FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

GROUP PROJECT

OPTIMAL SCHEDULING OF DRONE DELIVERIES IN A SMART CITY

COURSE NAME	**: DESIGN AND ANALYSIS OF ALGORITHM**
CODE	**: CSC4202**
GROUP	**: 6**
LECTURER NAME	**: DR. NUR ARZILAWATI BINTI MD YUNUS**

GROUP MEMBERS :

NAME	MATRIC
MUHAMMAD FARHAN BIN HASRAT NAZARUDIN	210924
MUHAMMAD ZAHIN BIN MAHAT	210270
MUHAMMAD NURI BIN MOHD ALI	205768

Code

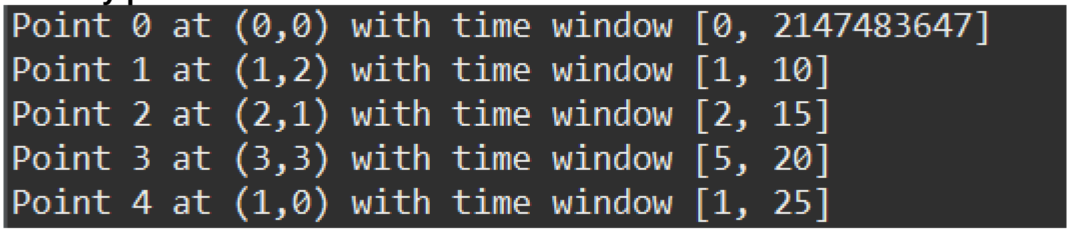
```
package Project;
import java.util.*;
public class dynamic {
    static final int INF = Integer.MAX_VALUE;

    static class DeliveryPoint {
        int id, x, y, startTime, endTime;

        public DeliveryPoint(int id, int x, int y, int startTime, int endTime) {
            this.id = id;
            this.x = x;
            this.y = y;
            this.startTime = startTime;
            this.endTime = endTime;
        }

        @Override
        public String toString() {
            return String.format("Point %d at (%d,%d) with time window [%d,
%d]", id, x, y, startTime, endTime);
        }
    }

    static int distance(DeliveryPoint a, DeliveryPoint b) {
        return Math.abs(a.x - b.x) + Math.abs(a.y - b.y);
    }

    public static Result findOptimalRoute(List<DeliveryPoint> points, int
base, int maxVisits) {
        int n = points.size();
        int[][] dist = new int[n][n];

        System.out.println("Calculating distances between points:");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                dist[i][j] = distance(points.get(i), points.get(j));
                System.out.printf("Distance from %s to %s: %d\n", points.get(i),
points.get(j), dist[i][j]);
            }
        }
        int[][] dp = new int[n][1 << n];
        int[][] prev = new int[n][1 << n];

        for (int[] row : dp) {
            Arrays.fill(row, INF);
        }
        for (int[] row : prev) {
            Arrays.fill(row, -1);
        }

        dp[base][1 << base] = 0;
        System.out.printf("Initialized dp[%d][%d] = %d\n", base, 1 << base,
0);

        for (int mask = 0; mask < (1 << n); mask++) {
            for (int u = 0; u < n; u++) {
                if ((mask & (1 << u)) == 0) continue;
                for (int v = 0; v < n; v++) {
                    if ((mask & (1 << v)) != 0) continue;
                    int newMask = mask | (1 << v);
                    if (dp[u][mask] != INF && points.get(v).startTime <=
dp[u][mask] + dist[u][v] &&
                        dp[u][mask] + dist[u][v] <= points.get(v).endTime) {
                        if (dp[v][newMask] > dp[u][mask] + dist[u][v]) {
                            dp[v][newMask] = dp[u][mask] + dist[u][v];
                            prev[v][newMask] = u;
                            System.out.printf("Updated dp[%d][%d] = %d (from
dp[%d][%d] + distance %d)\n",
                                              v, newMask, dp[v][newMask], u, mask,
dist[u][v]);
                        }
                    }
                }
            }
        }

        int minDistance = INF;
        int endNode = -1;
        for (int u = 0; u < n; u++) {
            if (dp[u][(1 << n) - 1] != INF && dp[u][(1 << n) - 1] + dist[u][base] <
minDistance) {
                minDistance = dp[u][(1 << n) - 1] + dist[u][base];
                endNode = u;
```

```java
            }
        }

        List<Integer> route = new ArrayList<>();
        if (endNode != -1) {
            int mask = (1 << n) - 1;
            while (endNode != -1) {
                route.add(endNode);
                int temp = endNode;
                endNode = prev[endNode][mask];
                mask ^= (1 << temp);
            }
            Collections.reverse(route);
        }

        return new Result(minDistance, route);
    }

    public static void main(String[] args) {
        List<DeliveryPoint> points = new ArrayList<>();
        points.add(new DeliveryPoint(0, 0, 0, 0, INF));  // Base
        points.add(new DeliveryPoint(1, 1, 2, 1, 10));
        points.add(new DeliveryPoint(2, 2, 1, 2, 15));
        points.add(new DeliveryPoint(3, 3, 3, 5, 20));
        points.add(new DeliveryPoint(4, 1, 0, 1, 25));

        int base = 0;
        int maxVisits = 3;

        System.out.println("Delivery Points:");
        for (DeliveryPoint point : points) {
            System.out.println(point);
        }

        Result result = findOptimalRoute(points, base, maxVisits);
        System.out.println("\nOptimal distance: " + result.minDistance);
        System.out.print("Optimal route: ");
        for (int point : result.route) {
            System.out.print(point + " ");
        }
        System.out.println();
    }
```

```
  static class Result {
      int minDistance;
      List<Integer> route;

      public Result(int minDistance, List<Integer> route) {
          this.minDistance = minDistance;
          this.route = route;
      }
  }
}
```

