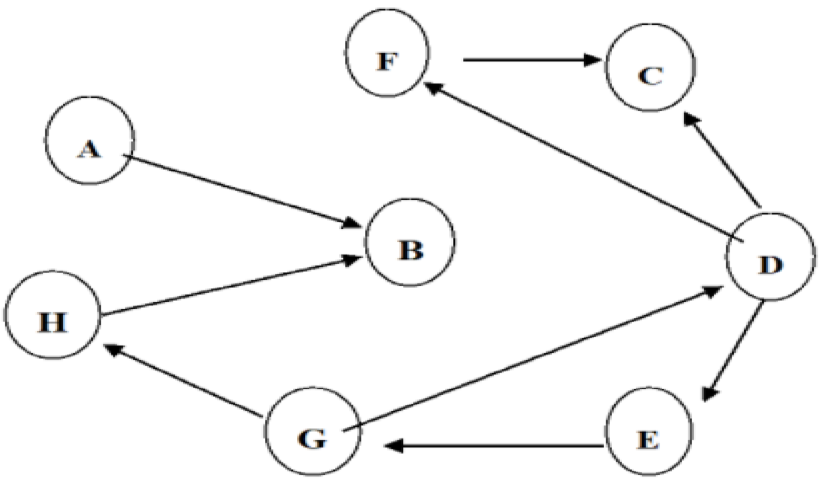
Graphs

1. Write a Program to implement DFS algorithm and print the DFS sequence for below graph start with node D.

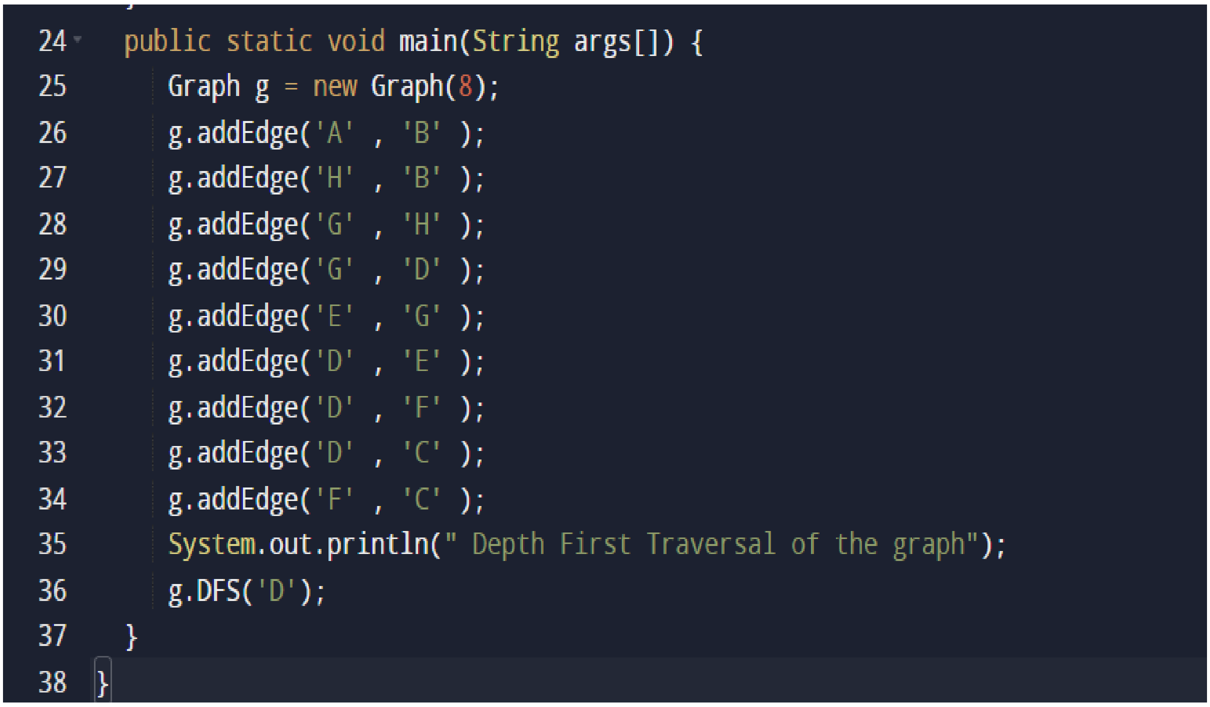


CODE-

```
import java.util.*;

class Graph {

 private LinkedList<Character> adjLists[];

 private boolean visited[];

 Graph(int vertices) {

  adjLists = new LinkedList[vertices];

  visited = new boolean[vertices];

  for (int i = 0; i < vertices; i++)

   adjLists[i] =new LinkedList<Character>();

  }

 void addEdge(char src, char dest) {
  adjLists[src%8].add(dest);
}
void DFS(int vertex) {
  visited[vertex%8] = true;
  System.out.print((char)vertex + " ");
  Iterator<Character> ite =adjLists[vertex%8].listIterator();
  while (ite.hasNext()) {
    int adj = ite.next();
    if (!visited[adj%8])
       DFS(adj);
  }
}
public static void main(String args[]) {
   Graph g = new Graph(8);
   g.addEdge('A' , 'B' );
   g.addEdge('H' , 'B' );
   g.addEdge('G' , 'H' );
   g.addEdge('G' , 'D' );
   g.addEdge('E' , 'G' );
   g.addEdge('D' , 'E' );
   g.addEdge('D' , 'F' );
```

