UNIVERSITY OF CONNECTICUT

Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

(CSE – 2050)

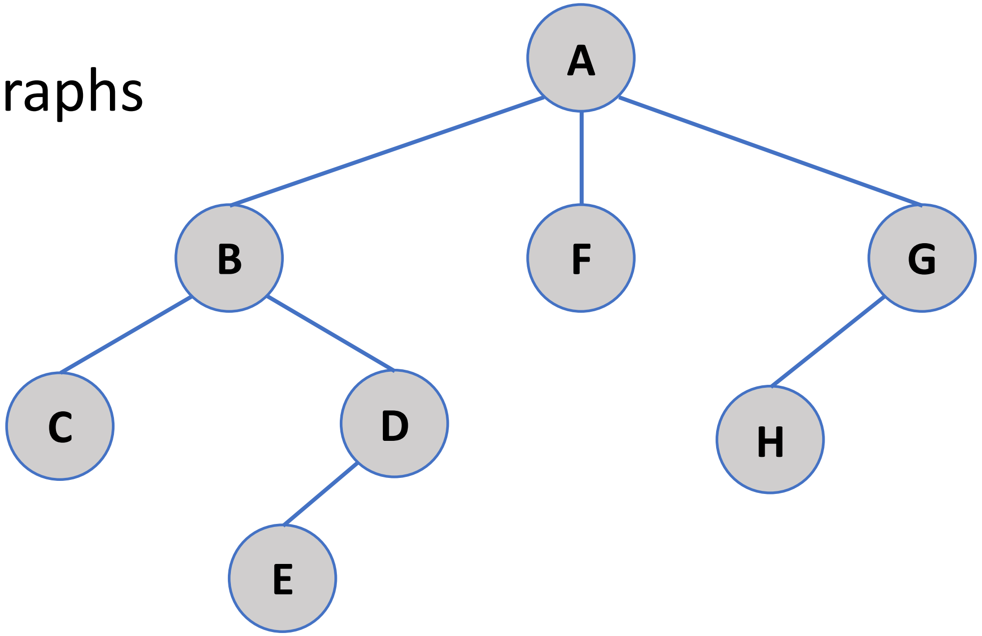
Hasan Baig

Office: UConn (Stamford), 305C
email: hasan.baig@uconn.edu

Quick Recap

Graph Traversal

- Breadth-First Search
  - Constructs a shortest path for unweighted graphs
  - Based on Queue (FIFO)
  - V = ABFGCDHE
- Depth-First Search
  - Used to solve mazes
  - Based on Stacks (LIFO)
  - ABCDEFGH



Quick Recap

- A priority queue stores a collection of items in a (key, value) pair
  - Key → defines the priority
  - Value → the actual data

- Highest priority element is the one to come out first

- **Heaps** are data structures that are used to implement priority queues (ADT)
  - Max Heap → highest priority element has the maximum value
  - Min Heap → highest priority element has the minimum value

Shortest Path Problem

*"In graph theory the **shortest path problem** is the problem of finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimized"*

Algorithms

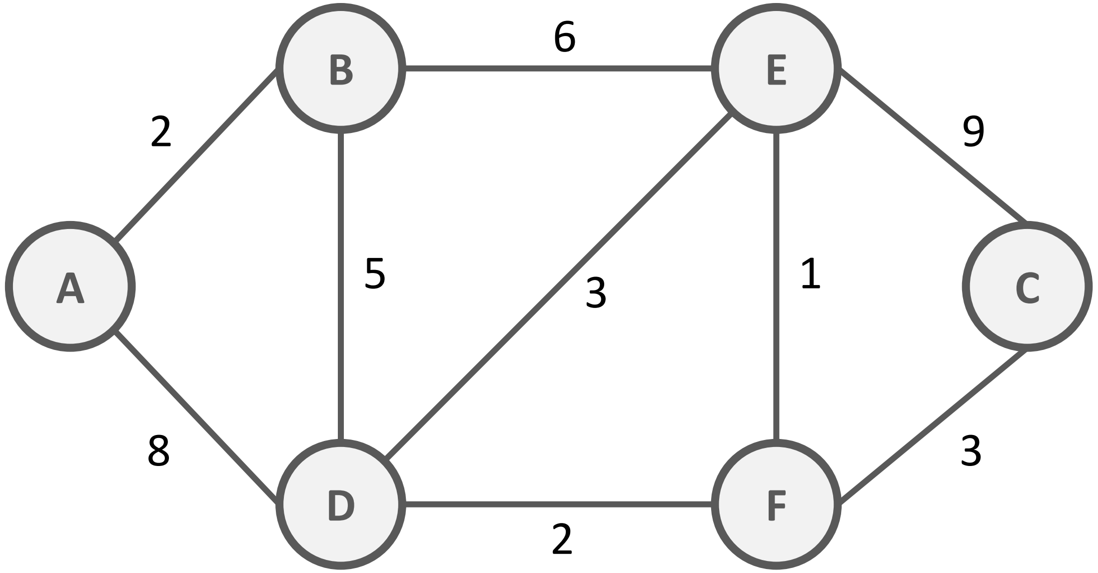
1. **Dijkstra's**
2. Bellman-Ford
3. A* search
4. Floyd-Warshall

Dijkstra’s Algorithm

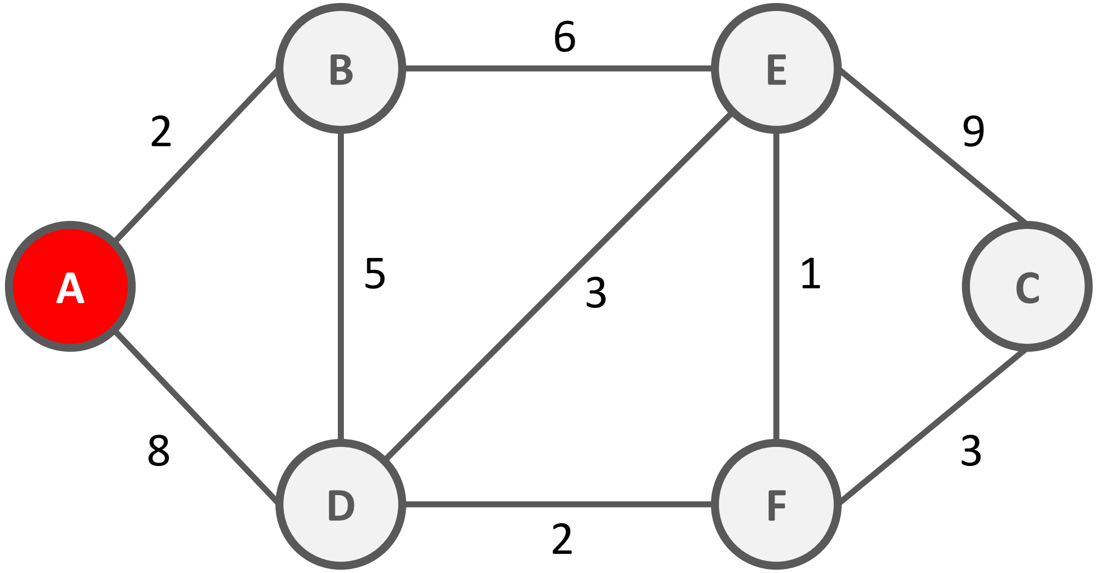
- Invented by Dutch scientist Edsger Dijkstra
- It can handle positive edge weights
- It can find the shortest path in a G(V, E) graph from vertex u to v, alongside constructing a shortest path tree as well

- It solves the problem using greedy approach
- During every iteration, it searches for the minimum distance to the next vertex
- The appropriate data structure is a Heap (Priority Queue)

Dijkstra's Algorithm



Dijkstra's Algorithm



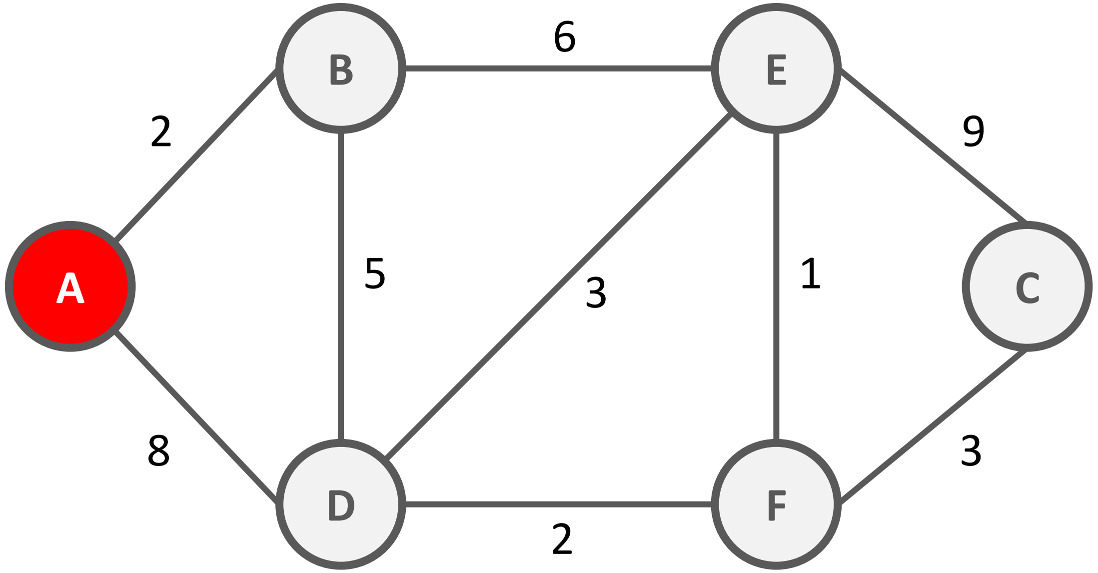
PQ = [A:0, B: ∞, C: ∞, D: ∞, E: ∞, F: ∞]

V = []

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

- Choose a starting vertex and set its minimum distance from itself to 0
- Assume all vertices are located at infinite distance from the starting vertex
- Maintain the list of visited vertices
- Keep updating the shortest distance and the predecessor of each vertex

Dijkstra's Algorithm



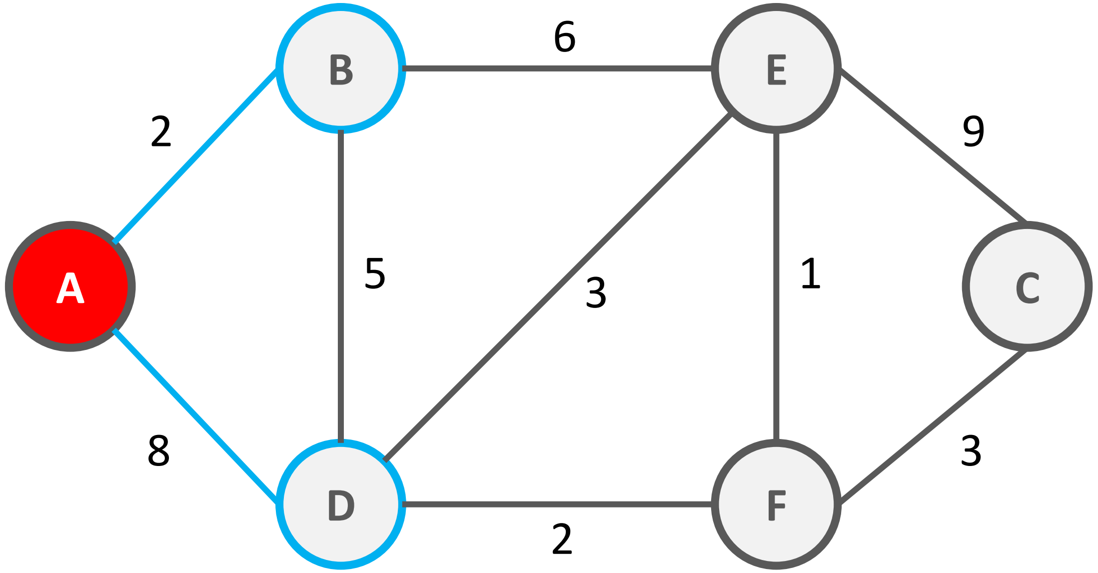
PQ = [A:0, B: ∞, C: ∞, D: ∞, E: ∞, F: ∞]

V = []

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors

Dijkstra's Algorithm



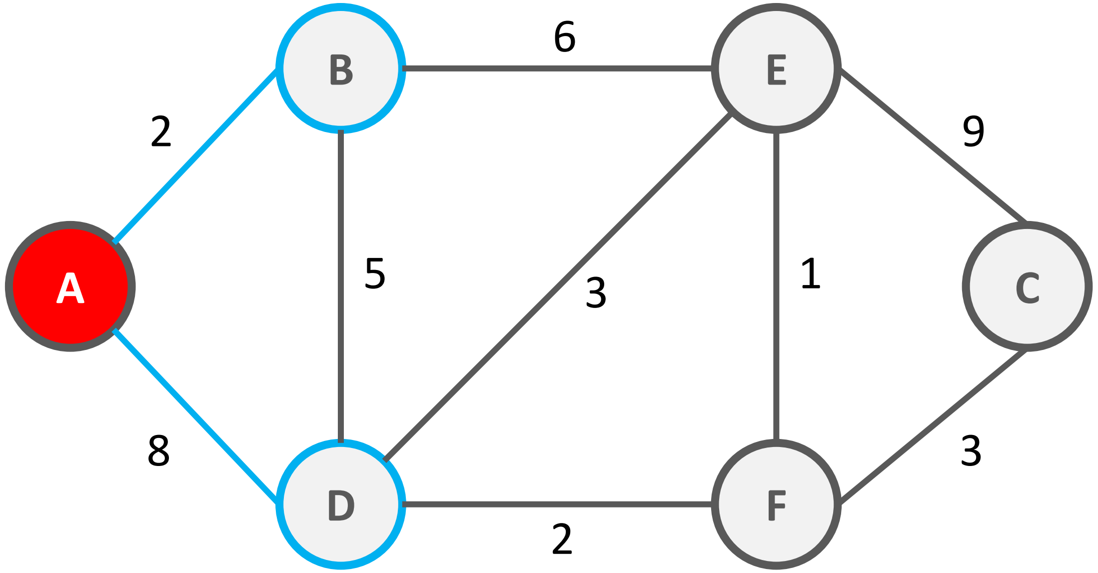
1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation

PQ = [A:0, B: ∞, C: ∞, D: ∞, E: ∞, F: ∞]
V = []

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

```
if D[u] + e(u, v) < D[v]:
        D[v] = D[u] + e(u, v)
        v.predecessor = u
```

Dijkstra's Algorithm



PQ = [A:0, B: ∞, C: ∞, D: ∞, E: ∞, F: ∞]

V = []

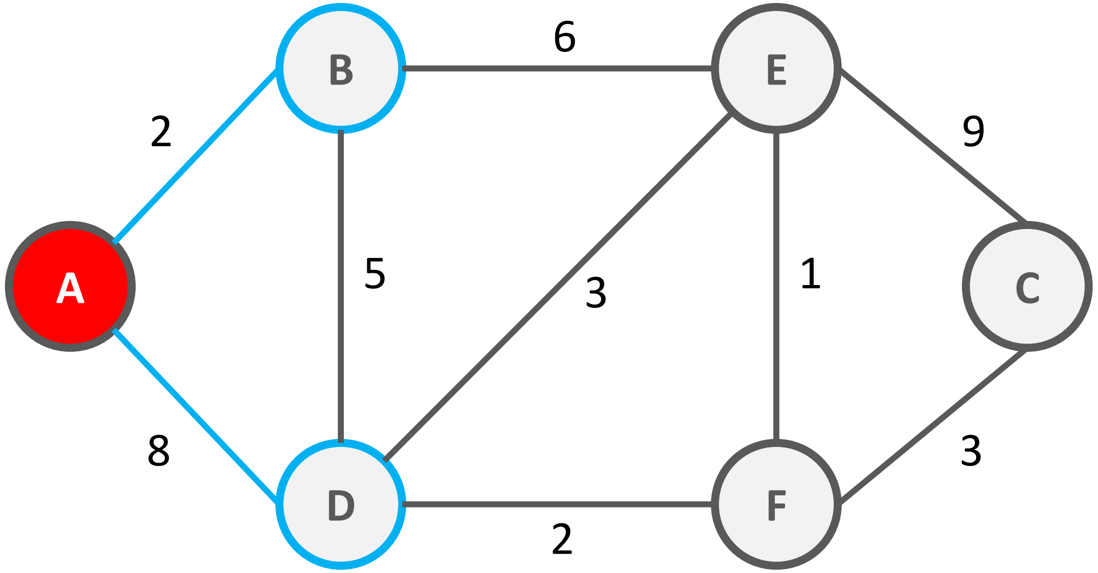
Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation

```
if D[A] + e(A, B) < D[B]:
        D[B] = D[A] + e(A, B)
        v.predecessor = u
```

```
if 0 + 2 < ∞ :
        D[B] = 0 + 2
        B.predecessor = A
```

Dijkstra's Algorithm



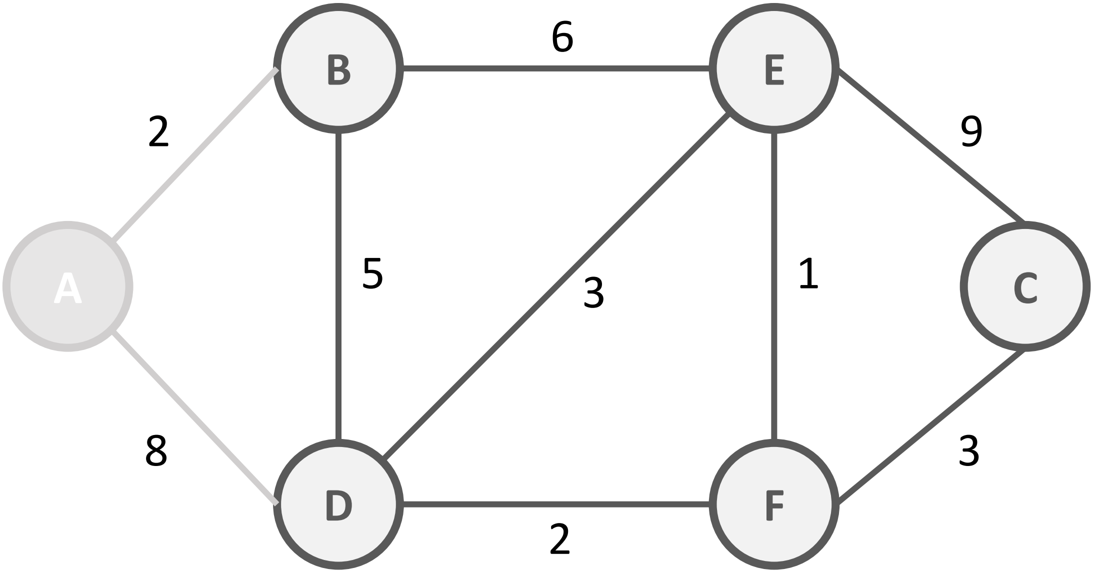
PQ = [A:0, B: 2, C: ∞, D: 8, E: ∞, F: ∞]

