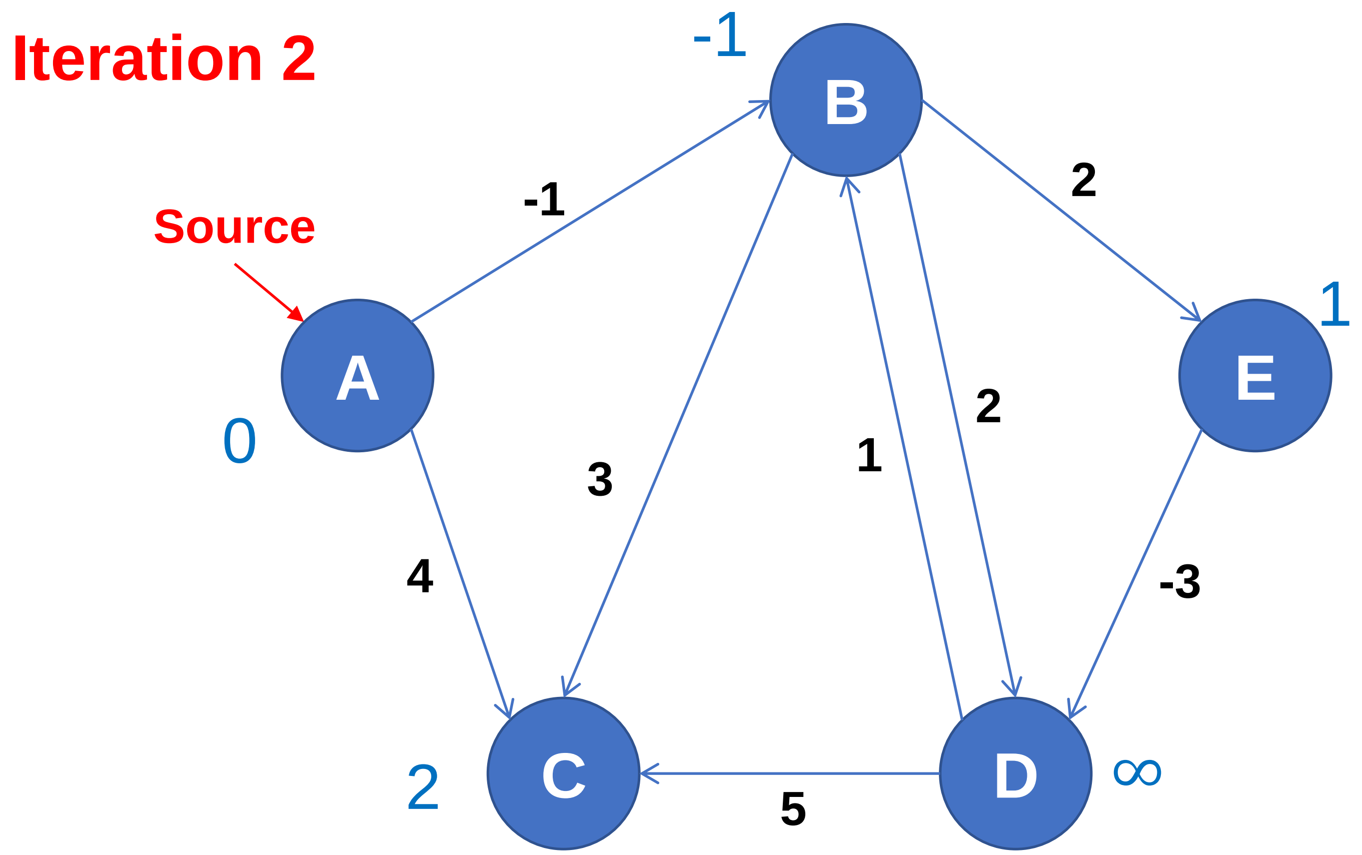
distance parent

Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	∞	NIL
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(D, B): d[u]+edge(u,v)=∞ = ∞

Bellman Ford's algorithm



distance ↑ parent ↑

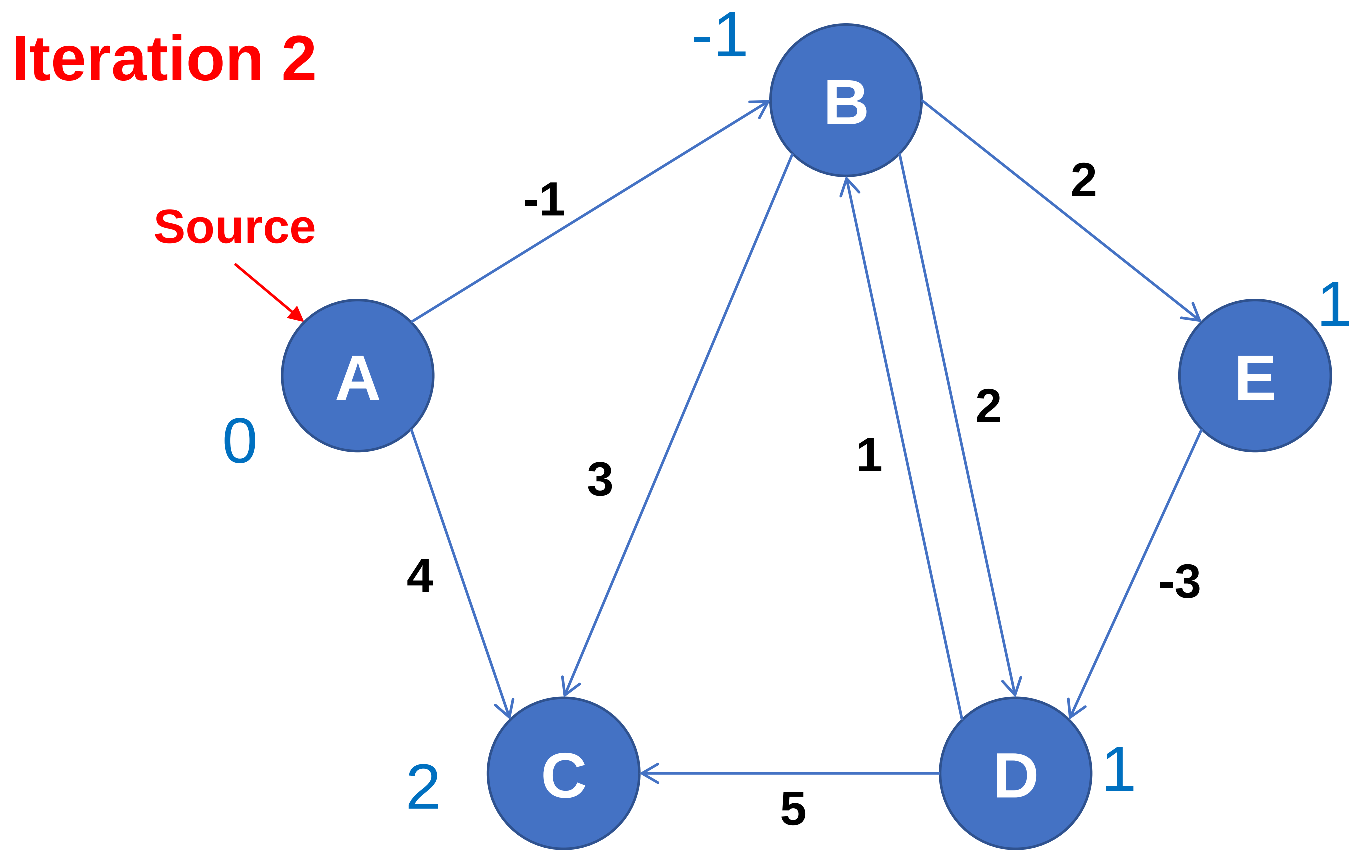
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	∞	NIL
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, D): d[u]+edge(u,v)=(-1)+2=1 < ∞

Bellman Ford's algorithm

Iteration 2



Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	1	B
E	1	B

distance

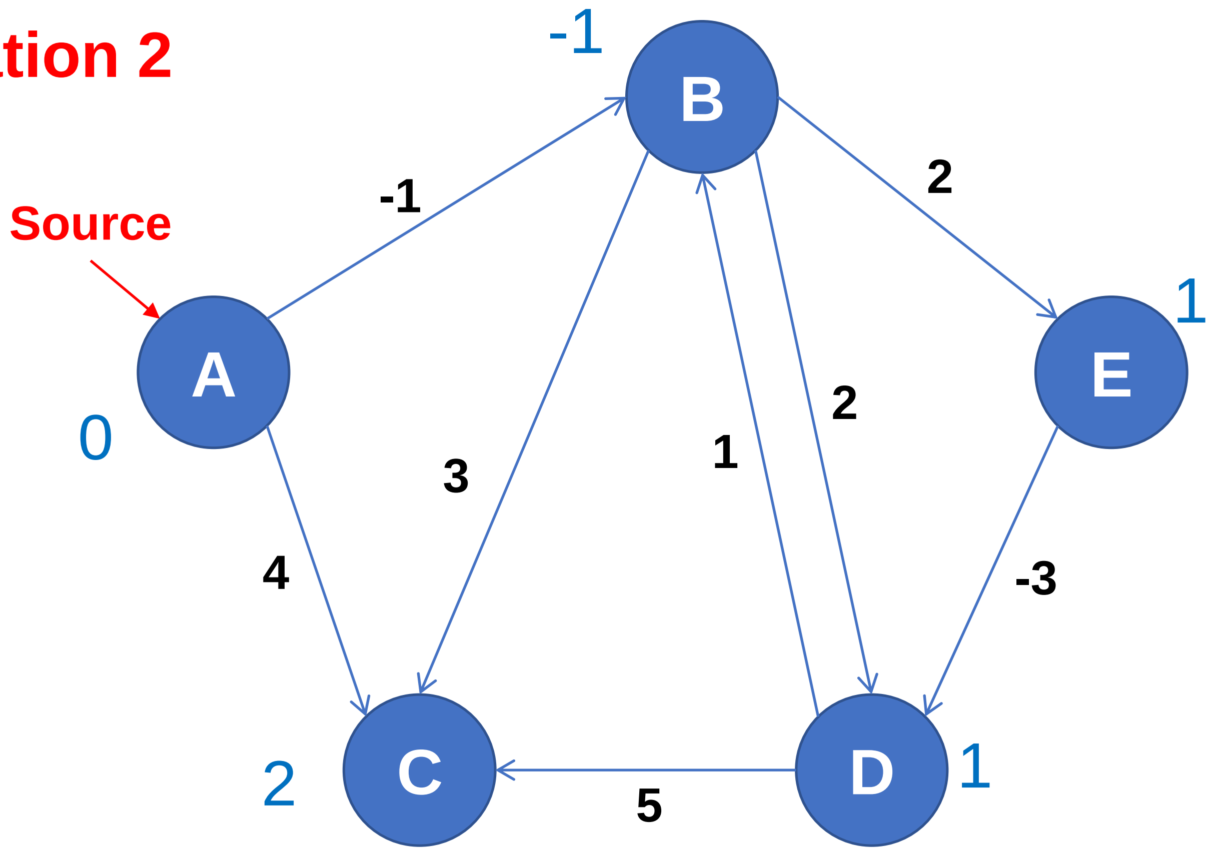
parent

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, D): d[u]+edge(u,v)=(-1)+2=1 < ∞

