```java
private static Graph kruskal(ArrayList<Edge> sortedGraphEdges, ArrayList<Vertex>
verticesGraph) {
    Graph A = new Graph();
    Vertex[][] S = new Vertex[verticesGraph.size()][verticesGraph.size()];
    int nA = 0;
    addVerticesToSack(verticesGraph, S, nA);
    int i = 0;
    while (nA < verticesGraph.size() - 1) {
        Edge edge = sortedGraphEdges.get(i);
        int Sp = locationVertexInS(edge.getOrigin(), S);
        int Sq = locationVertexInS(edge.getDestiny(), S);
        if (Sp != Sq && (Sp != -1 && Sq != -1)) {
            A.addEdge(edge);
            moveVertices(S, Sp, Sq);
            nA++;
        }
        i++;
    }
    return A;
}

private static void addVerticesToSack(ArrayList<Vertex> verticesGraph, Vertex[][] S,
int nA) {
    for (Vertex vertex : verticesGraph) {
        S[0][nA] = vertex;
        nA++;
    }
}

private static int locationVertexInS(Vertex vertex, Vertex[][] S) {
    for (int i = 0; i < S[0].length; i++) {
        for (Vertex[] vertices : S) {
            if (vertices[i] != null && vertices[i].equals(vertex)) {
                return i;
            }
        }
    }
    return -1;
}

private static void moveVertices(Vertex[][] S, int Sp, int Sq) {
    int i = 0;
    int j = 0;

    while (S[j][Sp] != null) {
        j++;
        if (j == S.length){
            break;
        }
    }

    while (S[i][Sq] != null && j < S.length) {
        S[j][Sp] = S[i][Sq];
        S[i][Sq] = null;
        i++;
        j++;

        if (j == S.length) {
            break;
        }

        while (S[j][Sp] != null) {
            j++;
            if (j == S.length) {
                break;
            }
        }
    }
}
private static Edge readLine(Scanner in) {
    if (in.hasNextLine()) {
        String line = in.nextLine();
        String[] parts = line.split(";");
        if (parts.length == 3) {
            String origin = parts[0];
            String destiny = parts[1];
            int cost = Integer.parseInt(parts[2]);
            return new Edge(new Vertex(origin), new Vertex(destiny), cost);
        }
    }
    return null;
}

private static ArrayList<Vertex> getVerticesGraph(ArrayList<Edge> graphEdges) {
    ArrayList<Vertex> verticesGraph = new ArrayList<>();
    for (Edge edge : graphEdges) {
        Vertex vertex1 = edge.getOrigin();
        Vertex vertex2 = edge.getDestiny();
        if (!verticesGraph.contains(vertex1)) {
            verticesGraph.add(vertex1);
        }
        if (!verticesGraph.contains(vertex2)) {
            verticesGraph.add(vertex2);
        }
    }
    return verticesGraph;
}
private static void bubbleSort(ArrayList<Edge> graphEdges) {
    int n = graphEdges.size();
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (graphEdges.get(j).getCost() > graphEdges.get(j + 1).getCost()) {

                Edge temp = graphEdges.get(j);
                graphEdges.set(j, graphEdges.get(j + 1));
                graphEdges.set(j + 1, temp);
            }
        }
    }
}
```