V = []

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	8	A
E	∞	
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation

Dijkstra's Algorithm



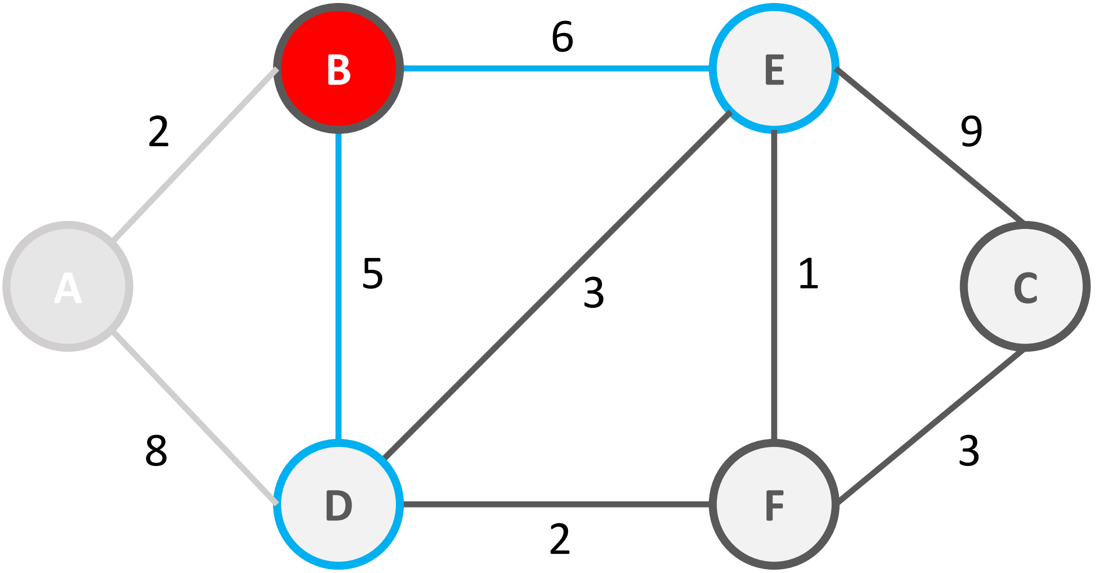
PQ = [B: 2, C: ∞, D: 8, E: ∞, F: ∞]

V = [A]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	8	A
E	∞	
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm

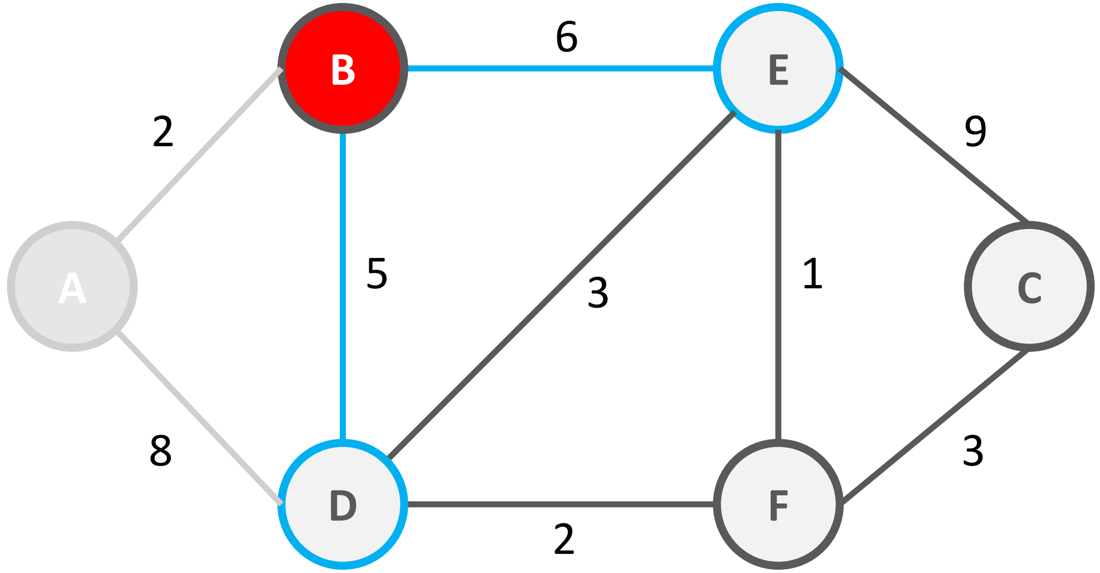


PQ = [B: 2, C: ∞, D: 8, E: ∞, F: ∞]
V = [A]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	8	A
E	∞	
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm



PQ = [B: 2, C: ∞, D: 8, E: ∞, F: ∞]

V = [A]

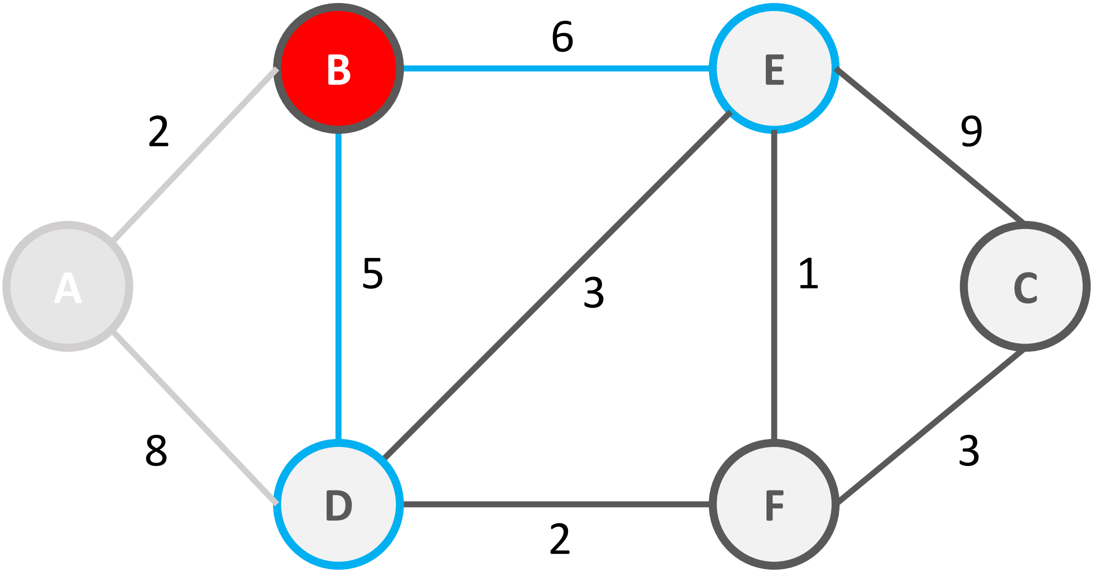
Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	8	A
E	∞	
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

```
if D[B] + e(B, D) < D[D]:
        D[D] = D[B] + e(B, D)
        D.predecessor = B
```

```
if 2 + 5 < 8 :
        D[B] = 2 + 5
        D.predecessor = B
```

Dijkstra's Algorithm



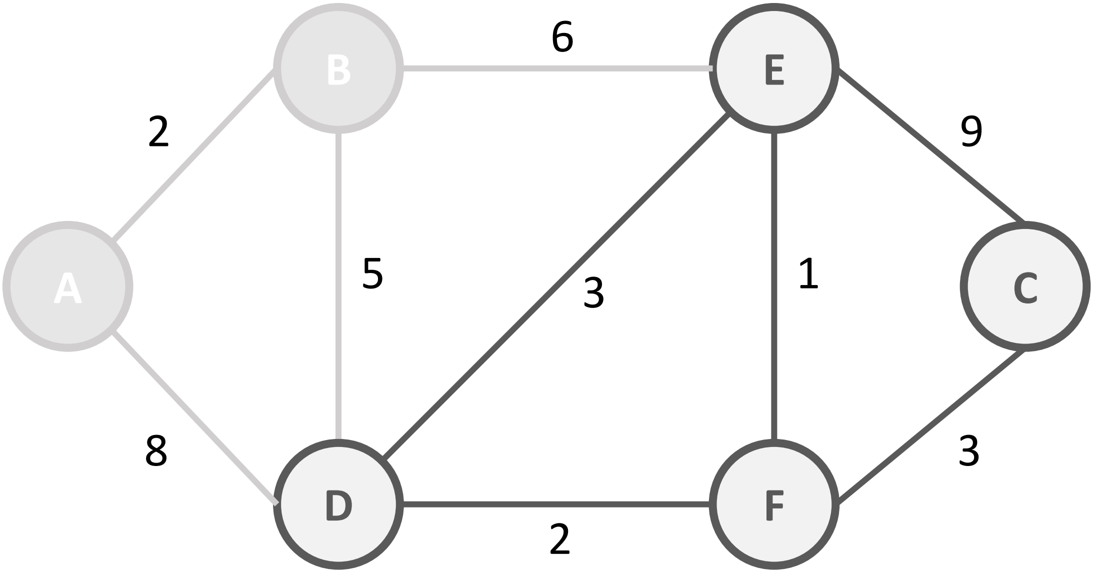
PQ = [B: 2, C: ∞, D: 7, E: 8, F: ∞]

V = [A]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm

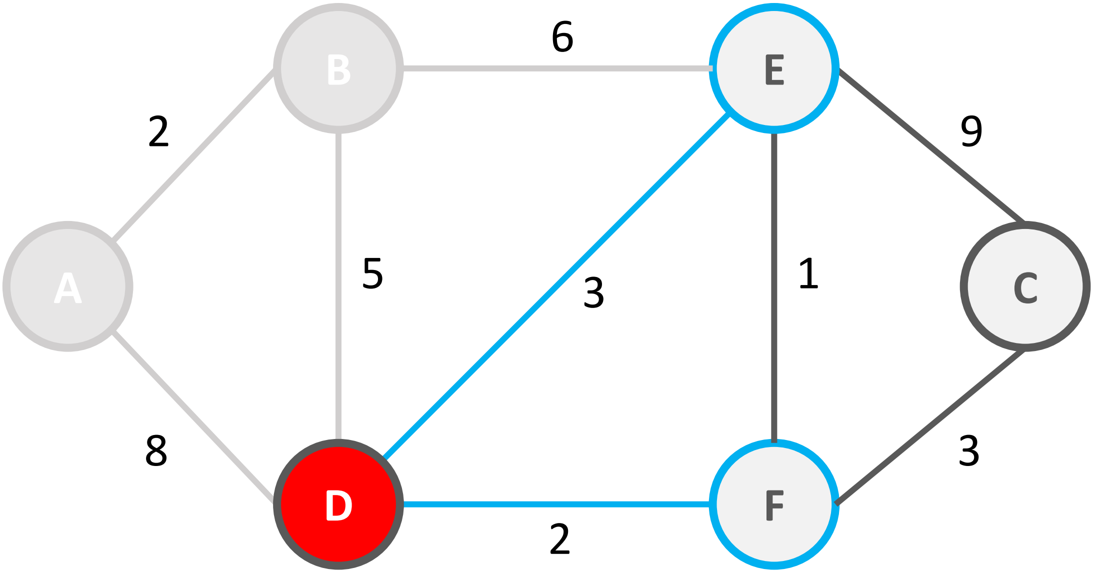


PQ = [C: ∞, D: 7, E: 8, F: ∞]
V = [A, B]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm



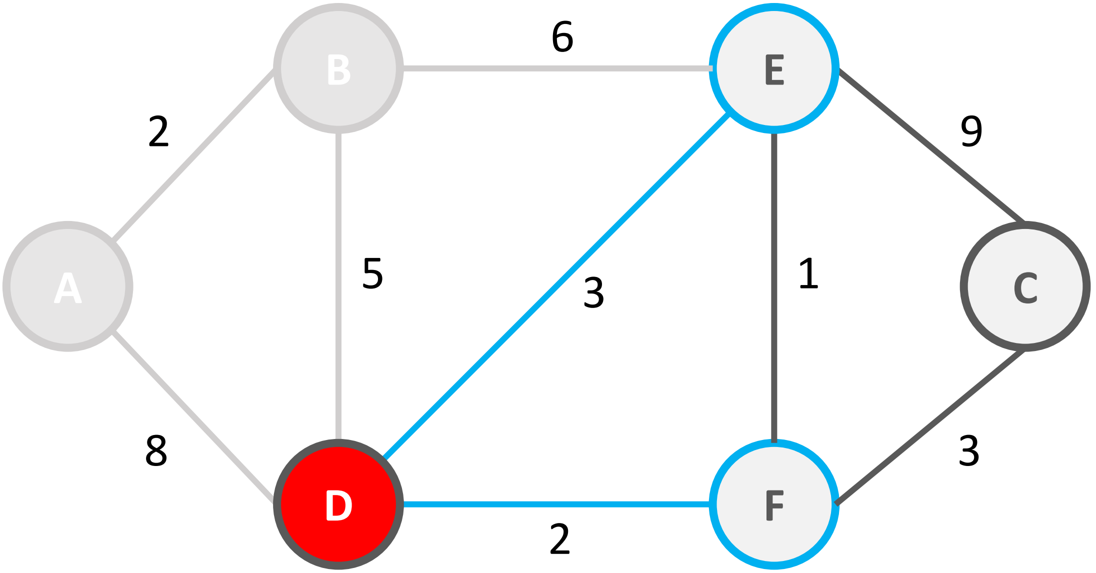
PQ = [C: ∞, D: 7, E: 8, F: ∞]

V = [A, B]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm



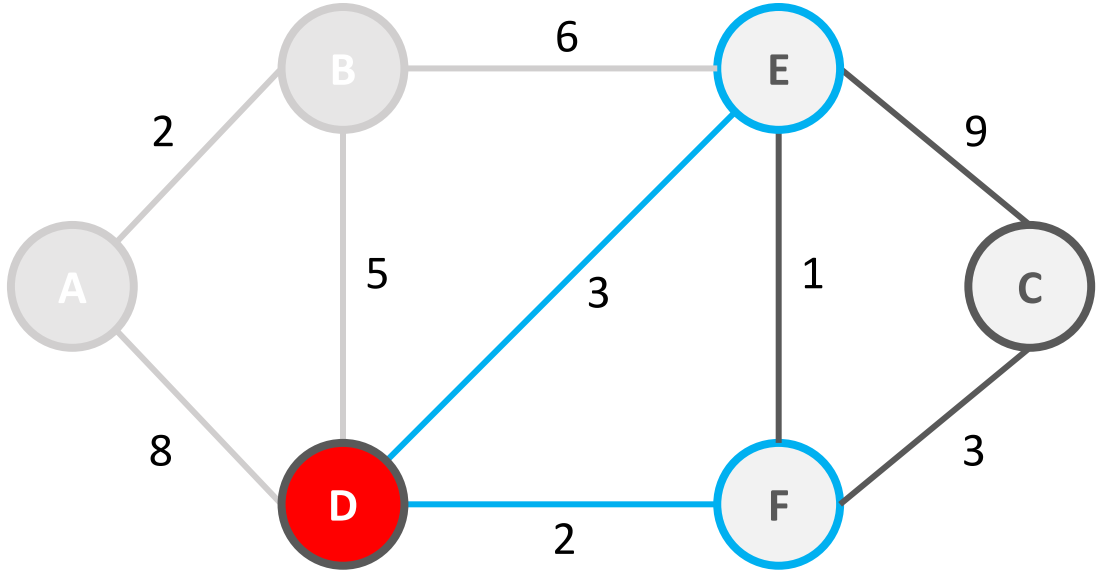
PQ = [C: ∞, D: 7, E: 8, F: ∞]
V = [A, B]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	∞	

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

```
if D[u] + e(u, v) < D[v]:
        D[v] = D[u] + e(u, v)
        v.predecessor = u
```

Dijkstra's Algorithm



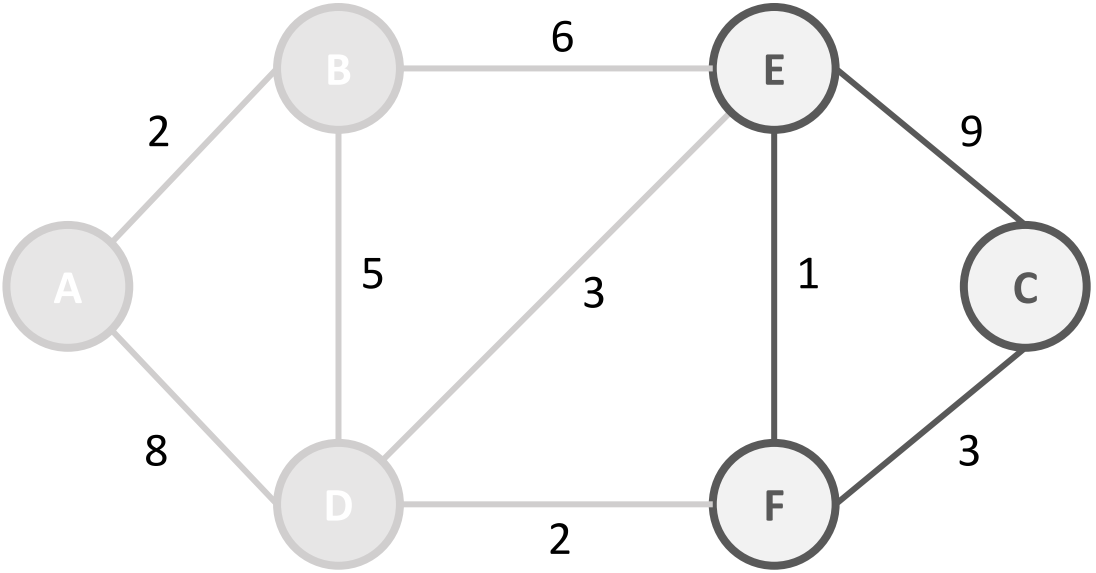
PQ = [C: ∞, D: 7, E: 8, F: 9]