Bellman Ford's algorithm

Iteration 2



distance ↑ d

parent ↑ π

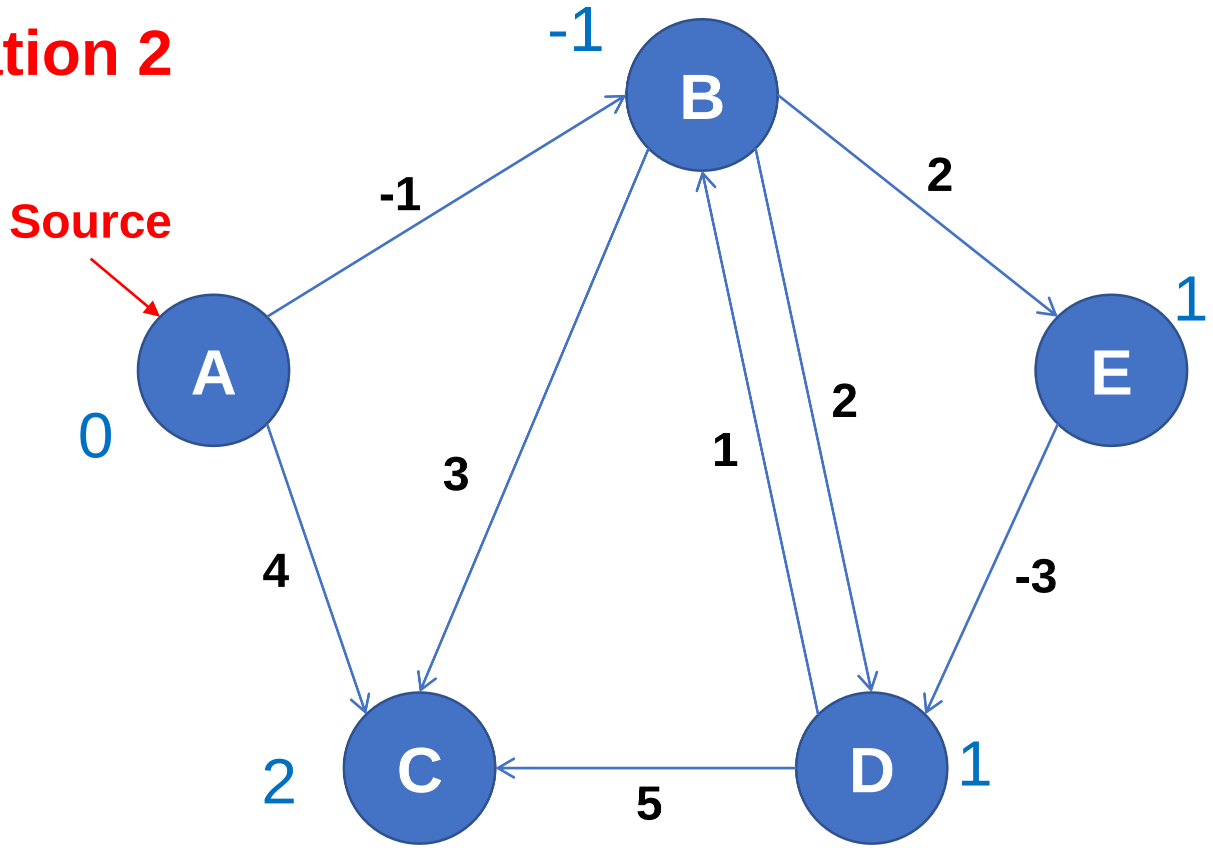
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	1	B
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(A, B): d[u]+edge(u,v)=0+(-1)=-1 = -1

Bellman Ford's algorithm

Iteration 2



distance

parent

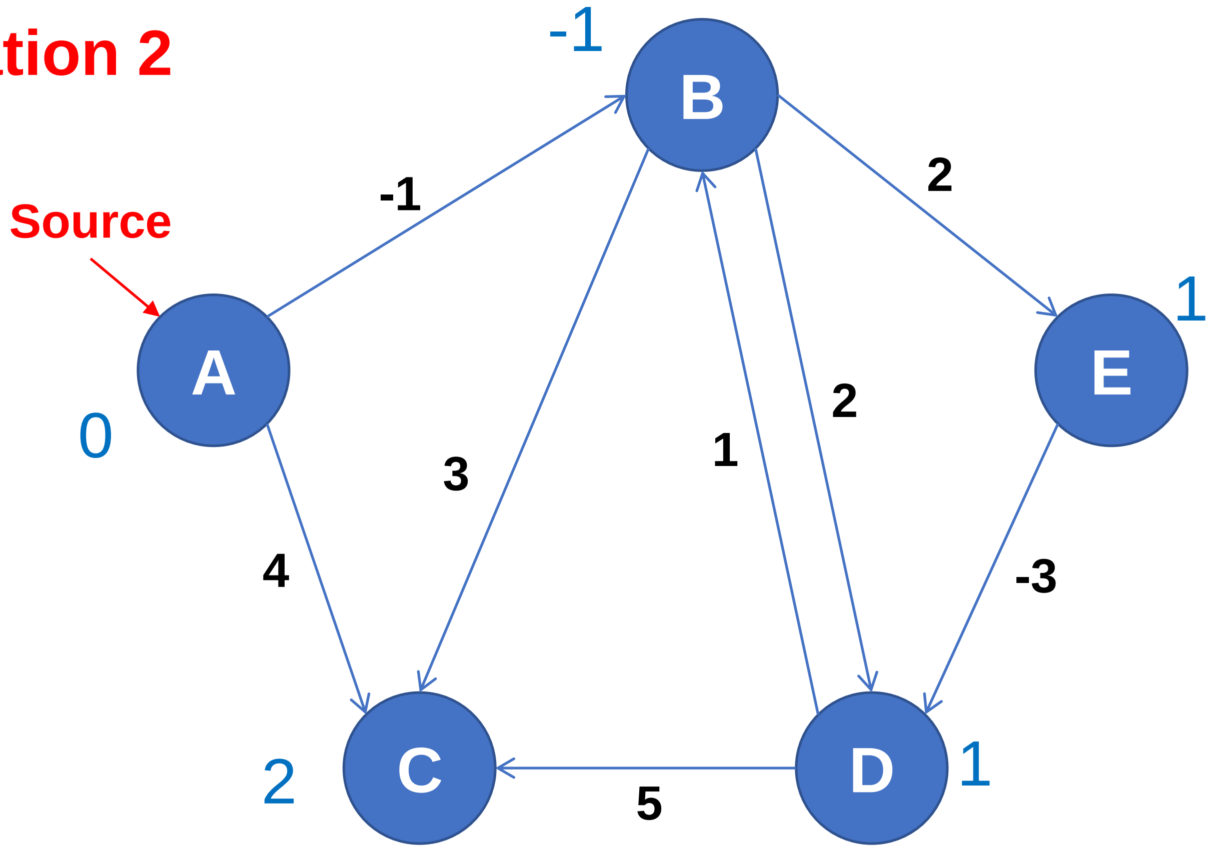
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	1	B
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(A, C): d[u]+edge(u,v)=0+4=4 > 2

