Problem Solving:

Graph algorithm

May 2019

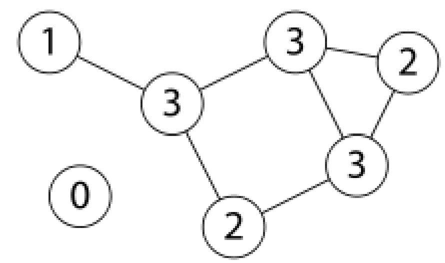
Honguk Woo

Degrees

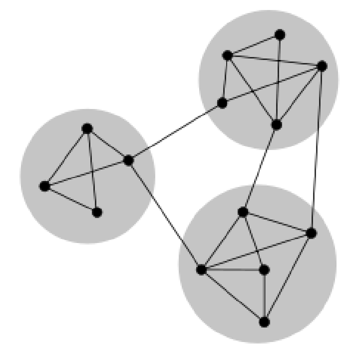
- Degree : number of edges connected to a vertex
- for undirected graphs
  - sum of all degrees = 2 X edges
- for directed graph
  - sum of in-degree = sum of out-degree

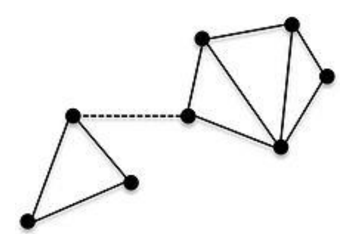
Connectivity

- Connected graph
  - where there exists a path between every pair of vertices (in an undirected graph)
  - Disconnected : not connected graph
  - *How to find the connected components in a graph ?*



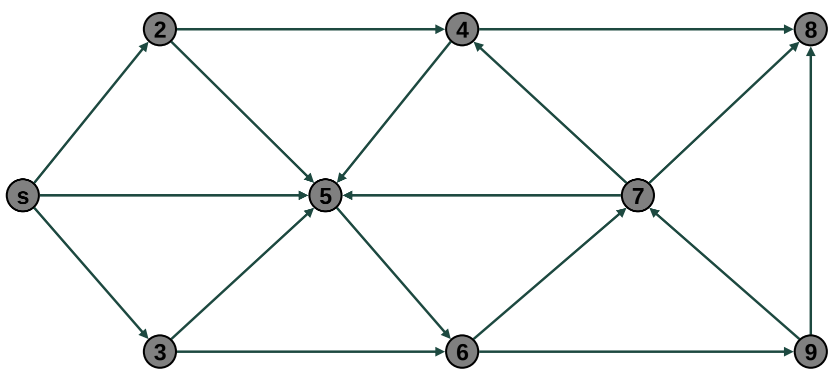
  - Articulation vertex : deleting this vertex makes the graph disconnected
    - a graph without any such vertex is biconnected
  - Bridge edge : deleting a bridge edge makes the graph disconnected
  - *How to find the articulation vertex (bridge edge) ?*

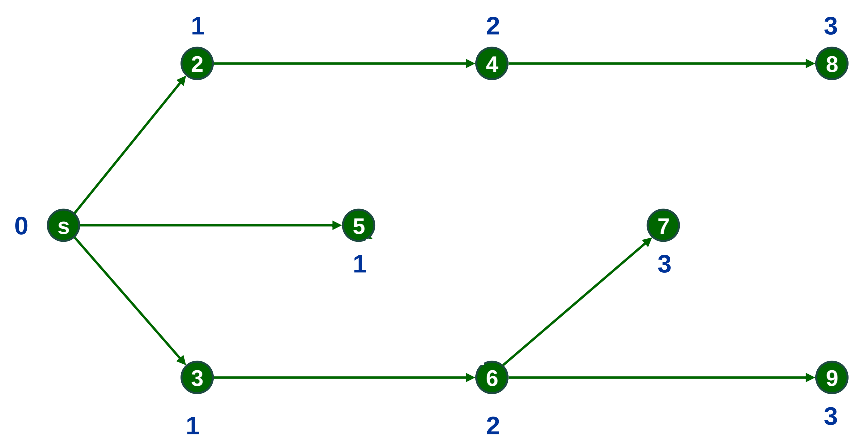




Spanning Tree

- Note that tree has no cycle
- Given a graph G = (V, E),
  Spanning tree T = (V, E′) such that
  - E′ ⊂ E
  - **Connecting all vertices of V**





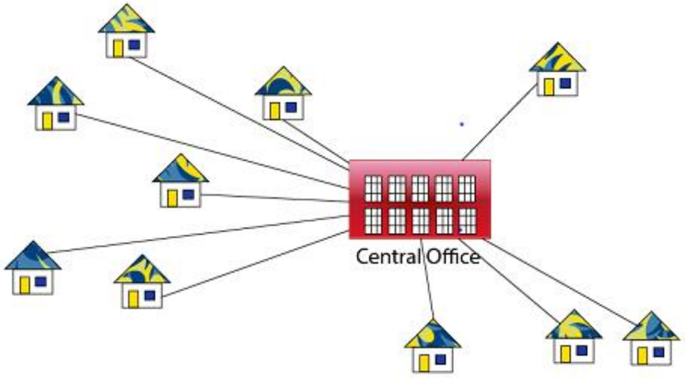
- A spanning tree can be constructed using DFS or BFS

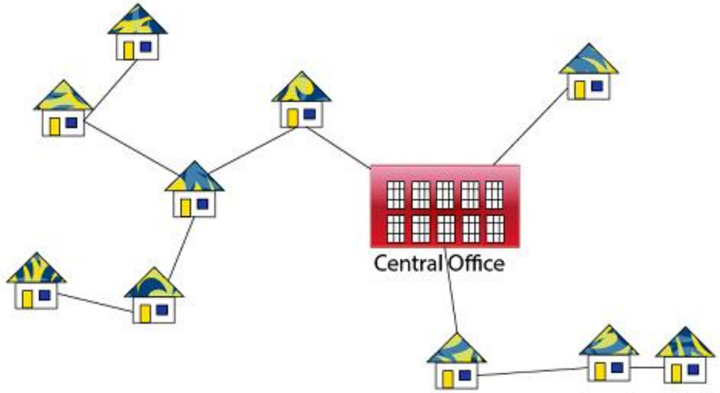
Q : minimize the total cost of connections

- Suppose we want to connect a set of houses for telephone lines (e.g., cable length = cost)

Q : minimize the total cost of connections

- Suppose we want to connect a set of houses for telephone lines (e.g., cable length = cost)

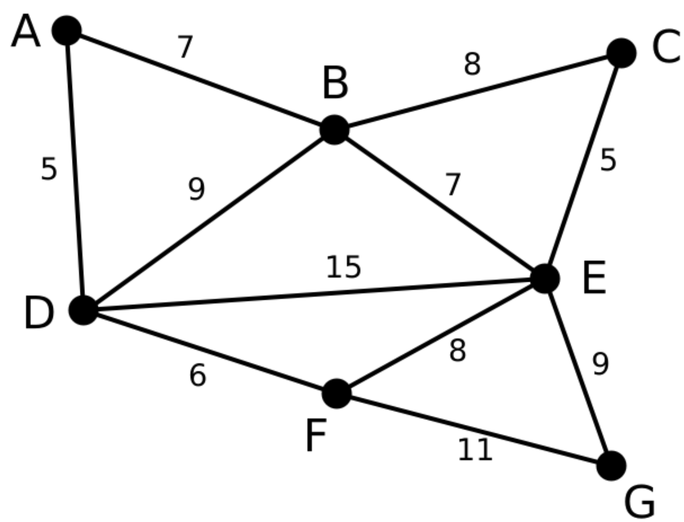




For n vertices,
How many edges ?
How many paths between two vertices ?
Two trees connected by a single edge

