ESINF

Relatório

Projeto Integrador
Sprint1/Sprint2

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

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).

US307

Importar a lista de cabazes.

Análise de Complexidade:

```java
public class FileReaderCabazes {
    public List<Cabaz> readCabazes(String filename) throws IOException {
        FileReader reader = new FileReader(filename);
        BufferedReader bufferedReader = new BufferedReader(reader);
        String line;
        String id = null;
        int k = 0;
        List<Cabaz> cabazes = new ArrayList<>();
        while ((line = bufferedReader.readLine()) != null) {
            if (k == 0) {
                String[] info = line.split( regex: ",");
                k = 1;
            } else {

                String[] info = line.split( regex: ",");
                if (info[0].charAt(0) == '\"' && info[0].charAt(info[0].length() - 1) == '\"') {
                    id = info[0].substring(1, info[0].length() - 1);
                } else id = info[0];
                if (filename.contains("small")) {

                    cabazes.add(new Cabaz(id, Integer.parseInt(info[1]), Double.parseDouble(info[2]), Double.parseDouble(i
                }
                else cabazes.add(new Cabaz(id, Integer.parseInt(info[1]), Double.parseDouble(info[2]), Double.parseDouble(

            }
        }
        return cabazes;
    }
}
```

Este método tem de complexidade O(n).

US308

Gerar lista de expedição diária.

Análise de Complexidade:

```
public List<Cabaz> CabazOnTheDay(int day, List<Cabaz> list) {

    List<Cabaz> res = new ArrayList<>();
    for (Cabaz c1 : list) {
        if (c1.getDay() == day && c1.getId().contains("C")) {
            res.add(c1);
        }

    }


    return res;


}
```

Este método tem de complexidade O(n).

```
public List<Cabaz> ProdByDayAndP(int day, List<Cabaz> res) {
    List<Cabaz> prodList = new ArrayList<>();
    for (Cabaz c1 : res) {

        if (c1.getDay() == day && c1.getId().contains("P")) {
            prodList.add(c1);


        }
    }
    return prodList;

}
```

Este método tem de complexidade O(n).

```java
public Map<String, Map<String, Double>> expeditionList(List<Cabaz> prodList, List<Cabaz> clientList) {


    Map<String, Map<String, Double>> fullMap = new HashMap<>();
    int i = 1;
    int count = 0;
    for (Cabaz p1 : clientList) {
        if (p1.getProd13() != null) {
            count = 1;
        } }

          // ------------------------------SMALL ---------------------------
    if (count == 0) {
        do {

            for (Cabaz c1 : clientList) {
                Map<String, Double> partialMap = new HashMap<>();
                Double aux = c1.getProdGeneric(i);


                for (Cabaz c2 : prodList) {

                    if (c1.getProdGeneric(i) == 0) {
                        partialMap.put(c2.getId(), 0.0);
                        fullMap.put("Cliente:" + c1.getId() + "   " + "Produto:" + i, partialMap);

                    } else {
                        Double aux2 = c2.getProdGeneric(i);
                        Double aux3 = aux;
                        aux = aux - aux2;
                        if (aux < 0) {
                            if (aux3 < 0) {
                                partialMap.put(c2.getId(), 0.0);
                                fullMap.put("Cliente:" + c1.getId() + "   " + "Produto:" + i, partialMap);

                            } else {
                                partialMap.put(c2.getId(), aux3);
                                fullMap.put("Cliente:" + c1.getId() + "   " + "Produto:" + i, partialMap);
                            }
```

Este método tem de complexidade O(n2).

US309

Gerar lista de expedição diária ordenada pela distância dos produtores ao hub de entrega do cliente.