V = [A, B]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm

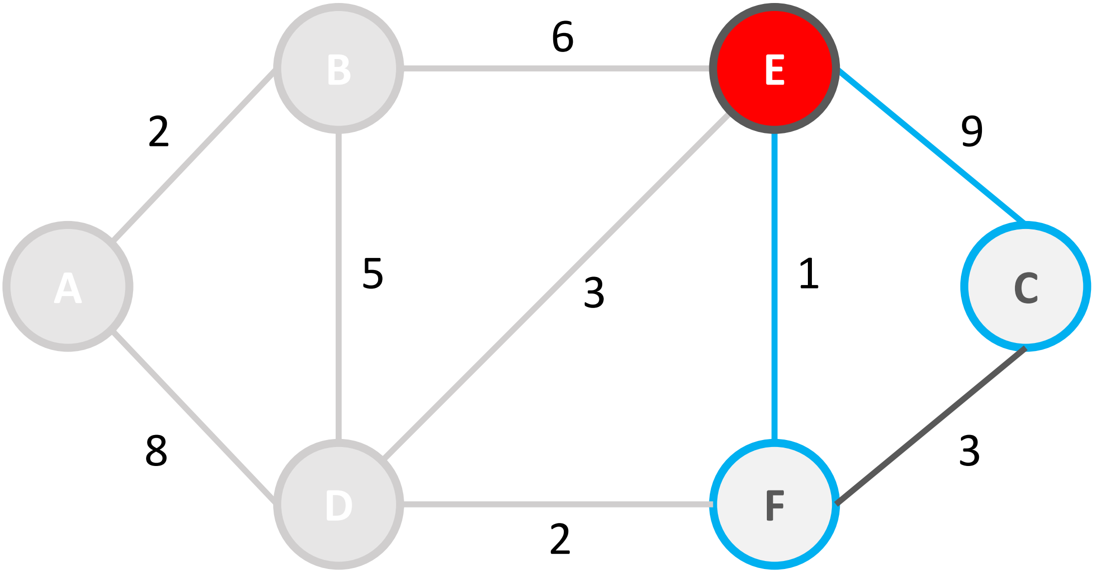


PQ = [C: ∞, E: 8, F: 9]
V = [A, B, D]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm

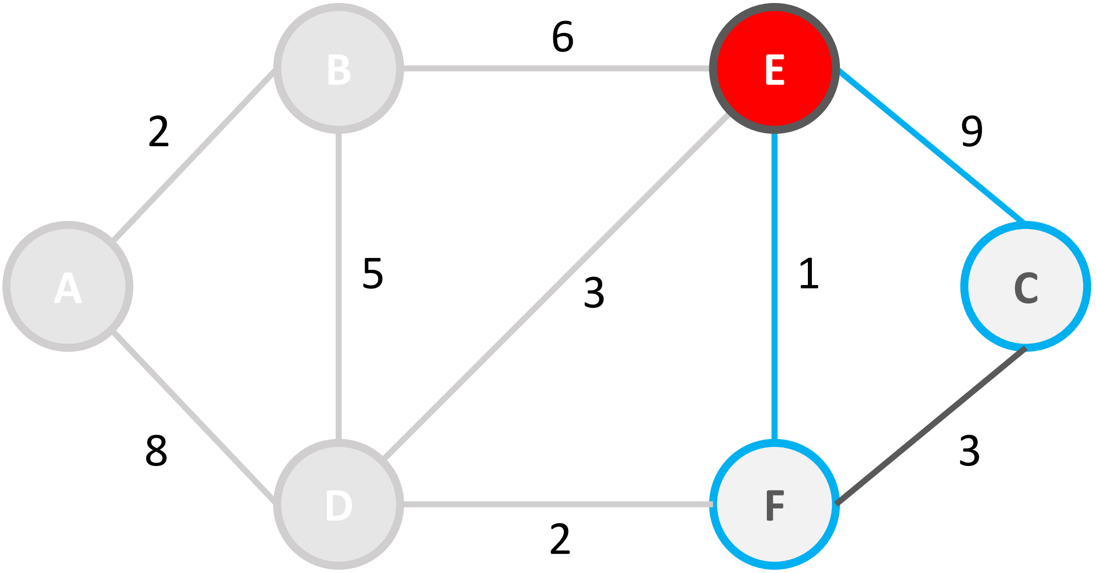


PQ = [C: ∞, E: 8, F: 9]
V = [A, B, D]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm



PQ = [C: ∞, E: 8, F: 9]

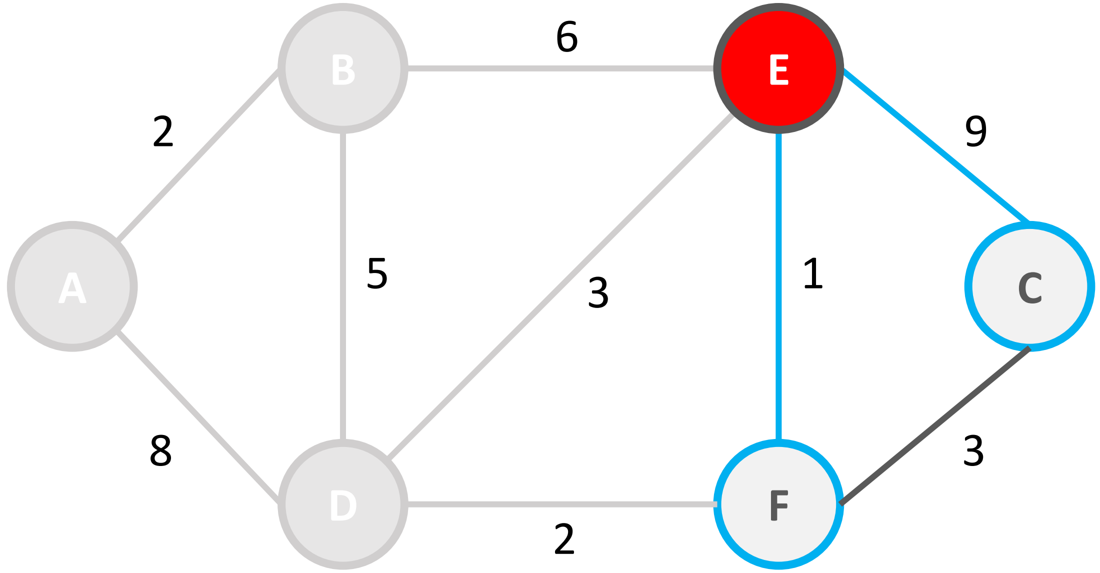
V = [A, B, D]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

```
if D[u] + e(u, v) < D[v]:
        D[v] = D[u] + e(u, v)
        v.predecessor = u
```

Dijkstra's Algorithm

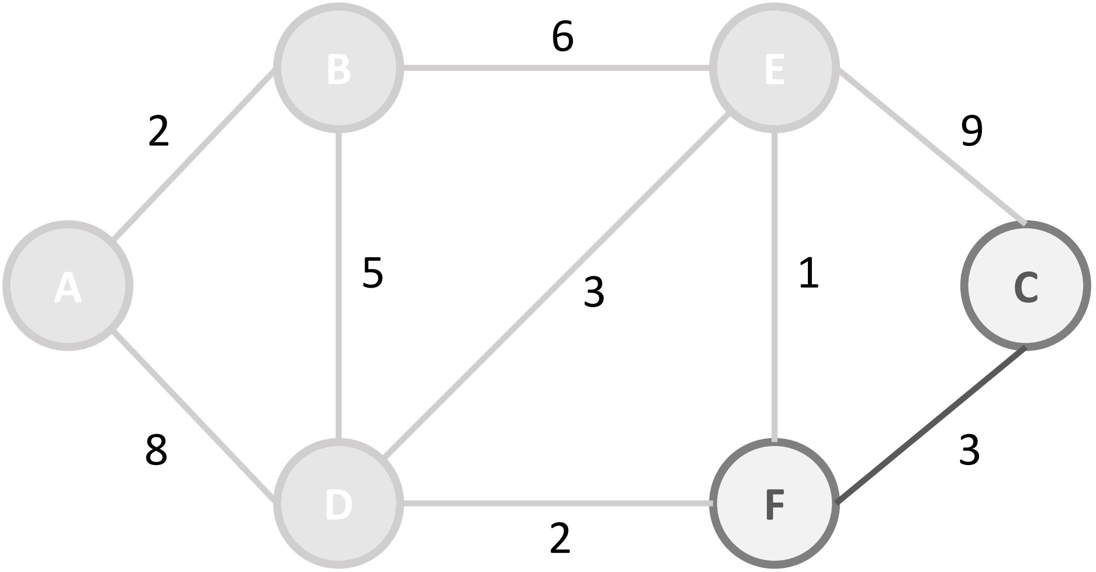


PQ = [C: 17, E: 8, F: 9]
V = [A, B, D]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	17	E
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm

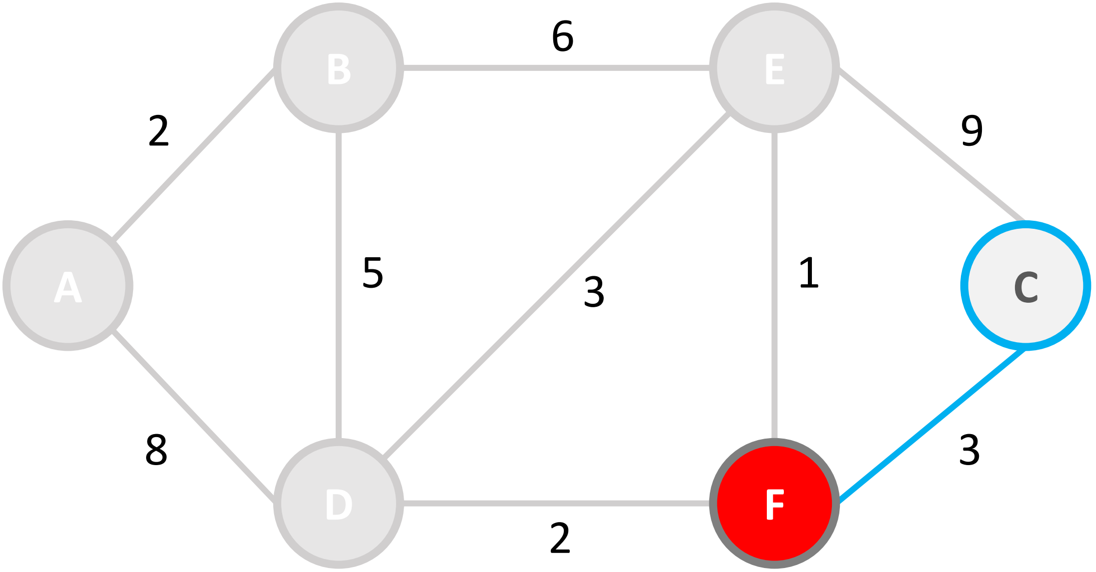


PQ = [C: 17, F: 9]
V = [A, B, D, E]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	17	E
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm

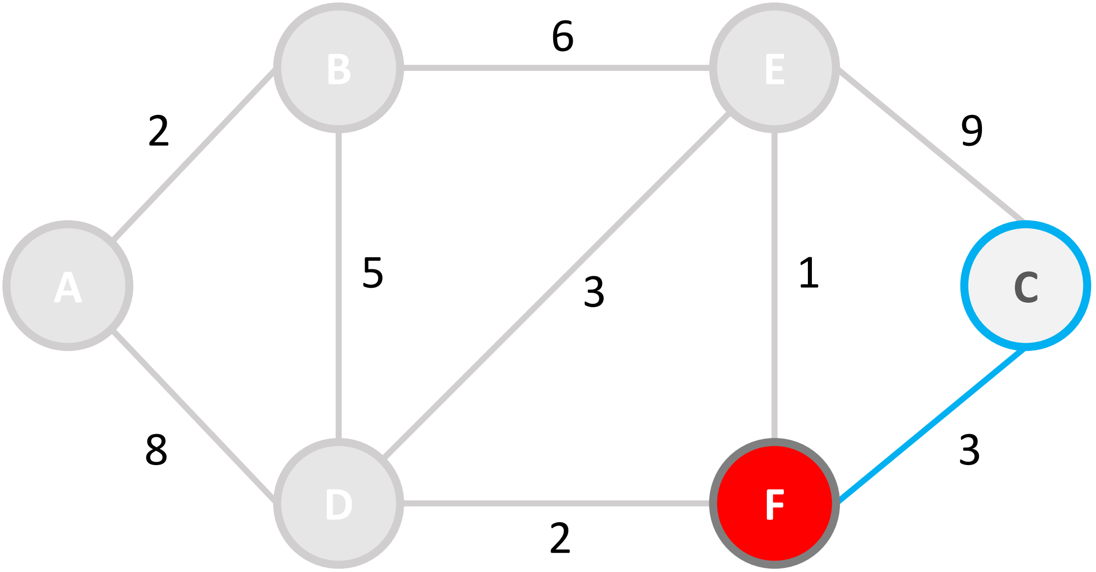


PQ = [C: 17, F: 9]
V = [A, B, D, E]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	17	E
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm



PQ = [C: 17, F: 9]

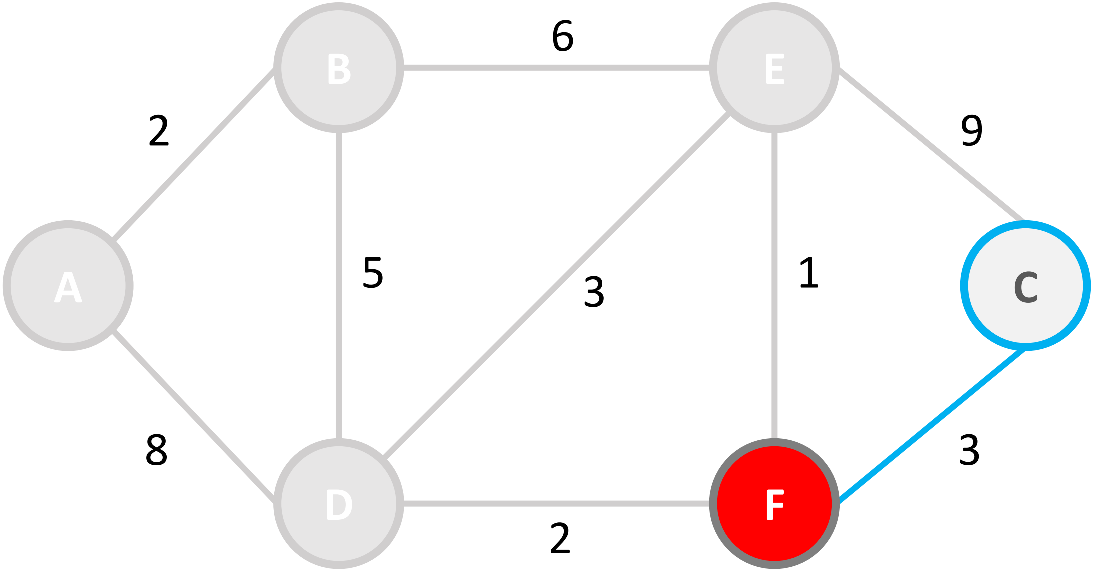
V = [A, B, D, E]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	17	E
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

```
if D[u] + e(u, v) < D[v]:
        D[v] = D[u] + e(u, v)
        v.predecessor = u
```

Dijkstra's Algorithm

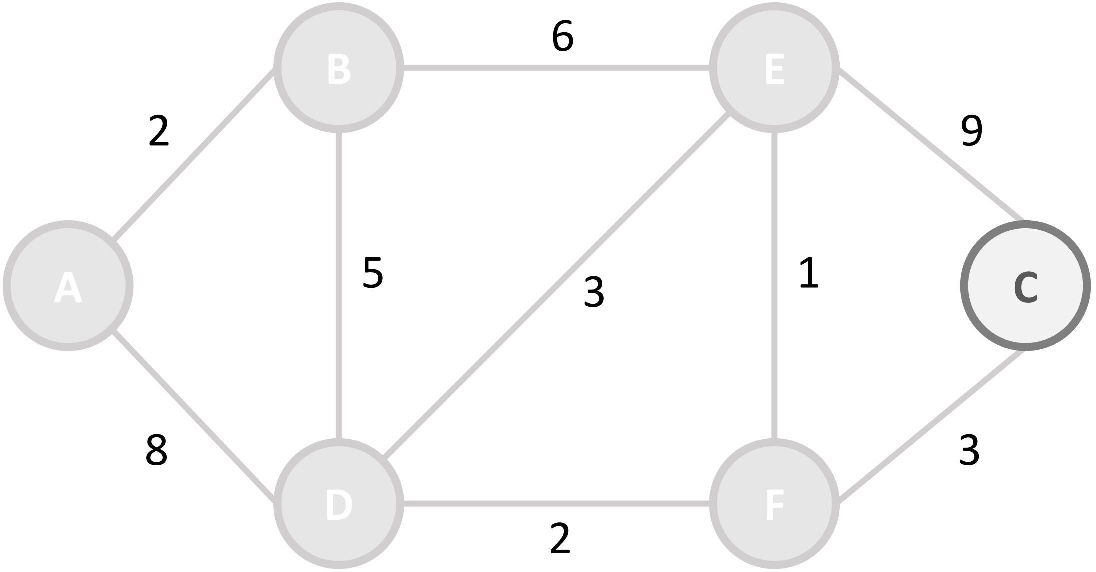


PQ = [C: 12, F: 9]
V = [A, B, D, E]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm