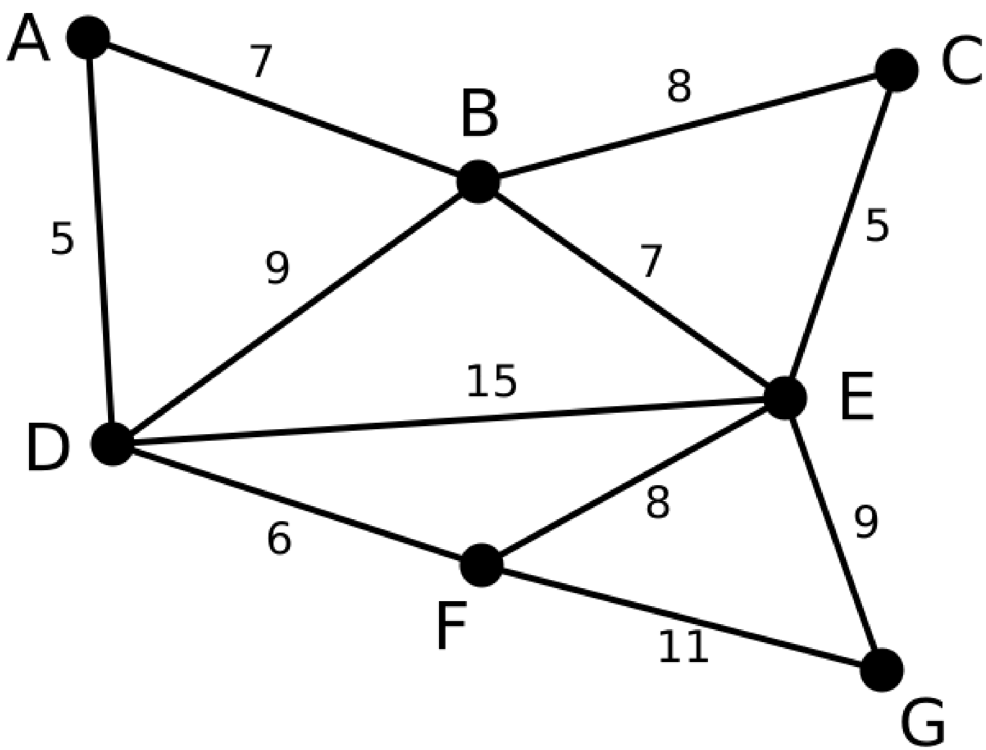
Minimal Spanning Tree (MST)

- Spanning tree whose sum of edge weights is minimal
  - The smallest connected graph in terms of edge weights → *How to find it ?*
  - If there is no weight, the number of edges is minimal ?
    - Spanning tree has n-1 edges (for n vertices)

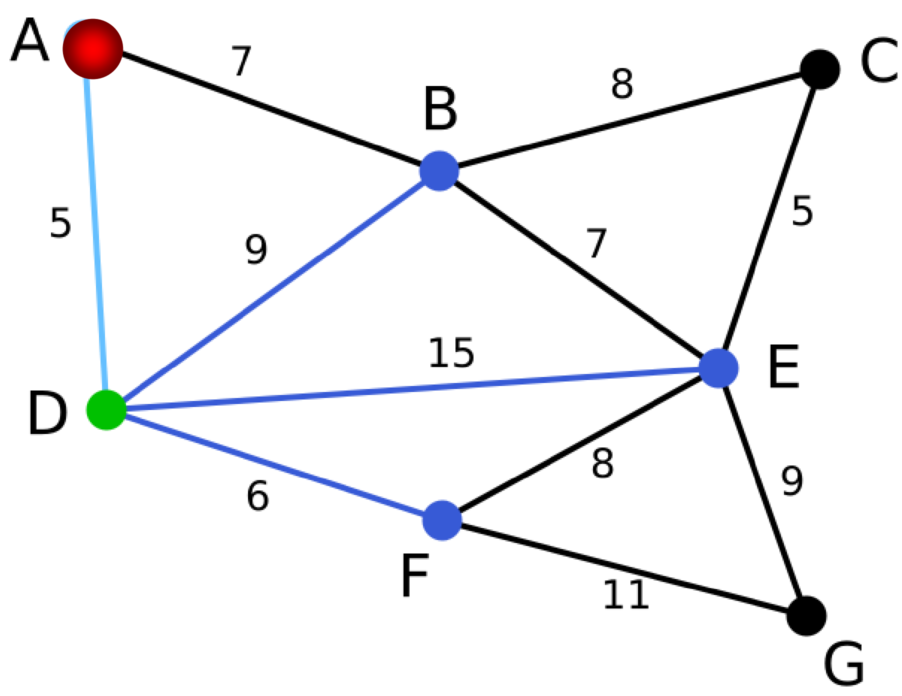


Example of MST: **Prim's Algorithm**

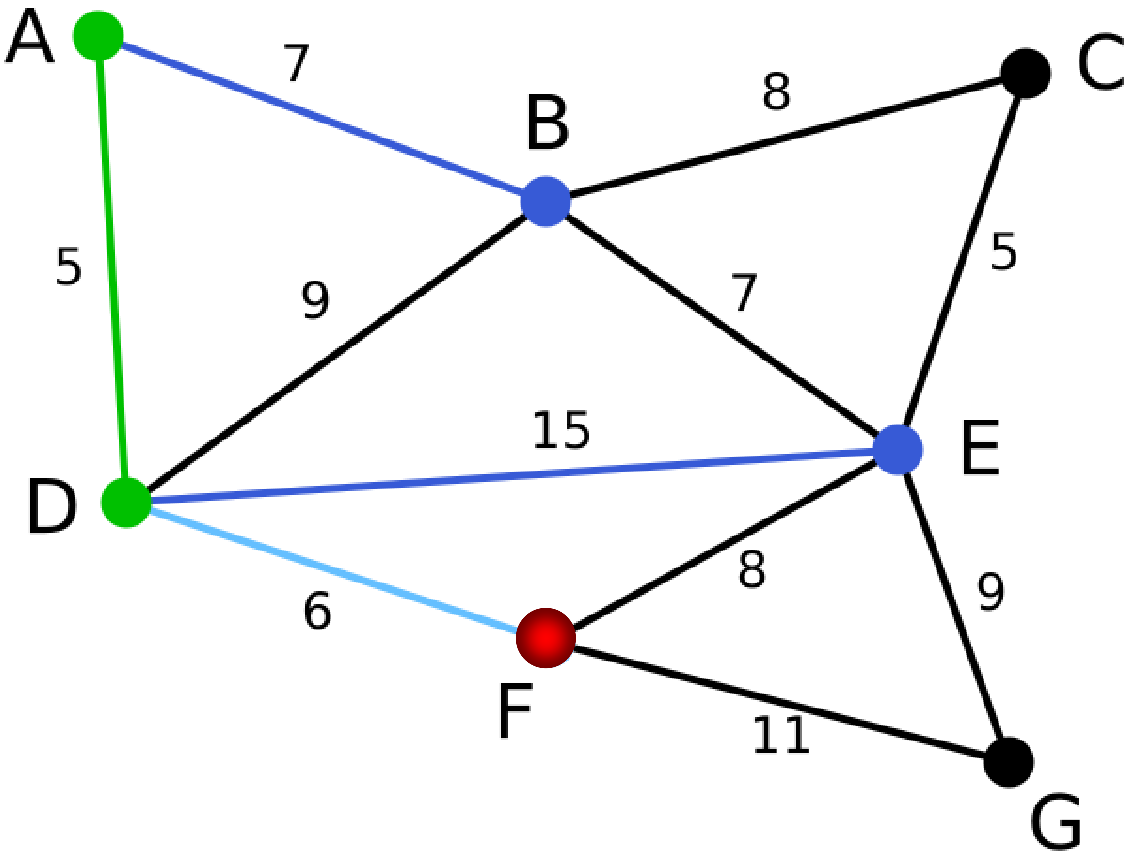
- **Greedy algorithm :** expanding MST by adding the nearest vertex