Análise de Complexidade:

```
public Map<String,Double> expeditionListProximity(List<Cabaz> prodList, List<Cabaz> clientList, MatrixGraph<Local, Double> matrixGraph) throws IOException {
    Map<String, Map<String, Double>> fullMap = controller.expeditionList(prodList,clientList);
    Map<String, Map<String, Double>> fullMapToSort = new HashMap<>();
    Map<String,Double> testMap = new HashMap<>();
    List<Empresa> empresaList1 = defineHubController.getClosestHubs(matrixGraph);
    List<Hub> hubList = defineHubController.mostCentralHubs( n: 5, empresaList1);
    Map<Local, Hub> mapa = defineHubController.getNearestHub(matrixGraph, hubList);
    Map<String,Double> prodAndDistance = new HashMap<>();


    for(Map.Entry<Local, Hub> en: mapa.entrySet()) {
        if (en.getKey().getType().contains("C") || en.getKey().getType().contains("P") ) {
            prodAndDistance.put(en.getKey().getType(), en.getValue().getCentralDistance());
        }
    }
    for(Map.Entry<String, Map<String, Double>> eg: fullMap.entrySet()){
        for(Map.Entry<String,Double> ef: eg.getValue().entrySet()){
            for(Map.Entry<String,Double> ep : prodAndDistance.entrySet()){
                if(ef.getKey().equals(ep.getKey()) || eg.getKey().contains(ep.getKey())){
                    Map<String, Double> partialMapToSort = new HashMap<>();
                    partialMapToSort.put(ef.getKey(),ep.getValue());
                    fullMapToSort.put(eg.getKey(),partialMapToSort);
                    testMap.put(eg.getKey() + " " + ef.getKey() +" --->" + "Distance",ep.getValue());}
            }
        }

    }
    List<Map.Entry<String,Double>> entryList = new ArrayList<>(testMap.entrySet());
    entryList.sort(Map.Entry.comparingByValue());

    return testMap;
}
```

Este método tem de complexidade O(n3).

US310

Para uma lista de expedição diária gerar o percurso de entrega que minimiza a distância total percorrida.

Análise de complexidade:

```
public List<Cabaz> CabazesByDay(int day, List<Cabaz> res) {
    List<Cabaz> cabazList = new ArrayList<>();
    for (Cabaz c1 : res) {

        if (c1.getDay() == day) {
            cabazList.add(c1);

        }

    }
    return cabazList;

}
```

Este método tem de complexidade O(n).

```
public Set<Integer> getDays(List<Cabaz>c){

    Set<Integer> days = new HashSet<Integer>();
    for(Cabaz cabaz : c){
        days.add(cabaz.day);
    }

    return days;
}
```

Este método tem de complexidade O(n).

```java
public List<Cabaz> removeClients(List<Cabaz> cabazs)
{    List<Cabaz> cabazPE = new ArrayList<>(cabazs);
    for (Cabaz l1 : cabazs)
    {
        if (l1.id.contains("C"))
        {
            cabazPE.remove(l1);
        }
    }

    return cabazPE;
}
```

Este método tem de complexidade O(n).

```java
public MatrixGraph<Local, Double> getDayGraph(MatrixGraph<Local, Double> graphList, List<Cabaz>cabazList){

    MatrixGraph<Local,Double> graph=new MatrixGraph<>( directed: false, cabazList.size());

    for(Local v : graphList.vertices())
    {
        for(Cabaz c : cabazList)
        {
            if (v.getType().contains(c.id))
            {
                graph.addVertex(v);
                break;
            }
        }
    }
    return graph;
}
```

Este método tem de complexidade $O(n^2)$.

```java
public List<Local> getShortest(List<Cabaz> cabazes, MatrixGraph<Local, Double> matrixGraph)
{
    List<Local> rtnList = new ArrayList<>();

    MatrixGraph<Local,Double> graph=new MatrixGraph<>( directed: false, cabazes.size());

    graph=getDayGraph(matrixGraph, cabazes);

    return Algorithms.minDistGraph(graph, Double::compare, Double::sum).vertices();
}
```

Este método tem de complexidade O(1).

```java
public List<Empresa> getDistance(MatrixGraph<Local, Double> graph, List<Cabaz>cabazes) {

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

    Double total_sum = 0.0;
    for (Local l1 : graph.vertices())
    {
        if (l1.getType().contains("E") || l1.getType().contains("P"))
        {
            for (Local l2 : graph.vertices())
            {
                if (!l2.getType().contains("C"))
                {
                    total_sum += Algorithms.shortestPath(graph, l1, l2, Double::compare, Double::sum,
                    hubsList.add(localList);
                }
            }
            Empresa futureHub = new Empresa(l1.getId(), l1.getLati(), l1.getLongi(), total_sum, l1.getType
            companyHubs.add(futureHub);
        }
    }
    System.out.println("Soma Total: ");
    System.out.println(total_sum);
    System.out.println("Soma entre pontos: ");
    return companyHubs;
}
```

Este método tem de complexidade $O(n^2)$.