Bellman Ford's algorithm

Iteration 2



distance ↑ d

parent ↑ π

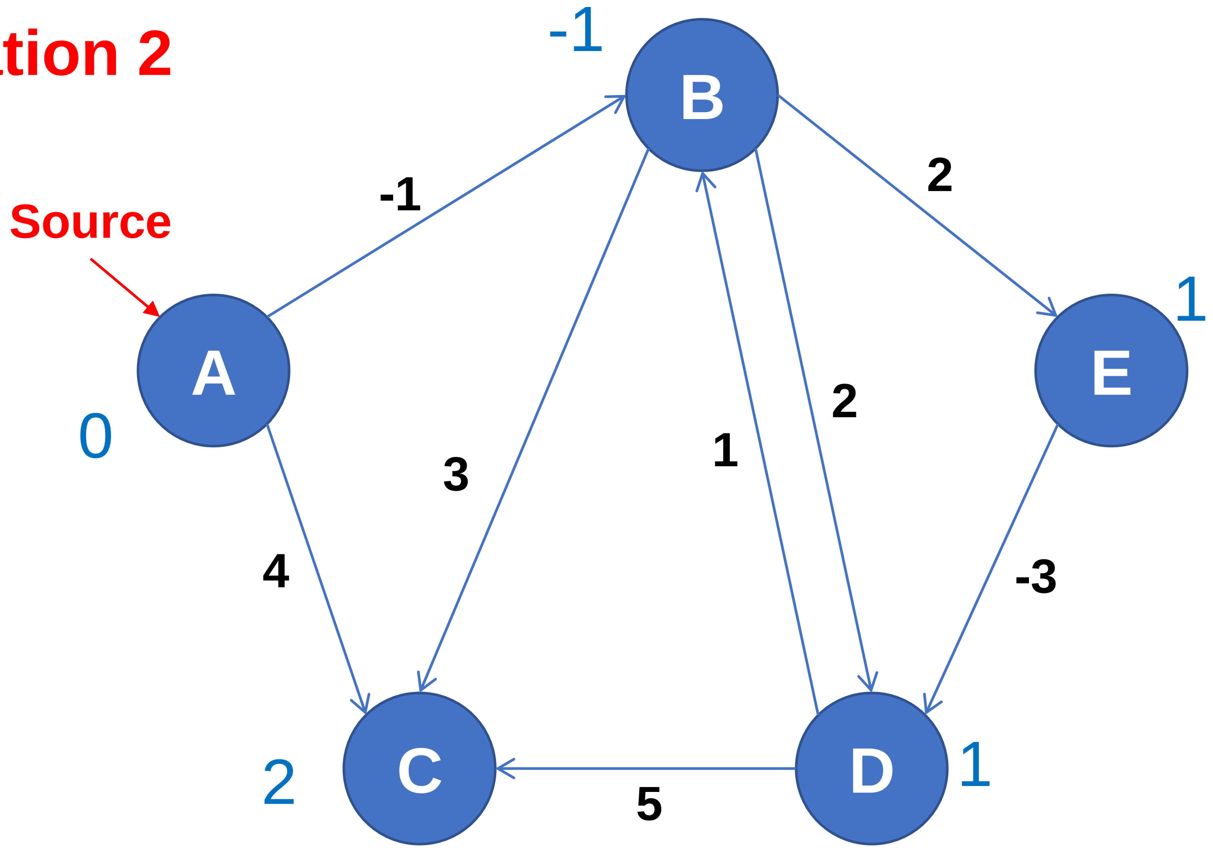
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	1	B
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(D, C): d[u]+edge(u,v)=1+5=6 > 2

Bellman Ford's algorithm

Iteration 2



distance

parent

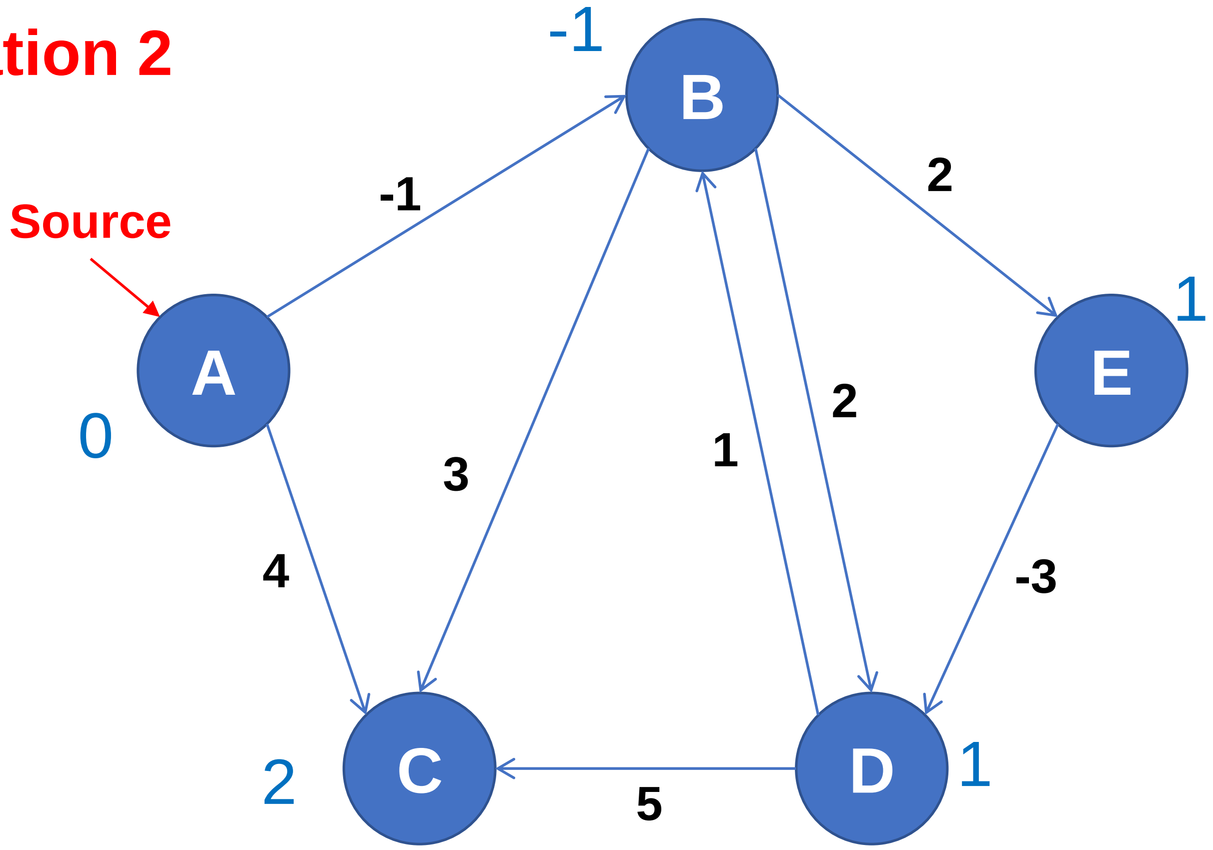
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	1	B
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, C): d[u]+edge(u,v)=(-1)+3=2 > 2

Bellman Ford's algorithm

Iteration 2



Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	1	B
E	1	B

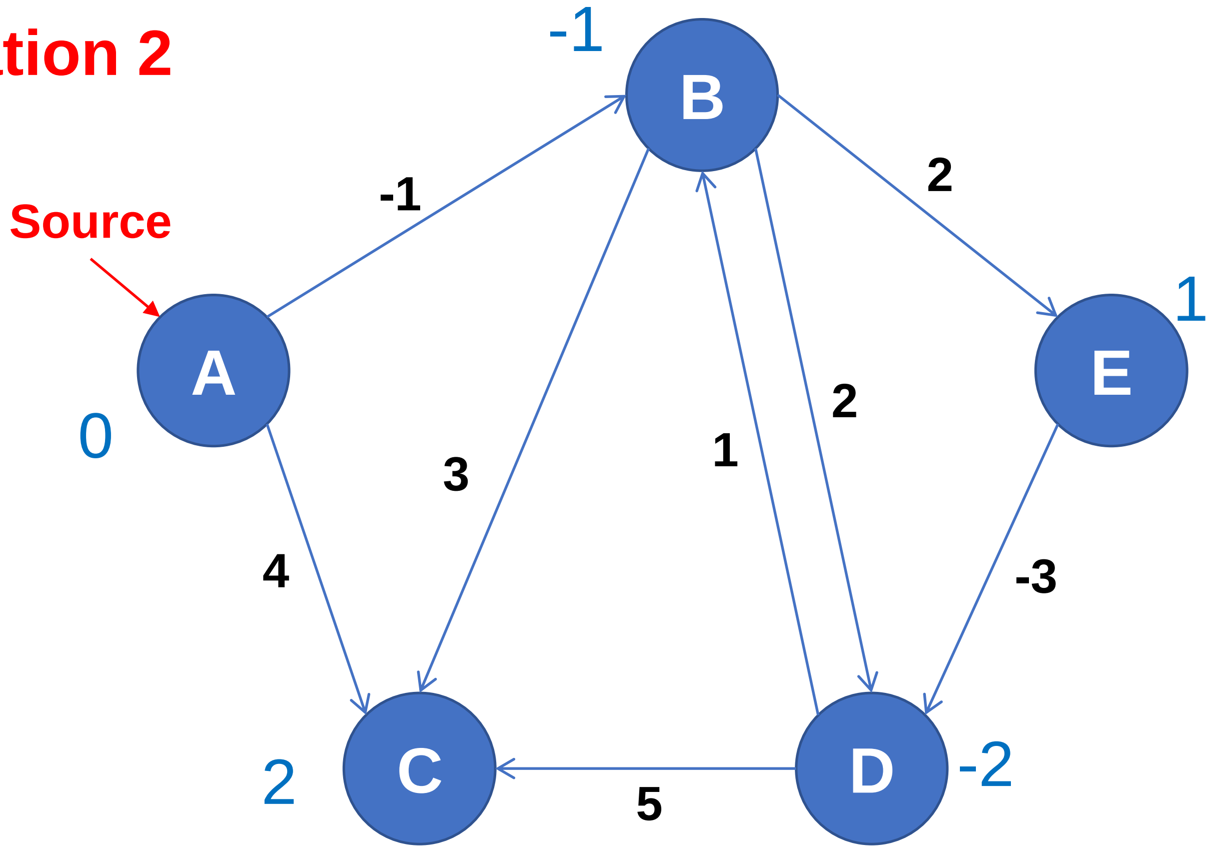
distance ↑ d; parent ↑ π

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(E, D): d[u]+edge(u,v)=1+(-3)=-2 < 1