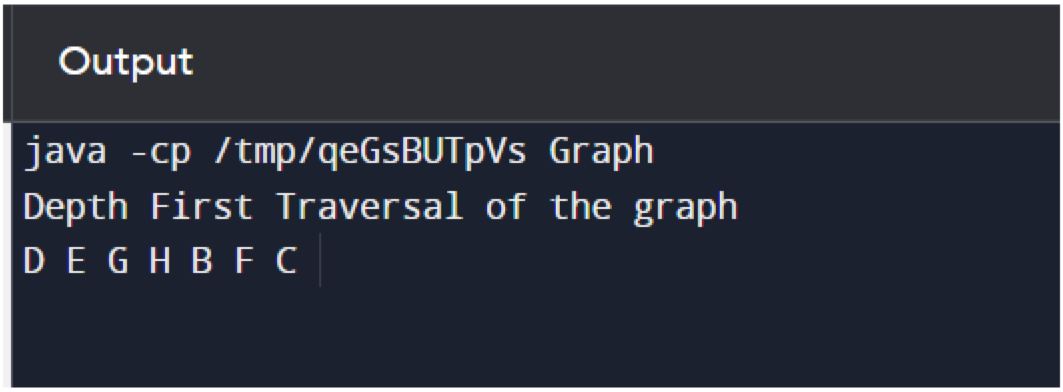
```
    g.addEdge('D' , 'C' );

    g.addEdge('F' , 'C' );

    System.out.println(" Depth First Traversal of the graph");

    g.DFS('D');

  }

}
```

```
import java.util.*;
class Graph {
  private LinkedList<Character> adjLists[];
  private boolean visited[];
  Graph(int vertices) {
    adjLists = new LinkedList[vertices];
    visited = new boolean[vertices];
    for (int i = 0; i < vertices; i++)
      adjLists[i] =new LinkedList<Character>();
    }
  void addEdge(char src, char dest) {
    adjLists[src%8].add(dest);
  }
  void DFS(int vertex) {
    visited[vertex%8] = true;
    System.out.print((char)vertex + " ");
    Iterator<Character> ite =adjLists[vertex%8].listIterator();
    while (ite.hasNext()) {
      int adj = ite.next();
      if (!visited[adj%8])
         DFS(adj);
    }
  }
  public static void main(String args[]) {
     Graph g = new Graph(8);
     g.addEdge('A' , 'B' );
     g.addEdge('H' , 'B' );
```



```
    public static void main(String args[]) {
        Graph g = new Graph(8);
        g.addEdge('A' , 'B' );
        g.addEdge('H' , 'B' );
        g.addEdge('G' , 'H' );
        g.addEdge('G' , 'D' );
        g.addEdge('E' , 'G' );
        g.addEdge('D' , 'E' );
        g.addEdge('D' , 'F' );
        g.addEdge('D' , 'C' );
        g.addEdge('F' , 'C' );
        System.out.println(" Depth First Traversal of the graph");
        g.DFS('D');
    }
}
```

OUTPUT-



```
java -cp /tmp/qeGsBUTpVs Graph
Depth First Traversal of the graph
D E G H B F C
```

2. Write a Program to implement BFS Algorithm and print the BFS sequence start with node A.

CODE-

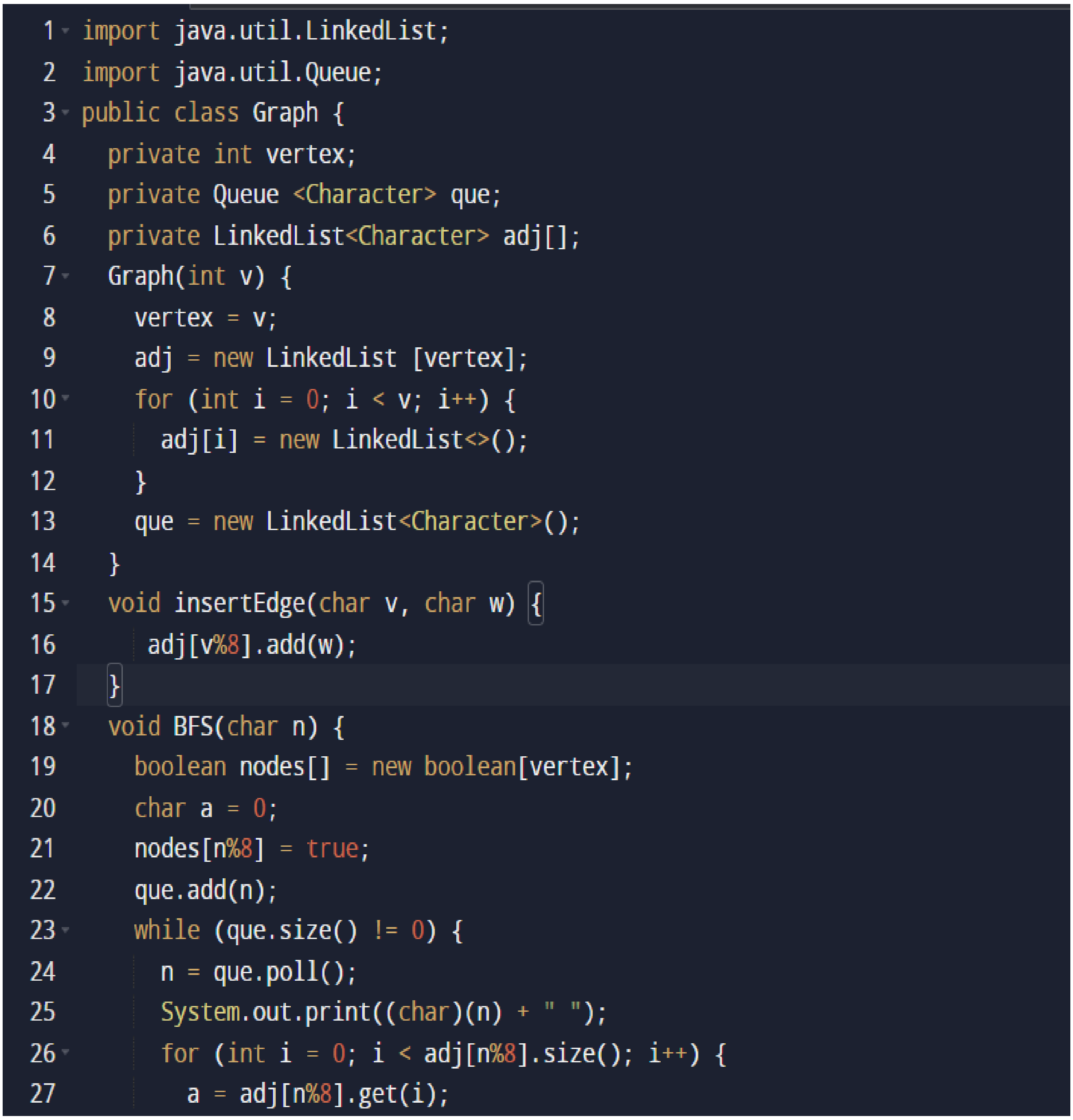
import java.util.LinkedList;

import java.util.Queue;

