Graphs

1. Creating Graph using Matrix

```
public class Graphs {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int vertex = sc.nextInt();
      int edge = sc.nextInt();
      int[][] adjMatrix = new int[vertex][vertex];
      for (int i = 0; i < edge; i++) {
        int v1 = sc.nextInt();
        int v2 = sc.nextInt();
        adjMatrix[v1][v2] = 1;
        adjMatrix[v2][v1] = 1;
      }
      printMatrix(adjMatrix, vertex);
   }

   static void printMatrix(int[][] adjMatrix, int n) {
      for (int[] matrix : adjMatrix) {
        for (int j = 0; j < n; j++) {
          System.out.print(matrix[j] + " ");
        }
      }
   }
}
```

2. Depth First Traversal

```
public class DFSTraversal {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int n = sc.nextInt();
      int e = sc.nextInt();
      int[][] adjMatrix = new int[n][n];
      for (int i = 0; i < e; i++) {
        int v1 = sc.nextInt();
        int v2 = sc.nextInt();
        adjMatrix[v1][v2] = 1;
        adjMatrix[v2][v1] = 1;
      }
      dfsTraversal(adjMatrix);
   }

   static void dfsTraversal(int[][] adjMatrix) {
      boolean[] visited = new boolean[adjMatrix.length];
      dfsTraversal(adjMatrix, 0, visited);
   }

   static void dfsTraversal(int[][] adjMatrix, int currentVertex,
boolean[] visited) {
      visited[currentVertex] = true;
      System.out.print(currentVertex + " ");
      for (int i = 0; i < adjMatrix.length; i++) {
        if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
          // means i is neighbour of current Vertex
```

```
            dfsTraversal(adjMatrix, i, visited);
        }
      }
   }
}
```

3. Breadth First Traversal

```
public class BFSTraversal {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int n = sc.nextInt();
      int e = sc.nextInt();
      int[][] adjMatrix = new int[n][n];
      for (int i = 0; i < e; i++) {
        int v1 = sc.nextInt();
        int v2 = sc.nextInt();
        adjMatrix[v1][v2] = 1;
        adjMatrix[v2][v1] = 1;
      }
      bfsTraversal(adjMatrix);
   }

   static void bfsTraversal(int[][] adjMatrix) {
      Queue<Integer> pendingVertices = new LinkedList<>();
      boolean[] visited = new boolean[adjMatrix.length];
      visited[0] = true;
      pendingVertices.add(0);
      while (!pendingVertices.isEmpty()) {
        int currentVertex = pendingVertices.poll();
        System.out.print(currentVertex + " ");
        for (int i = 0; i < adjMatrix.length; i++) {
          if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
             pendingVertices.add(i);
             visited[i] = true;
          }
        }
      }
   }
}
```

4. Has Path from source to destination

```
public class HasPath {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      int vert = scanner.nextInt();
      int edge = scanner.nextInt();

      int[][] adjMatrix = new int[vert][vert];
      for (int i = 0; i < edge; i++) {
        int vertex1 = scanner.nextInt();
        int vertex2 = scanner.nextInt();
        adjMatrix[vertex1][vertex2] = 1;
        adjMatrix[vertex2][vertex1] = 1;
      }
```

```
        int source = scanner.nextInt();
        int destination = scanner.nextInt();
        System.out.println(hasPathBFS(adjMatrix, source, destination));
    }

    // using BFS
    public static boolean hasPathBFS(int[][] adjMatrix, int source, int
destination) {
        boolean[] visited = new boolean[adjMatrix.length];
        Queue<Integer> pending = new LinkedList<>();
        pending.add(source);
        visited[source] = true;
        while (!pending.isEmpty()) {
          int currentEle = pending.poll();
          for (int i = 0; i < adjMatrix.length; i++) {
            if (adjMatrix[currentEle][i] == 1 && !visited[i]) {
               pending.add(i);
               visited[i] = true;
               if (i == destination)
                 return true;
            }
          }
        }
        return false;
    }

    // using DFS
    public static boolean hasPathDFS(int[][] adjMatrix, int source, int
destination) {
        boolean[] visited = new boolean[adjMatrix.length];
        return hasPathDFSHelper(adjMatrix, source, destination, visited);
    }

    public static boolean hasPathDFSHelper(int[][] adjMatrix, int source,
int destination, boolean[] visited) {
        visited[source] = true;
        for (int i = 0; i < adjMatrix.length; i++) {
          if (adjMatrix[source][i] == 1 && !visited[source]) {
            if (i == destination)
               return true;
            hasPathDFSHelper(adjMatrix, i, destination, visited);
          }
        }
        return false;
    }
}
```