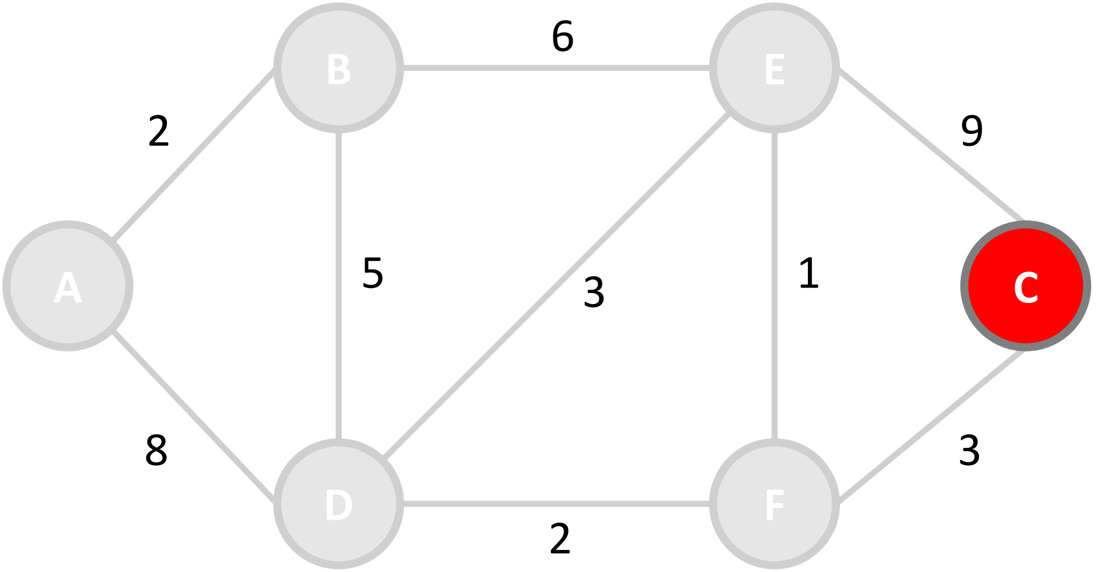
PQ = [C: 12]

V = [A, B, D, E, F]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm



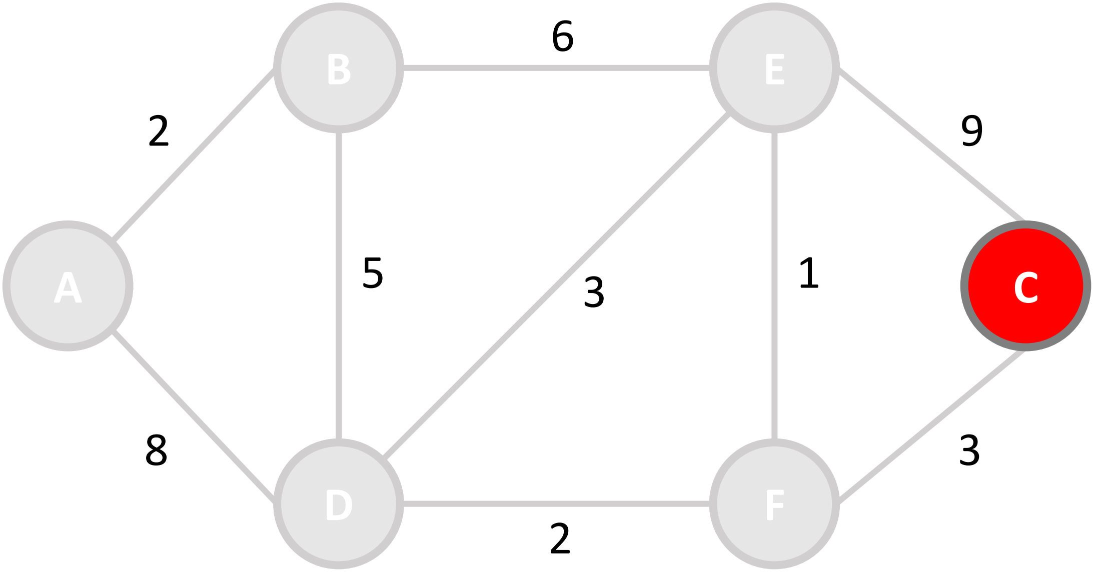
PQ = [C: 12]

V = [A, B, D, E, F]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm

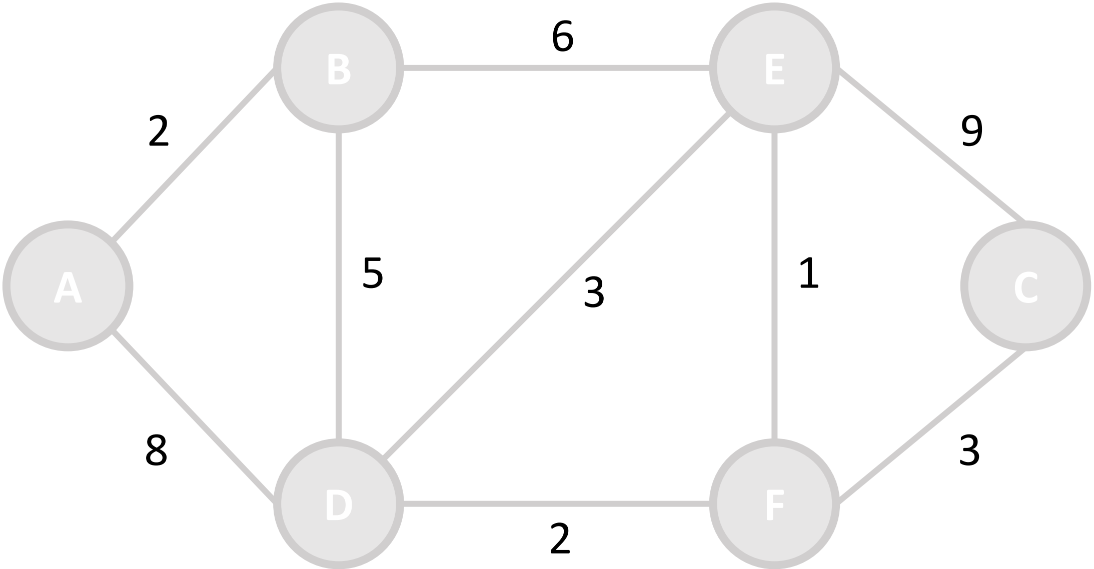


PQ = [C: 12]
V = [A, B, D, E, F]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm



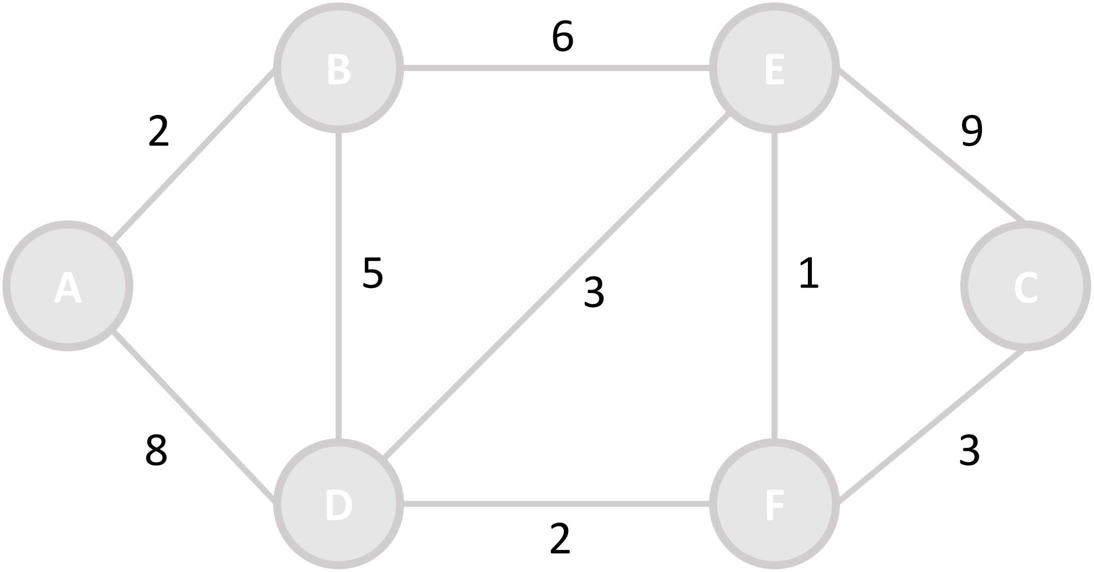
PQ = []

V = [A, B, D, E, F, C]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

1. Choose the highest priority element from PQ and obtain all unvisited neighbors
2. Edge Relaxation
3. Dequeue and put the vertex in the visited list

Dijkstra's Algorithm



PQ = []

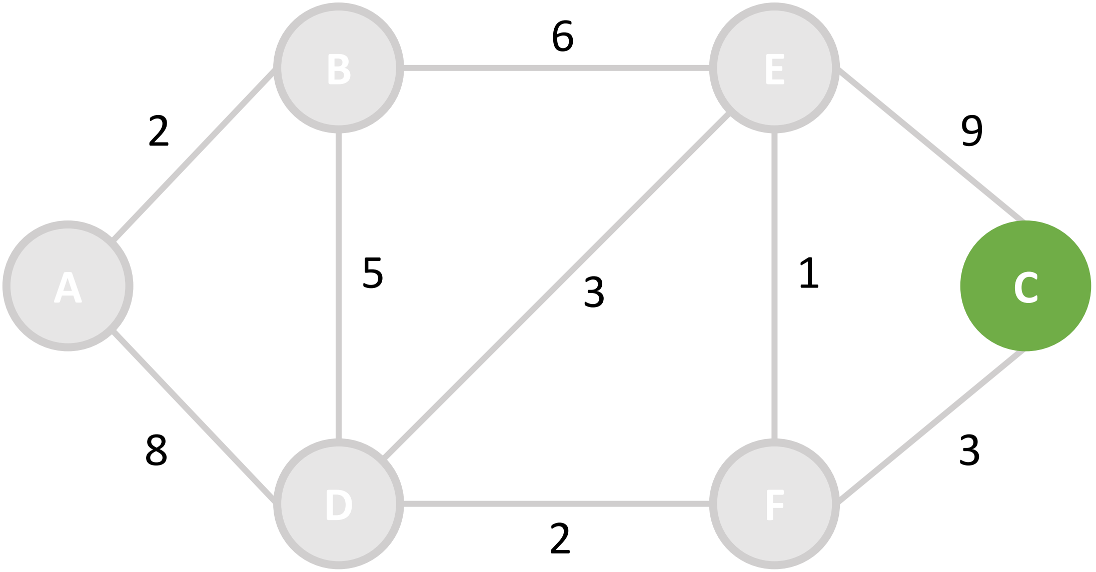
V = [A, B, D, E, F, C]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

To determine the shortest path from A to C

Starting from C, Check the predecessors periodically until the predecessor value becomes equal to the start vertex

Dijkstra's Algorithm

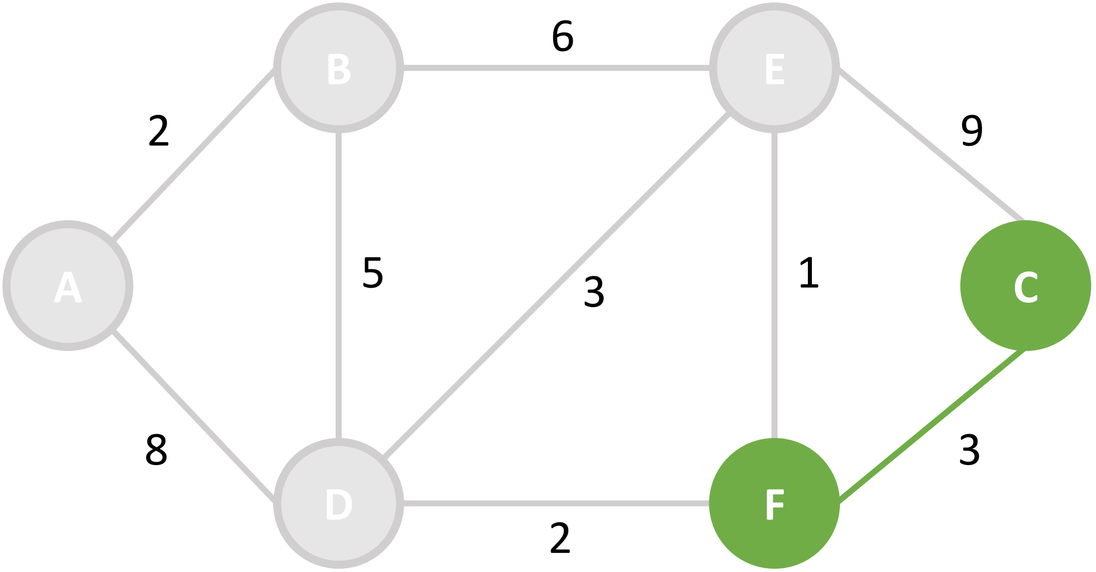


PQ = []

V = [A, B, D, E, F, C]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

Dijkstra's Algorithm

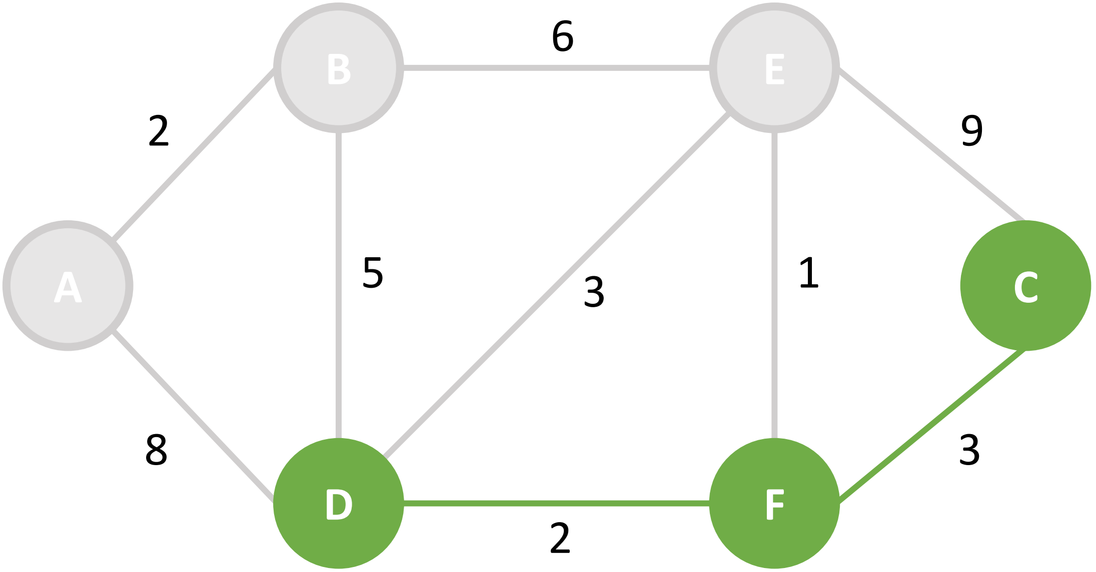


PQ = []

V = [A, B, D, E, F, C]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

Dijkstra's Algorithm

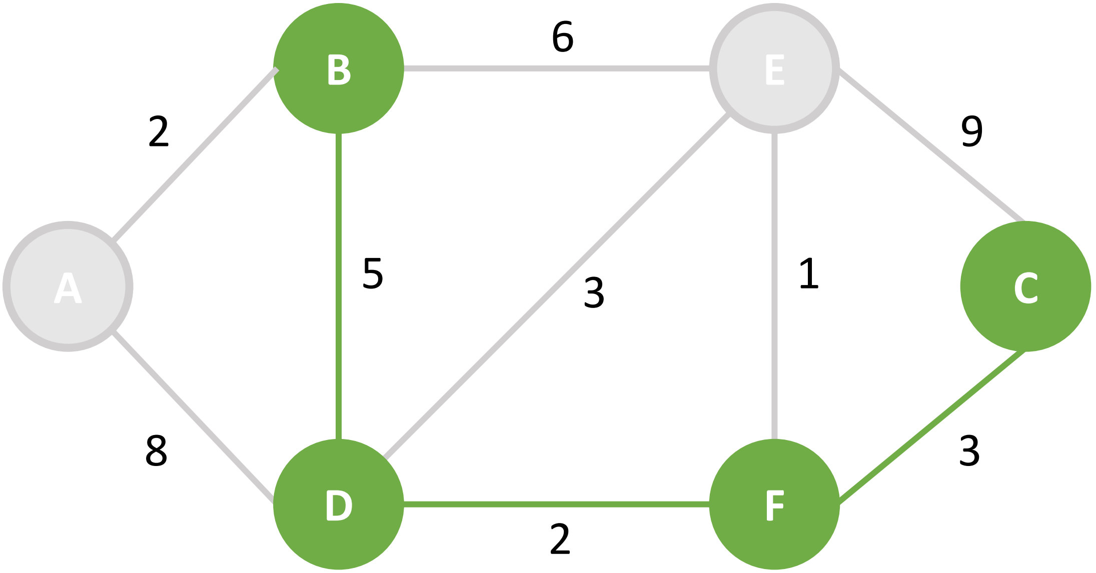


PQ = []

V = [A, B, D, E, F, C]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

Dijkstra's Algorithm

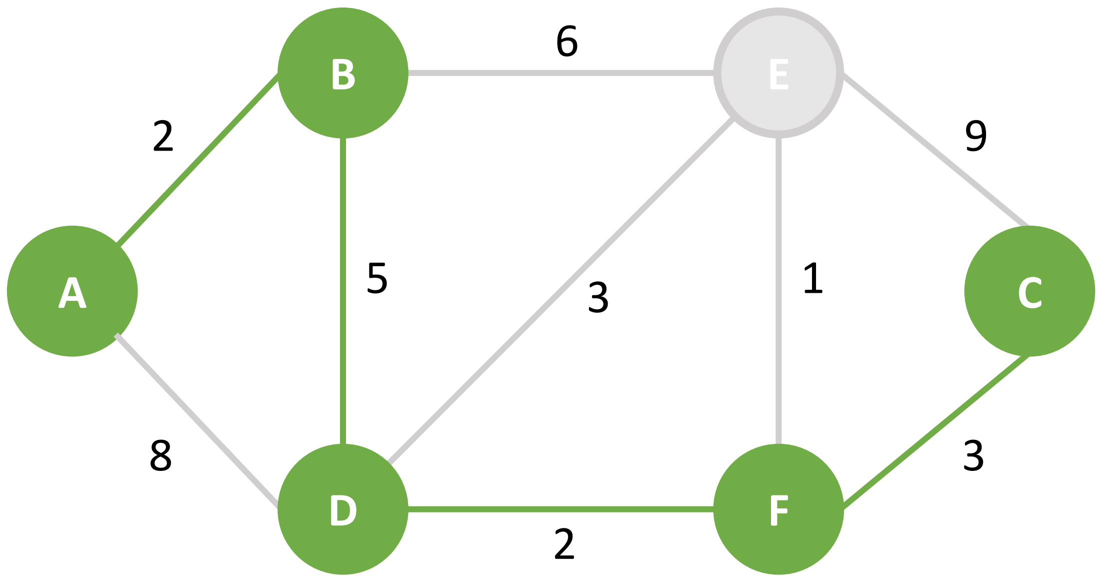


PQ = []

