ESINF

Relatório

Projeto Integrador
Sprint1

João Gomes, 1210818
André Gonçalves, 1210804
Miguel Oliveira, 1200874
Manuel Silva, 1200585
Jorge Moreira, 1201458
Alexandre Vieira, 1211551

Turma 2DL

isep Instituto Superior de Engenharia do Porto

Índice

US301 ..... **3**

US302 ..... **5**

US303 ..... **6**

US304 ..... **8**

US305 ..... **9**

US301

Construir a rede de distribuição de cabazes a partir da informação fornecida nos ficheiros.

Análise de Complexidade:

```
public String addVertex(List<Local> list) {
    int numV = 0;
    for (Local l : list) {
        graph.addVertex(l);
        numV++;
    }
    return numV + "added";
}
```

Esta função tem de complexidade O(n).

```
public String addEdges(List<Length> list) {
    int numE = 0;
    for (Length l : list) {
        double dist = l.getLength();
        int key1 = checkKey(l.id1);
        int key2 = checkKey(l.id2);

        Local local = graph.vertex(key1);
        Local local1 = graph.vertex(key2);
        graph.addEdge(local, local1, dist);
        numE++;
    }
    return numE + "added";
}
```

Esta função tem de complexidade O(n).

```java
public int checkKey(String id) {
    int k = 0;
    boolean check = false;
    while (check == false) {
        Local local = graph.vertex(k);
        if (local.id.equals(id)) {
            check = true;
        } else k++;
    }
    return k;
}
```

Esta função tem de complexidade O(n).

```java
public MatrixGraph<Local, Double> returnGraph() {
    graph.resizeMatrix();
    return graph;
}
```

Esta função tem de complexidade O(1).

US302

Verificar se o grafo carregado é conexo e devolver o número mínimo de ligações necessário para nesta rede qualquer cliente/produtor conseguir contactar um qualquer outro.

Análise de Complexidade:

```
public boolean checkConnectivity(MatrixGraph<Local,Double> g) {
    return Algorithms.isConnected(g);

}
```

Esta função tem de complexidade $O(n^2)$.

US303

Definir os hubs da rede de distribuição, ou seja, encontrar as N empresas mais próximas de todos os pontos da rede (clientes e produtores agrícolas).

Análise de Complexidade:

```
public List<Empresa> getClosestHubs(MatrixGraph<Local, Double> matrixGraph) {


    LinkedList<Local> localList = new LinkedList<>();
    ArrayList<LinkedList<Local>> hubsList = new ArrayList<>();

    Double sum = 0.0;
    int i = 0;
    Double average = 0.0;
    Double total_sum = 0.0;
    double zero = 0.0;
    for (Local l1 : matrixGraph.vertices()) {
        if (l1.getType().contains("C") || l1.getType().contains("P")) {
        } else {
            for (Local l2 : matrixGraph.vertices()) {
                if (l2.getType().contains("E")) {

                } else {

                    sum = Algorithms.shortestPath(matrixGraph, l1, l2, Double::compare, Double::sum, zero, localList);
                    i++;
                    total_sum = total_sum + sum;
                    hubsList.add(localList);

                }

            }
            average = total_sum / i;
            Empresa futureHub = new Empresa(l1.getId(), l1.getLati(), l1.getLongi(), average, l1.getType());
            companyHubs.add(futureHub);
            i = 0;
        }

    }
```

Esta função tem de complexidade $O(n^2)$.

```java
public List<Hub> mostCentralHubs(Integer n, List<Empresa> companyHubs) {

    Collections.sort(companyHubs, new Comparator<Empresa>() {

        @Override
        public int compare(Empresa o1, Empresa o2) {
            return (o1.getCentralDistance().compareTo(o2.getCentralDistance()));
        }


    });
    for (int i = 0; i < n; i++) {
        Hub hubSorted = new Hub(companyHubs.get(i).getId(), companyHubs.get(i).getLati(),
        companyHubsSorted.add(hubSorted);
    }


    return companyHubsSorted;

}
```

Esta função tem de complexidade O(n).

US304

Para cada cliente (particular ou empresa) determinar o hub mais próximo.

Análise de Complexidade:

```java
public Map<Local, Hub> getNearestHub(MatrixGraph<Local, Double> matrixGraph, List<Hub> companyHubsSorted) throws IOException {

    LinkedList<Local> localList = new LinkedList<>();
    Double sum = 0.0;
    double zero = 0;

    for (Local l1 : matrixGraph.vertices()) {
        if (l1.getType().contains("C") || l1.getType().contains("P")) {
            for (Local l2 : matrixGraph.vertices()) {
                for (Hub h1 : companyHubsSorted) {
                    if (l2.getType().equals(h1.getType())) {
                        sum = Algorithms.shortestPath(matrixGraph, l1, l2, Double::compare, Double::sum, zero, localList);
                        Hub closestIndex = new Hub(l2.getId(), l2.getLati(), l2.getLongi(), sum, l2.getType());
                        listHubsClient.add(closestIndex);

                    }

                }
                Collections.sort(listHubsClient, new Comparator<Hub>() {

                    @Override
                    public int compare(Hub o1, Hub o2) {
                        return (o1.getCentralDistance().compareTo(o2.getCentralDistance()));
                    }
                });
            }
            Local client = new Local(l1.getId(), l1.getLati(), l1.getLongi(), l1.getType());
            getNearestHub.put(client, listHubsClient.get(0));
        }
    }
    return getNearestHub;
}
```

Esta função tem de complexidade $O(n^3)$.

US305

Determinar a rede que conecte todos os clientes e produtores agrícolas com uma distância total mínima.

Análise de Complexidade:

```
    public List<Local> getShortestPath(MatrixGraph<Local, Double> matrixGraph)
    {
        MatrixGraph<Local, Double> graph = new MatrixGraph<>(matrixGraph);
        for (Local l1 : graph.vertices())
        {
            if (l1.getType().contains("E"))
            {
                graph.removeVertex(l1);
            }
        }
       return Algorithms.minDistGraph(graph, Double::compare, Double::sum).vertices();
    }
}
```

Esta função tem de complexidade O(n).