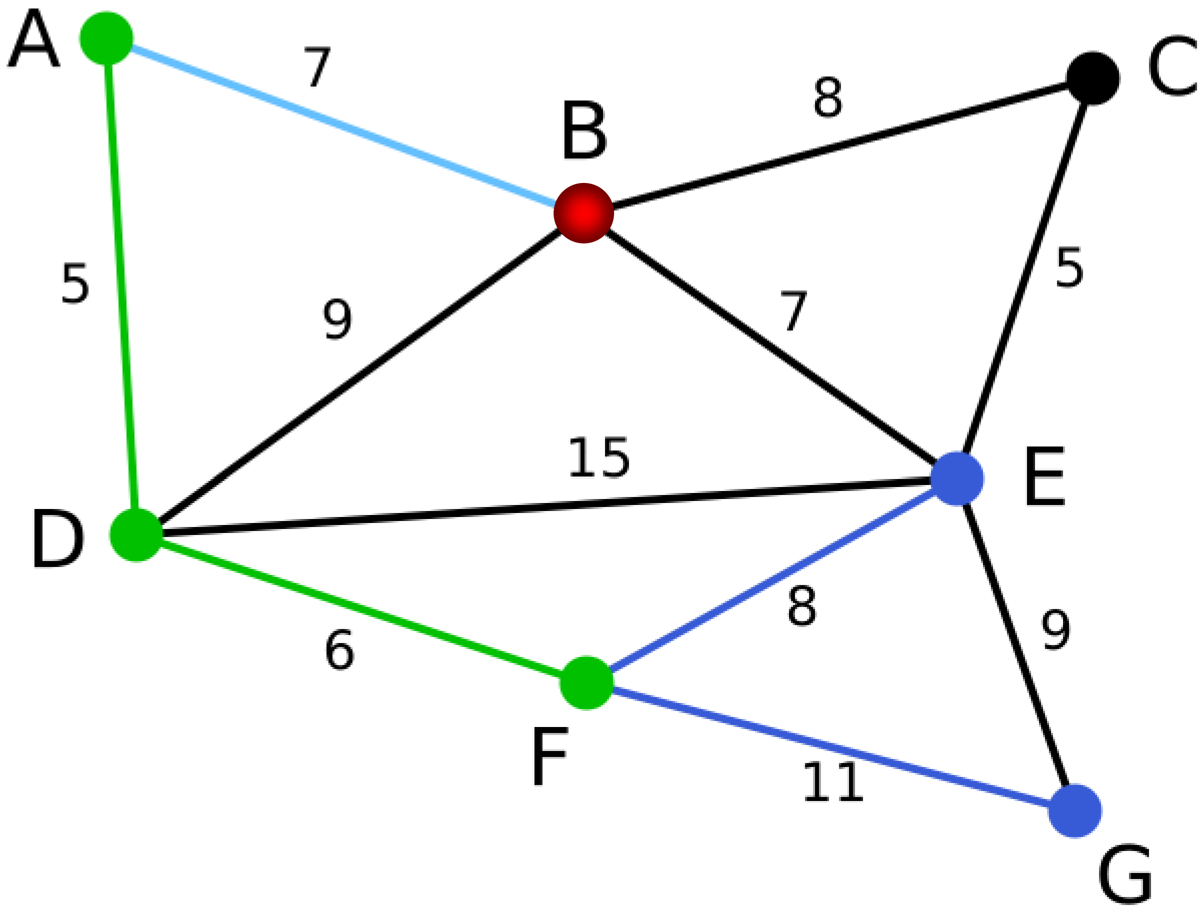
1. Vertex D has been chosen as a starting point
   ① Vertices A, B, E, F are connected to D through a single edge.
   ② A is the nearest to D and thus chosen as the 2nd vertex along with the edge AD



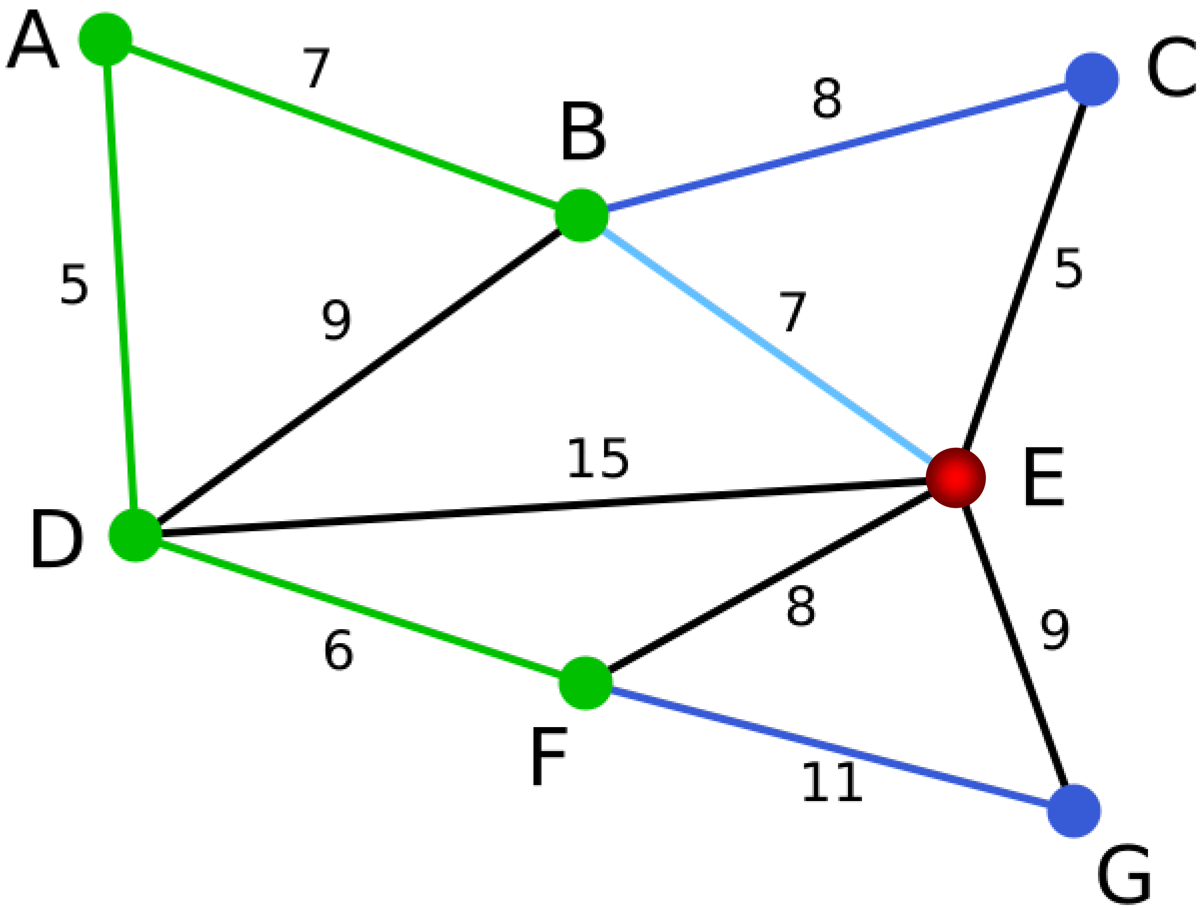
2. The next vertex chosen is the vertex nearest to either D or A. So the vertex F is chosen along with the edge DF



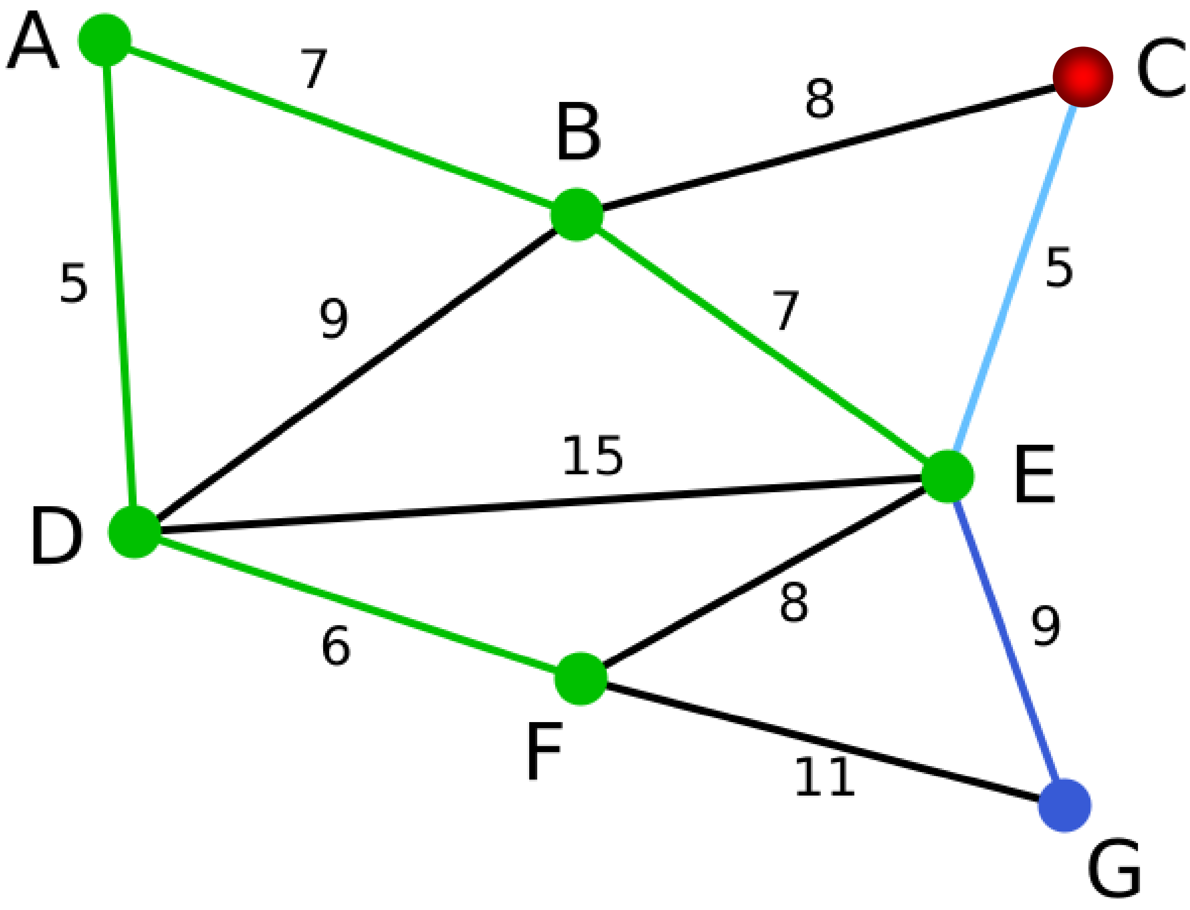
3. same as 2, Vertex B is chosen.