Bellman Ford's algorithm

Iteration 2



distance ↑ d   parent ↑ π

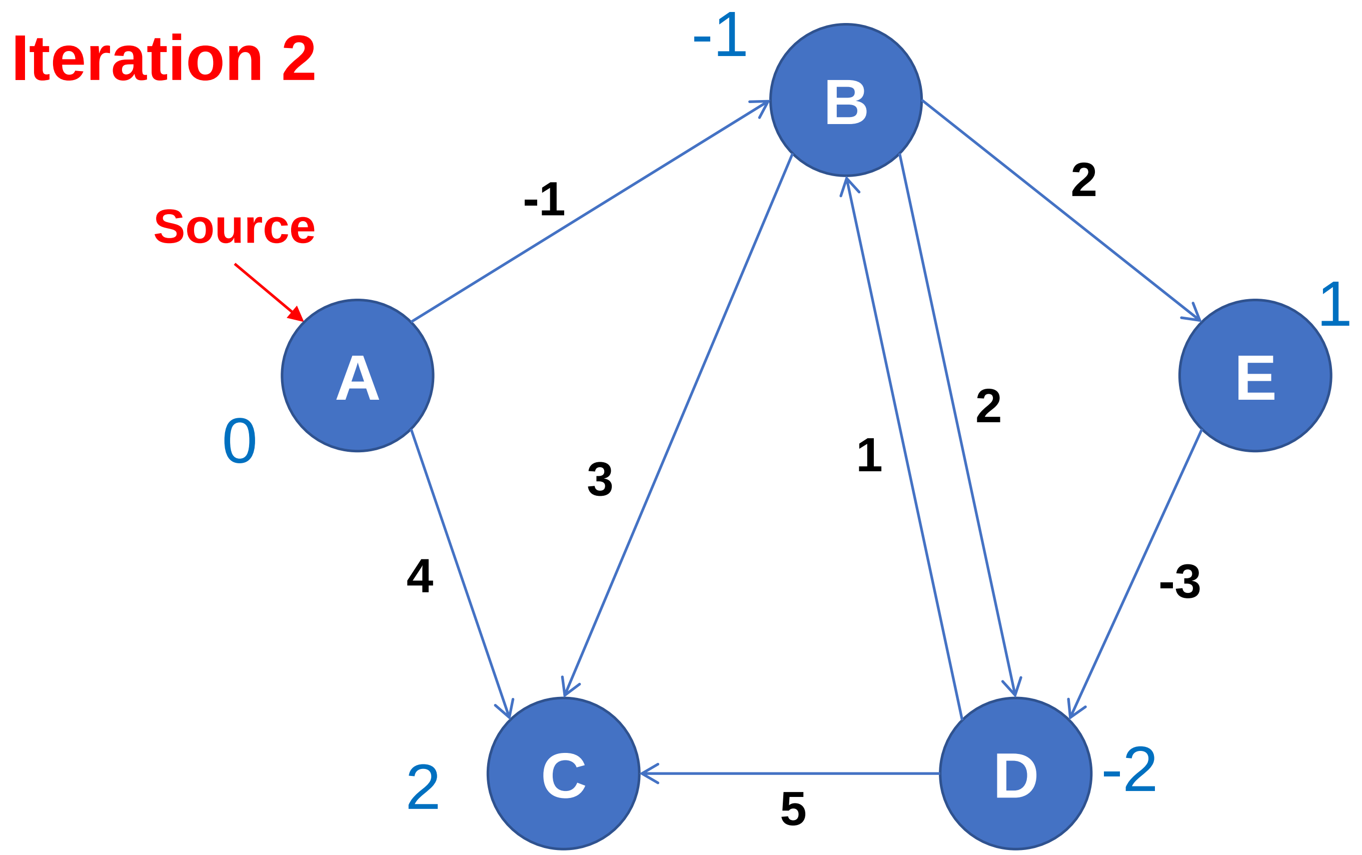
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(E, D): d[u]+edge(u,v)=1+(-3)=-2 < 1

Bellman Ford's algorithm

Iteration 2



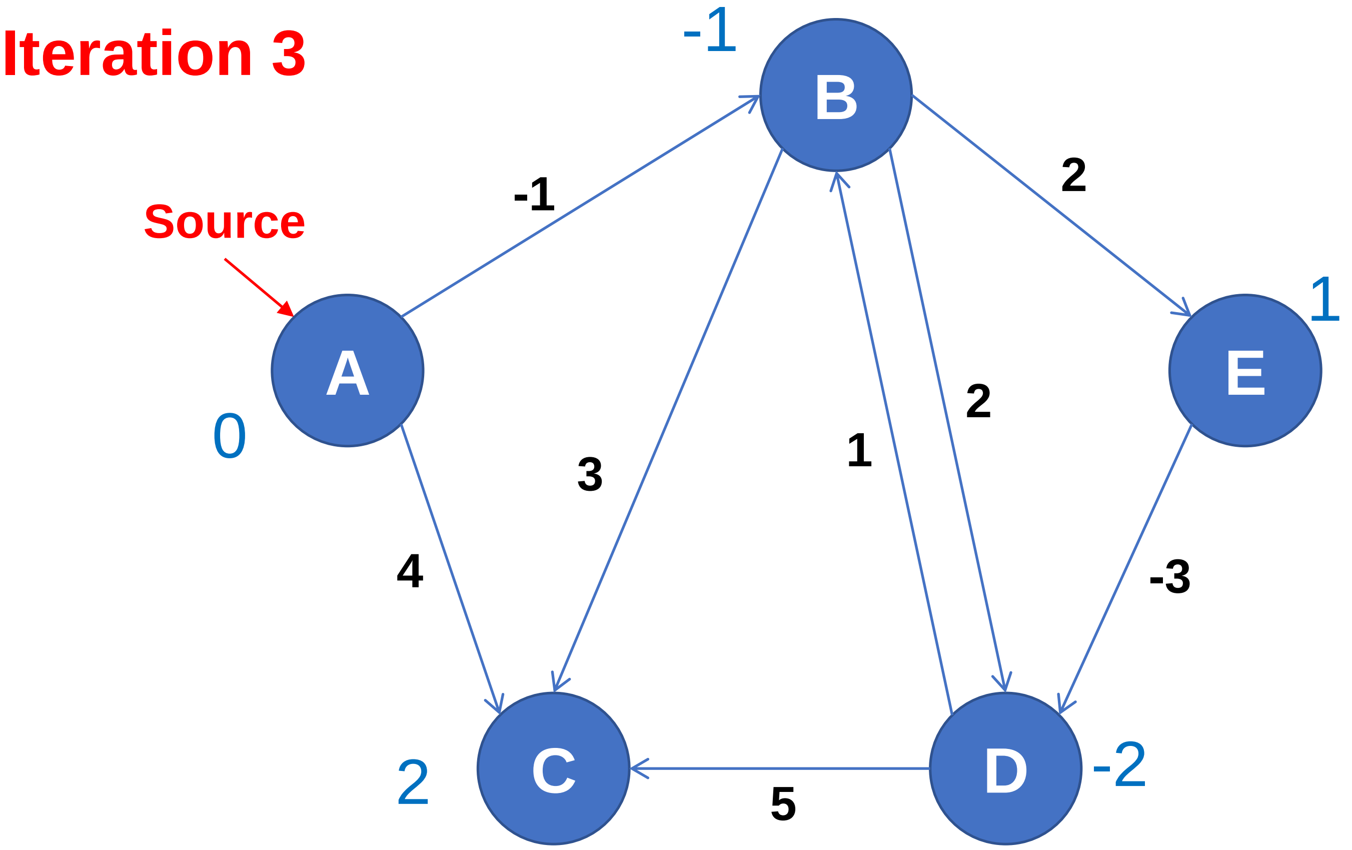
distance

parent

Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

Bellman Ford's algorithm



distance ↑ d, parent ↑ π

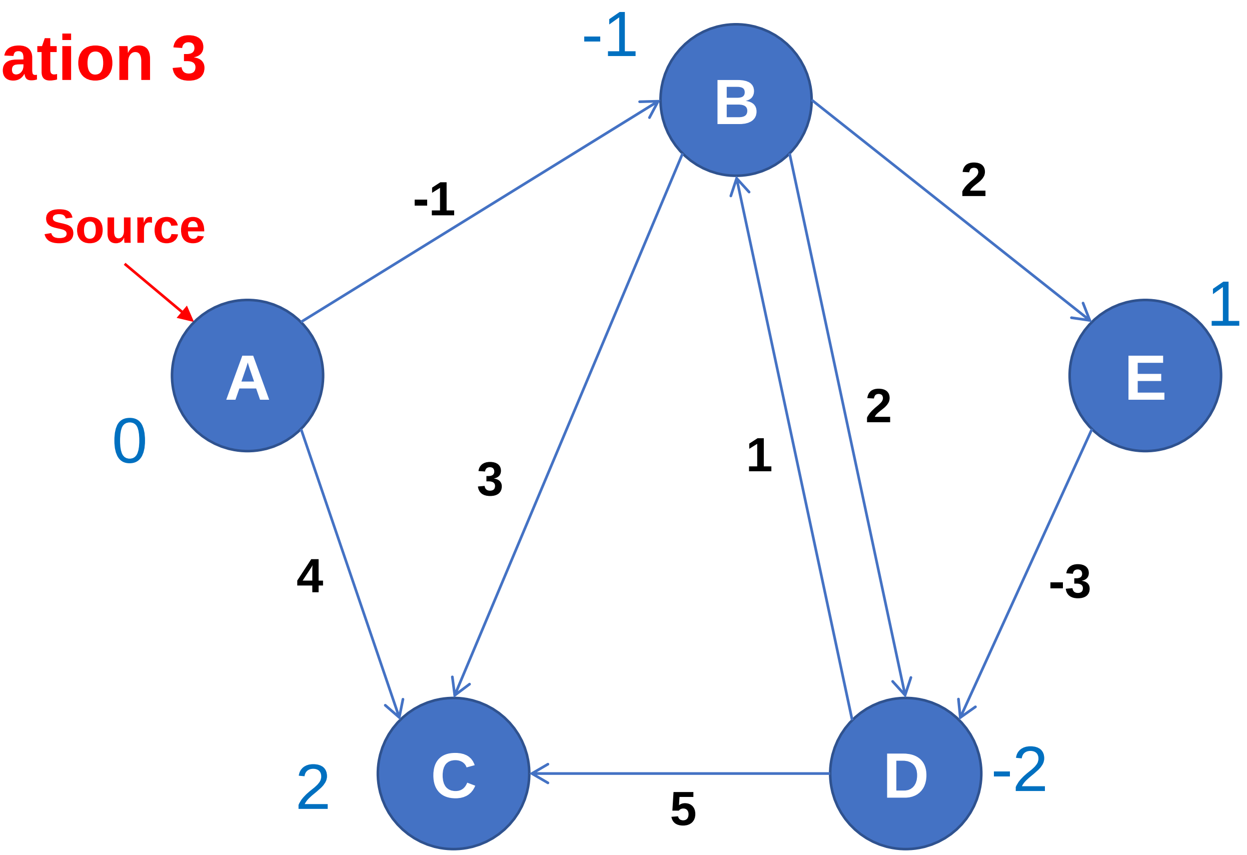
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, E): d[u]+edge(u,v)=(-1)+2=1 = 1