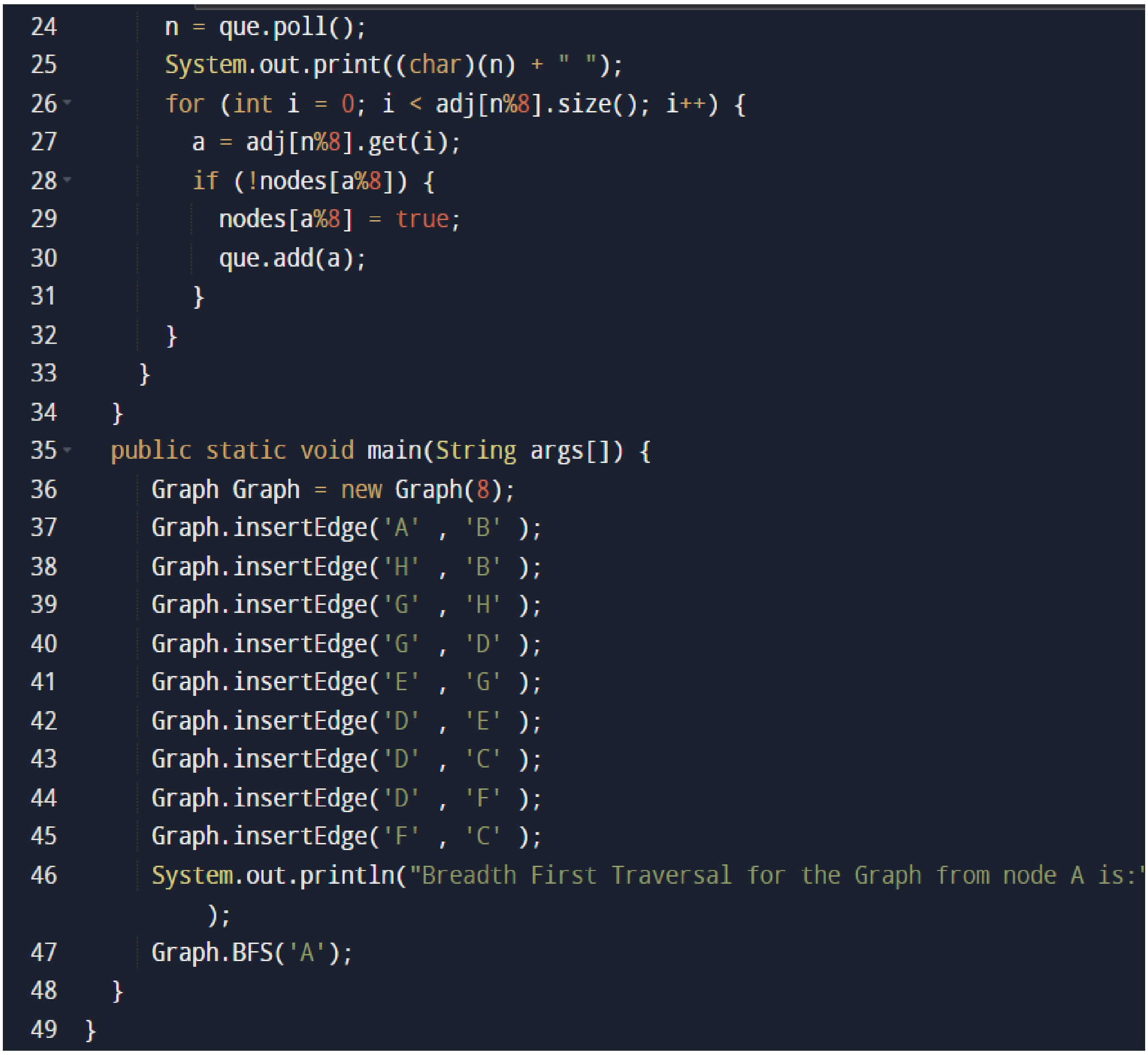
```
public class Graph {
  private int vertex;
  private Queue <Character> que;
  private LinkedList<Character> adj[];
  Graph(int v) {
    vertex = v;
    adj = new LinkedList [vertex];
    for (int i = 0; i < v; i++) {
      adj[i] = new LinkedList<>();
    }
    que = new LinkedList<Character>();
  }
  void insertEdge(char v, char w) {
     adj[v%8].add(w);
  }
  void BFS(char n) {
    boolean nodes[] = new boolean[vertex];
    char a = 0;
    nodes[n%8] = true;
    que.add(n);
    while (que.size() != 0) {
```

```
    n = que.poll();
    System.out.print((char)(n) + " ");
    for (int i = 0; i < adj[n%8].size(); i++) {
      a = adj[n%8].get(i);
      if (!nodes[a%8]) {
        nodes[a%8] = true;
        que.add(a);
      }
    }
  }
}
public static void main(String args[]) {
   Graph Graph = new Graph(8);
   Graph.insertEdge('A' , 'B' );
   Graph.insertEdge('H' , 'B' );
   Graph.insertEdge('G' , 'H' );
   Graph.insertEdge('G' , 'D' );
   Graph.insertEdge('E' , 'G' );
   Graph.insertEdge('D' , 'E' );
   Graph.insertEdge('D' , 'C' );
   Graph.insertEdge('D' , 'F' );
```

```
    Graph.insertEdge('F' , 'C' );
    System.out.println("Breadth First Traversal for the Graph from node A is:");
    Graph.BFS('A');
  }
}
```