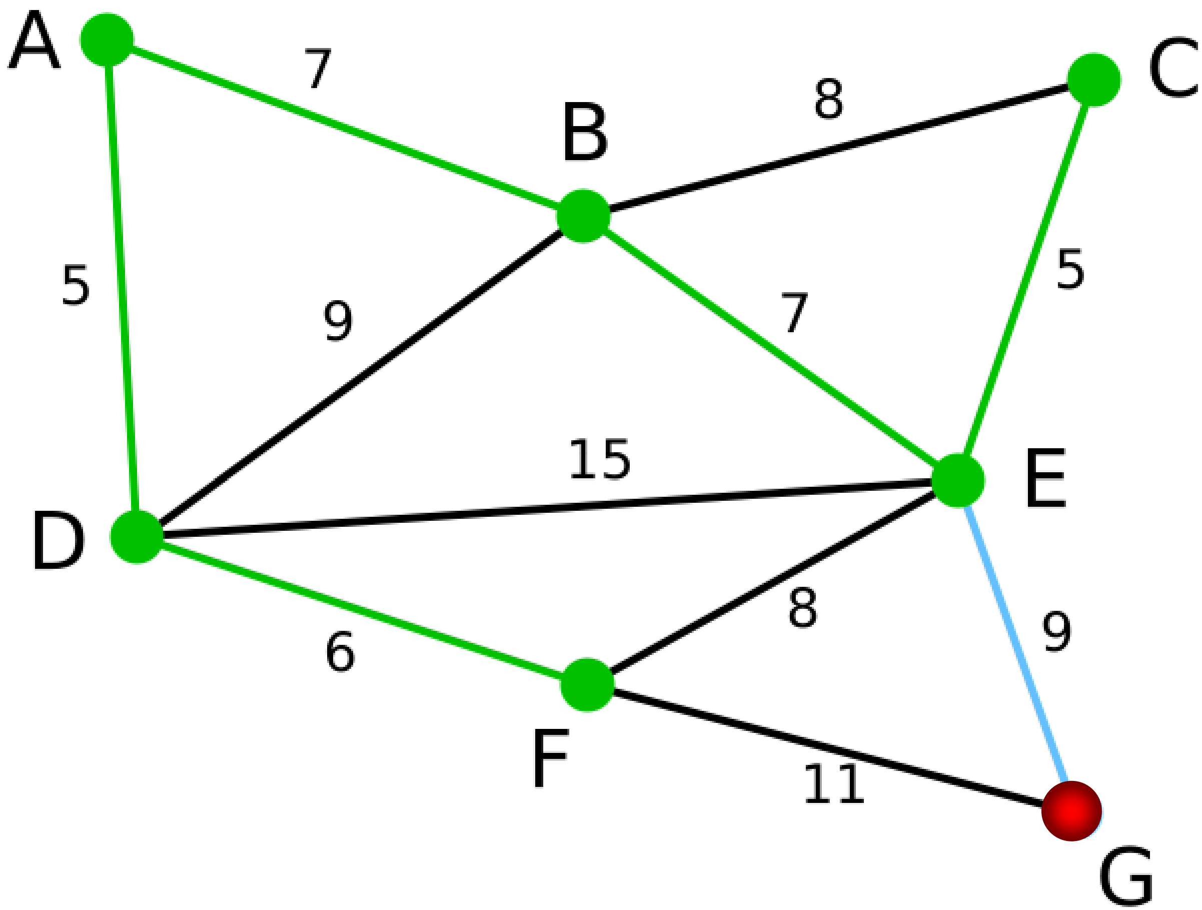
4. among C, E, G, E is chosen.



5. among C and G, C is chosen.

