```java
import java.util.*;

class Node {
    String id;
    int heuristicValue;
    List<Edge> edges = new ArrayList<>();

    Node(String id, int heuristicValue) {
        this.id = id;
        this.heuristicValue = heuristicValue;
    }

    void addEdge(Node node, int weight) {
        edges.add(new Edge(node, weight));
    }

    @Override
    public String toString() {
        return id;
    }
}

class Edge {
    Node node;
    int weight;

    Edge(Node node, int weight) {
        this.node = node;
        this.weight = weight;
    }
}

class AStar {
    static class NodeComparator implements Comparator<Node> {
        Map<Node, Integer> gScore;

        NodeComparator(Map<Node, Integer> gScore) {
            this.gScore = gScore;
        }

        @Override
        public int compare(Node node1, Node node2) {
            return (gScore.get(node1) + node1.heuristicValue) - (gScore.get(node2) +
                    node2.heuristicValue);
        }
    }

    static List<Node> aStarAlgo(Node startNode, Node stopNode) {
        Set<Node> openSet = new HashSet<>();
        Set<Node> closedSet = new HashSet<>();
        Map<Node, Integer> gScore = new HashMap<>();
        Map<Node, Node> parents = new HashMap<>();

        gScore.put(startNode, 0);
        parents.put(startNode, startNode);
        openSet.add(startNode);

        while (!openSet.isEmpty()) {
            Node n = Collections.min(openSet, new NodeComparator(gScore));

            if (n == stopNode) {
                List<Node> path = new ArrayList<>();
                while (parents.get(n) != n) {
                    path.add(n);
                    n = parents.get(n);
                }
                path.add(startNode);
                Collections.reverse(path);
                System.out.println("Path found: " + path);
                return path;
            }

            openSet.remove(n);
            closedSet.add(n);

            for (Edge edge : n.edges) {
                Node m = edge.node;
                int weight = edge.weight;

                if (!openSet.contains(m) && !closedSet.contains(m)) {
                    openSet.add(m);
                    parents.put(m, n);
                    gScore.put(m, gScore.get(n) + weight);
                } else if (gScore.get(m) > gScore.get(n) + weight) {
                    gScore.put(m, gScore.get(n) + weight);
                    parents.put(m, n);

                    if (closedSet.contains(m)) {
                        closedSet.remove(m);
                        openSet.add(m);
                    }
                }
            }
        }

        System.out.println("Path does not exist!");
        return null;
    }
}

public class Main {
    public static void main(String[] args) {
        Node A = new Node("A", 11);
        Node B = new Node("B", 6);
        Node C = new Node("C", 5);
        Node D = new Node("D", 7);
        Node E = new Node("E", 3);
        Node F = new Node("F", 6);
        Node G = new Node("G", 5);
        Node H = new Node("H", 3);
        Node I = new Node("I", 1);
        Node J = new Node("J", 0);

        A.addEdge(B, 6);
        A.addEdge(F, 3);
        B.addEdge(A, 6);
        B.addEdge(C, 3);
        B.addEdge(D, 2);
        C.addEdge(B, 3);
        C.addEdge(D, 1);
        C.addEdge(E, 5);
        D.addEdge(B, 2);
        D.addEdge(C, 1);
        D.addEdge(E, 8);
        E.addEdge(C, 5);
        E.addEdge(D, 8);
        E.addEdge(I, 5);
        E.addEdge(J, 5);
        F.addEdge(A, 3);
        F.addEdge(G, 1);
        F.addEdge(H, 7);
        G.addEdge(F, 1);
        G.addEdge(I, 3);
        H.addEdge(F, 7);
        H.addEdge(I, 2);
        I.addEdge(E, 5);
        I.addEdge(G, 3);
        I.addEdge(H, 2);
        I.addEdge(J, 3);

        AStar.aStarAlgo(A, J);
    }
}
```

Output

```
Path found: [A, F, G, I, J]
```