Objective(s):

a. To practice representing undirected, weighted graphs.

b. To implement and understand a shortest path algorithm.

save Lab12b_Graph2_XXYYYY.java to package Lab12**ab** (You won't need sub package pack today)

Task 1: Implementing Dijkstra's Algorithm

Dijkstra's algorithm finds the shortest path from a source node to all other nodes in a graph. It works by expanding from the current node, u, to all its unvisited adjacent nodes, v. If the path to v through u is shorter than any previously known path, the algorithm updates the distance and predecessor arrays (dist and prev) for v. An updated distance for v is then added to a priority queue to ensure the next node chosen is always the one with the shortest known distance.

Given the adjacency matrix below, complete the q4_dijkstra_pq() and q4_extractPath() methods. The q4_extractPath() method should trace the path from a destination back to the source using the prev array.

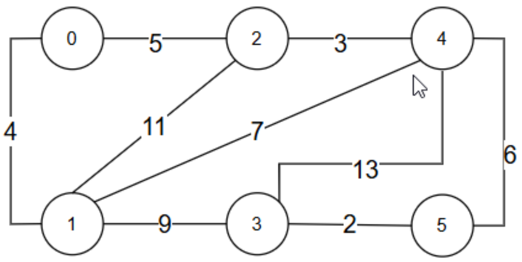
```
static void q4() {
  int [][] q4_distanceBetween = { {   0,   4,    5,  INF,  INF,  INF},
                                  {   4,   0,   11,    9,    7,  INF},
                                  {   5,  11,    0,  INF,    3,  INF},
                                  { INF,    9, INF,    0,   13,    2},
                                  { INF,    7,   3,   13,    0,    6},
                                  { INF,  INF, INF,    2,    6,    0} };
  int A, B, C, D, E, F; A = 0; B = 1; C = 2; D = 3; E = 4; F = 5;
  System.out.println("djikstra from A");
  q4_dijkstra_pq(q4_distanceBetween, A);
      // exploring 0 [0, 4, 5, 2147483647, 2147483647, 2147483647]
      // exploring 1 [0, 4, 5, 13, 11, 2147483647]
      // exploring 2 [0, 4, 5, 13, 8, 2147483647]
      // exploring 4 [0, 4, 5, 13, 8, 14]
      // exploring 3 [0, 4, 5, 13, 8, 14]
      // exploring 5 [0, 4, 5, 13, 8, 14]
      // prev= [-1, 0, 0, 1, 2, 4]
      // 0->2->4
}
```

Task 2: Weighted undirected graph and Finding the Longest Shortest Path

Modify your Dijkstra adjacencyMatrix implementation.

Complete q5b_dijstra_adjaList.

q5c_furthest_distance(dist) called in q5b() returns {city, distance} representing the city id and the maximum time required to reach the city from the starting source.



```
static void q5_call_dijkstra_adjacencyList() {
    // int [][] adjacencyMatrix = { {   0,    4,     5,  INF,  INF,  INF},
    //                              {   4,    0,   11,     9,     7,  INF},
    //                              {   5,   11,     0,  INF,     3,  INF},
    //                              { INF,     9, INF,     0,    13,     2},
    //                              { INF,     7,    3,   13,     0,     6},
    //                              { INF,  INF, INF,     2,     6,     0} };
    List<int[]> edges = Arrays.asList(new int[][]{
                                                {0,1,4}, {0,2,5},
                                                {1,2,11}, {1,3,9}, {1,4,7},
                                                {2,4,3},
                                                {3,4,13}, {3,5,2},
                                                {4,5,6}});
    int A, B, C, D, E, F; A = 0; B = 1; C = 2; D = 3; E = 4; F = 5;
    int numVertices = 6;  // else loop through both edge[0] and edge[1]
                          //  from edges to find maxId
                          // -> numVertices = maxId + 1;
    List<List<int[]>> graph = new ArrayList<>();
    for (int i = 0; i < numVertices; i++)
        graph.add(new ArrayList<>());     // List of <List of weighted edges>

    for (int[] edge: edges) {
        int i = edge[0];
        int j = edge[1];
        int weight = edge[2];
        graph.get(i).add(new int[]{weight, j});
        graph.get(j).add(new int[]{weight, i});
    }

    int startNode = C;  // 2
    q5b_dijstra_adjaList(graph, startNode);
        // ???
        // ???
        // prev= ???
        // Furthest city is 3, distance= 11
}
```

Submission: Lab12b_Graph2_XXYYYY.java where. XX are the first two digit of your student id and YYYY are the last four.

Due date: TBA