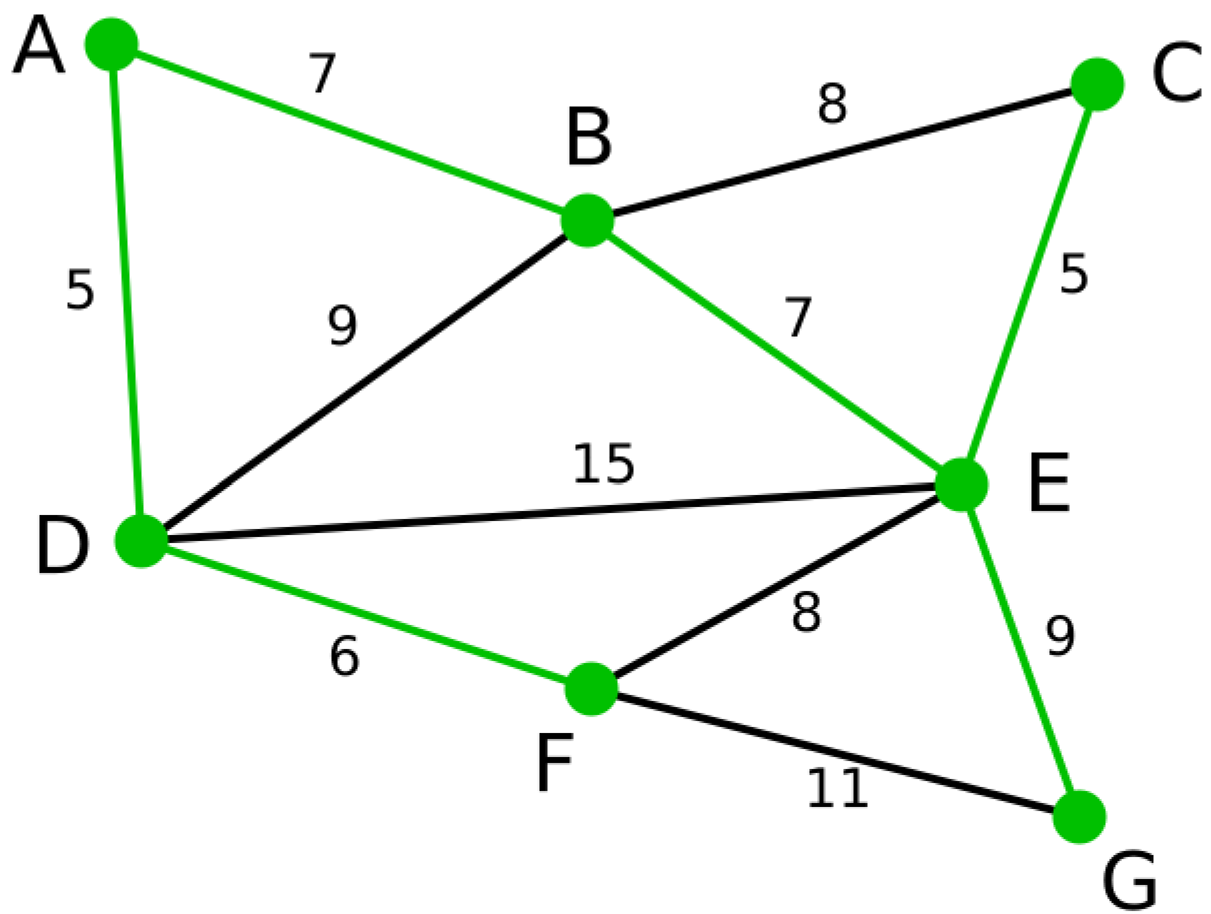


6. G is the only remaining vertex.



7. The finally obtained minimum spanning tree

$\Rightarrow$ the total weight is 39

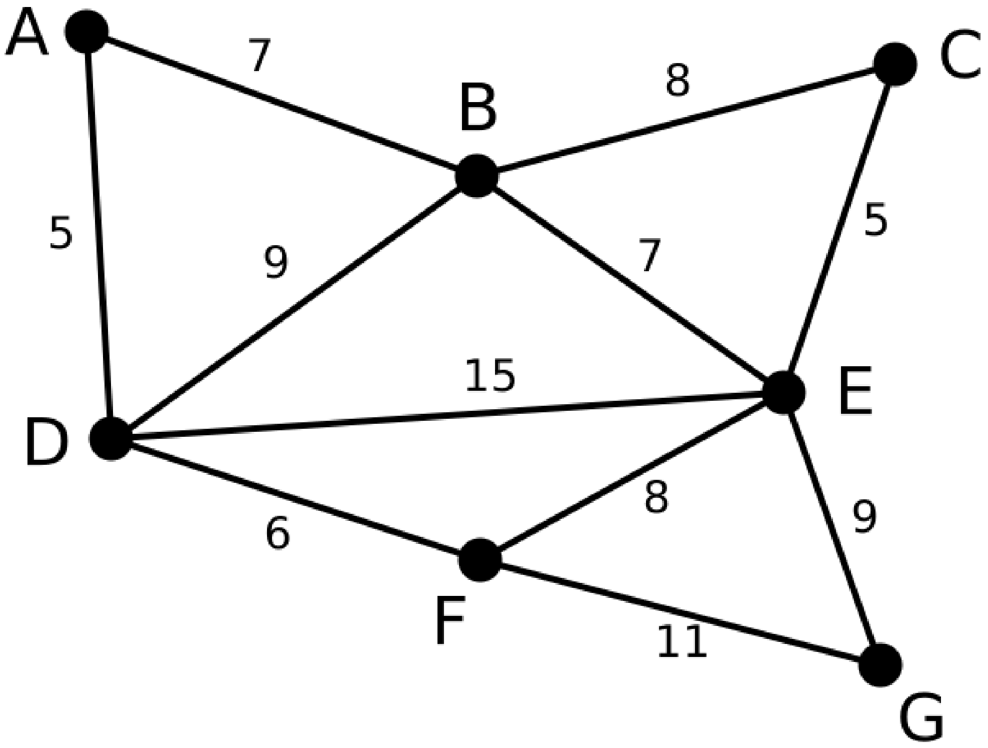


Prim's algorithm

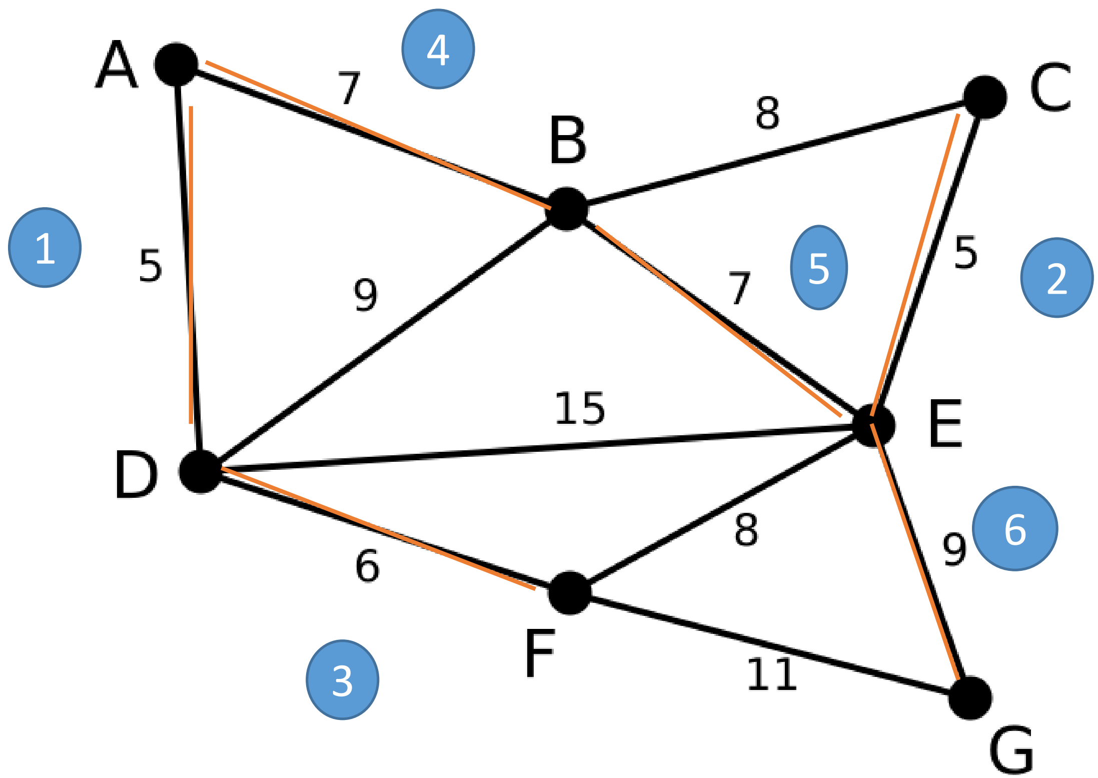
```
ReachSet = {0};          // You can use any node
  UnReachSet = {1, 2, ..., N-1};
  SpanningTree = {};

  while ( UnReachSet ≠ empty )
  {
     Find edge e = (x, y) such that:
     1. x ∈ ReachSet
     2. y ∈ UnReachSet
     3. e has smallest cost

     SpanningTree = SpanningTree ∪ {e};

     ReachSet   = ReachSet ∪ {y};
     UnReachSet = UnReachSet - {y};
  }
```

Another algorithm : Kruskal

- Prim (vertex-centric), Kruskal (edge-centric)
- What if choosing the smallest edge and adding it to MST (as long as it connects two trees in a single tree, expanding MST) ?



- What if choosing the smallest edge and adding it to MST (as long as it does not create a cycle) ?



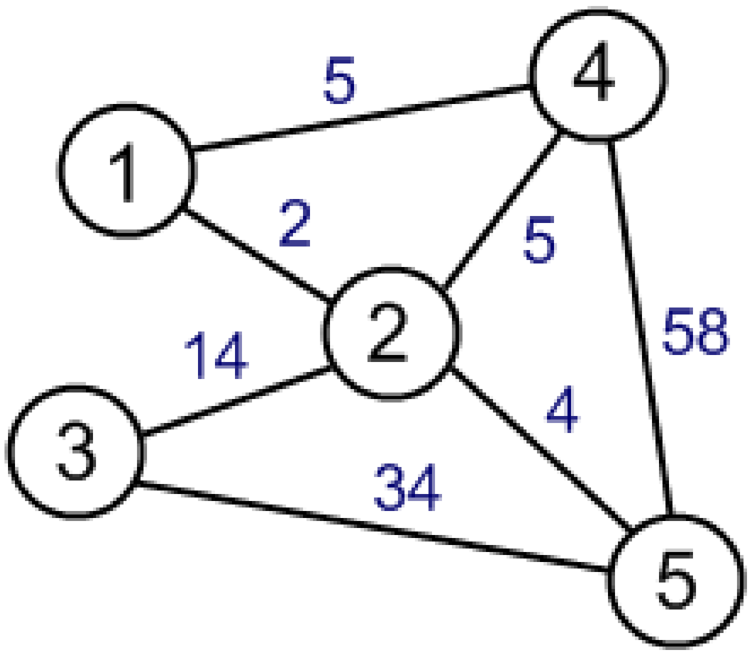
Q : what kind of spanning tree ?

- Suppose we hire an evil telephone company to connect a bunch of houses together, and that this company will be paid a price proportional to the amount of wire they install.