5. BFS DFS For Disconnected Graph

```
public class BFS_DFS_For_Disconnected_Graph {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int vertex = scanner.nextInt();
        int edge = scanner.nextInt();
        int[][] adjMatrix = new int[vertex][vertex];
        for (int i = 0; i < edge; i++) {
```

```
        int v1 = scanner.nextInt();
        int v2 = scanner.nextInt();
        adjMatrix[v1][v2] = 1;
        adjMatrix[v2][v1] = 1;
      }
      bfs(adjMatrix);
      System.out.println("\nDFS traversal");
      dfs(adjMatrix);
   }

   static void bfs(int[][] adjMatrix) {
      boolean[] visited = new boolean[adjMatrix.length];
      for (int i = 0; i < adjMatrix.length; i++) {
        if (!visited[i])
          bfsHelper(adjMatrix, i, visited);
      }
   }

   static void bfsHelper(int[][] adjMatrix, int source, boolean[] visited)
{
      Queue<Integer> pending = new LinkedList<>();
      pending.add(source);
      visited[source] = true;
      while (!pending.isEmpty()) {
        int current = pending.poll();
        System.out.print(current + " ");
        for (int i = 0; i < adjMatrix.length; i++) {
          if (adjMatrix[current][i] == 1 && !visited[i]) {
             pending.add(i);
             visited[i] = true;
          }
        }
      }
   }

   static void dfs(int[][] ajdMatrix) {
      boolean[] visited = new boolean[ajdMatrix.length];
      for (int i = 0; i < ajdMatrix.length; i++) {
        if (!visited[i]) {
          dfsHelper(ajdMatrix, i, visited);
          System.out.println();
        }
      }
   }

   static void dfsHelper(int[][] adjMatrix, int currentVertex, boolean[]
visited) {
      visited[currentVertex] = true;
      System.out.print(currentVertex + " ");
      for (int i = 0; i < adjMatrix.length; i++) {
        if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
          dfsHelper(adjMatrix, i, visited);
        }
      }
   }
}
```

6. Get Path DFS and BFS

```
public static ArrayList<Integer> getPathDFS(int[][] adjMatrix, int start,
int end) {
    boolean[] visited = new boolean[adjMatrix.length];
    ArrayList<Integer> path = new ArrayList<>();
    if (getPathDFSHelper(adjMatrix, start, end, visited, path)) {
      return path;
    }
    else {
      return new ArrayList<>();
    }
  }

  public static boolean getPathDFSHelper(
         int[][] adjMatrix, int start, int end, boolean[] visited,
         ArrayList<Integer> path) {
    visited[start] = true;
    path.add(start);
    if (start == end)
      return true;
    for (int i = 0; i < adjMatrix.length; i++) {
      if (adjMatrix[start][i] == 1 && !visited[i]) {
       if (getPathDFSHelper(adjMatrix, i, end, visited, path))
         return true;
      }
    }
    path.removeLast();
    return false;
  }
}
```

```
public static ArrayList<Integer> getPathBFS(int[][] adjMatrix, int start,
int end) {
Queue<Integer> pendingVertices = new LinkedList<>();
HashMap<Integer, Integer> map = new HashMap<>();
boolean[] visited = new boolean[adjMatrix.length];
visited[start] = true;
pendingVertices.add(start);
map.put(start, -1);
boolean pathFount = false;
while (!pendingVertices.isEmpty()) {
  int currentVertex = pendingVertices.poll();
  for (int i = 0; i < adjMatrix.length; i++) {
    if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
      pendingVertices.add(i);
      visited[i] = true;
      map.put(i, currentVertex);
      if (i == end) {
       pathFount = true;
       break; // path found
      }
    }
  }
}
if (pathFount) {
```

```
  ArrayList<Integer> path = new ArrayList<>();
  int currentVertex = end;
  while (currentVertex != -1) {
    path.add(currentVertex);
    currentVertex = map.get(currentVertex);
  }
  return path;
}
else
  return null;
 }
```