Bellman Ford's algorithm

Iteration 3



distance ↑ parent ↑

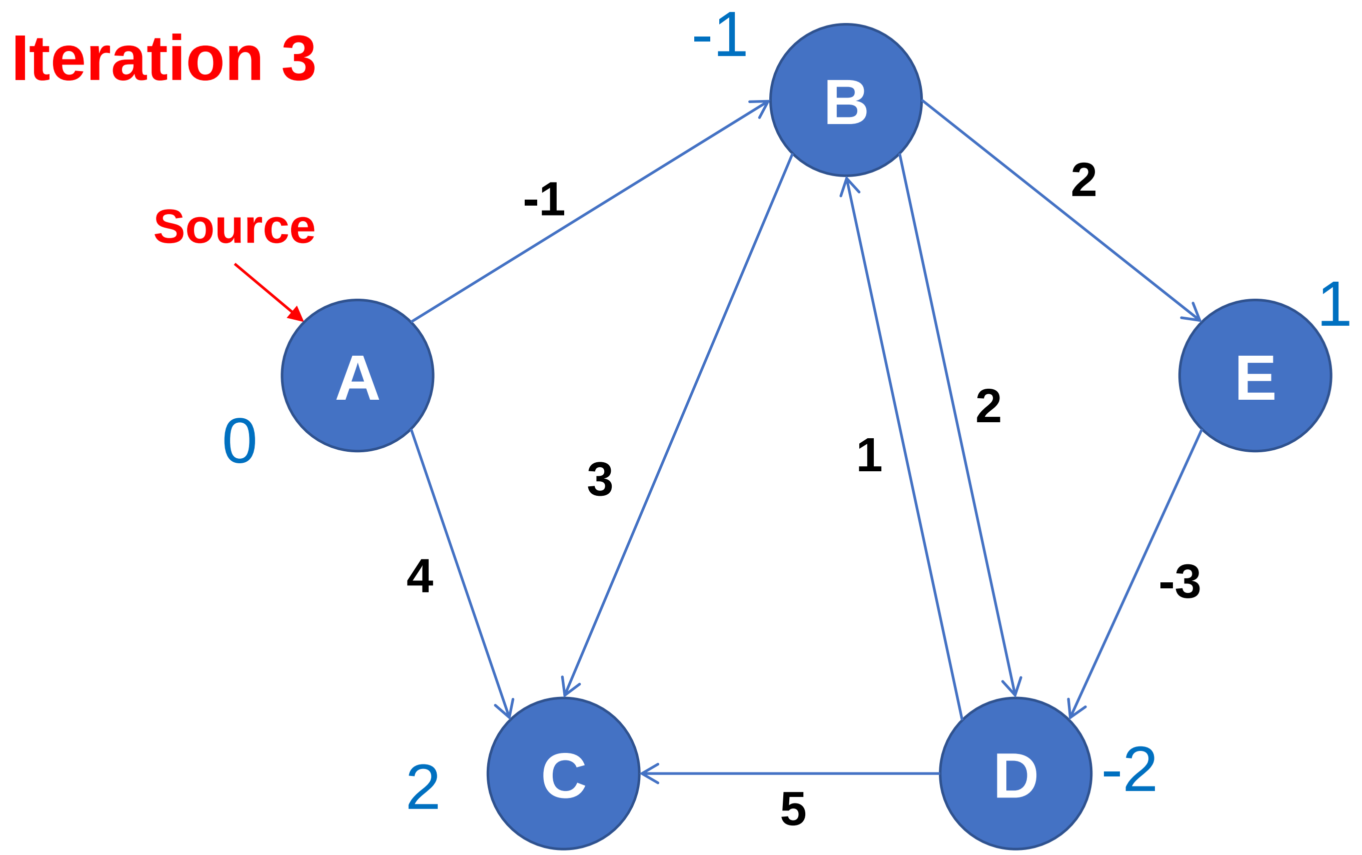
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(D, B): d[u]+edge(u,v)=(-2)+1=-1 = -1

Bellman Ford's algorithm

Iteration 3



Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

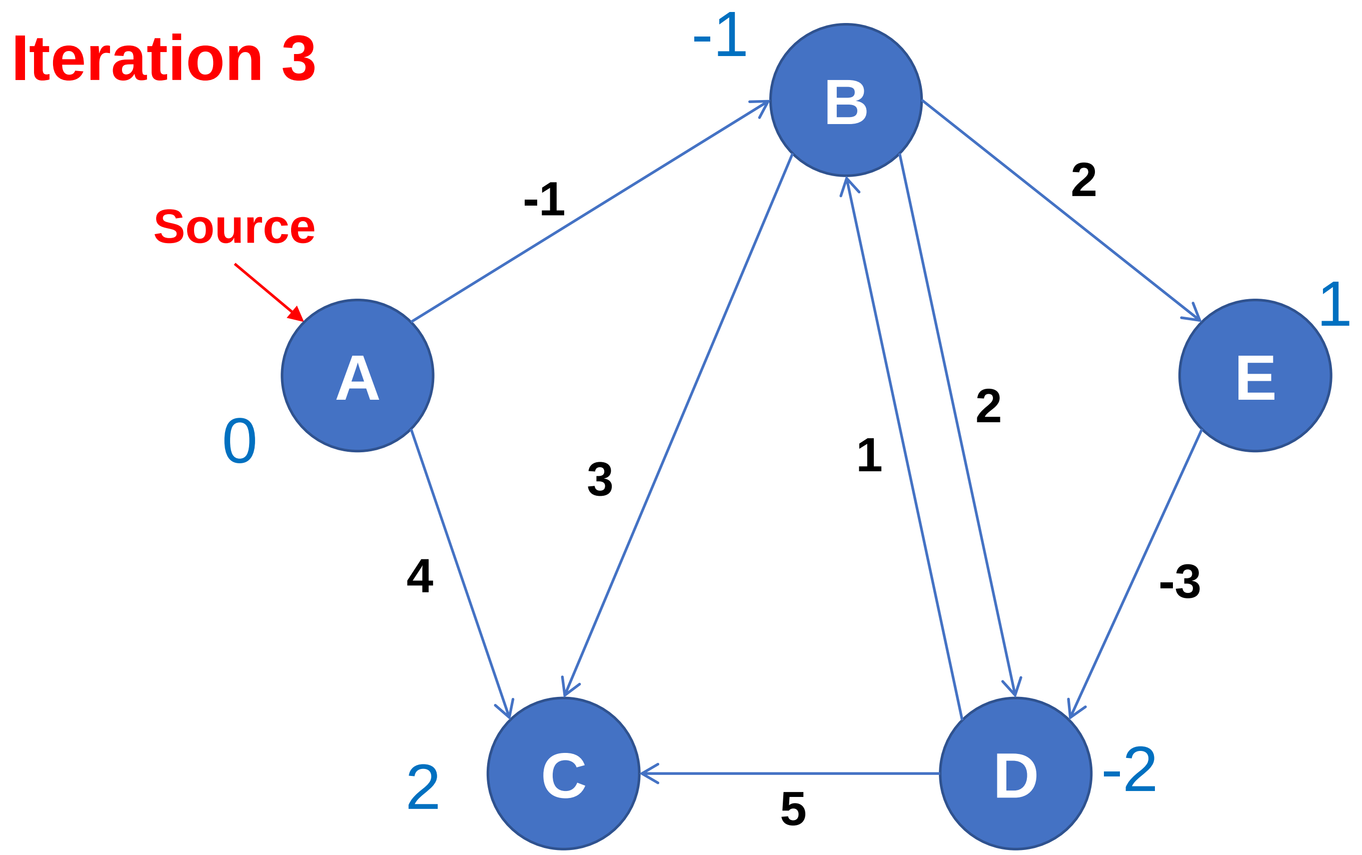
distance ↑ d

parent ↑ π

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, D): d[u]+edge(u,v)=(-1)+2=1 > -2