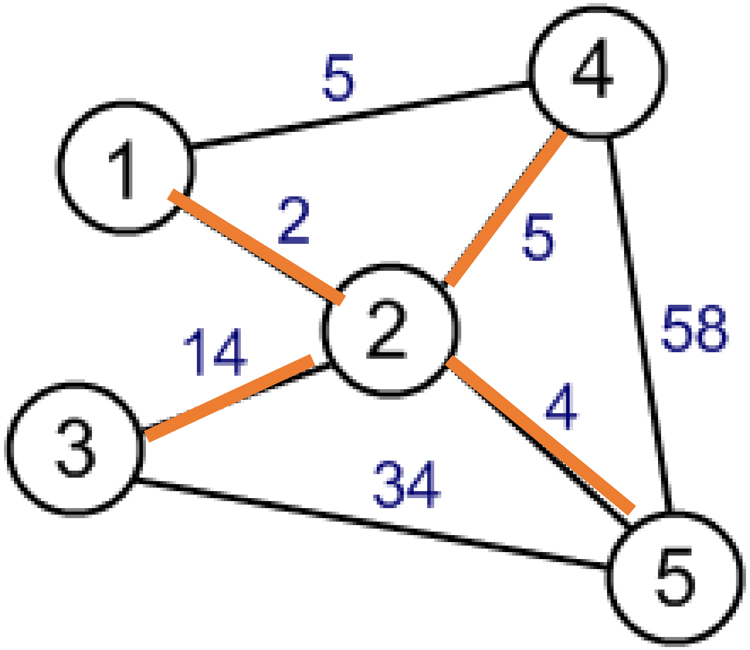
Shortest path problem

- The shortest path problem is about finding a path between vertices in a graph such that the total sum of the edges weights is minimum
  - Single-source : the shortest path from a source vertex to all other vertices
    - Dijkstra's algorithm : in case of non-negative edge weight
  - All-pairs : the shortest paths between every pair of vertices
    - Floyd-Warshall

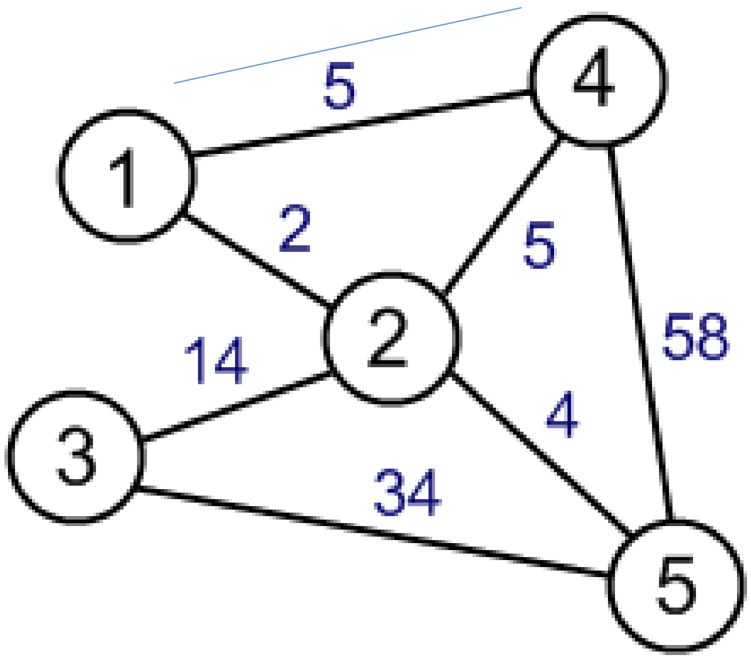
Q : Shortest path is different from MST ?



