```
public class Question4 {  O(1)

   public static boolean question4(int[][] matrix) {

      for (int i = 0; i < matrix.length; i++) { O(n)

         for (int j = 0; j < matrix.length; j++) { O(n)

            if (matrix[i][j] != matrix[j][i]) { O(1)

               return true; O(1)
            }
         }
      }
      return false; O(1)
   }

   public static void main(String[] args) {

      System.out.println("Directed or undirected checker: "); O(1)

      System.out.println();O(1)

      int[][] input = { O(1)
            {0, 1, 0},
            {1, 0, 1},
            {0, 0, 1}
      };
      System.out.println("Test 1 - Is the matrix representing a directed graph? : " + question4(input)); //this should return true O(1)

      System.out.println("----------------------------------------------------");O(1)

      int[][] input2 = { O(1)
            {0, 1, 1},
            {1, 0, 1},
            {1, 1, 0}
      };

      System.out.println("Test 2 - Is the matrix representing a directed graph? : " + question4(input2)); //this should return false O(1)


      System.out.println("----------------------------------------------------");O(1)


      int[][] input3 = { O(1)
            {0, 1, 0, 0},
            {0, 0, 1, 0},
            {0, 0, 0, 1},
            {1, 0, 0, 0}
      };
```

Question 4

Time Complexity: O(n^2)

- Nested for-loop, where n is the length of the matrix.

Space Complexity: O(1)

- No additional data structures.

```
        System.out.println("Test 3 - Is the matrix representing a directed graph? : " + question4(input3)); //this should return false O(1)




    }

}
```

Question 5 is on the next page!

```
import java.util.*; O(1)

public class Question5 {
   static class edge {

      String destination; O(1)
      int weight; O(1)

      edge(String destination, int weight) {
         this.destination = destination;
         this.weight = weight;
      }
   }

   static class graph {
      private final Map<String, List<edge>> adjacent = new HashMap<>(); O(1)

      public void addEdge(String from, String destination, int weight) {
         if (!adjacent.containsKey(from)) { O(1)
            adjacent.put(from, new ArrayList<>()); O(1)
         }
         adjacent.get(from).add(new edge(destination, weight)); O(1)

      }

      public void printweight(String start, String end, int targetWeight) {
         Deque<String> path = new ArrayDeque<>(); O(1)
         Set<String> visited = new HashSet<>(); O(1)
         depthfirst(start, end, 0, targetWeight, path, visited); O(1)
      }

      private void depthfirst(String current, String end, int currentWeight, int targetWeight, Deque<String> path, Set<String> visited) { O(V
+ E)
         path.addLast(current);
         visited.add(current);

         if (current.equals(end) && currentWeight == targetWeight) {
            System.out.println(new ArrayList<>(path));
         }

         if (adjacent.containsKey(current)) {
            for (edge edge : adjacent.get(current)) {
               if (!visited.contains(edge.destination) && currentWeight + edge.weight <= targetWeight) {
                  depthfirst(edge.destination, end, currentWeight + edge.weight, targetWeight, path, visited);
               }
            }
         }

         path.removeLast();O(1)
         visited.remove(current); O(1)
      }
   }

   public static void main(String[] args) {
      graph test = new graph();O(1)
```

Question 5

Time Complexity: O(V+E)

- Worst case for the DFS method is O(V+E).

Space Complexity: O(V+E)

- Worst case for the DFS method is O(V+E).

```
        test.addEdge("A", "B", 3);
        test.addEdge("A", "C", 4);
        test.addEdge("A", "F", 5);
        test.addEdge("B", "D", 2);
        test.addEdge("B", "E", 4);
        test.addEdge("C", "D", 3);
        test.addEdge("C", "F", 2); O(1)
        test.addEdge("D", "E", 2);
        test.addEdge("D", "F", 1);
        test.addEdge("E", "F", 6);
        String u = "A";
        String w = "D"; O(1)

        System.out.println("Start: " + u);
        System.out.println("End: " + w); O(1)
        System.out.println("PATHS: ");
        test.printweight(u, w, 7);
    }
}
```

Question 6 is on the next page!

```
import java.util.*;

public class Question6 {


   public static void graph(String input){ O(1)
      if (input == null) {
         return;
      }

      input = input.replaceAll("[\\[\\]{}()|| ]", "").trim(); O(m)


      String[] parts = input.split("\\s*,\\s*(?=[a-z A-Z]\\s*,)"); O(n)

      int n = parts.length; O(1)
      int[][] matrix = new int[n][n]; O(1)
      String[] vertices = new String[n]; O(1)
      int[] jumps = new int[n]; O(1)


      for (int i = 0; i < n; i++) { O(n)

         String[] pair = parts[i].split(",");

         vertices[i] = pair[0].trim();
         jumps[i] = Integer.parseInt(pair[1].trim());
      }


      for (int i = 0; i < n; i++) { O(n)

         int left = (i - jumps[i] + n) % n;
         int right = (i + jumps[i]) % n;

         matrix[i][left] = 1;
         matrix[i][right] = 1;
      }

      System.out.print("   ");

      for (int i = 0; i < vertices.length; i++) { O(n^2) (nested for-loop)
         System.out.print(vertices[i] + " ");
      }
      System.out.println();

      for (int i = 0; i < n; i++) {

         System.out.print(vertices[i] + "| ");

         for (int j = 0; j < n; j++) {
            System.out.print(matrix[i][j] + " ");
         }

         System.out.println();
      }

   }


   public static void main(String[] args) {
```

Question 6

Time Complexity: O(m+n^2)

- Parsing the input is O(m), processing the array and creating the matrix is O(n), printing the matrix is O(n^2).
- Thus, O(m+n+n+n^2), chip off the slowest growing we get O(m+n^2).

Space Complexity: O(n^2)

- n x n matrix = O(n^2)

```
        System.out.println("Question 6: ");
        System.out.println();

        String input = "[(I,2),(A,5),(E,4),(F,2),(T,2),(S,3)]"; O(1)


        System.out.println("INPUT: " + input);

        System.out.println("_____");

        System.out.println("OUTPUT: ");

        graph(input);
    }


}
```