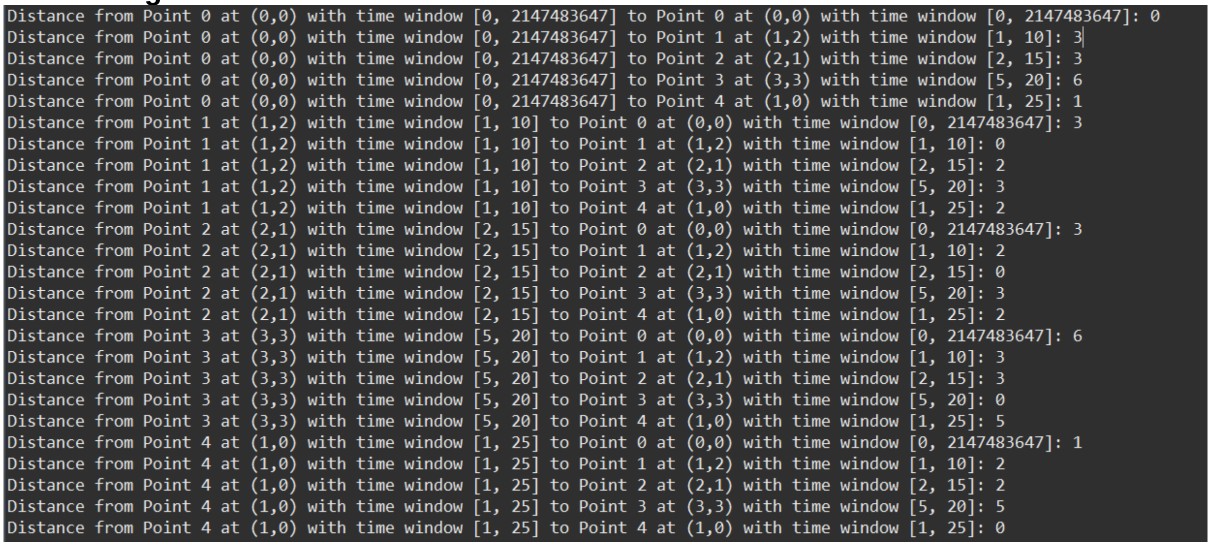
Output

Delivery points:



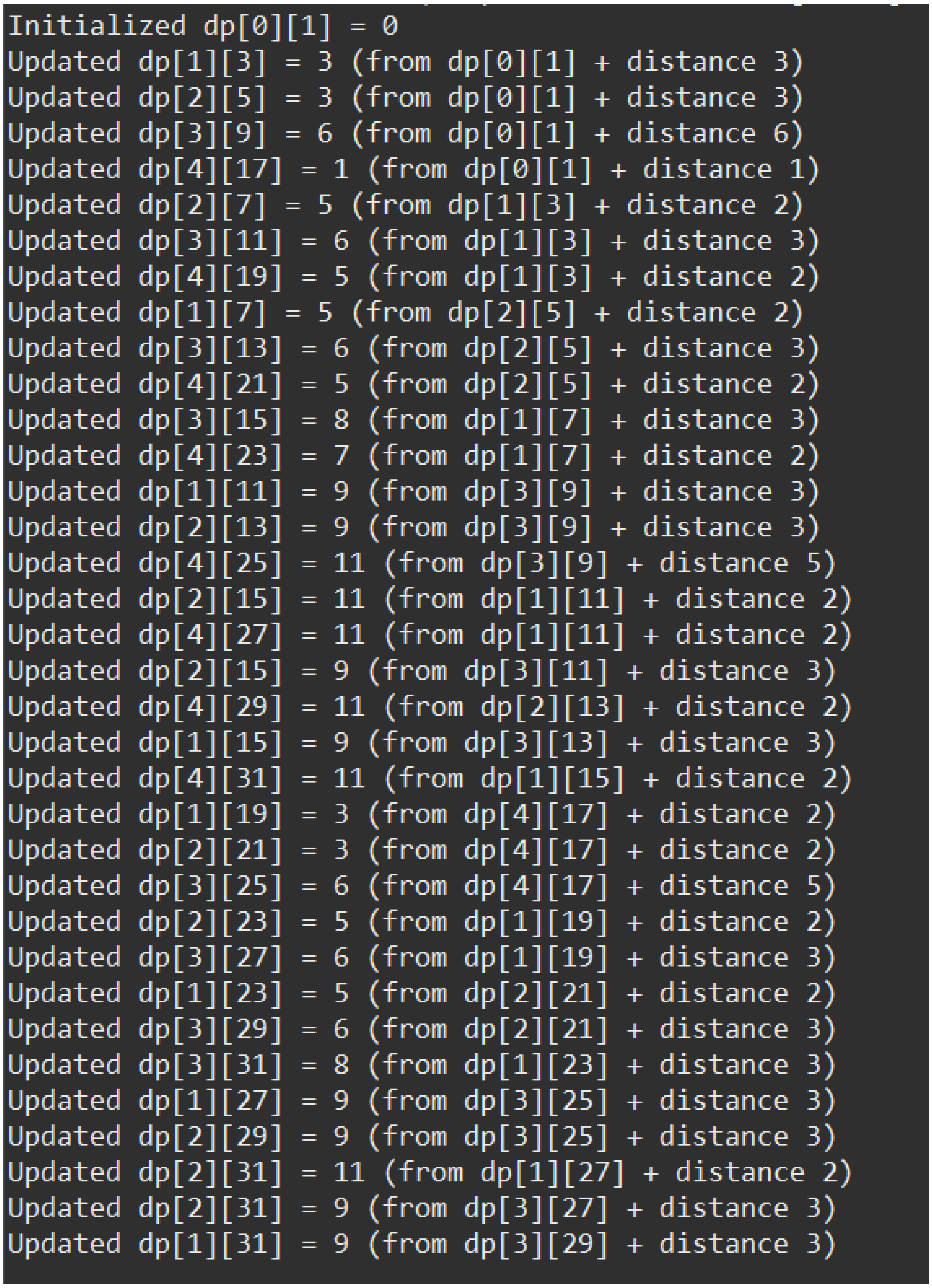
```
Point 0 at (0,0) with time window [0, 2147483647]
Point 1 at (1,2) with time window [1, 10]
Point 2 at (2,1) with time window [2, 15]
Point 3 at (3,3) with time window [5, 20]
Point 4 at (1,0) with time window [1, 25]
```

Calculating Distance Between Point:



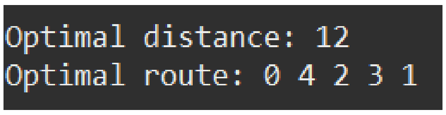
```
Distance from Point 0 at (0,0) with time window [0, 2147483647] to Point 0 at (0,0) with time window [0, 2147483647]: 0
Distance from Point 0 at (0,0) with time window [0, 2147483647] to Point 1 at (1,2) with time window [1, 10]: 3
Distance from Point 0 at (0,0) with time window [0, 2147483647] to Point 2 at (2,1) with time window [2, 15]: 3
Distance from Point 0 at (0,0) with time window [0, 2147483647] to Point 3 at (3,3) with time window [5, 20]: 6
Distance from Point 0 at (0,0) with time window [0, 2147483647] to Point 4 at (1,0) with time window [1, 25]: 1
Distance from Point 1 at (1,2) with time window [1, 10] to Point 0 at (0,0) with time window [0, 2147483647]: 3
Distance from Point 1 at (1,2) with time window [1, 10] to Point 1 at (1,2) with time window [1, 10]: 0
Distance from Point 1 at (1,2) with time window [1, 10] to Point 2 at (2,1) with time window [2, 15]: 2
Distance from Point 1 at (1,2) with time window [1, 10] to Point 3 at (3,3) with time window [5, 20]: 3
Distance from Point 1 at (1,2) with time window [1, 10] to Point 4 at (1,0) with time window [1, 25]: 2
Distance from Point 2 at (2,1) with time window [2, 15] to Point 0 at (0,0) with time window [0, 2147483647]: 3
Distance from Point 2 at (2,1) with time window [2, 15] to Point 1 at (1,2) with time window [1, 10]: 2
Distance from Point 2 at (2,1) with time window [2, 15] to Point 2 at (2,1) with time window [2, 15]: 0
Distance from Point 2 at (2,1) with time window [2, 15] to Point 3 at (3,3) with time window [5, 20]: 3
Distance from Point 2 at (2,1) with time window [2, 15] to Point 4 at (1,0) with time window [1, 25]: 2
Distance from Point 3 at (3,3) with time window [5, 20] to Point 0 at (0,0) with time window [0, 2147483647]: 6
Distance from Point 3 at (3,3) with time window [5, 20] to Point 1 at (1,2) with time window [1, 10]: 3
Distance from Point 3 at (3,3) with time window [5, 20] to Point 2 at (2,1) with time window [2, 15]: 3
Distance from Point 3 at (3,3) with time window [5, 20] to Point 3 at (3,3) with time window [5, 20]: 0
Distance from Point 3 at (3,3) with time window [5, 20] to Point 4 at (1,0) with time window [1, 25]: 5
Distance from Point 4 at (1,0) with time window [1, 25] to Point 0 at (0,0) with time window [0, 2147483647]: 1
Distance from Point 4 at (1,0) with time window [1, 25] to Point 1 at (1,2) with time window [1, 10]: 2
Distance from Point 4 at (1,0) with time window [1, 25] to Point 2 at (2,1) with time window [2, 15]: 2
Distance from Point 4 at (1,0) with time window [1, 25] to Point 3 at (3,3) with time window [5, 20]: 5
Distance from Point 4 at (1,0) with time window [1, 25] to Point 4 at (1,0) with time window [1, 25]: 0
```

Progress Update :



```
Initialized dp[0][1] = 0
Updated dp[1][3] = 3 (from dp[0][1] + distance 3)
Updated dp[2][5] = 3 (from dp[0][1] + distance 3)
Updated dp[3][9] = 6 (from dp[0][1] + distance 6)
Updated dp[4][17] = 1 (from dp[0][1] + distance 1)
Updated dp[2][7] = 5 (from dp[1][3] + distance 2)
Updated dp[3][11] = 6 (from dp[1][3] + distance 3)
Updated dp[4][19] = 5 (from dp[1][3] + distance 2)
Updated dp[1][7] = 5 (from dp[2][5] + distance 2)
Updated dp[3][13] = 6 (from dp[2][5] + distance 3)
Updated dp[4][21] = 5 (from dp[2][5] + distance 2)
Updated dp[3][15] = 8 (from dp[1][7] + distance 3)
Updated dp[4][23] = 7 (from dp[1][7] + distance 2)
Updated dp[1][11] = 9 (from dp[3][9] + distance 3)
Updated dp[2][13] = 9 (from dp[3][9] + distance 3)
Updated dp[4][25] = 11 (from dp[3][9] + distance 5)
Updated dp[2][15] = 11 (from dp[1][11] + distance 2)
Updated dp[4][27] = 11 (from dp[1][11] + distance 2)
Updated dp[2][15] = 9 (from dp[3][11] + distance 3)
Updated dp[4][29] = 11 (from dp[2][13] + distance 2)
Updated dp[1][15] = 9 (from dp[3][13] + distance 3)
Updated dp[4][31] = 11 (from dp[1][15] + distance 2)
Updated dp[1][19] = 3 (from dp[4][17] + distance 2)
Updated dp[2][21] = 3 (from dp[4][17] + distance 2)
Updated dp[3][25] = 6 (from dp[4][17] + distance 5)
Updated dp[2][23] = 5 (from dp[1][19] + distance 2)
Updated dp[3][27] = 6 (from dp[1][19] + distance 3)
Updated dp[1][23] = 5 (from dp[2][21] + distance 2)
Updated dp[3][29] = 6 (from dp[2][21] + distance 3)
Updated dp[3][31] = 8 (from dp[1][23] + distance 3)
Updated dp[1][27] = 9 (from dp[3][25] + distance 3)
Updated dp[2][29] = 9 (from dp[3][25] + distance 3)
Updated dp[2][31] = 11 (from dp[1][27] + distance 2)
Updated dp[2][31] = 9 (from dp[3][27] + distance 3)
Updated dp[1][31] = 9 (from dp[3][29] + distance 3)
```

Result:



```
Optimal distance: 12
Optimal route: 0 4 2 3 1
```