Bellman Ford's algorithm



distance ↑ d

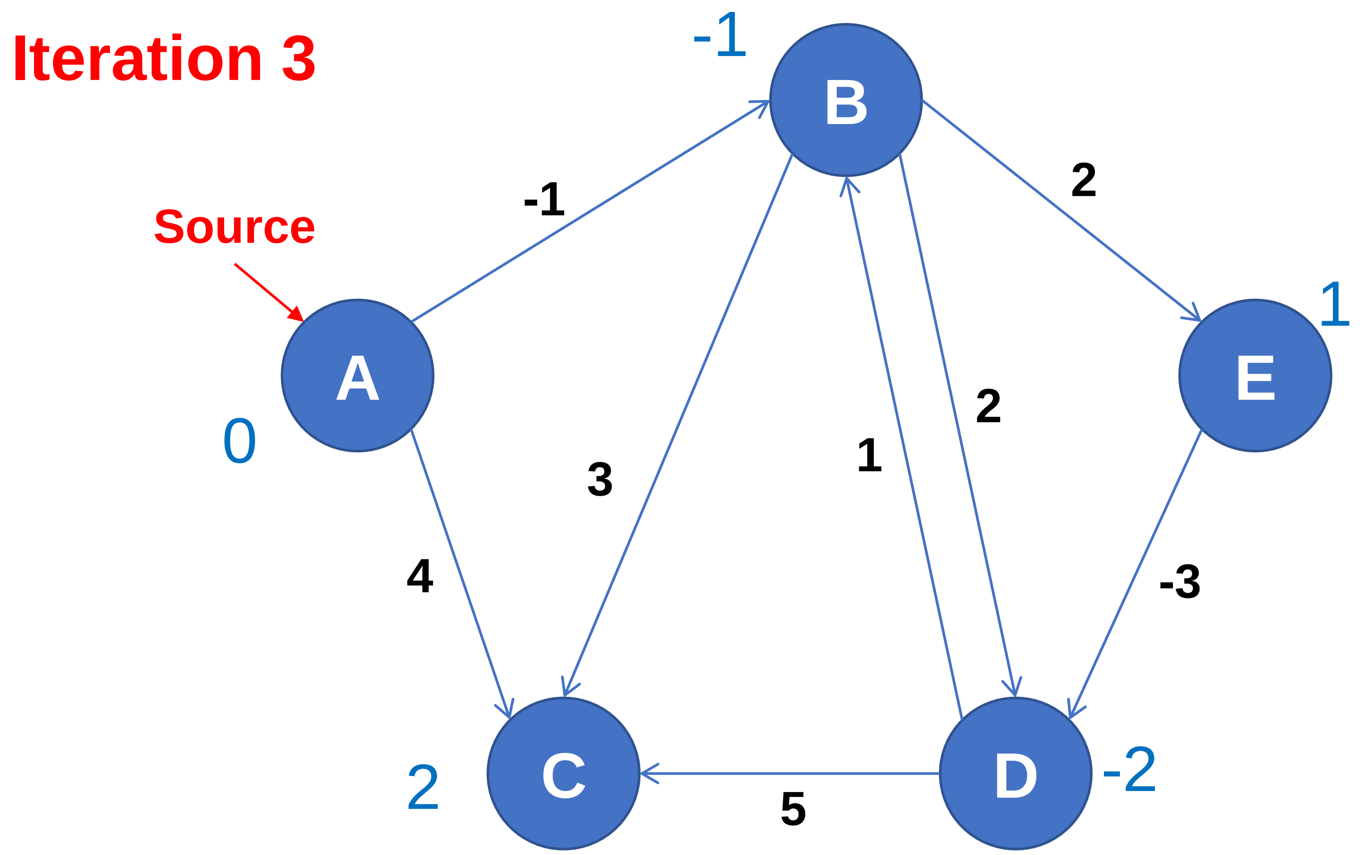
parent ↑ π

Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(A, B): d[u]+edge(u,v)=0+(-1)=-1 = -1

Bellman Ford's algorithm



Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

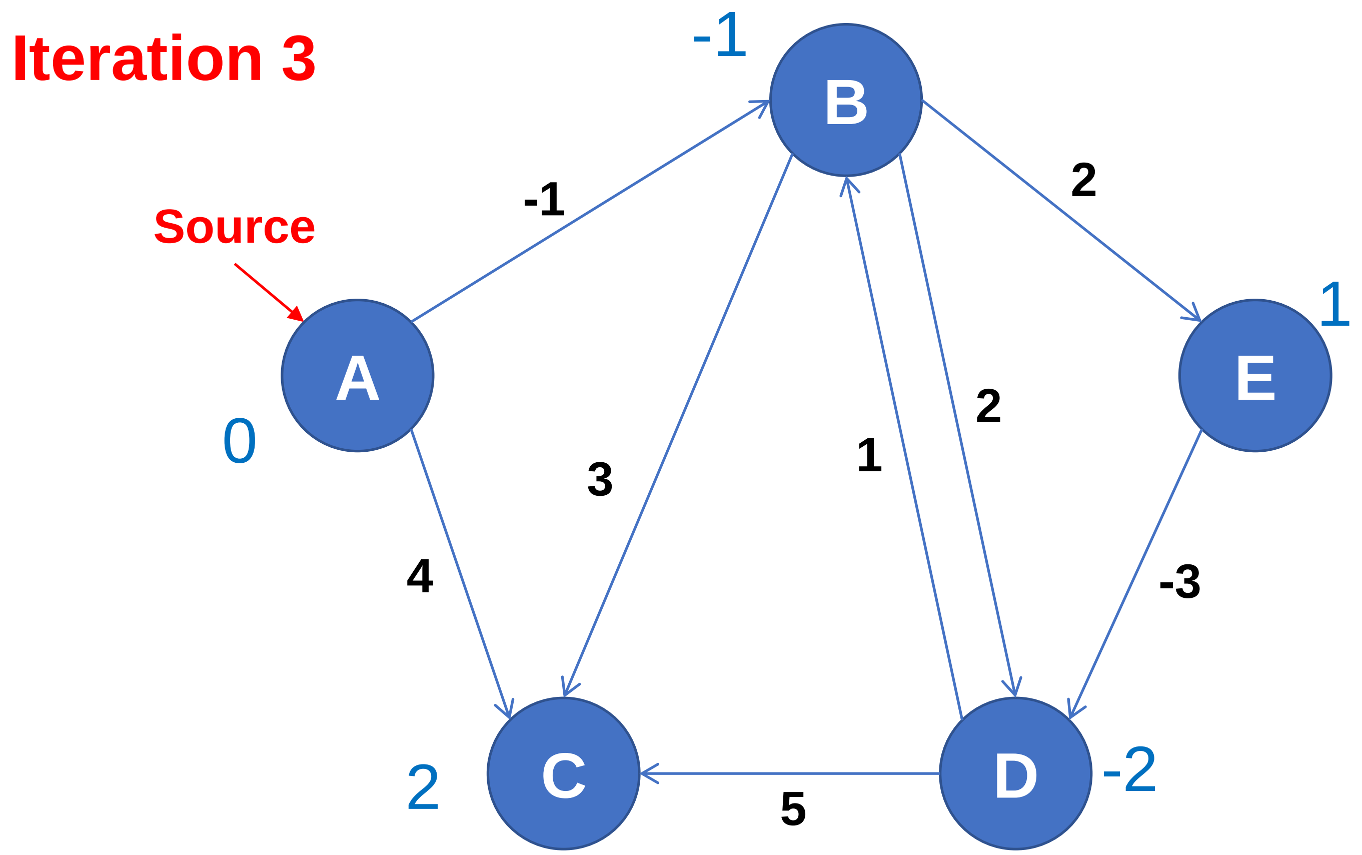
distance → d, parent → π

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(A, C): d[u]+edge(u,v)=0+2=2 = 2

Bellman Ford's algorithm

Iteration 3



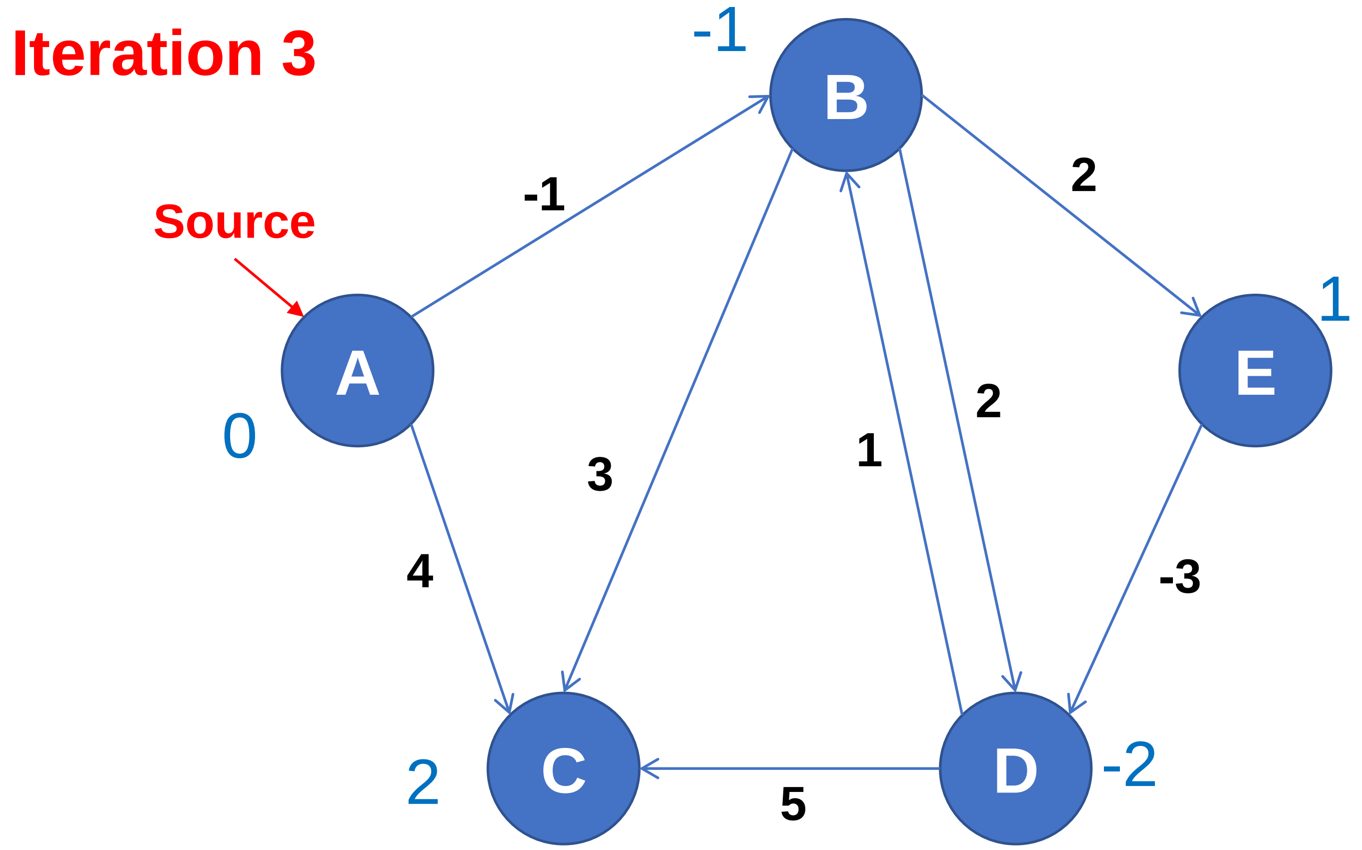
distance parent

Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(D, C): d[u]+edge(u,v)=(-2)+5=3 > 2