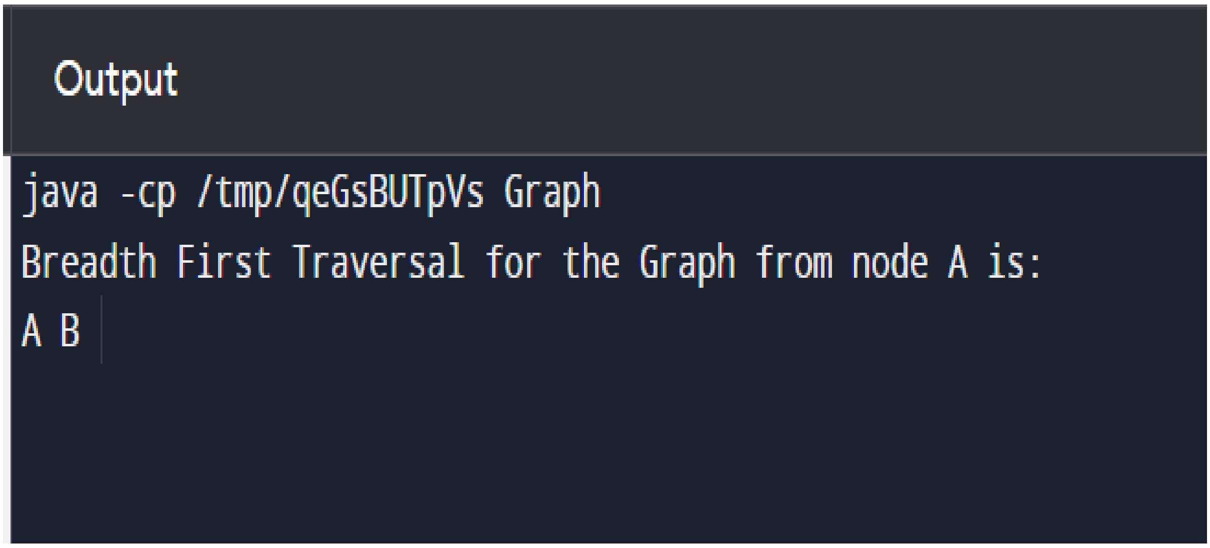


```
import java.util.LinkedList;
import java.util.Queue;
public class Graph {
  private int vertex;
  private Queue <Character> que;
  private LinkedList<Character> adj[];
  Graph(int v) {
    vertex = v;
    adj = new LinkedList [vertex];
    for (int i = 0; i < v; i++) {
      adj[i] = new LinkedList<>();
    }
    que = new LinkedList<Character>();
  }
  void insertEdge(char v, char w) {
     adj[v%8].add(w);
  }
  void BFS(char n) {
    boolean nodes[] = new boolean[vertex];
    char a = 0;
    nodes[n%8] = true;
    que.add(n);
    while (que.size() != 0) {
      n = que.poll();
      System.out.print((char)(n) + " ");
      for (int i = 0; i < adj[n%8].size(); i++) {
        a = adj[n%8].get(i);
```



```
      n = que.poll();
      System.out.print((char)(n) + " ");
      for (int i = 0; i < adj[n%8].size(); i++) {
        a = adj[n%8].get(i);
        if (!nodes[a%8]) {
          nodes[a%8] = true;
          que.add(a);
        }
      }
    }
  }
  public static void main(String args[]) {
     Graph Graph = new Graph(8);
     Graph.insertEdge('A' , 'B' );
     Graph.insertEdge('H' , 'B' );
     Graph.insertEdge('G' , 'H' );
     Graph.insertEdge('G' , 'D' );
     Graph.insertEdge('E' , 'G' );
     Graph.insertEdge('D' , 'E' );
     Graph.insertEdge('D' , 'C' );
     Graph.insertEdge('D' , 'F' );
     Graph.insertEdge('F' , 'C' );
     System.out.println("Breadth First Traversal for the Graph from node A is:"
         );
     Graph.BFS('A');
  }
}
```

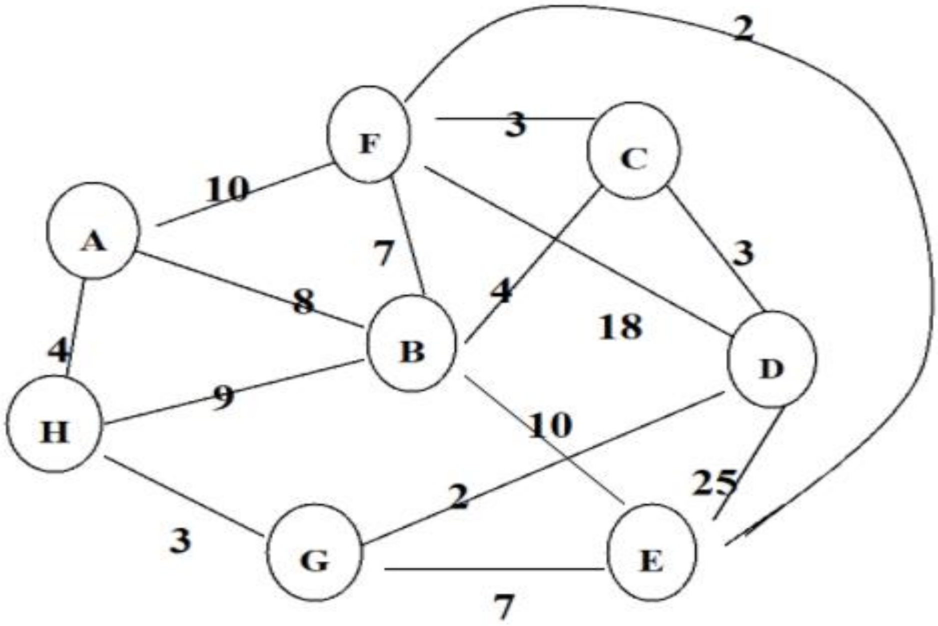
OUTPUT-



```
java -cp /tmp/qeGsBUTpVs Graph
Breadth First Traversal for the Graph from node A is:
A B
```