V = [A, B, D, E, F, C]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

Dijkstra's Algorithm

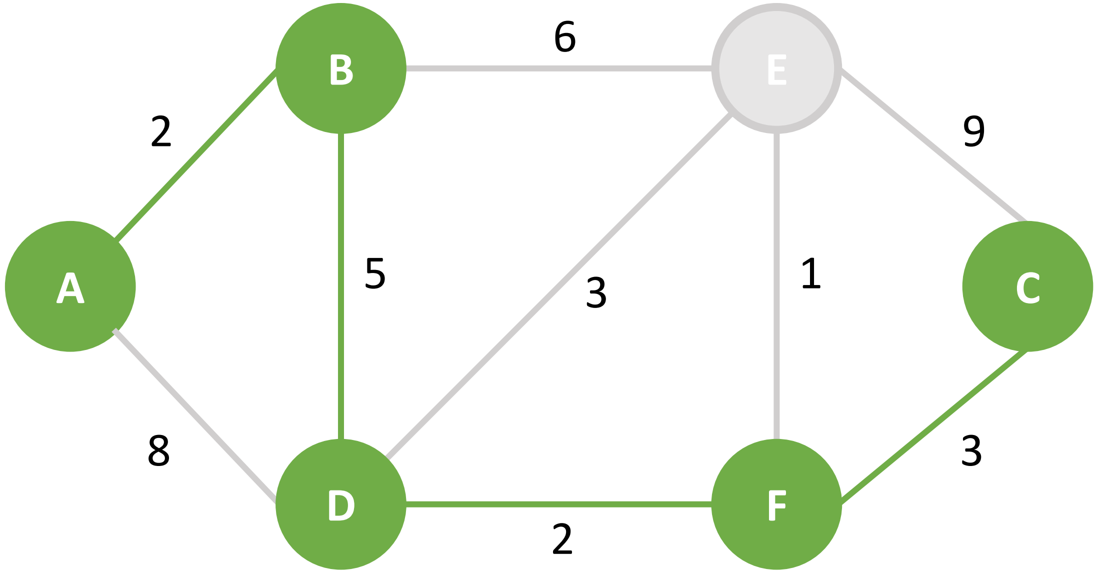


PQ = []

V = [A, B, D, E, F, C]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

Dijkstra's Algorithm



PQ = []

V = [A, B, D, E, F, C]

Vertex	Shortest Distance	Predecessor Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

Shortest path from A to C:
A-B-D-F-C = 12

Minimum Spanning Tree

- Given an undirected graph G with weighted edges, a minimum spanning tree (MST) is a subset of the edges in the graph which:
  - connects all vertices together
  - have no cycles
  - Include edges with minimum weight only

Minimum Spanning Tree | Prim's Algorithm

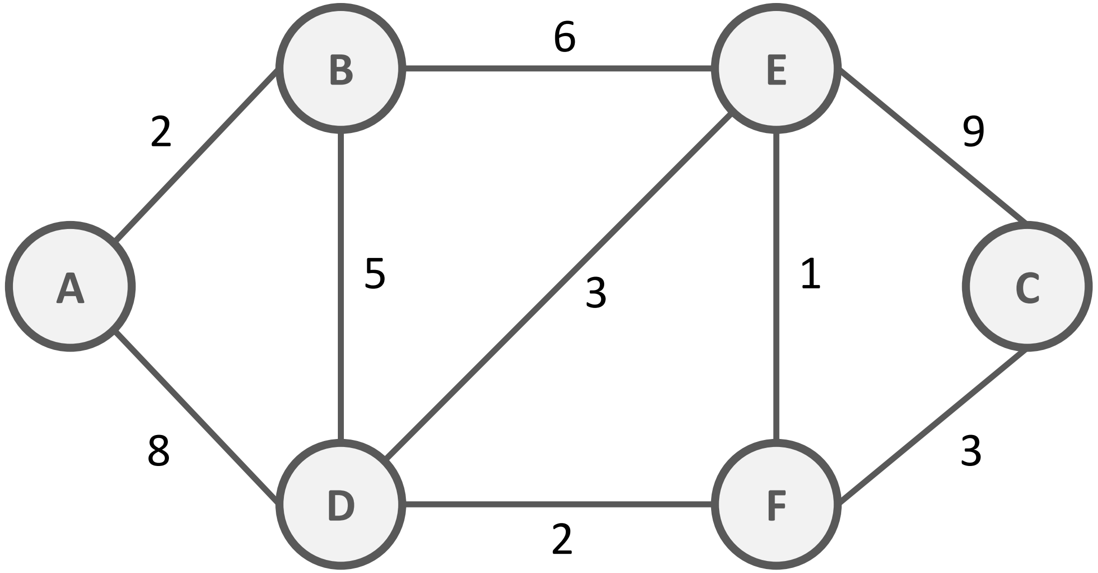
- Maintain a Priority Queue (PQ) of edges
- Start algorithm from any vertex V
  - Mark V as visited
  - Iterate over all edges of V and add them to PQ

While PQ is not empty and MST has not been formed (Total edges = V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited), skip and poll again (step 1)
3. Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