US311

```
public void statisticsPerCabaz(List<Cabaz> cabazList) {
    Map<String, List<Double>> results = new HashMap<>();
    int produtosSatisfeitos = 0, produtosParcSatisfeitos = 0, produtosNaoSatisfeitos = 0, i = 0, produtosNoCabaz = 0;
    System.out.println("Por Cabaz:\n");
    for (int day = 1; day < 6; day++) {
        Set<String> prodInvolved = new HashSet<>();
        List<Cabaz> clientList = expeditionListController.CabazOnTheDay(day, cabazList);
        List<Cabaz> prodList = expeditionListController.ProdByDayAndP(day, cabazList);
        Map<String, Map<String, Double>> expeditionList = expeditionListController.expeditionList(prodList, clientList);

        for (Cabaz c : clientList) {
            System.out.printf("ID: %s, Day: %d\n", c.id, c.day);

            Iterator<Map.Entry<String, Map<String, Double>>> it = expeditionList.entrySet().iterator();
            //Map.Entry<String, Map<String, Double>> entry = it.next();
            //System.out.println("\n" + entry.getKey() + ": " + c.getId());

            for (Map.Entry<String, Map<String, Double>> entry : expeditionList.entrySet()) {
                if (entry.getKey().contains(c.getId())) {
                    //System.out.println(entry.getKey() + "               :           " + c + "\n");

                    for (int j = 12; j > 0; j--) {
                        Iterator<Map.Entry<String, Map<String, Double>>> itR = expeditionList.entrySet().iterator();
                        String s3 = "Produto:", s4 = Integer.toString(j), s1 = "Cliente:", s2 = c.id;
                        String s = s1 + s2 + "  " + s3 + s4;
```

```
        if (c.getProdGeneric(j) != 0) {
            Iterator<Map.Entry<String, Map<String, Double>>> itEntry = expeditionList.entrySet().iterator();
            while (!(entry.getKey().equals(s)) && it.hasNext()) {
                //System.out.println("hiii");
                //System.out.println(entry.getKey());
                entry = it.next();
            }

            if (entry.getKey().equals(s)) {
                //System.out.println("u made it");
                Iterator<Map.Entry<String, Double>> it2 = entry.getValue().entrySet().iterator();
                Iterator<Map.Entry<String, Double>> it88 = itEntry.next().getValue().entrySet().iterator();
                Map.Entry<String, Double> entry2 = it2.next();
                Map.Entry<String, Double> biggestEntry2 = entry2;

                for (Map.Entry<String, Double> uiii : entry.getValue().entrySet()) {
                    if (biggestEntry2.getValue() < uiii.getValue()) {
                        biggestEntry2 = uiii;
                    }
                }

                if (biggestEntry2.getValue() >= c.getProdGeneric(j)) {
                    //System.out.println("\n" + entry2);
                    produtosSatisfeitos++;
                    produtosNoCabaz++;
                    prodInvolved.add(entry2.getKey());

                } else {
```

```
                    } else {
                        if (biggestEntry2.getValue() < c.getProdGeneric(j) && biggestEntry2.getValue() > 0) {
                            //System.out.println("\n" + entry2);
                            produtosParcSatisfeitos++;

                            prodInvolved.add(entry2.getKey());
                        }

                        if (biggestEntry2.getValue() == 0) {
                            produtosNaoSatisfeitos++;
                            produtosNoCabaz++;
                        }
                    }
                }
            }
        }

    }
}
System.out.printf("Número de produtos satisfeitos: %d\n", produtosSatisfeitos / 2);
System.out.printf("Número de produtos parcialmente satisfeitos: %d\n", produtosParcSatisfeitos / 2);
System.out.printf("Número de produtos não satisfeitos: %d\n", produtosNaoSatisfeitos / 2);
if (produtosNoCabaz == 0)
    produtosNoCabaz = 1;
double percentSatisfeitos = (((produtosSatisfeitos) + (produtosParcSatisfeitos) / 2) / produtosNoCabaz) * 50;
System.out.printf("Percentagem de Cabaz Satisfeito: %.2f\n", percentSatisfeitos);
System.out.printf("Número de produtores envolvidos no cabaz: %d\n", prodInvolved.size());
System.out.println("-------------------------------------------------");
```

```
        List<Double> resultProd = new ArrayList<>();
        resultProd.add((double) produtosSatisfeitos);
        resultProd.add((double) produtosParcSatisfeitos);
        resultProd.add((double) produtosNaoSatisfeitos);
        resultProd.add(percentSatisfeitos);
        resultProd.add((double) prodInvolved.size());
        results.put(c.id, resultProd);
        produtosSatisfeitos = 0;
        produtosParcSatisfeitos = 0;
        produtosNaoSatisfeitos = 0;
        produtosNoCabaz = 0;
    }
}
```

Este método tem de complexidade $O(n^2)$.