Bellman Ford's algorithm



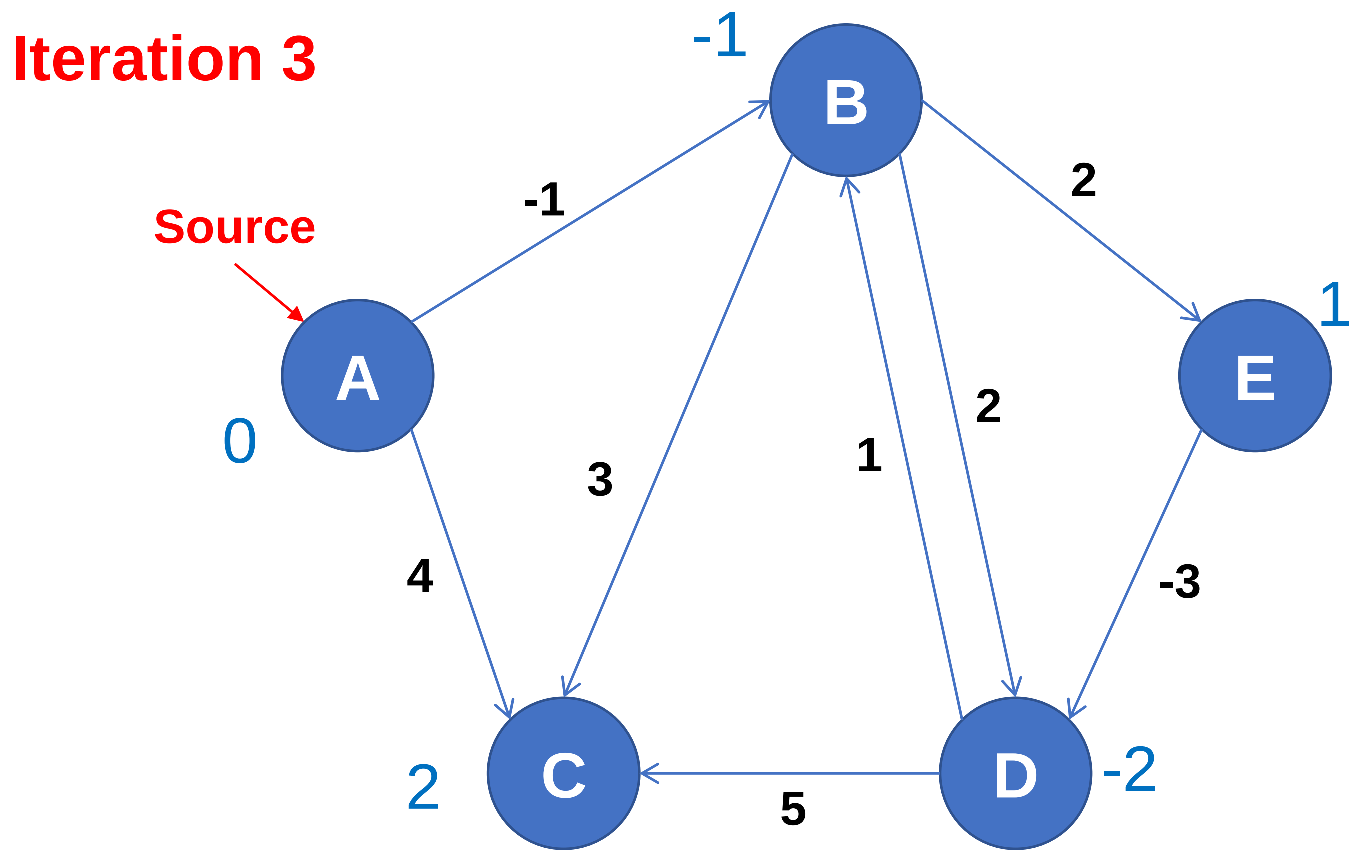
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

distance ↑ d; parent ↑ π

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(B, C): d[u]+edge(u,v)=(-1)+3=2 = 2

Bellman Ford's algorithm



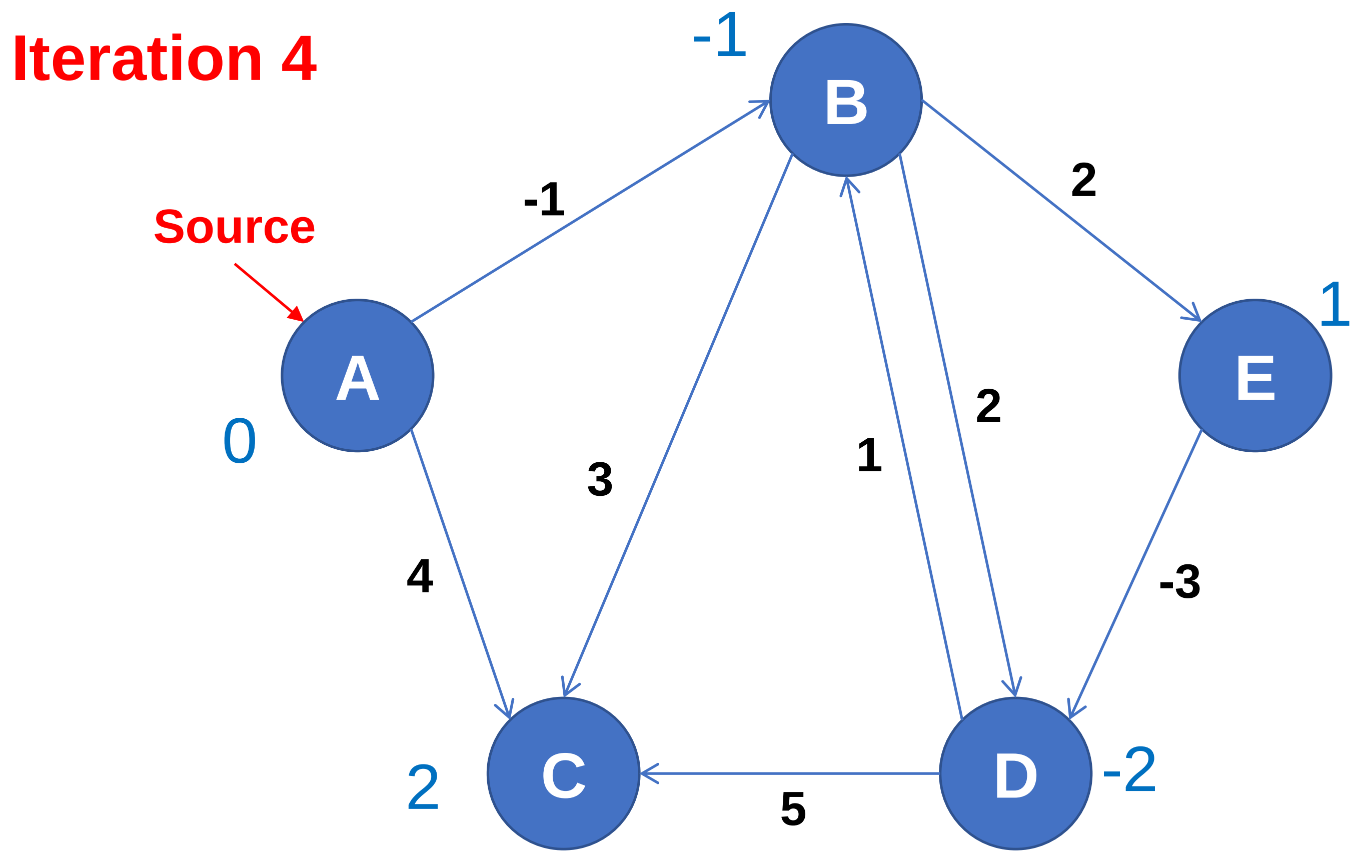
	distance	parent
Vertex	d	*π*
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

(E, D): d[u]+edge(u,v)=1+(-3)=-2 = -2

Bellman Ford's algorithm

Iteration 4



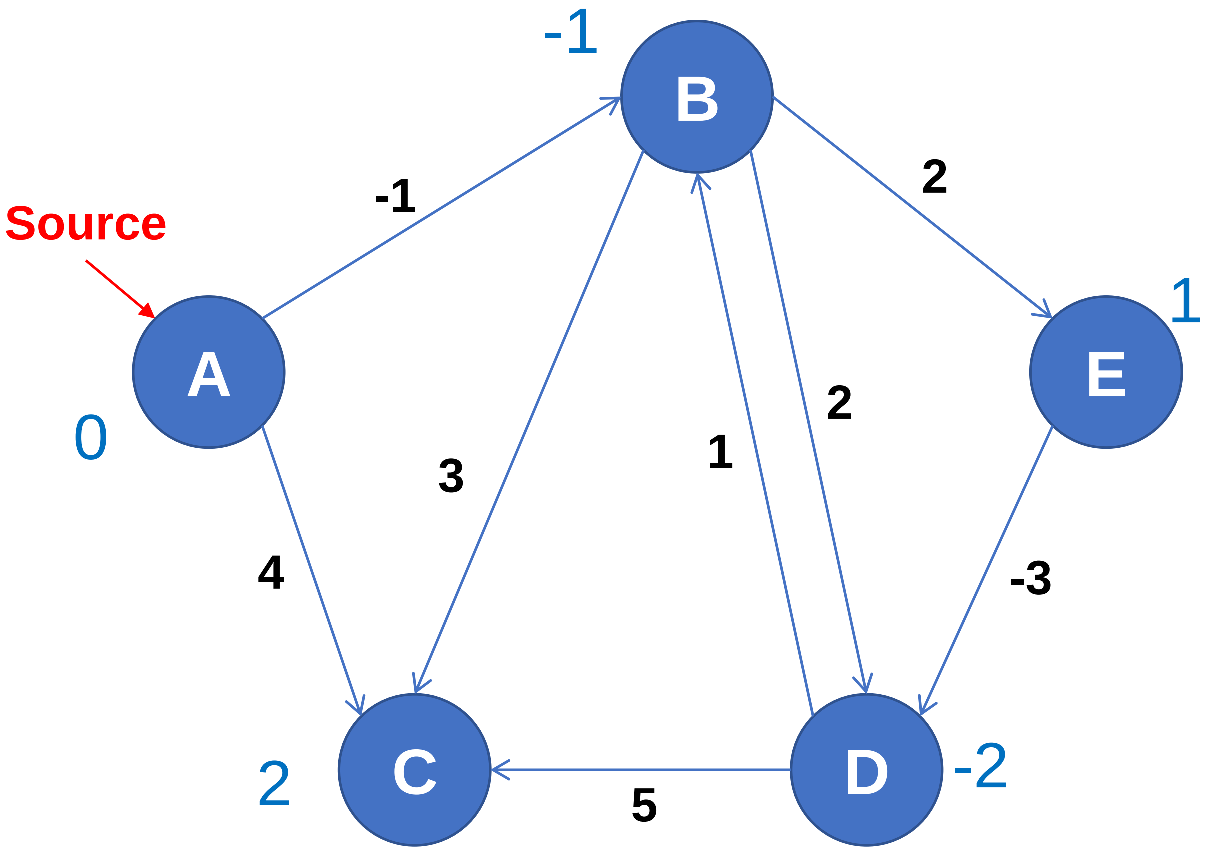
Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

