```java
package graph;

import java.util.Queue;
import java.util.LinkedList;

class BFS {

    public static void BFSUsingAdjacentMatrix(AdjacencyMatrix adjacencyMatrix) {

        Integer[][] innerAdjacencyMatrix = adjacencyMatrix.getInnerAdjacencyMatrix();

        boolean[] hasVisited = new boolean[innerAdjacencyMatrix.length];

        for (int i = 1; i < hasVisited.length; i++) {
            BFSUsingAdjacentMatrixHelper(i, innerAdjacencyMatrix, hasVisited,
            adjacencyMatrix);
        }
    }

    private static void BFSUsingAdjacentMatrixHelper(int currentVertexIndex,
                                                     Integer[][] innerAdjacencyMatrix,
                                                     boolean[] hasVisited,
                                                     AdjacencyMatrix
                                                     adjacencyMatrix) {

        if (hasVisited[currentVertexIndex]) {
            return;
        }

        Queue<Integer> queue = new LinkedList<Integer>();
        queue.offer(currentVertexIndex);

        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                Integer index = queue.poll();
                if (!hasVisited[index]) {
                    hasVisited[index] = true;
                    System.out.println(adjacencyMatrix.getVertexName(index));
                }
                for (int j = 0; j < innerAdjacencyMatrix[index].length; j++) {
                    if (innerAdjacencyMatrix[index][j] != null && !hasVisited[j]) {
                        queue.offer(j);
                    }
                }
            }
        }
    }


    public static void BFSUsingAdjacentTable(AdjacencyTable adjacencyTable) {

        LinkedList<LinkedList<AdjacencyTable.Node>> innerAdjacencyTable
                = adjacencyTable.getInnerAdjacencyTable();

        boolean[] hasVisited = new boolean[innerAdjacencyTable.size()];

        for (int i = 1; i < hasVisited.length; i++) {
            BFSUsingAdjacentTableHelper(adjacencyTable, hasVisited,
                    i, innerAdjacencyTable);
        }
    }

    private static void BFSUsingAdjacentTableHelper(AdjacencyTable adjacencyTable,
    boolean[] hasVisited,
                                                    int currentIndex,

                                                    LinkedList<LinkedList<AdjacencyTab
                                                    le.Node>> innerAdjacencyTable) {

        if (hasVisited[currentIndex]) {
            return;
        }
```

```
69
70          Queue<AdjacencyTable.Node> queue = new LinkedList<>();
71          queue.offer(innerAdjacencyTable.get(currentIndex).get(0));
72
73          while (!queue.isEmpty()) {
74              int size = queue.size();
75              for (int i = 0; i < size; i++) {
76                  AdjacencyTable.Node tempNode = queue.poll();
77                  if (!hasVisited[adjacencyTable.getIndex(tempNode.name)]) {
78                      hasVisited[adjacencyTable.getIndex(tempNode.name)] = true;
79                      System.out.println(tempNode.name);
80                  }
81
82                  int row = adjacencyTable.getIndex(tempNode.name);
83
84                  for (int j = 1; j < innerAdjacencyTable.get(row).size(); j++) {
85                      int index = adjacencyTable.getIndex(
86                              innerAdjacencyTable.get(row).get(j).name);
87                      if (!hasVisited[index]) {
88                          queue.offer(innerAdjacencyTable.get(row).get(j));
89                      }
90                  }
91              }
92          }
93      }
94
95
96      public static void main(String[] args) {
97  /*
98          String[] vertex = {"1", "2", "3", "4", "5", "6", "7"};
99
100         AdjacencyMatrix adjacencyMatrix = new AdjacencyMatrix(vertex);
101         adjacencyMatrix.updateEdge("1", "2", 1);
102         adjacencyMatrix.updateEdge("3", "1", 6);
103         adjacencyMatrix.updateEdge("4", "1", 3);
104         adjacencyMatrix.updateEdge("2", "3", 4);
105         adjacencyMatrix.updateEdge("2", "4", 4);
106         adjacencyMatrix.updateEdge("4", "3", 9);
107         adjacencyMatrix.updateEdge("6", "2", 7);
108         adjacencyMatrix.updateEdge("7", "6", 7);
109         adjacencyMatrix.updateEdge("7", "4", 7);
110         adjacencyMatrix.updateEdge("5", "7", 7);
111         adjacencyMatrix.updateEdge("5", "6", 7);
112
113         BFSUsingAdjacentMatrix(adjacencyMatrix);*/
114
115         String[] vertex = {"1", "2", "3", "4", "5", "6", "7"};
116         AdjacencyTable adjacencyTable = new AdjacencyTable(vertex);
117
118         adjacencyTable.updateEdge("1", "2", 1);
119         adjacencyTable.updateEdge("3", "1", 6);
120         adjacencyTable.updateEdge("4", "1", 3);
121         adjacencyTable.updateEdge("2", "3", 4);
122         adjacencyTable.updateEdge("2", "4", 4);
123         adjacencyTable.updateEdge("4", "3", 9);
124         adjacencyTable.updateEdge("6", "2", 7);
125         adjacencyTable.updateEdge("7", "6", 7);
126         adjacencyTable.updateEdge("7", "4", 7);
127         adjacencyTable.updateEdge("5", "7", 7);
128         adjacencyTable.updateEdge("5", "6", 7);
129
130         BFSUsingAdjacentTable(adjacencyTable);
131     }
132 }
```