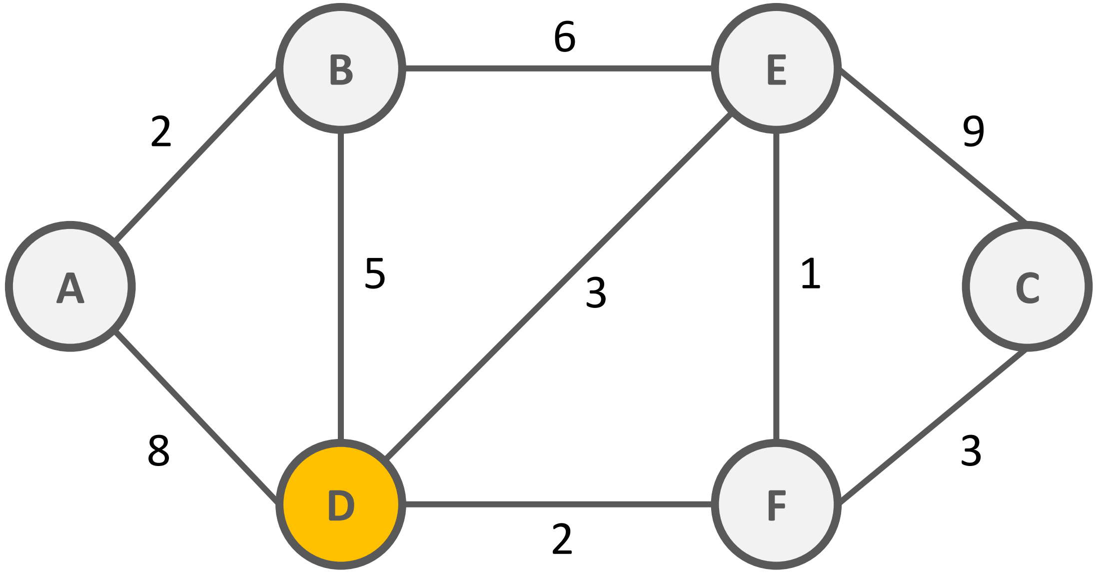
Visited

PQ

MST Edges = 0

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



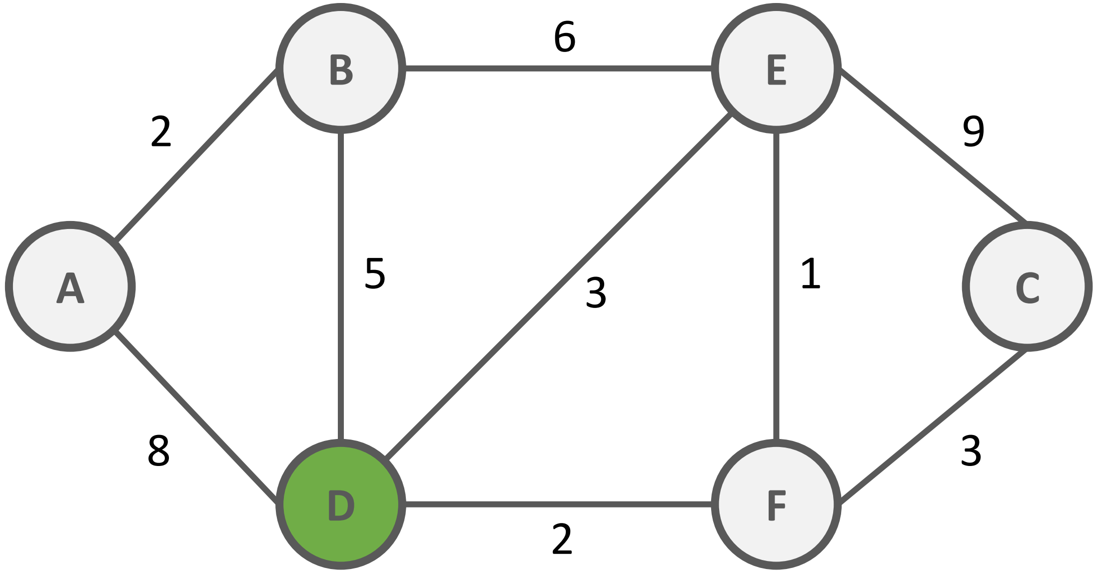
Visited

PQ

MST Edges = 0

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5

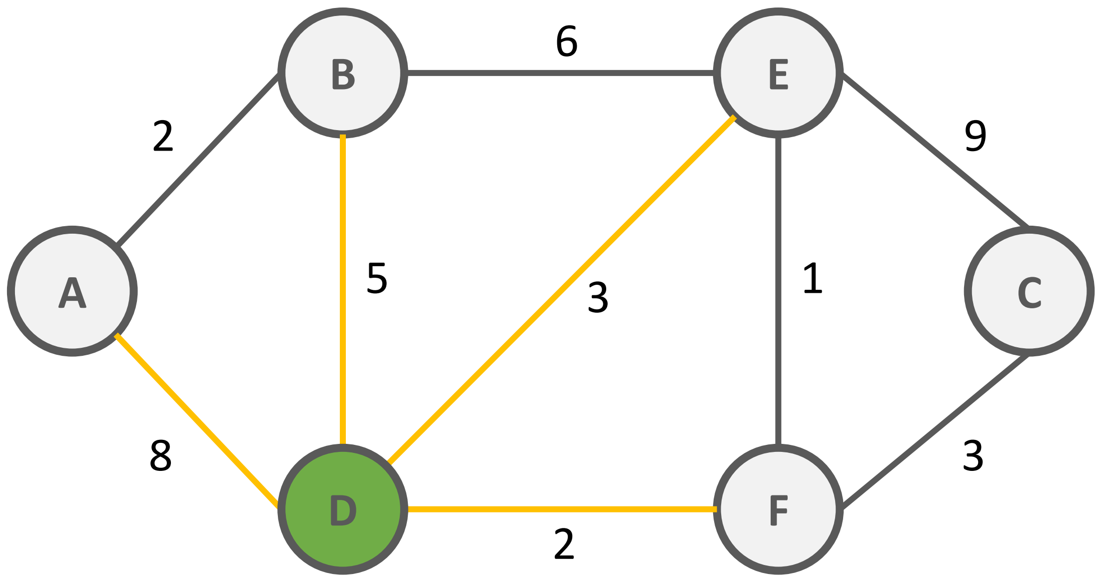


Visited	PQ
D	

MST Edges = 0

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



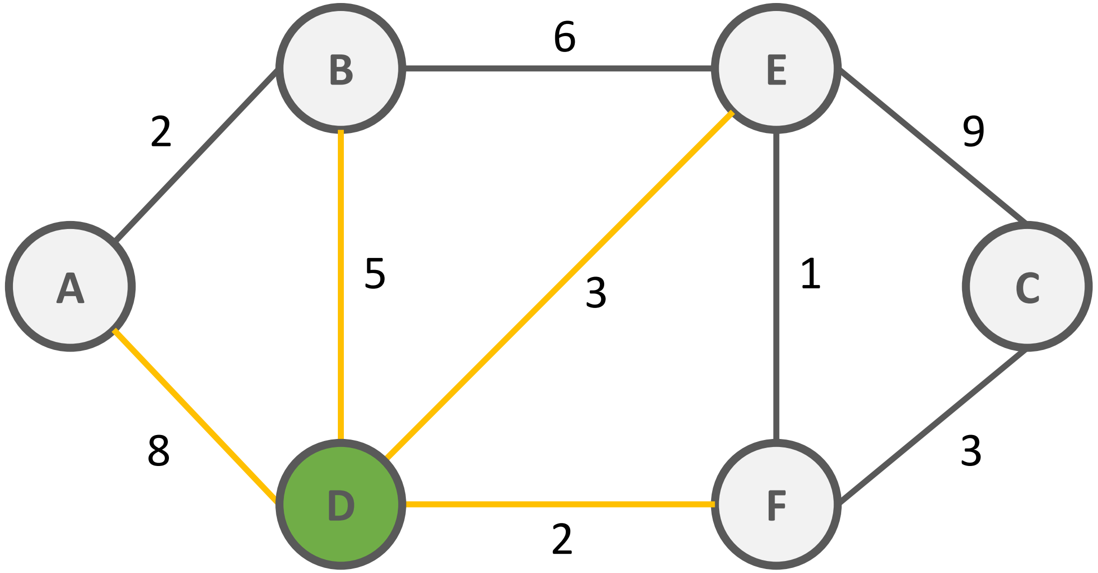
Visited
D

PQ
D-A, 8
D-B, 5
D-E, 3
D-F, 2

MST Edges = 0

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D

PQ
D-A, 8
D-B, 5
D-E, 3
D-F, 2

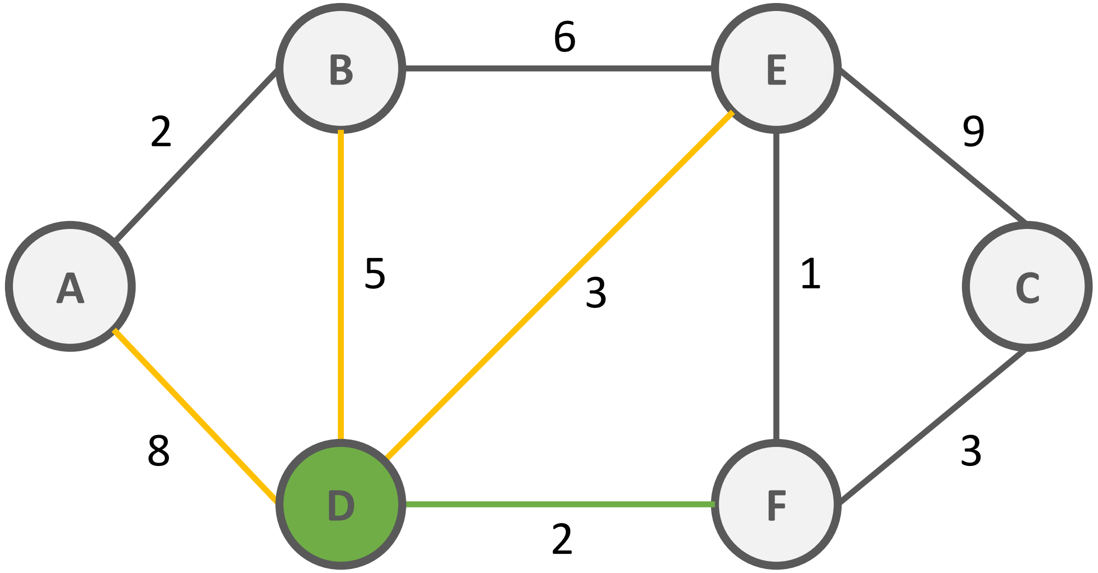
MST Edges = 0

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D

PQ
D-A, 8
D-B, 5
D-E, 3
D-F, 2

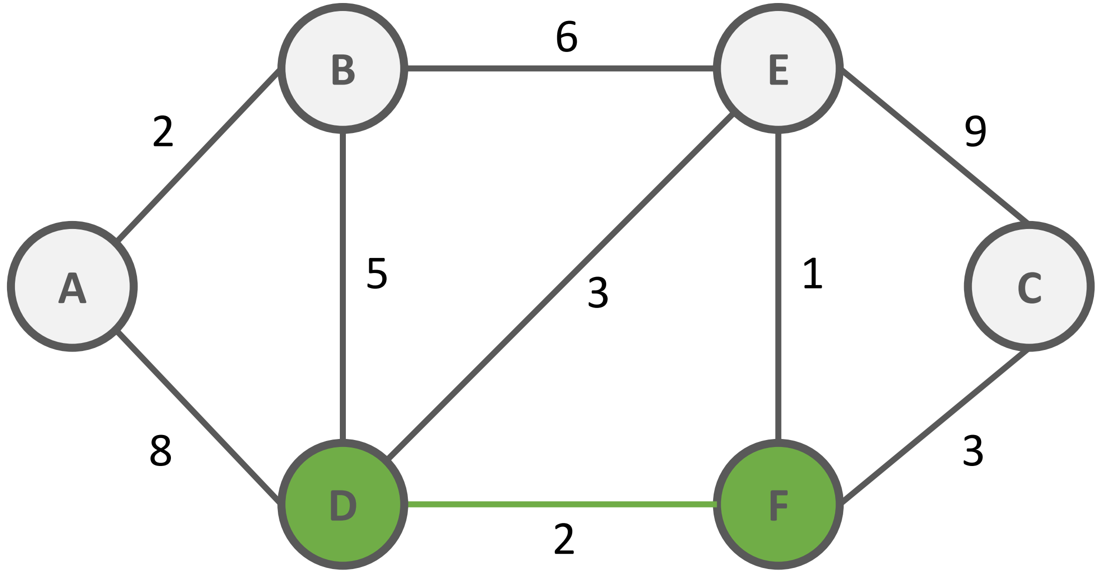
MST Edges = 0

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F

PQ
D-A, 8
D-B, 5
D-E, 3

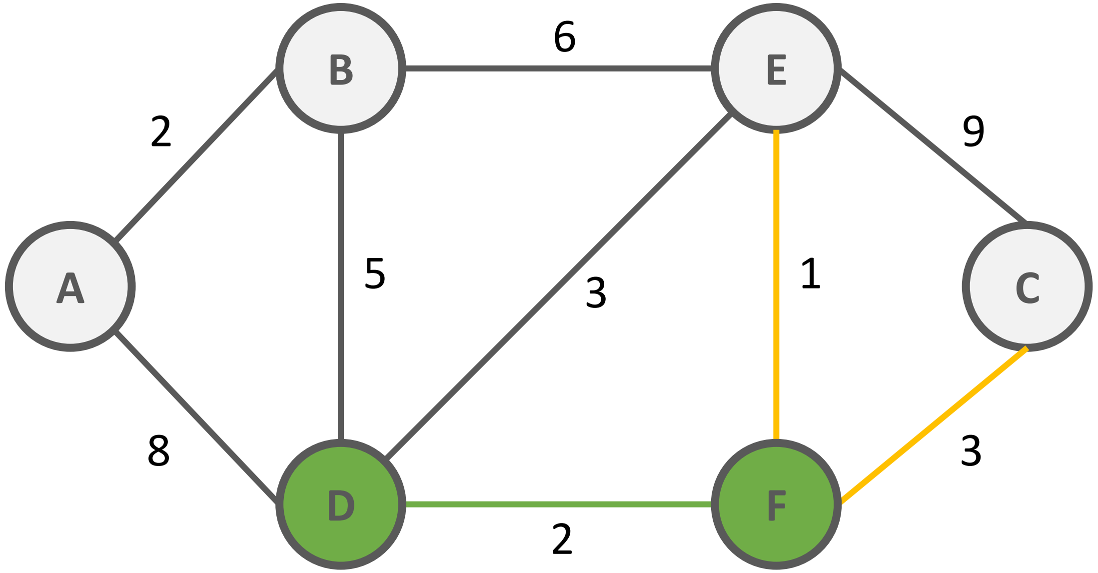
MST Edges = 1

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F

PQ
D-A, 8
D-B, 5
D-E, 3
F-E, 1
F-C, 3

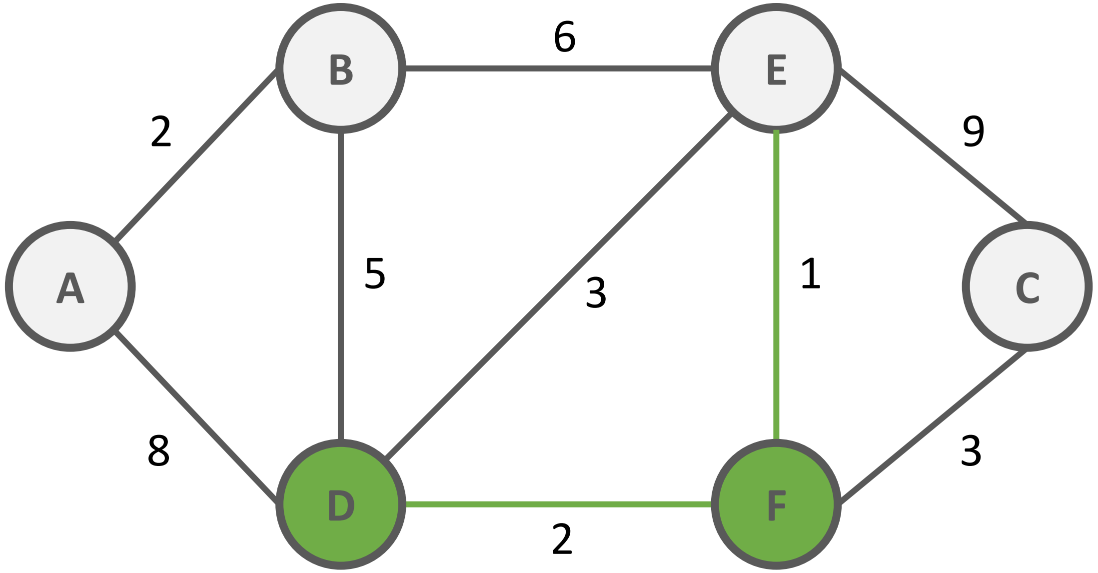
MST Edges = 1

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F

PQ
D-A, 8
D-B, 5
D-E, 3
F-E, 1
F-C, 3

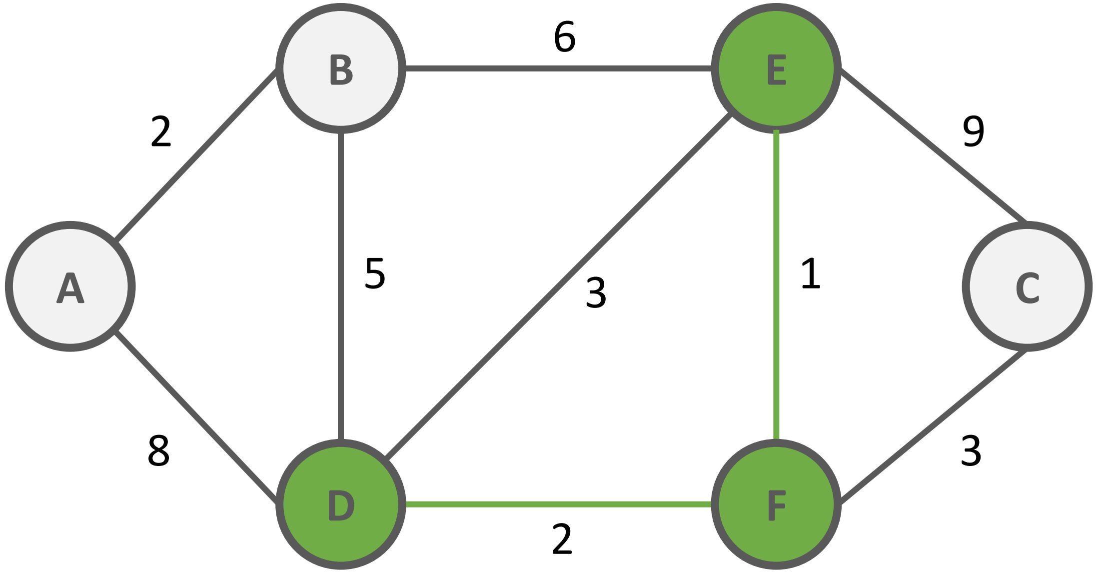
MST Edges = 1

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E

PQ
D-A, 8
D-B, 5
D-E, 3
F-C, 3

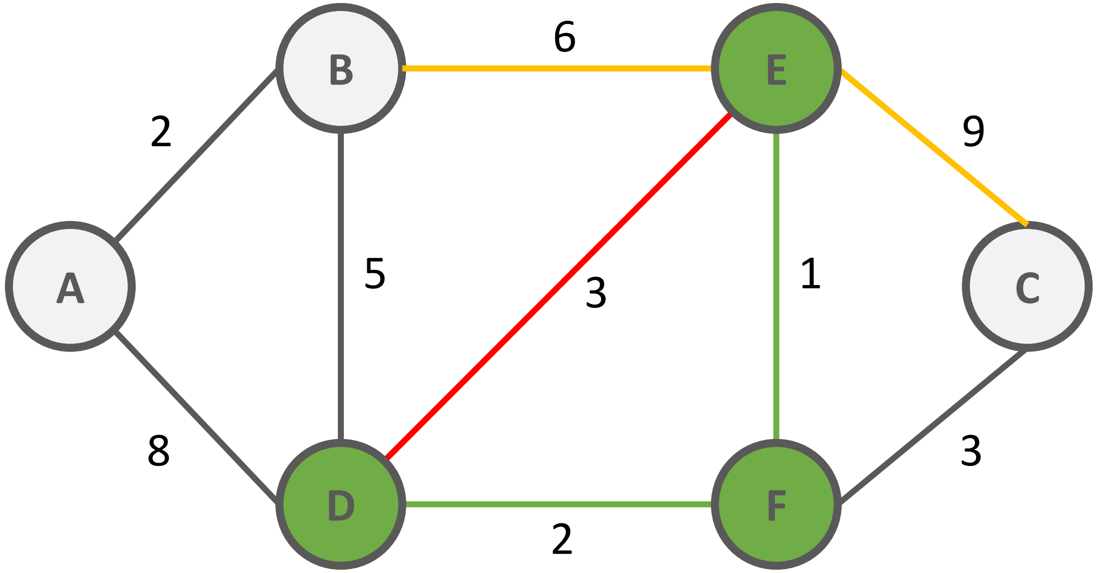
MST Edges = 2

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E

PQ
D-A, 8
D-B, 5
D-E, 3
F-C, 3
E-B, 6
E-C, 9

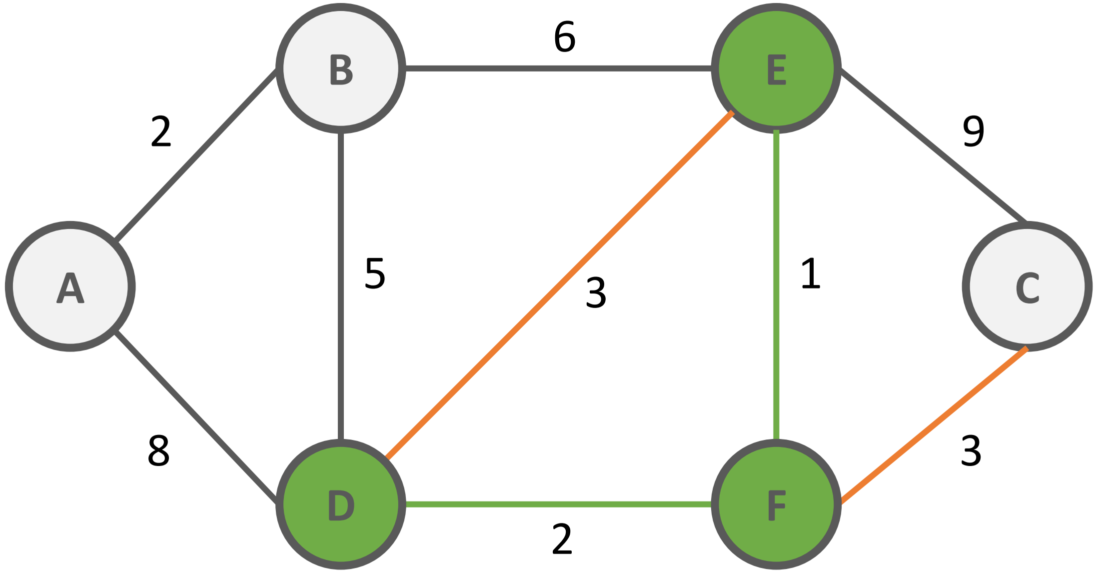
MST Edges = 2

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E

PQ
D-A, 8
D-B, 5
D-E, 3
F-C, 3
E-B, 6
E-C, 9

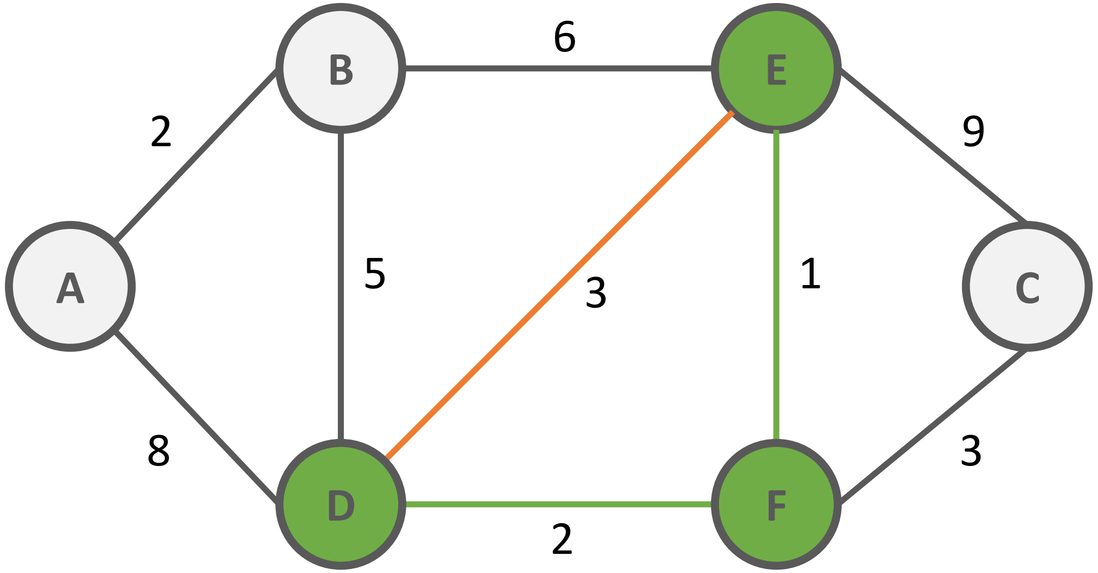
MST Edges = 2

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E

PQ
D-A, 8
D-B, 5
D-E, 3
F-C, 3
E-B, 6
E-C, 9

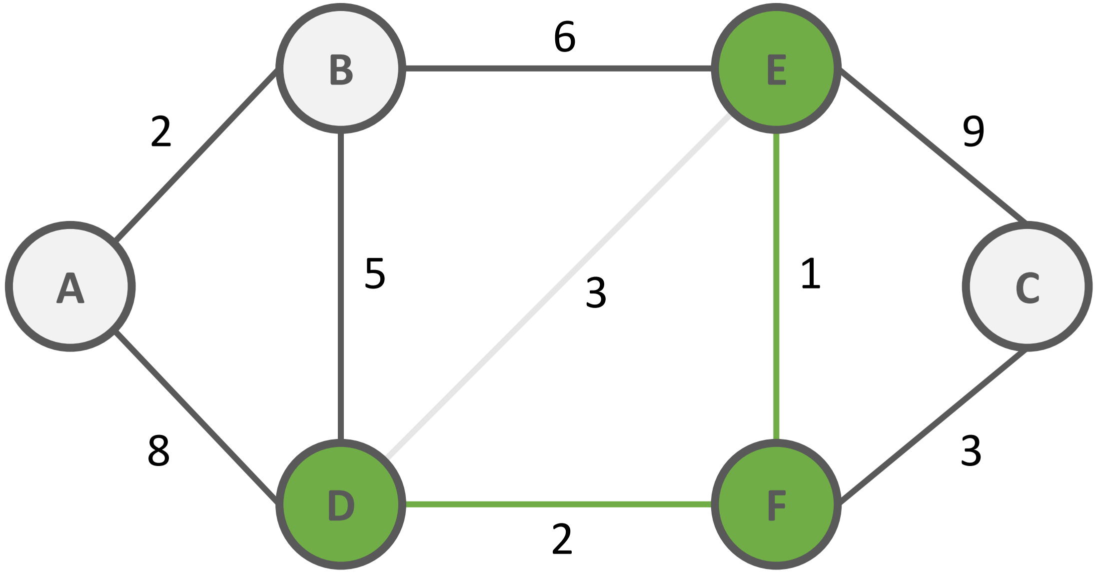
MST Edges = 2

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E

PQ
D-A, 8
D-B, 5
F-C, 3
E-B, 6
E-C, 9

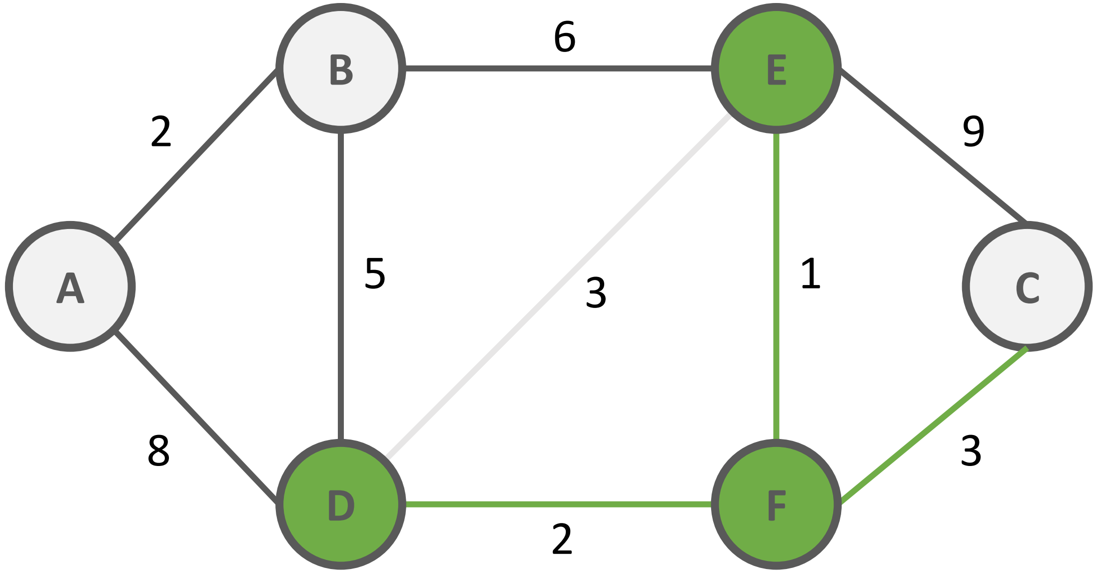
MST Edges = 2

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E

PQ
D-A, 8
D-B, 5
F-C, 3
E-B, 6
E-C, 9

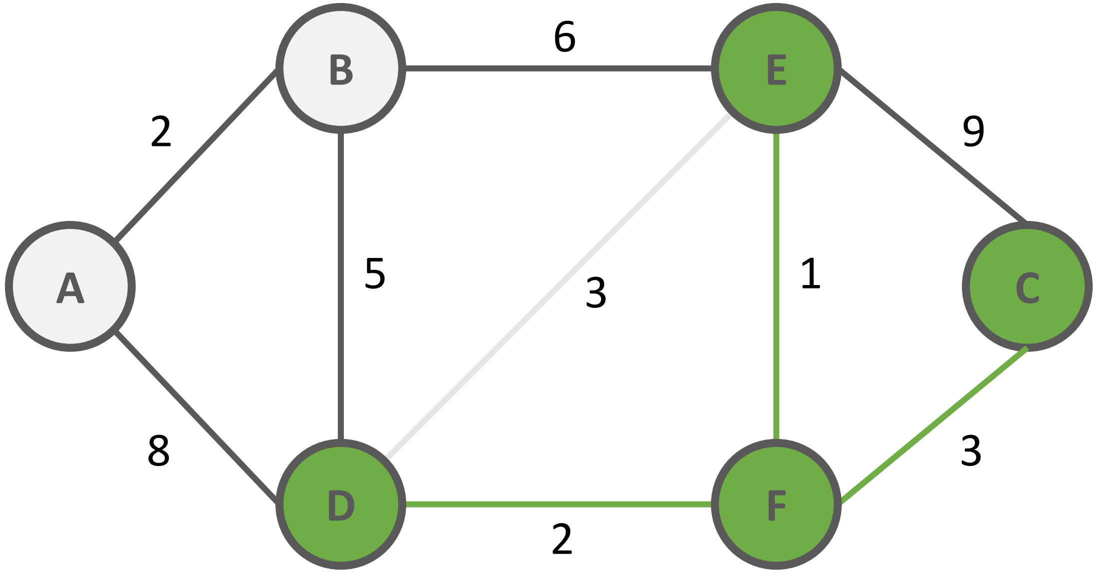
MST Edges = 2

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E
C

PQ
D-A, 8
D-B, 5
E-B, 6
E-C, 9

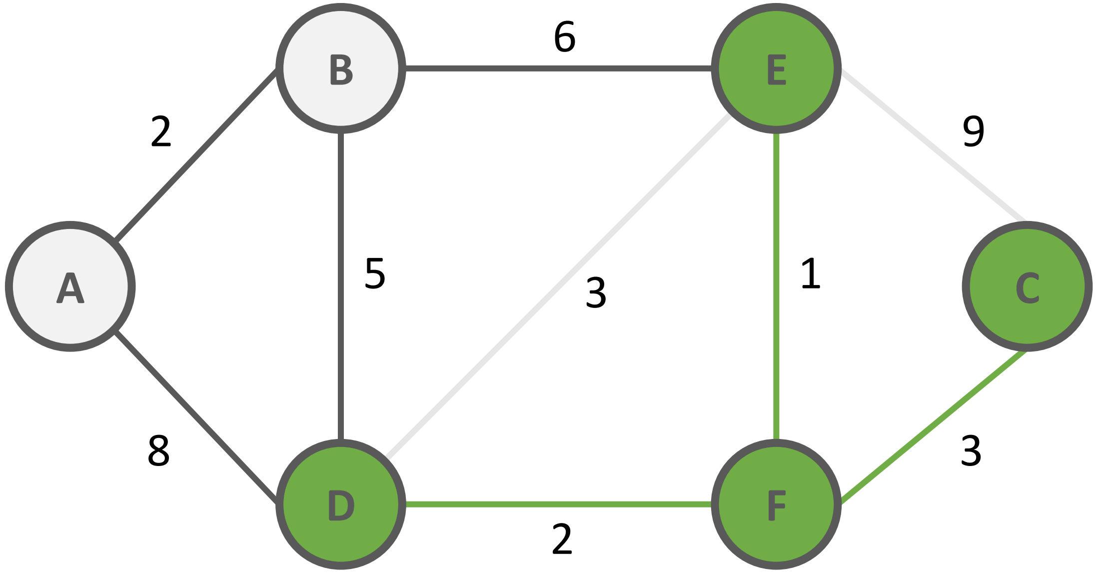
MST Edges = 3

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E
C

PQ
D-A, 8