Challenging:

3. Write a Program to Implement Prim’s Algorithm and find the minimum spanning tree for the given graph.



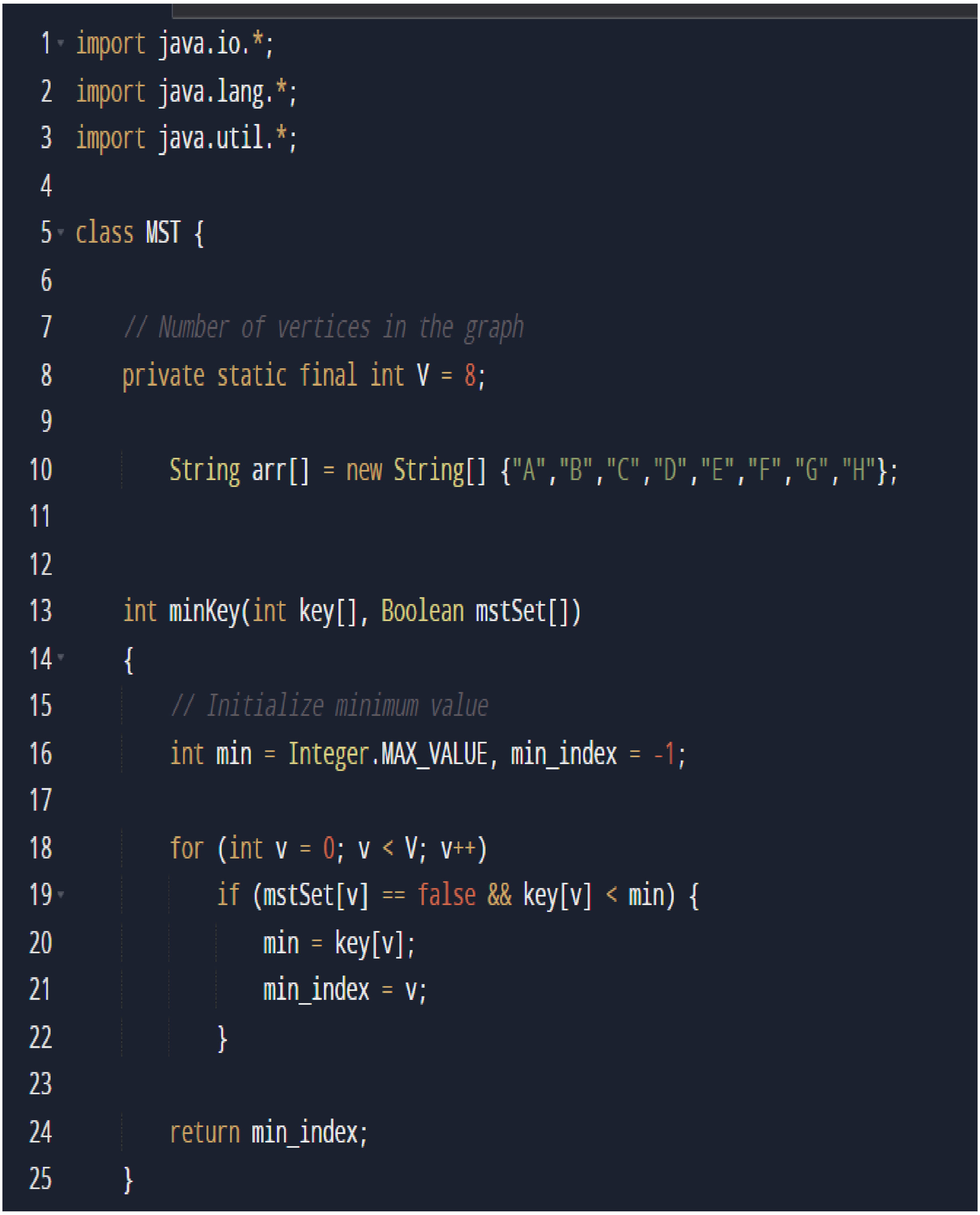
CODE-

```
import java.io.*;

import java.lang.*;

import java.util.*;


class MST {


   // Number of vertices in the graph

   private static final int V = 8;
```

