A* ALGORITHM

```
import java.util.*;
public class AStarAlgorithm {
    static class Node {
        int x, y;
        int g, h;
        Node parent;
        Node(int x, int y) {
            this.x = x;
            this.y = y;
        }
        int f() {
            return g + h;
        }
    }
    static int[][] grid = {
            {0, 0, 0, 0, 0},
            {0, 1, 1, 1, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 1, 1, 1},
            {0, 0, 0, 0, 0}
    };
    static int start_x = 0, start_y = 0;
    static int end_x = 4, end_y = 4;
    static int[] dx = {1, 0, -1, 0};
    static int[] dy = {0, 1, 0, -1};
    static boolean isValid(int x, int y) {
        return x >= 0 && x < grid.length && y >= 0 && y < grid[0].length;
    }
    static int heuristic(int x, int y) {
        return Math.abs(end_x - x) + Math.abs(end_y - y);
    }
    static void aStar() {
        PriorityQueue<Node> pq = new
PriorityQueue<>(Comparator.comparingInt(o -> o.f()));
        boolean[][] visited = new boolean[grid.length][grid[0].length];
        Node start = new Node(start_x, start_y);
        start.g = 0;
        start.h = heuristic(start_x, start_y);
        pq.offer(start);
        while (!pq.isEmpty()) {
            Node current = pq.poll();
            visited[current.x][current.y] = true;
            if (current.x == end_x && current.y == end_y) {
                // Path found, reconstruct path
                LinkedList<Node> path = new LinkedList<>();
                while (current != null) {
                    path.addFirst(current);
                    current = current.parent;
                }
                System.out.println("Path: " + path);
            return;
          }
          for (int i = 0; i < 4; i++) {
            int next_x = current.x + dx[i];
            int next_y = current.y + dy[i];
            if (isValid(next_x, next_y) && grid[next_x][next_y] == 0 &&
!visited[next_x][next_y]) {
              Node neighbor = new Node(next_x, next_y);
              neighbor.g = current.g + 1;
              neighbor.h = heuristic(next_x, next_y);
              neighbor.parent = current;
              pq.offer(neighbor);
            }
          }
        }
        System.out.println("No path found!");
      }
      public static void main(String[] args) {
        aStar();
      }
    }
```

OUTPUT:

```
Path: [Node{x=0, y=0}, Node{x=1, y=0}, Node{x=2,
y=0}, Node{x=3, y=0}, Node{x=4, y=0}, Node{x=4, y=1},
Node{x=4, y=2}, Node{x=4, y=3}, Node{x=4, y=4}]
```