MST

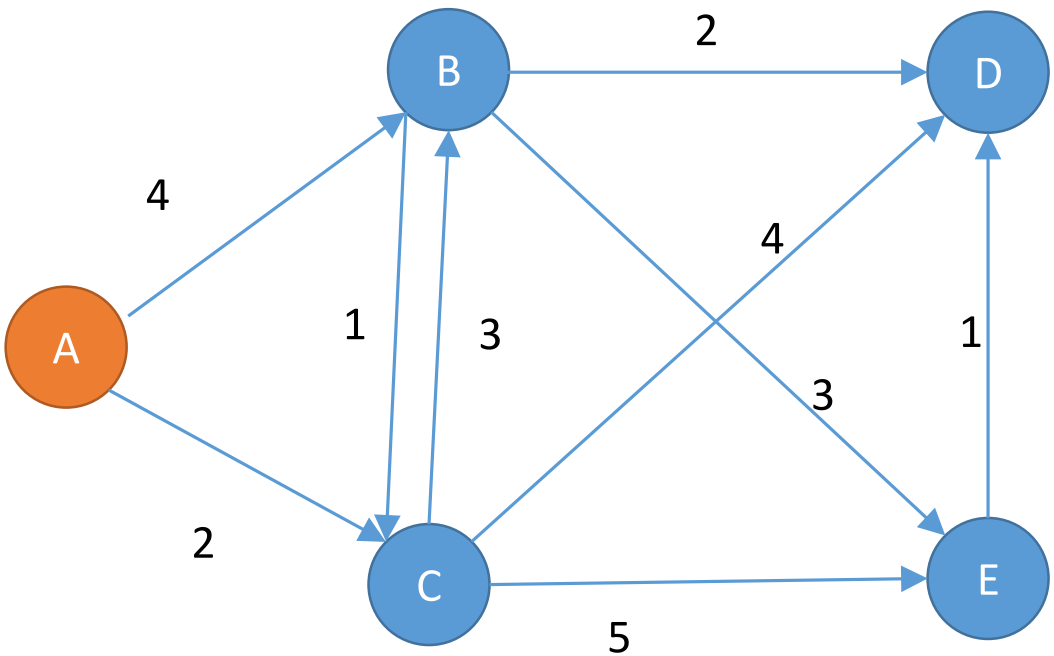


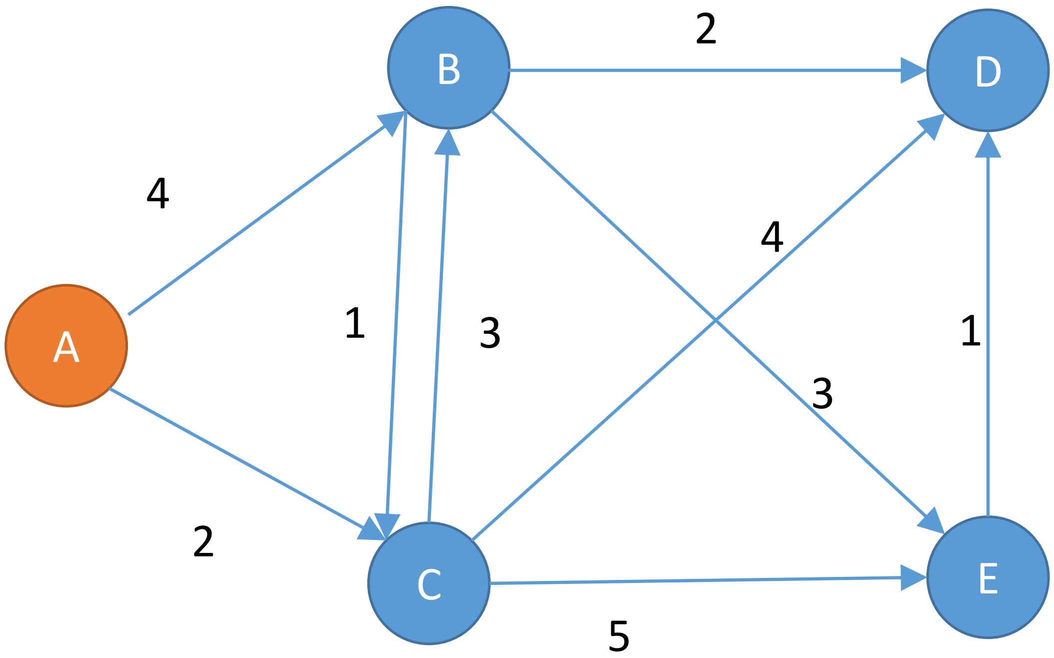
Shortest path
between 1 and 4



Dijkstra’s Algorithm

- Goal: Find the shortest path from **s (source) to t (all the other vertices)**





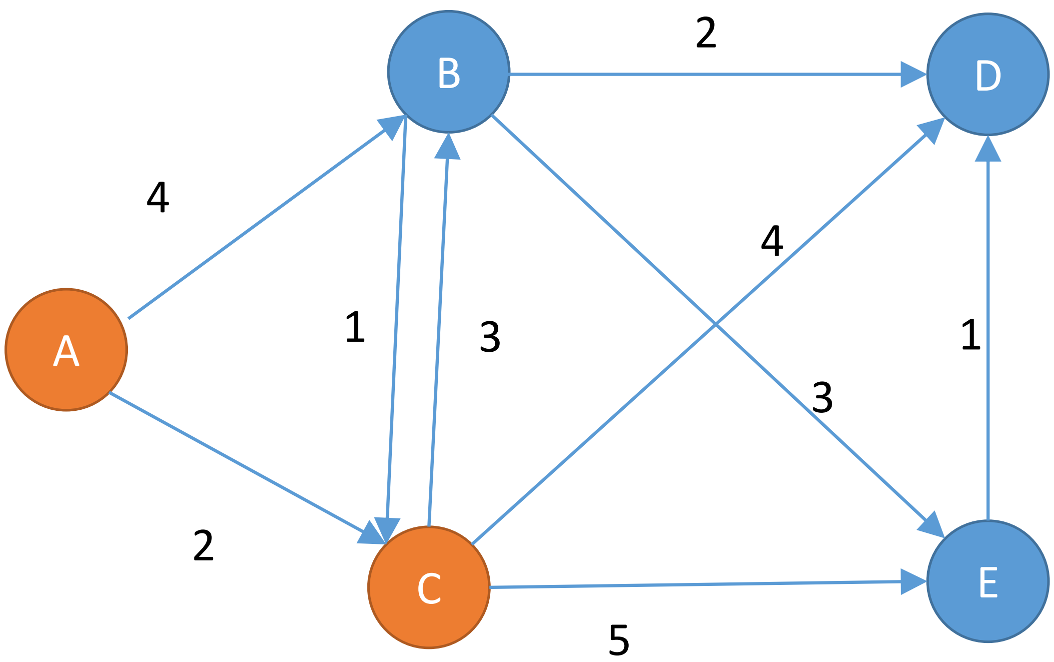
Unvisited : ~~A~~ B C D E

Distance :

A	B	C	D	E
0	IF	IF	IF	IF

3. Add C in SP
(C was the smallest path among unvisited)

Unvisited : ~~A~~ B ~~C~~ D E

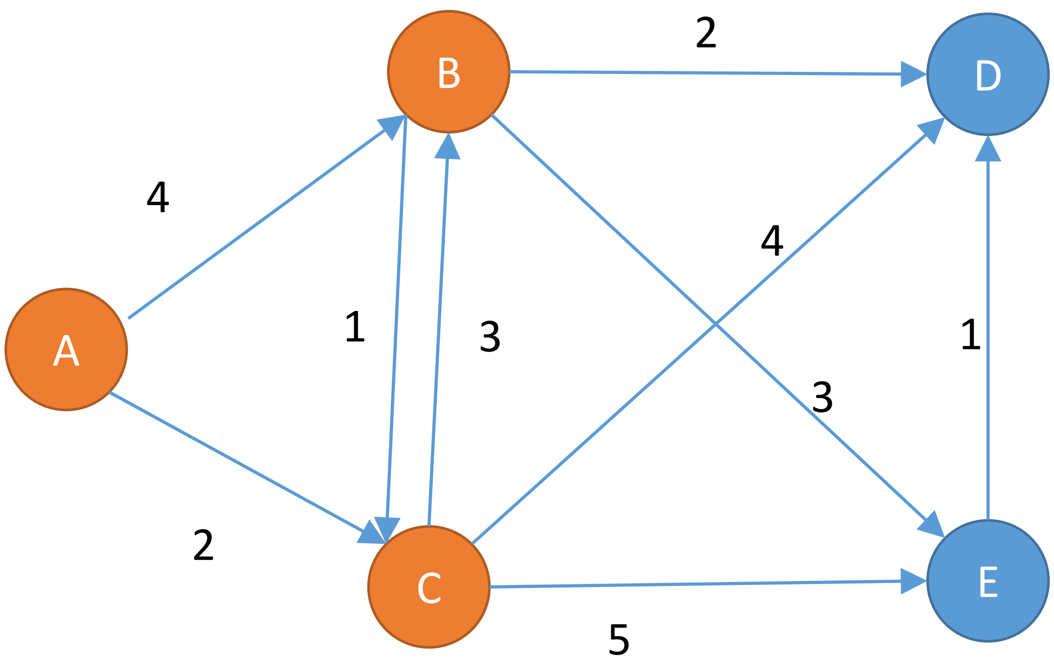


Distance :

A	*B*	*C*	D	E
0	*4*	*2*	IF	IF

2. Update the distance

1. Examine the edge from A



3. Add B in SP
(B was the smallest path among unvisited)

Unvisited : ~~A~~ ~~B~~ ~~C~~ D E

Distance :

A	*B*	C	*D*	*E*
0	*3*	2	*6*	*7*

2. Update the distance

1. Examine the edge from C

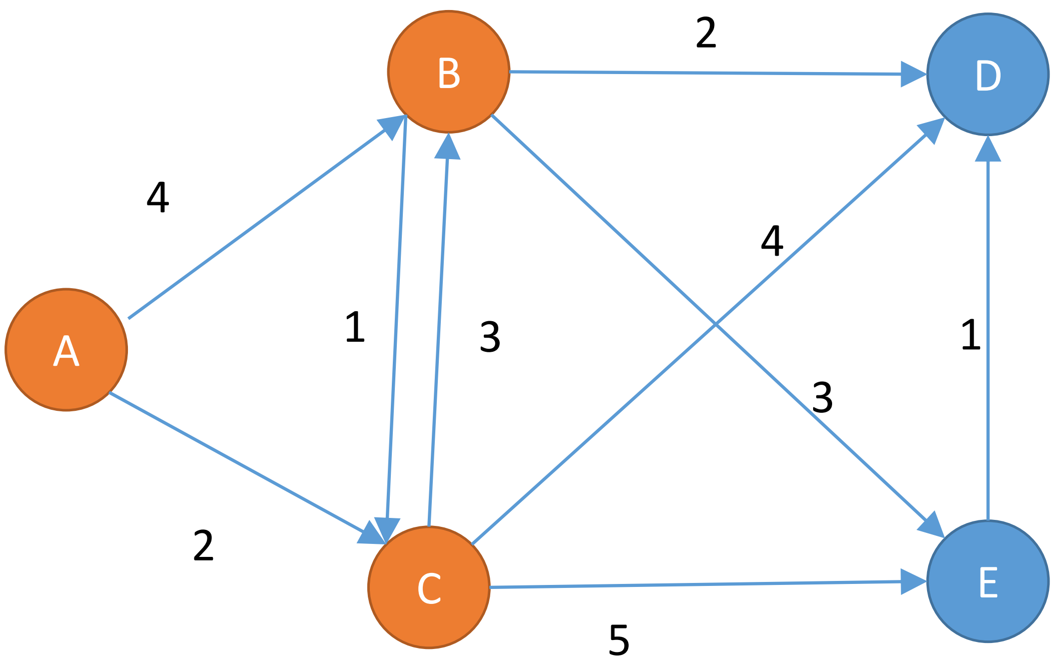
Unvisited : ~~A~~ ~~B~~ ~~C~~ D E

Distance :