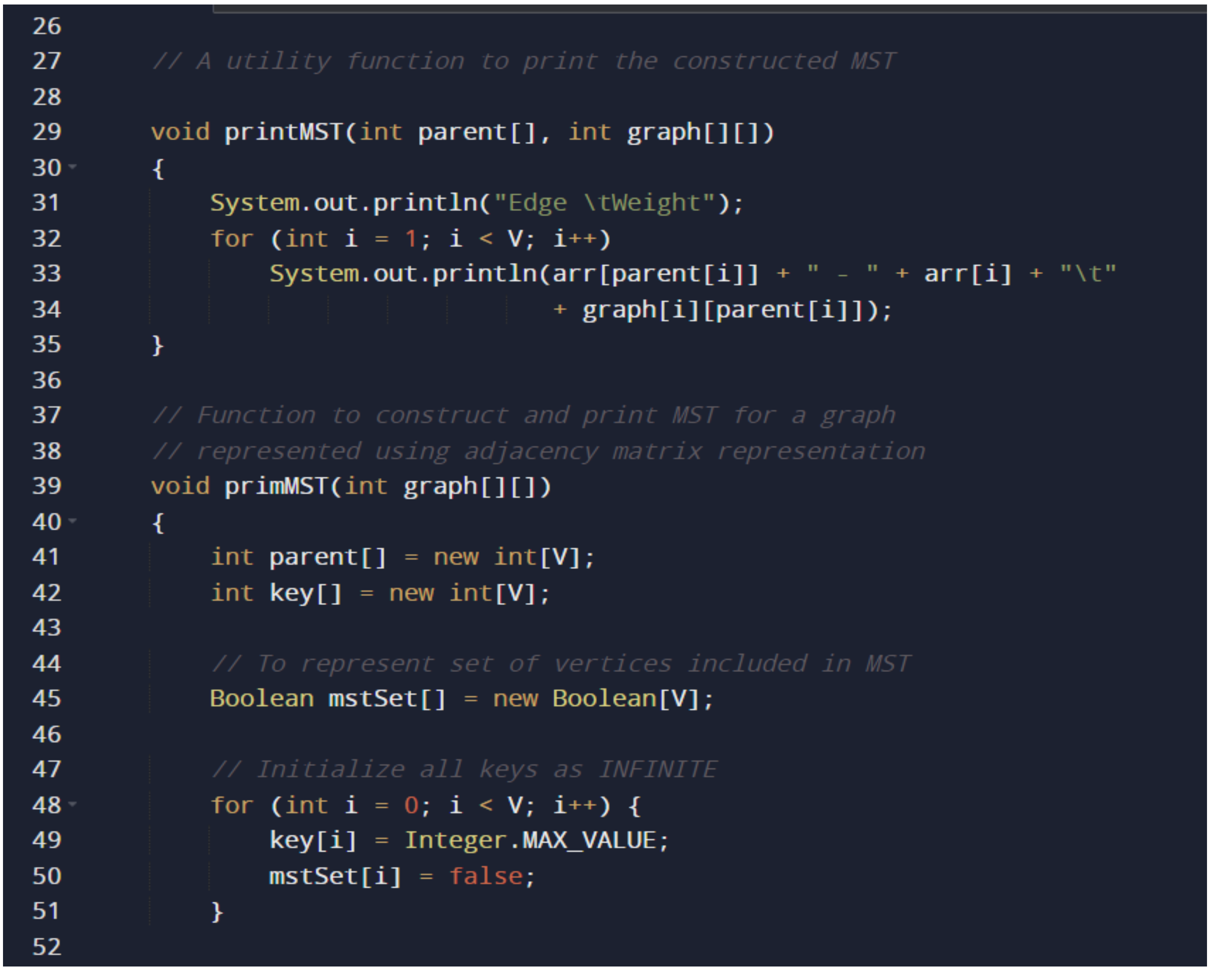
```
                String arr[] = new String[]
{"A","B","C","D","E","F","G","H"};



    int minKey(int key[], Boolean mstSet[])

    {

        // Initialize minimum value

        int min = Integer.MAX_VALUE, min_index = -1;


        for (int v = 0; v < V; v++)

            if (mstSet[v] == false && key[v] < min) {

                min = key[v];

                min_index = v;

            }


        return min_index;

    }


    // A utility function to print the constructed MST


    void printMST(int parent[], int graph[][])
```

```
{
   System.out.println("Edge \tWeight");
   for (int i = 1; i < V; i++)
      System.out.println(arr[parent[i]] + " - " + arr[i] + "\t"
                     + graph[i][parent[i]]);
}

// Function to construct and print MST for a graph
// represented using adjacency matrix representation
void primMST(int graph[][])
{
   int parent[] = new int[V];
   int key[] = new int[V];

   // To represent set of vertices included in MST
   Boolean mstSet[] = new Boolean[V];

   // Initialize all keys as INFINITE
   for (int i = 0; i < V; i++) {
      key[i] = Integer.MAX_VALUE;
      mstSet[i] = false;
```