D-B, 5
E-B, 6
E-C, 9

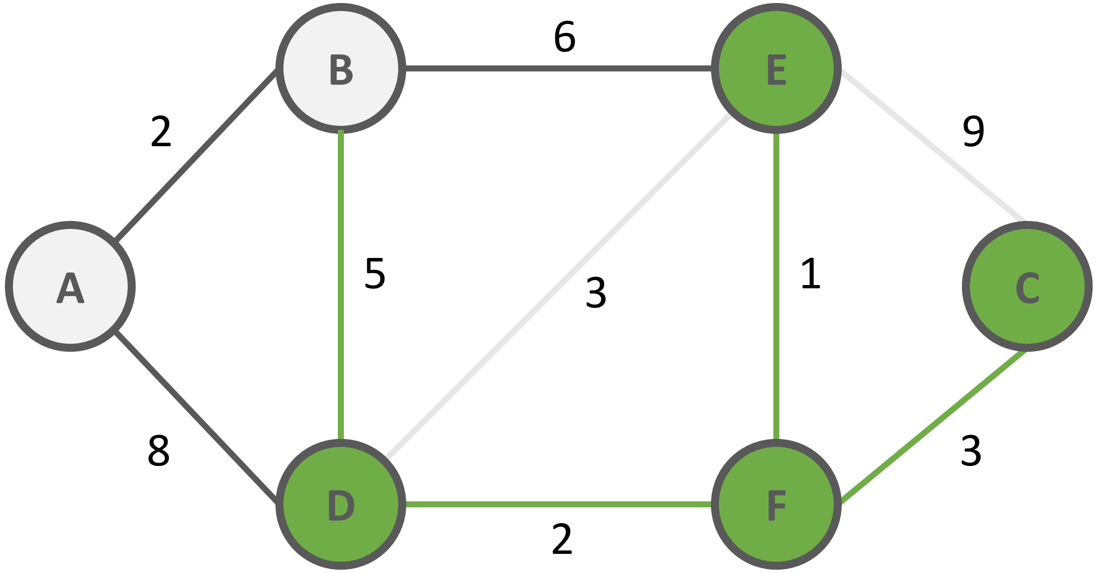
MST Edges = 3

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E
C

PQ
D-A, 8
D-B, 5
E-B, 6
E-C, 9

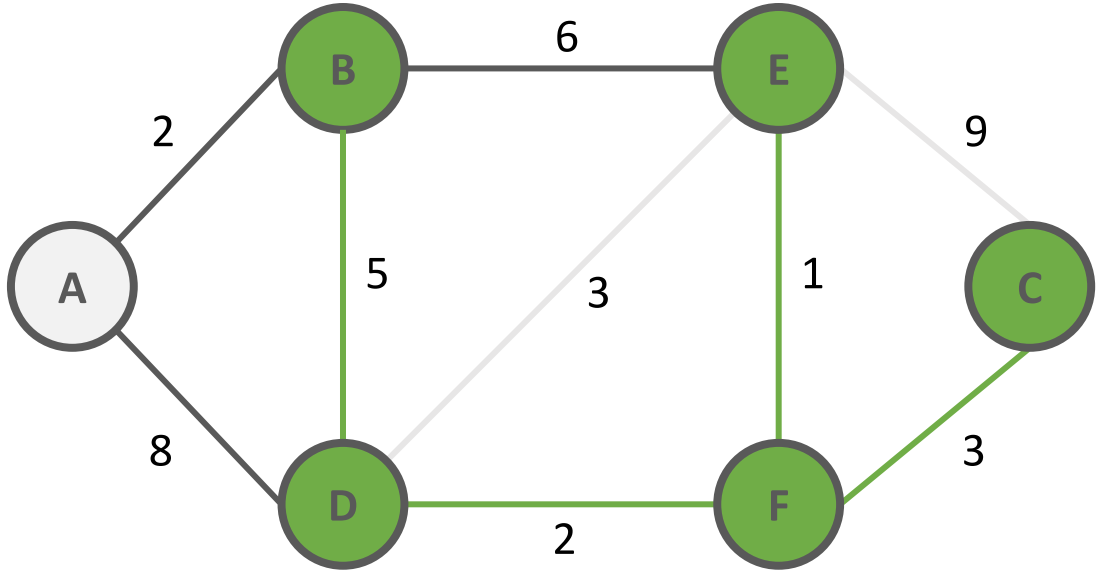
MST Edges = 3

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E
C
B

PQ
D-A, 8
E-B, 6
E-C, 9

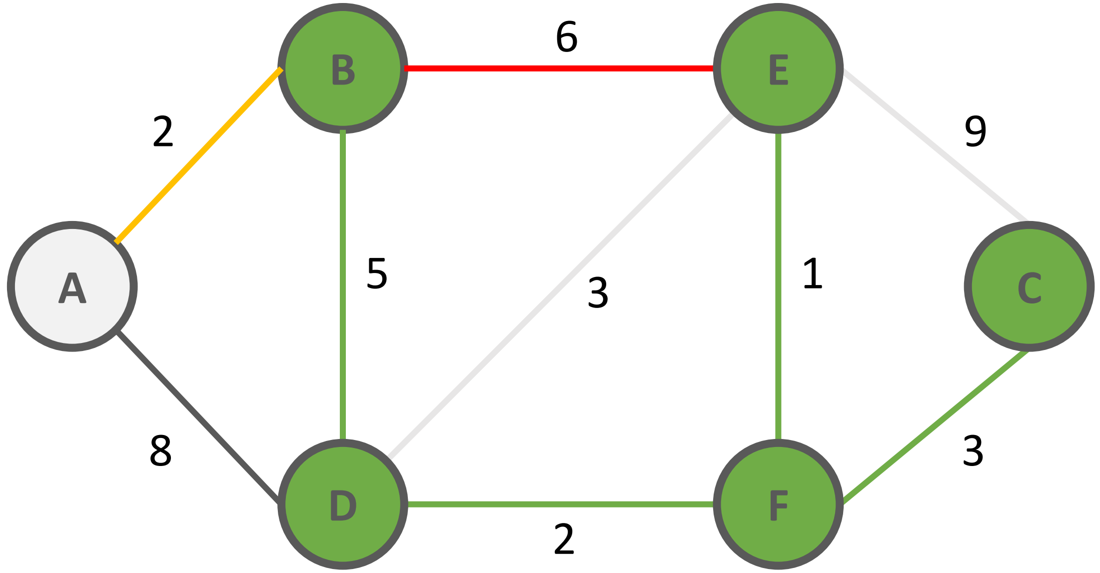
MST Edges = 4

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E
C
B

PQ
D-A, 8
E-B, 6
E-C, 9
B-A, 2

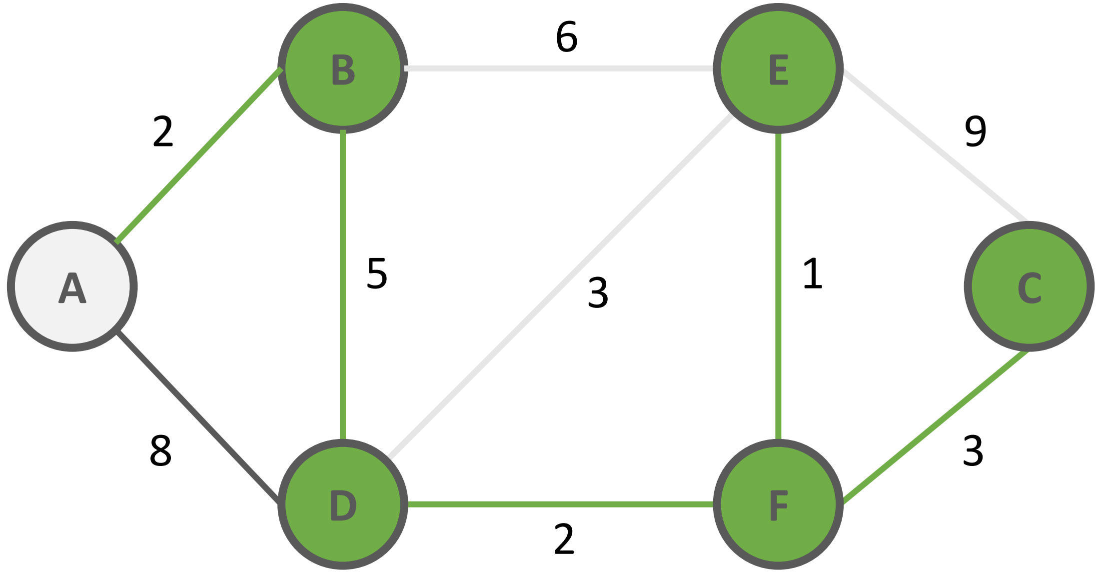
MST Edges = 4

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E
C
B

PQ
D-A, 8
E-B, 6
E-C, 9
B-A, 2

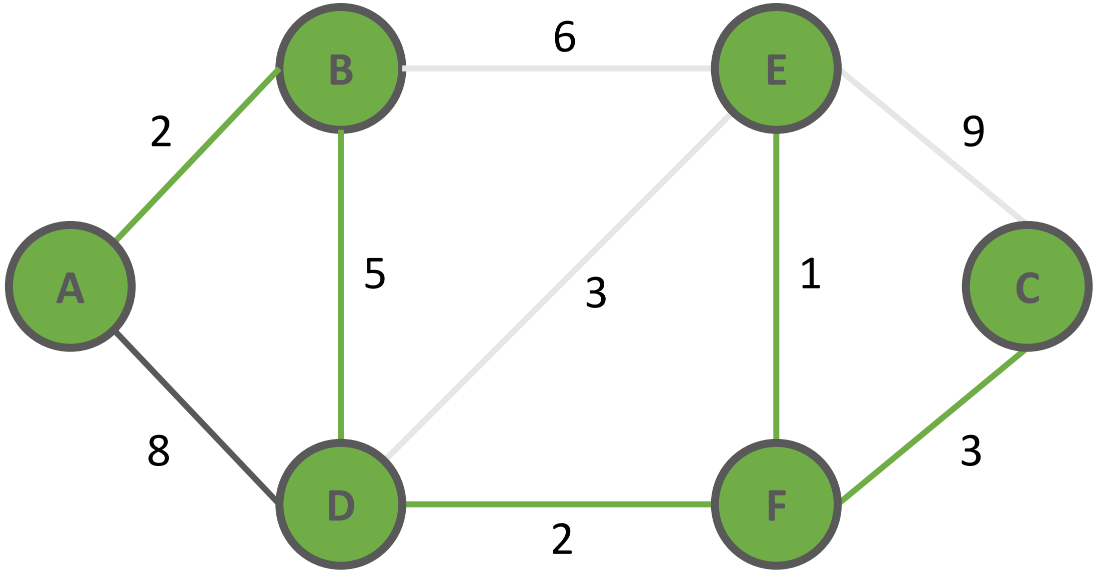
MST Edges = 4

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E
C
B
A

PQ
D-A, 8
E-B, 6
E-C, 9

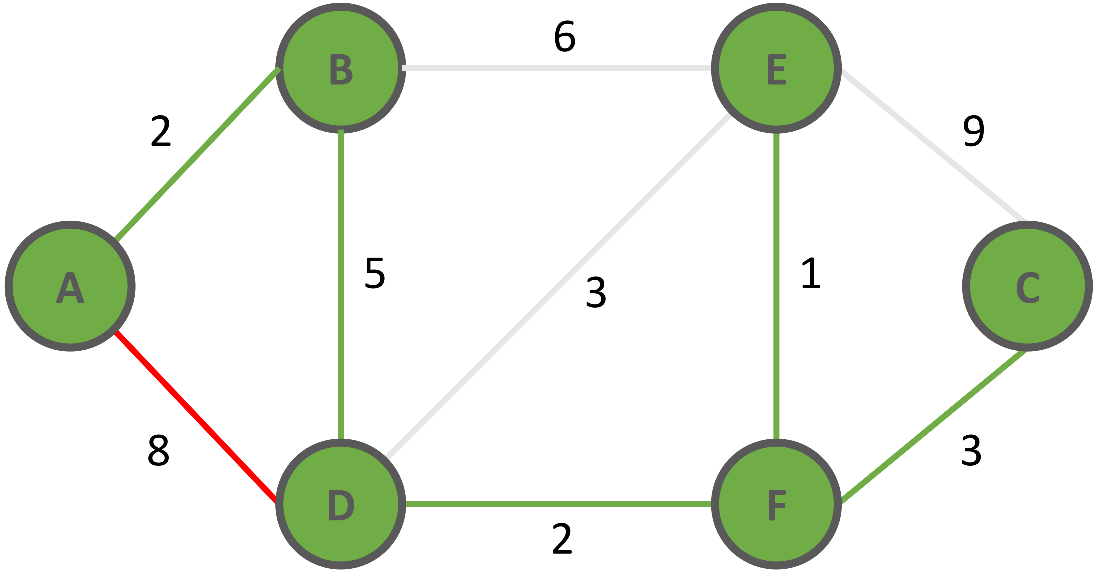
MST Edges = 5

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited
D
F
E
C
B
A

PQ
D-A, 8
E-B, 6
E-C, 9

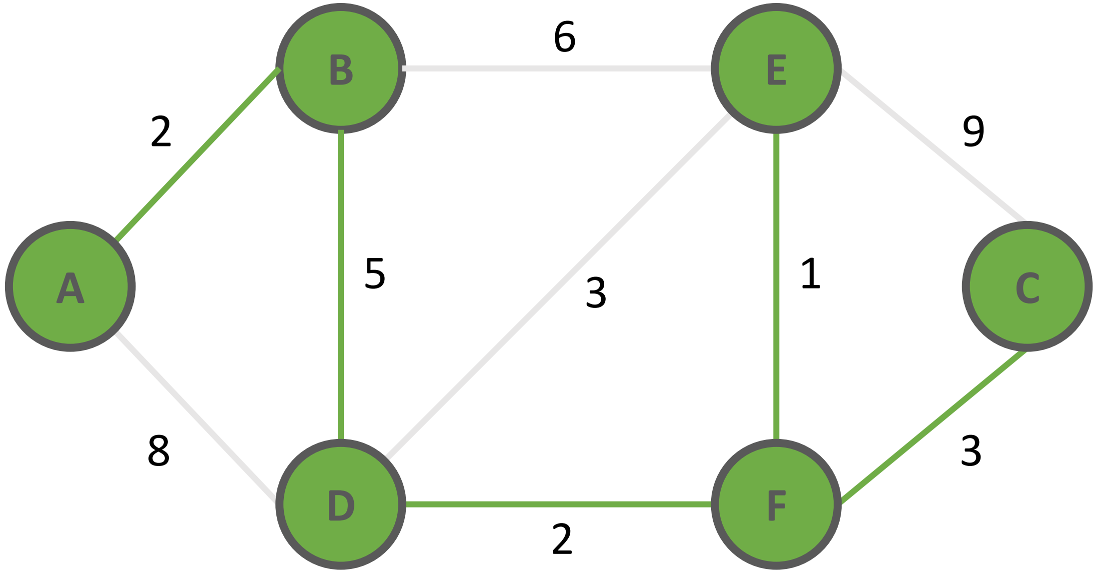
MST Edges = 5

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



Visited	PQ
D	D-A, 8
F	E-B, 6
E	E-C, 9
C	
B	
A	

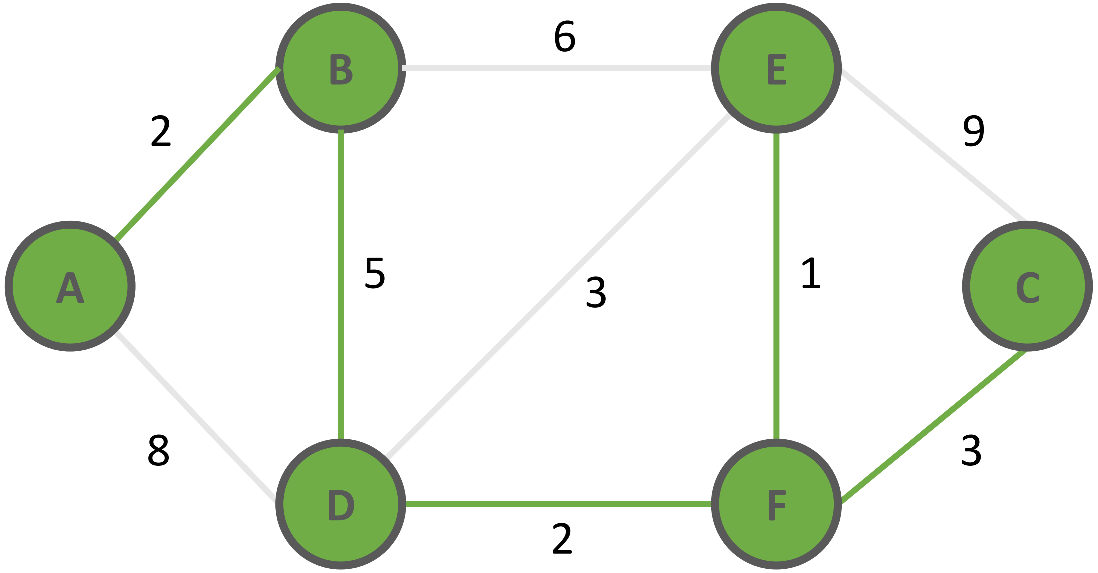
MST Edges = 5

While (PQ is not empty and MSTEdges != V - 1)

1. Dequeue the next cheapest edge from PQ
2. If the dequeued edge is outdated (i.e., destination vertex is already visited)
   skip and poll again (step 1)
3. Else
   Mark the current vertex V as visited and add the edge to MST.
4. Add V's edges to PQ which do not point to already visited nodes

Minimum Spanning Tree | Prim's Algorithm

Total V = 6
Tree Edges = V – 1 = 5



2 + 5 + 2 + 1 + 3 = 13

Visited
D
F
E
C
B
A

PQ
D-A, 8
E-B, 6
E-C, 9

MST Edges = 5

Activity # 1

Draw a graph containing the following vertices and edges

V = {A, Z, C, D, E, F, G, H}

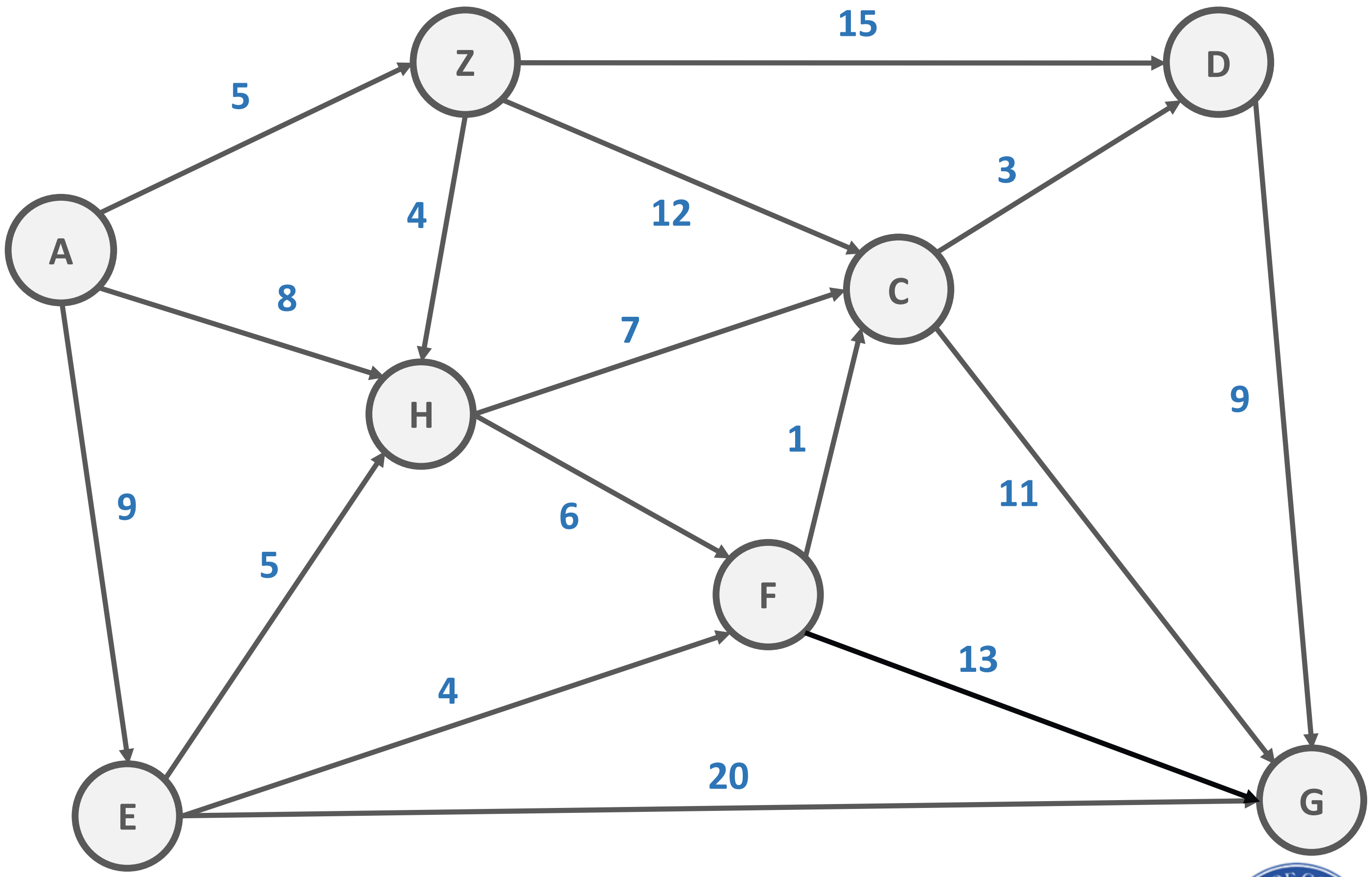
nbrs = { A: { (5, Z), (8, H), (9, E) },
C: { (3, D), (11, G) },
D: { (9, G) },
E: { (4, F), (20, G), (5, H) },
F: { (1, C), (13, G) },
H: { (7, C), (6, F) },
Z: { (4, H), (12, C), (15, D) },
}

Activity # 1 Solution

Draw a graph containing the following vertices and edges

V = {A, B, C, D, E, F, G, H}

nbrs = { A: { (5, Z), (8, H), (9, E) },
C: { (3, D), (11, G) },
D: { (9, G) },
E: { (4, F), (20, G), (5, H) },
F: { (1, C), (13, G) },
H: { (7, C), (6, F) },
Z: { (4, H), (12, C), (15, D) },
}



Activity # 2