distance ↑ d

parent ↑ π

(B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

Bellman Ford's algorithm



distance ↑ d

parent ↑ π

Vertex	d	π
A	0	NIL
B	-1	A
C	2	B
D	-2	E
E	1	B

Complexity

Time Complexity: O(VE)

Pseudo-Code

```
**for** $v$ in $V$:
    $v.d = \infty$
    $v.\pi = \text{None}$
$s.d = 0$
**for** $i$ from 1 to $|V| - 1$:
    **for** $(u, v)$ in $E$:
        **relax**$(u, v)$:
            **if** $v.d > u.d + w(u, v)$:
                $v.d = u.d + w(u, v)$
                $v.\pi = u$
```

Shortest Path Algorithms

	BFS	Dikstra’s	Bellman Ford
Complexity	O(V+E)	O((V+E)logV)	O(VE)
Recommended graph size	Large	Large/Medium	Medium/Small
Good for APSP	Only unweighted graphs	Ok	Bad
Can detect negative cycles	No	No	Yes
SP on graph with weighted edges	Incorrect SP answer	Best algorithm	Works
SP on graph with unweighted edges	Best algorithm	Ok	Bad

Competitive Programming 3, P. 161, Steven & Felix Halim

NP Complete

Classes of problems

P (Polynomial-time)

- Most of the algorithms studied are polynomial time.
- On the input size of *n,* the running time is $O(n^k)$

  for some constant k.
- All the problems can be solved in P? Answer is no.

NP (Nondeterministic polynomial time)

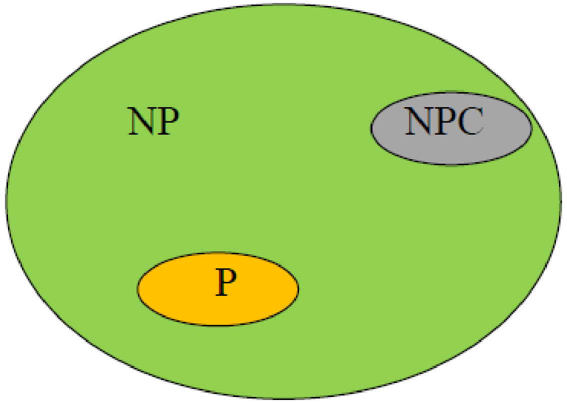
- Decision problems that are verifiable in polynomial time.
- Super-polynomial time for solving problems.
- Any problem in P is also in NP.

Classes of problems

NPC (NP-complete)

- A problem is in NP.
- It is as hard as any problem in NP.
- Can NPC problems be solved in polynomial time?

Not found yet.



Current view
$P \cap NPC = \emptyset$

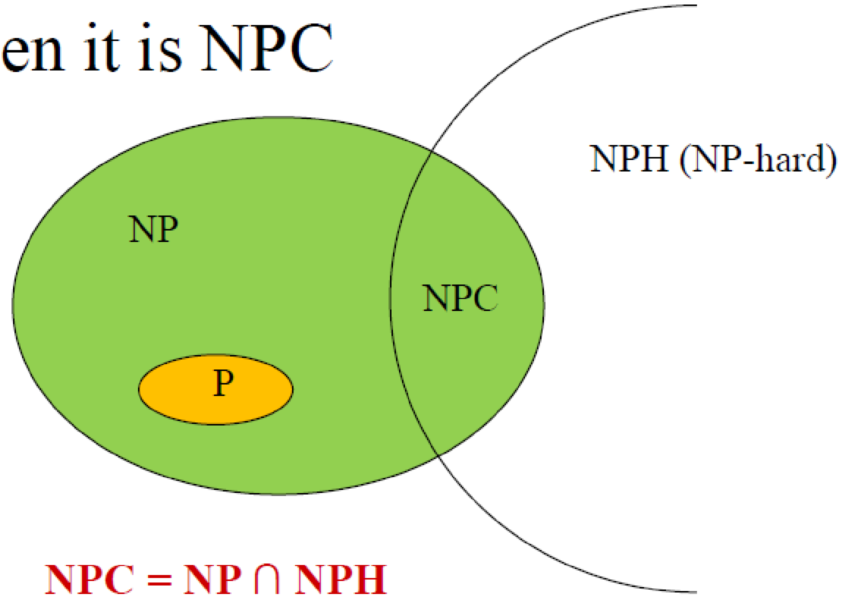
Classes of problems

NPH (NP-hard) and NPC

- A problem X is in NPH if every problem in NP reduces to the problem X.

(X is NP-hard if every problem Y∈NP reduces to X => (this implies that) X∉P unless P=NP)

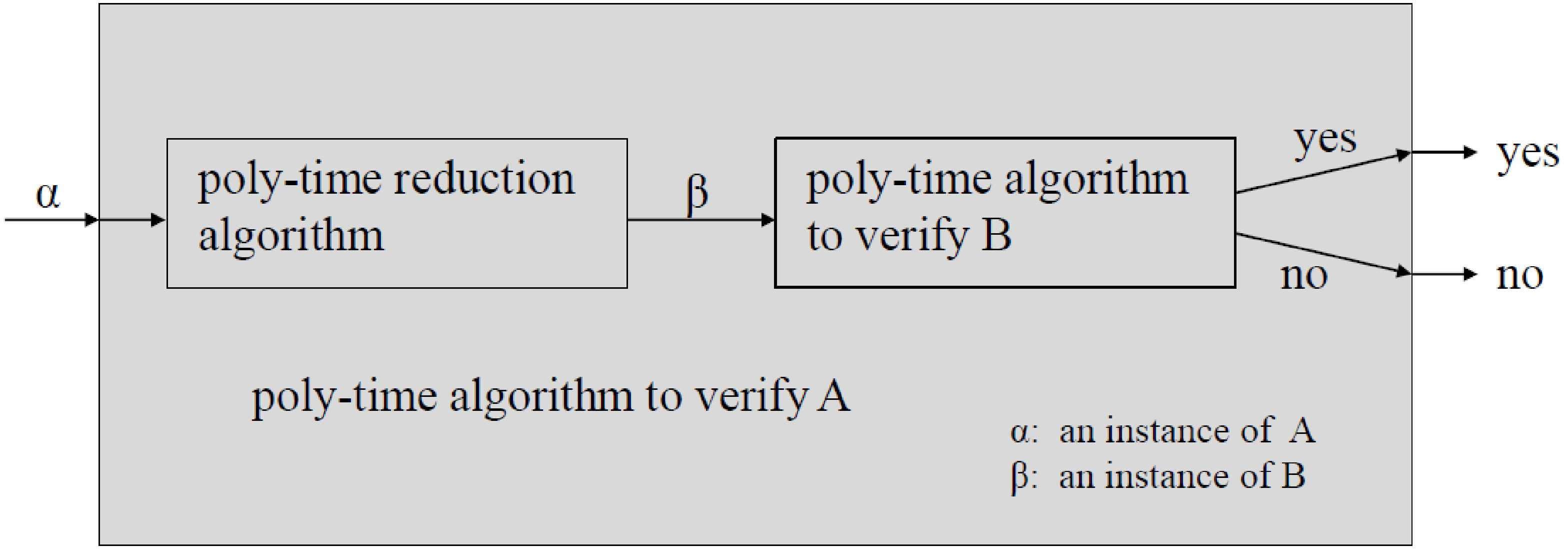
- If X is NP then it is NPC



NPC = NP ∩ NPH

Overview of showing NP-complete

Reductions


α
poly-time reduction algorithm
β
poly-time algorithm to verify B
yes
no
yes
no
poly-time algorithm to verify A
α: an instance of A
β: an instance of B

- A way to verify A in polynomial time (reducing verifying A to verifying B).

1. Given α of A, use the polynomial time reduction algorithm to transform it to β of B.
2. Run the polynomial time decision algorithm for B on β.
3. Use the answer for β as the answer for α.

Showing first NP-completeness

- A problem is NP-complete if

  1. It is NP
  2. It is NPH (Every problem in NP reduces to it in poly-time.)

- A problem Y is NP-complete if

  1. Y ∈ NP
  2. X ≤p Y for every X ∈ NP

     (Where, ≤p is the polynomial time reduction.)

- Circuit satisfiability problem: The first NP-complete problem.

NP-completeness proofs

Prove that a problem Y is NP-complete without directly reducing every problem in NP to Y.

1. If a problem X in NPC reduces to a problem Y then Y is NPH.
2. If Y is NP then Y is NPC.

Reference

- Charles Leiserson and Piotr Indyk, “*Introduction to Algorithms”,* September 29, 2004

- https://www.geeksforgeeks.org