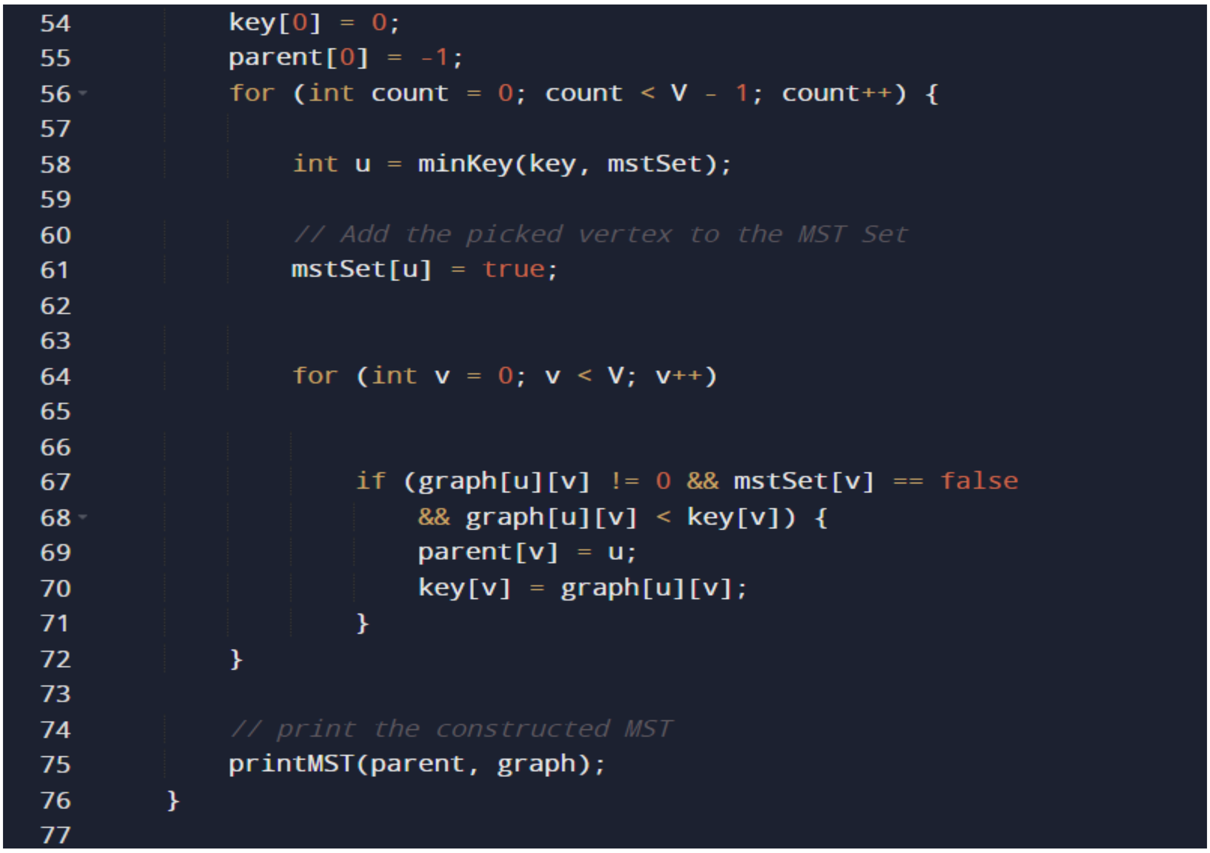
```
}


key[0] = 0;
parent[0] = -1;
for (int count = 0; count < V - 1; count++) {

    int u = minKey(key, mstSet);

    // Add the picked vertex to the MST Set
    mstSet[u] = true;


    for (int v = 0; v < V; v++)


        if (graph[u][v] != 0 && mstSet[v] == false
            && graph[u][v] < key[v]) {
            parent[v] = u;
            key[v] = graph[u][v];
        }
        }


        // print the constructed MST
        printMST(parent, graph);
    }


    public static void main(String[] args)
    {
        MST t = new MST();
        int graph[][] = new int[][] { { 0, 8, 0, 0, 0, 10, 0, 4 }, //A
                                     { 8, 0, 4, 0, 10, 7, 0, 9 }, //B
                                     { 0, 4, 0, 3, 0, 3, 0, 0 }, //C
                                     { 0, 0, 3, 0, 25, 0, 2, 0 }, //D
                                                                                    { 0, 10, 0,
25, 0, 2, 7, 0 }, //E
                                     { 10, 7, 3, 0, 2, 0, 0, 0 }, //F
                                     { 0, 0, 0, 2, 7, 0, 0, 3 }, //G
                                     { 4, 9, 0, 0, 0, 0, 3, 0 }, //H
                                   };


        // Print the solution
        t.primMST(graph);
    }
}
```



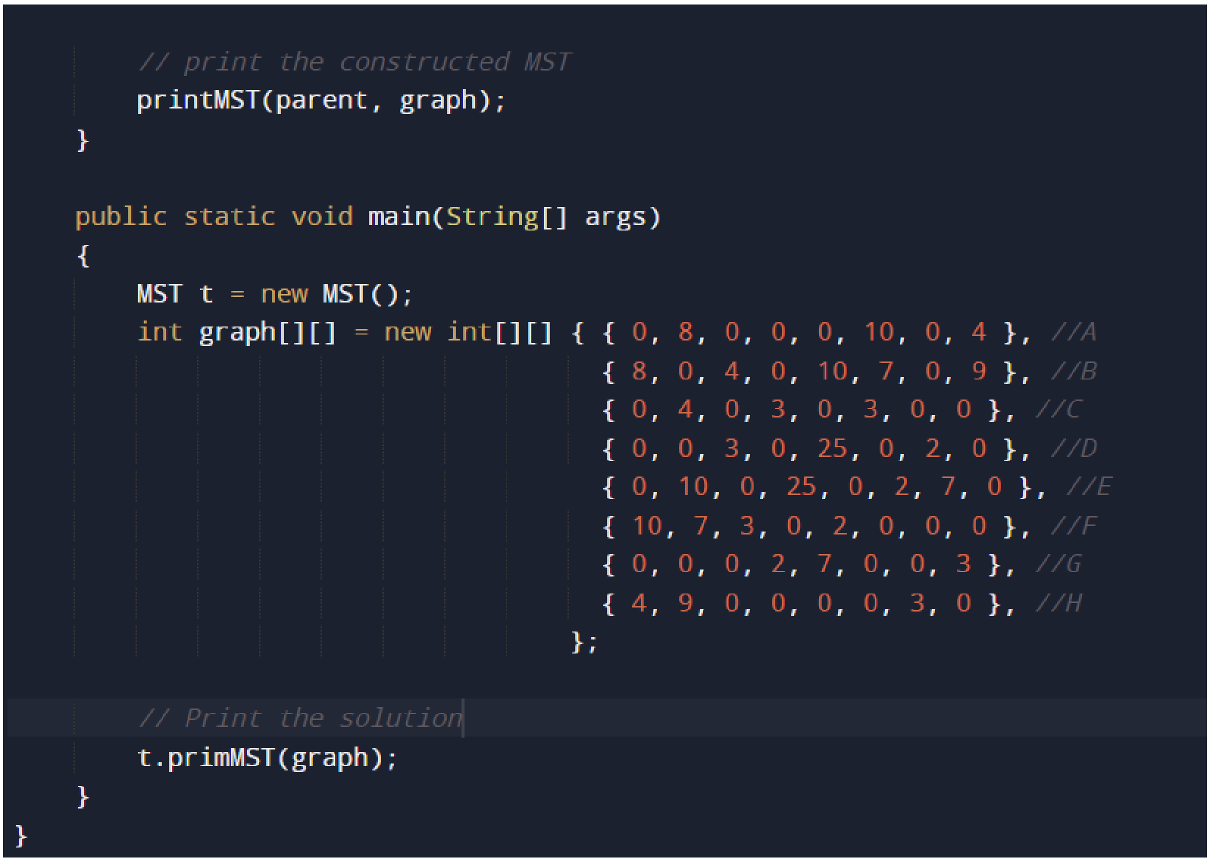
```
import java.io.*;
import java.lang.*;
import java.util.*;

class MST {

    // Number of vertices in the graph
    private static final int V = 8;

        String arr[] = new String[] {"A","B","C","D","E","F","G","H"};


    int minKey(int key[], Boolean mstSet[])
    {
        // Initialize minimum value
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++)
            if (mstSet[v] == false && key[v] < min) {
                min = key[v];
                min_index = v;
            }

        return min_index;
    }
```



```java

    // A utility function to print the constructed MST

    void printMST(int parent[], int graph[][])
    {
        System.out.println("Edge \tWeight");
        for (int i = 1; i < V; i++)
            System.out.println(arr[parent[i]] + " - " + arr[i] + "\t"
                               + graph[i][parent[i]]);
    }

    // Function to construct and print MST for a graph
    // represented using adjacency matrix representation
    void primMST(int graph[][])
    {
        int parent[] = new int[V];
        int key[] = new int[V];

        // To represent set of vertices included in MST
        Boolean mstSet[] = new Boolean[V];

        // Initialize all keys as INFINITE
        for (int i = 0; i < V; i++) {
            key[i] = Integer.MAX_VALUE;
            mstSet[i] = false;
        }