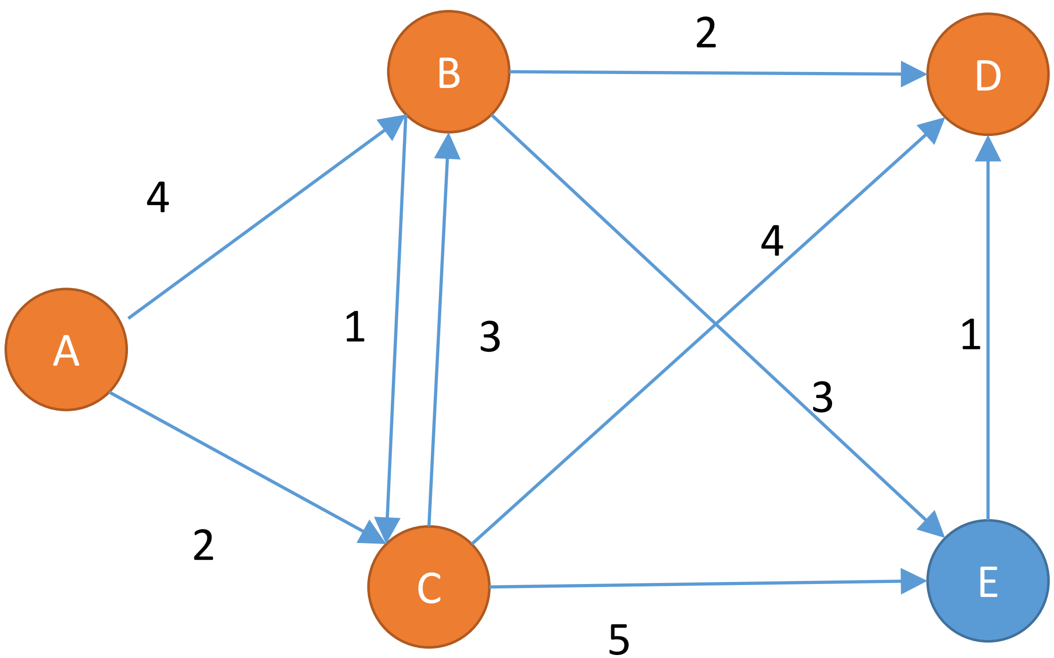
A	B	C	D	E
0	3	2	5	6

2. Update the distance

D, E are updated
C is not



1. Examine the edge from B



3. Add D in SP
(D was the smallest path among unvisited)

Unvisited : ~~A~~ ~~B~~ ~~C~~ ~~D~~ E

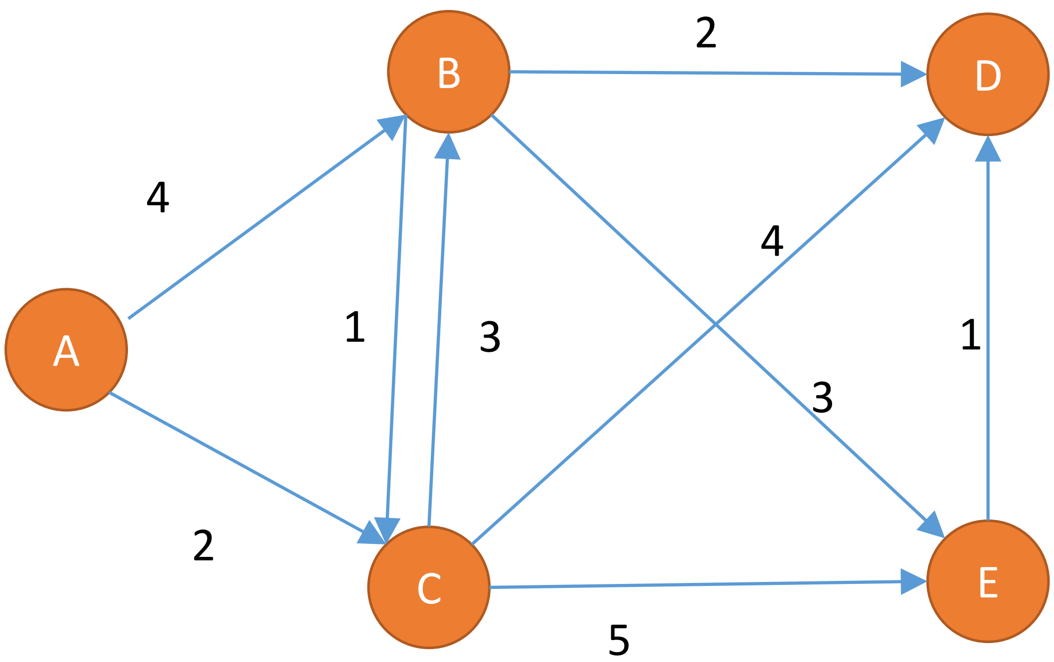
Distance :

A	B	C	D	E
0	3	2	5	6

2. Update the distance

1. Examine the edge from B

3. Add E in SP



Unvisited : ~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~

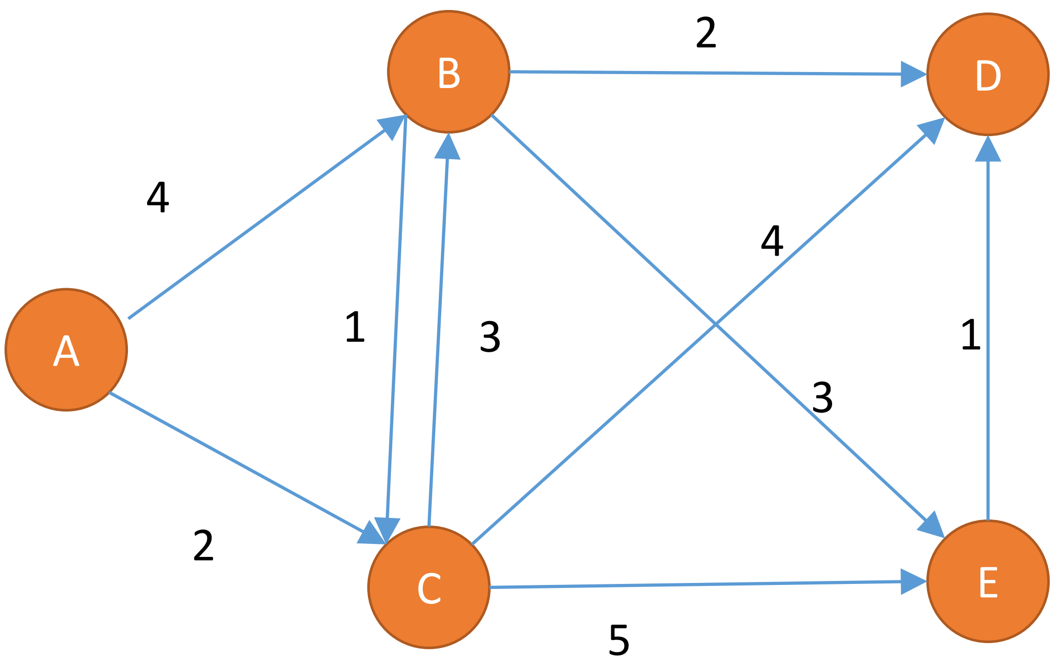
Distance :

A	B	C	D	E
0	3	2	5	6

2. No update by D

1. Examine the edge from D

3. Add E in SP



Unvisited : ~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~

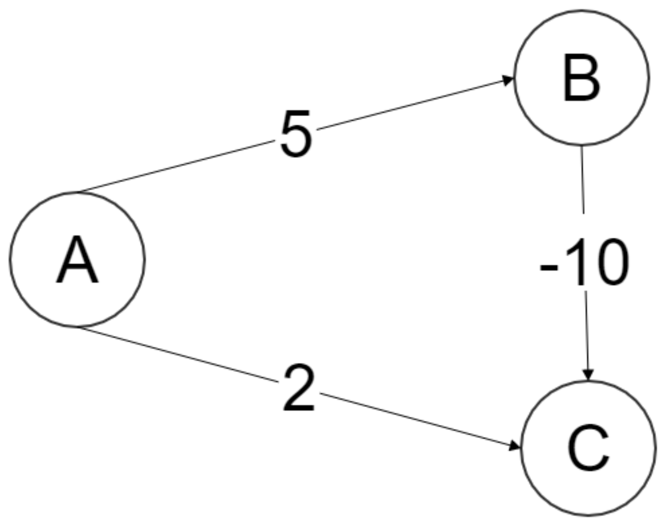
Distance :

A	B	C	D	E
0	3	2	5	6

2. No update by D

1. Examine the edge from D

Q: What if having negative edges ?



All Pairs Shortest Paths

- Use Dijkstra's method for all the vertices (for every source)
  - complexity?

- Floyd-Warshall algorithm
  - Given the adjacency matrix with vertices numbered (1..n)

$$W[i,j]^k = \min(W[i,j]^{k-1}, W[i,k]^{k-1} + W[k,j]^{k-1})$$

  - Can contain negative edge weights

Floyd-Warshall

```
Procedure Floyd-Warshall(Graph):
for k from 1 to V     // V : number of vertex
    for i from 1 to V
        for j from 1 to V
            if distance[i][j] > distance[i][k] + distance[k][j]
                distance[i][j] = distance[i][k] + distance[k][j]
                path[i][j] = path[k][j]
            end if
        end
    end
end
```