- For the graph created in activity # 1, fill the following table and find the shortest path from vertex A to G. Also indicate the value of shortest distance from A to G?

S No.	Current Vertex	Shortest Distance	Predecessor Vertex		
				PQ= A:0, C:∞, D:∞, E:∞, F:∞, G:∞, H:∞, Z:∞	V =
0	A	0		PQ= C:∞, D:∞, **E:9**, F:∞, G:∞, **H:8**, **Z:5**	V = **A**
1	Z	~~∞~~, **5**,	A	PQ=	V = A,
2	H	~~∞~~, **8**,	A	PQ=	V = A,
3	E	~~∞~~, **9**,	A	PQ=	V = A,
4	F	∞		PQ=	V = A,
5	C	∞		PQ=	V = A,
6	D	∞		PQ=	V = A,
7	G	∞		PQ=	V = A,

Activity # 2 Solution

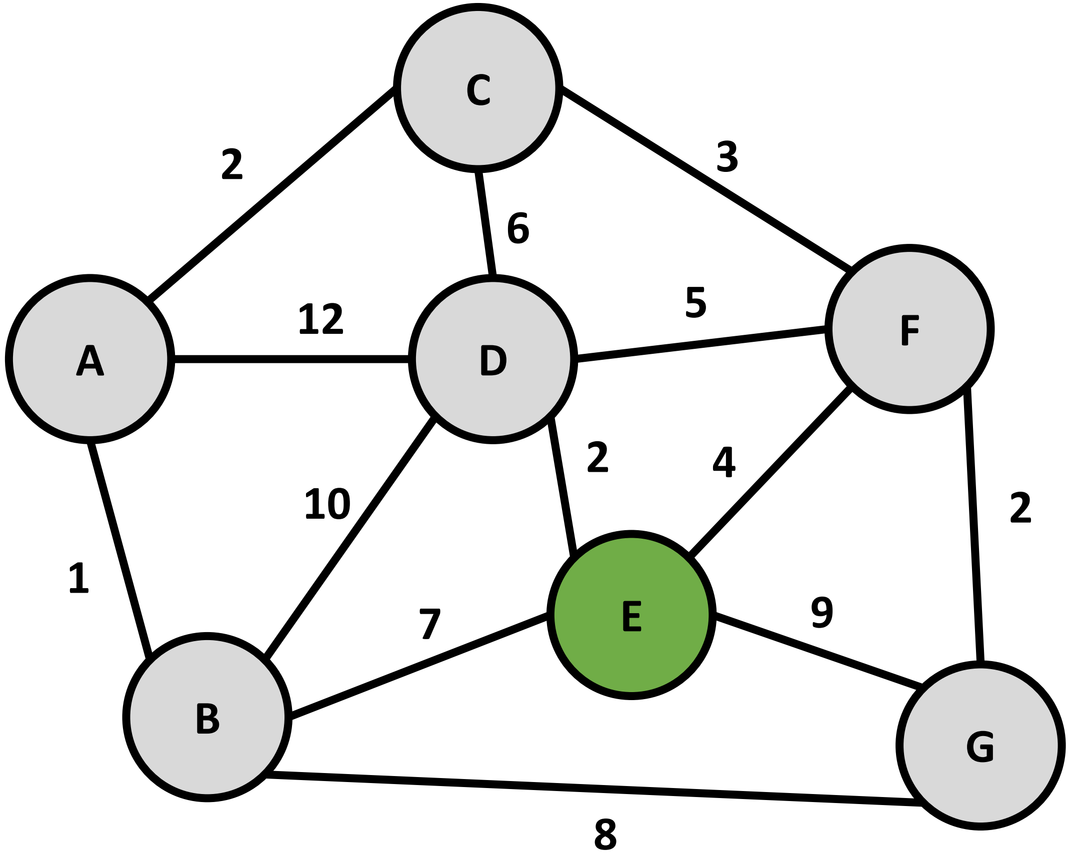
- For the graph created in activity # 1, fill the following table and find the shortest path from vertex A to G. Also indicate the value of shortest distance from A to G?

S No.	Current Vertex	Shortest Distance	Predecessor Vertex		
				PQ= A:0, C:∞, D:∞, E:∞, F:∞, G:∞, H:∞, Z:∞	V =
0	A	0		PQ= C:∞, D:∞, **E:9**, F:∞, G:∞, **H:8**, **Z:5**	V = **A**
1	Z	~~∞~~, 5,	A	PQ= **C:17**, **D:20**, E:9, F:∞, G:∞, H:8	V = A, **Z**
2	H	~~∞~~, 8,	A	PQ= **C:15**, D:20, E:9, **F:14**, G:∞	V = A, Z, **H**
3	E	~~∞~~, 9,	A	PQ= C:15, D:20, **F:13**, **G:29**	V = A, Z, H, **E**
4	F	~~∞~~, ~~14~~, 13,	~~H~~, E	PQ= **C:14**, D:20, **G:26**	V = A, Z, H, E, **F**
5	C	~~∞~~, ~~17~~, ~~15~~, 14	~~Z~~, ~~H~~, F	PQ= **D:17, G:25**	V = A, Z, H, E, F, **C**
6	D	~~∞~~, ~~20~~, 17	~~Z~~, C	PQ= G:25	V = A, Z, H, E, F, C, **D**
7	G	~~∞~~, ~~29~~, ~~26~~, 25	~~E~~, ~~D~~, ~~F~~, C	PQ=	V = A, Z, H, E, F, C, D, **G**

A-E-F-C-G = 25

Activity # 3

(a) Starting from vertex E, find the minimum spanning tree for the following graph. Fill out the tables below. Cross-out the dequeued entries from PQ.



Visited
E

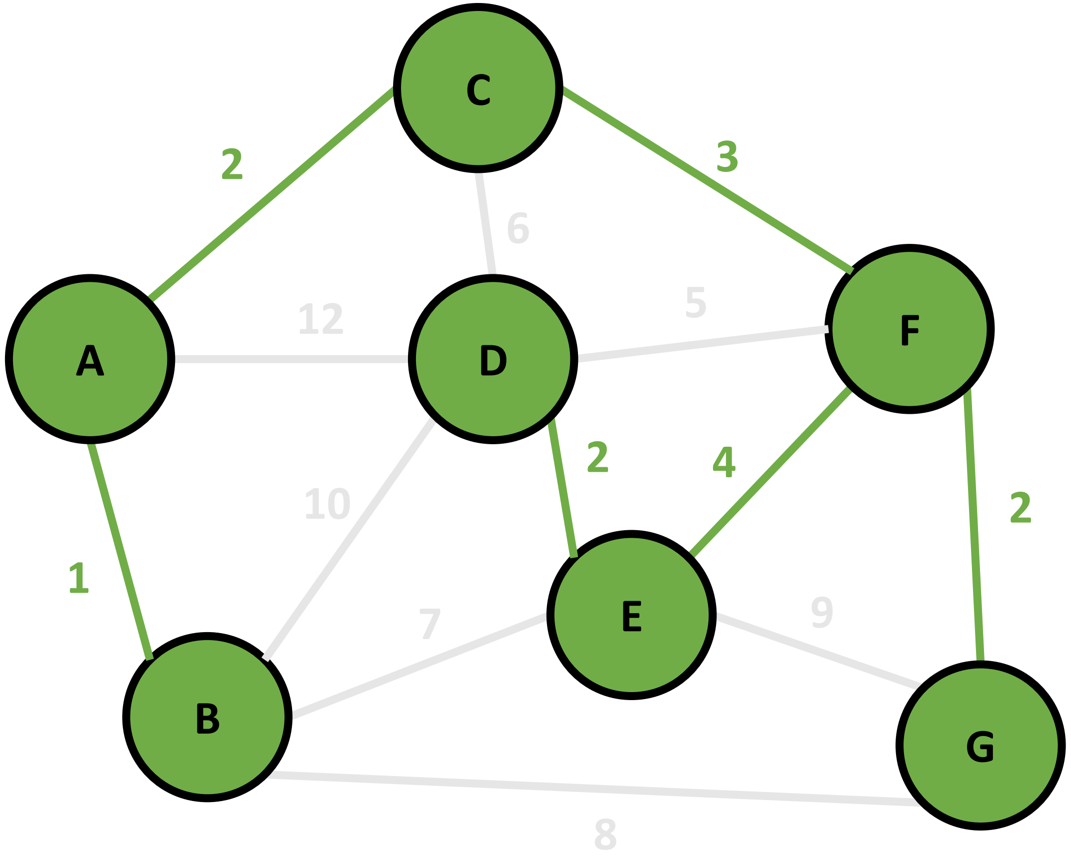
S. No.	PQ
1	E-D, 2
2	E-F, 4
3	E-B, 7
4	E-G, 9
5	
6	
7	
8	
9	
10	
11	
12	
13	

(b) What is the weight of MST?

(c) Also, use tuples (e.g. (A, B), (C, D)) to indicate the order in which edges will be added to MST.

Activity # 3 Solution

(a) Starting from vertex E, find the minimum spanning tree for the following graph. Fill out the tables below. Cross-out the dequeued entries from PQ. (b) What is the weight of MST?



Visited
E
D
F
G
C
A
B

S. No.	PQ
1	~~E-D, 2~~
2	~~E-F, 4~~
3	E-B, 7
4	E-G, 9
5	D-F, 5
6	D-C, 6
7	D-B, 10
8	D-A, 12
9	~~F-G, 2~~
10	~~F-C, 3~~
11	G-B, 8
12	~~C-A, 2~~
13	~~A-B, 1~~

(b). MST Weight = 2 + 4 + 2 + 3 + 2 + 1 = 14

(c). (E-D), (E-F), (F-G), (F-C), (C-A), (A-B)