```

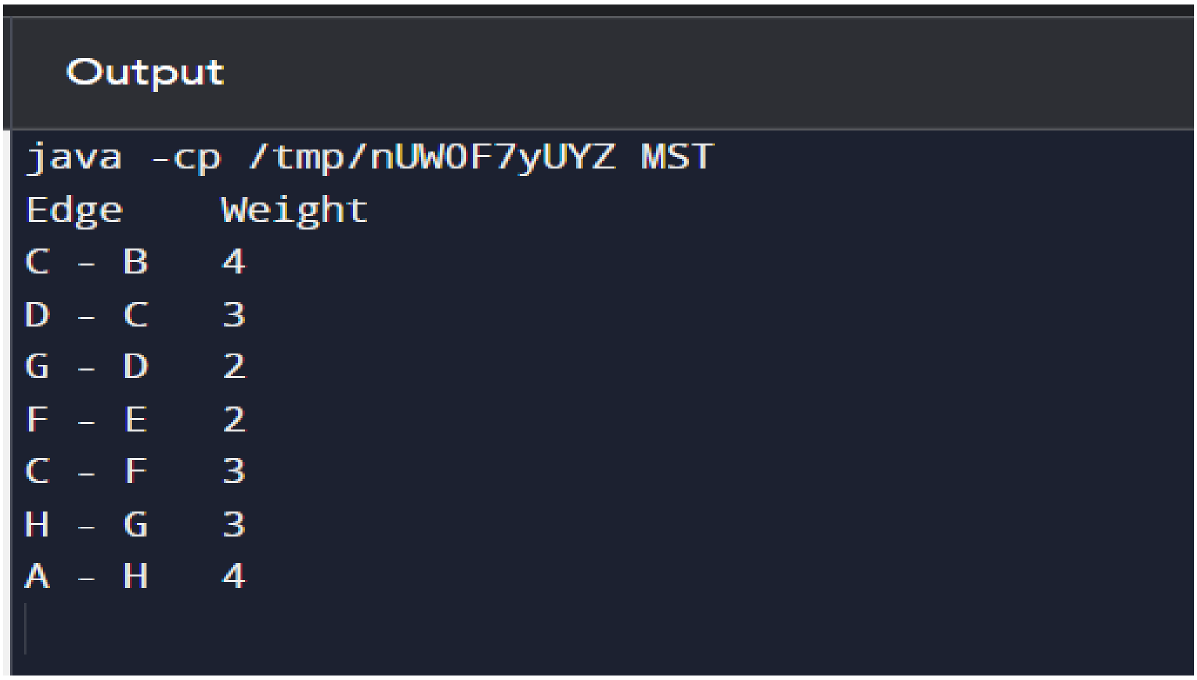


```java
        key[0] = 0;
        parent[0] = -1;
        for (int count = 0; count < V - 1; count++) {

            int u = minKey(key, mstSet);

            // Add the picked vertex to the MST Set
            mstSet[u] = true;


            for (int v = 0; v < V; v++)


                if (graph[u][v] != 0 && mstSet[v] == false
                    && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
        }

        // print the constructed MST
        printMST(parent, graph);
    }

```



```
        // print the constructed MST
        printMST(parent, graph);
    }

    public static void main(String[] args)
    {
        MST t = new MST();
        int graph[][] = new int[][] { { 0, 8, 0, 0, 0, 10, 0, 4 }, //A
                                      { 8, 0, 4, 0, 10, 7, 0, 9 }, //B
                                      { 0, 4, 0, 3, 0, 3, 0, 0 }, //C
                                      { 0, 0, 3, 0, 25, 0, 2, 0 }, //D
                                      { 0, 10, 0, 25, 0, 2, 7, 0 }, //E
                                      { 10, 7, 3, 0, 2, 0, 0, 0 }, //F
                                      { 0, 0, 0, 2, 7, 0, 0, 3 }, //G
                                      { 4, 9, 0, 0, 0, 0, 3, 0 }, //H
                                    };

        // Print the solution
        t.primMST(graph);
    }
}
```

OUTPUT-



```
java -cp /tmp/nUWOF7yUYZ MST
Edge    Weight
C - B   4
D - C   3
G - D   2
F - E   2
C - F   3
H - G   3
A - H   4
```

----------------------------------X--------------------------------------

Thank you!