Instituto Superior de Engenharia do Porto

Projeto Integrador

Sprint 2

ESINF – Licenciatura em Engenharia Informática

Relatório

Turma 2DM e 2DN

Beatriz Santos - 1211178 (2DM)
Daniela Soares - 1211229 (2DM)
Duarte Ramalho – 1221806 (2DM)
Daniel Braga – 1200801 (2DN)
Rúben Silva – 1200546 (2DM)

Índice

US301

Construir a rede de distribuição através da informação fornecida nos ficheiros

No método “read” começamos por ler a primeira linha do ficheiro: ”clientes-produtores_small.csv”, como esta é o cabeçalho ignoramos. De seguida dividimos os valores dos ficheiros pela virgula. Se for uma empresa cria um vértice como sendo uma “Company” se não, cria como sendo um “User” e guarda tudo numa lista.

```
public void read(String filename) {
    networkGraph = new MapGraph<>( directed: false);

    try (BufferedReader input = new BufferedReader(new FileReader(filename))){
        String line = input.readLine();
        String[] header = line.split( regex: ",");//ler primeira linha e ignorar
        System.out.println(Arrays.toString(header));

        while((line = input.readLine()) != null){
            String separator = ",";
            String[] lineFields = line.split(separator);

            Localization localization = new Localization(lineFields[0],Float.valueOf(lineFields[1]),Float.valueOf(lineFields[2]));
            if(lineFields[3].contains("E")){
                Company company = new Company(localization,lineFields[3]);
                if(!networkGraph.validVertex(company)){
                    networkGraph.addVertex(company);
                }
            }else {
                User user = new User(localization,lineFields[3]);
                if(!networkGraph.validVertex(user)){
                    networkGraph.addVertex(user);
                }
            }
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

No método “readDistances” começamos por ler a primeira linha do ficheiro, como esta é o cabeçalho ignoramos depois, dividimos os valores dos ficheiros pela virgula. De seguida, Para um “User” dentro da lista, se o seu código de Localização for igual ao primeiro campo referente ao código de localização do ficheiro: “distancias_small.csv”, vai fazer a mesma comparação para o segundo campo referente ao código de localização, se for igual adiciona o ramo com o seu respetivo peso.

```
public void readDistances(String filename){
    try (BufferedReader input = new BufferedReader(new FileReader(filename))){
        String line = input.readLine();
        String[] header = line.split( regex: ",");//ler primeira linha e ignorar
        System.out.println(Arrays.toString(header));

        while((line = input.readLine()) != null){
            String separator = ",";
            String[] lineFields = line.split(separator);
            for(User u : networkGraph.vertices()){
                if(u.getLocalization().getCodLoc().equalsIgnoreCase(lineFields[0])){
                    for(User u1: networkGraph.vertices()){
                        if(u1.getLocalization().getCodLoc().equalsIgnoreCase(lineFields[1])){
                            networkGraph.addEdge(u,u1,Float.valueOf(lineFields[2]));
                        }
                    }
                }
            }
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Análise Complexidade

O método “public void read ()” (linhas 20 a 48) é determinístico pois só existe um caso. Sendo assim este método tem complexidade de O(n), pois das linhas 28 a 44 existe um ciclo “for” com complexidade O(n) e dentro desse ciclo existem operações de complexidade O(n) e O(1) como não são ciclos não se multiplica sendo assim a soma O(n+n+1)=O(2n+1)=O(n).

O método “readDistances()” (linhas 50 a 76) também é determinístico existindo apenas um caso. A complexidade deste método é de O(n^3) pois existe um ciclo “while” que tem complexidade O(n) das linhas 56 a 68 onde se encontram mais 2 ciclos desta vez ciclos “for” cada um com complexidade O(n), ou seja, a sua multiplicação dá complexidade O(n^3).

US302

Verificar se o grafo é conexo

```
private static <V,E> void connectComps (Graph<V,E> g, ArrayList<LinkedList<V>> ccs, boolean[] visited){

    int vKey;
    LinkedList<V> conComp = new LinkedList<>();

    for (V vert : g.vertices()){
        vKey = g.key(vert);
        visited[vKey] = false;
    }

    for (V vert : g.vertices()){
        vKey = g.key(vert);
        if (!visited[vKey]) {                          //in some previous BFS
            conComp = BreadthFirstSearch(g, vert);     //discovers precisely V's
            ccs.add(conComp);                           //connected component
        }
        assert conComp != null;
        for (V v : conComp) {
            vKey = g.key(v);
            visited[vKey] = true;
        }
    }
}
```

Primeiro certificamo-nos de que todos os vértices de g ainda não foram visitados, através de um ciclo for. De seguida vamos percorrer novamente todos os vértices de g. Caso o vértice em análise ainda não tenha sido visitado, através do *BreadthFirstSearch* vamos guardar numa LinkedList todos os vértices alcançáveis através do vértice de origem. Adicionamos a lista obtida ao ArrayList css e mudamos o estado dos vértices que estavam na LinkedList para visitados. Desta forma, no css vão estar todos os conjuntos de vértices que estão ligados entre sim.

```java
//se existir algum vértice que não faz parte do arrayList conComp, então o grafo não é conexo

public static <V, E> boolean connectComps(Graph<V, E> g) {

    ArrayList<LinkedList<V>> ccs = new ArrayList<>();
    boolean[] visited = new boolean[g.numVertices()];

    connectComps(g, ccs, visited);

    /*se o primeiro elemento (LinkedList) de css tiver o número de elementos igual
    ao número de vértices de g então o grafo é conexo*/
    return ccs.get(0).size() == g.numVertices();

}
```

Assim, pegando no ArrayList css, caso a primeira LinkedList tenha o mesmo número de vértices do grafo, então significa que todos os vértices estão conectados e, consequentemente, que o grafo é conexo, sendo retornado true. Caso contrário, o método retorna false.

Análise Complexidade

O algoritmo *connectComps* é determinístico e tem complexidade de $O(V^3)$ ou $O(n^3)$.

Número mínimo de ligações

Para descobrir o número mínimo de ligações foi utilizado o método shortestPaths da classe Algotithms.

```java
/**
 * Shortest-path between a vertex and all other vertices
 *
 * @param g     graph
 * @param vOrig start vertex
 * @param ce    comparator between elements of type E
 * @param sum   sum two elements of type E
 * @param zero  neutral element of the sum in elements of type E
 * @param paths returns all the minimum paths
 * @param dists returns the corresponding minimum distances
 * @return if vOrig exists in the graph true, false otherwise
 */
public static <V, E> boolean shortestPaths(Graph<V, E> g, V vOrig, Comparator<E> ce, BinaryOperator<E> sum, E zero,
                                           ArrayList<LinkedList<V>> paths, ArrayList<E> dists) {

    if (!g.validVertex(vOrig)) {
        return false;
    }

    dists.clear();
    paths.clear();

    int numVerts = g.numVertices();
```

O shortestPaths guarda num ArrayList de LinkedList todos os caminhos mínimos entre o vértice enviado por parâmetro e todos os outros vértices do grafo.

Na classe DistributionNetWork temos o método numeroLigacoesMin. Este método vai descobrir qual é o caminho de maior comprimento que será o diâmetro do grafo.

```
//US302
public int numeroLigacoesMin (Graph<User, Float> g){
    int diâmetro = 0;
    //LinkedList<User> pathRes = null;
    for (User v : g.vertices()){
        ArrayList<LinkedList<User>> paths = new ArrayList<>();
        ArrayList<Float> dists = new ArrayList<>();
        Algorithms.shortestPaths(g, v, Float::compare, Float::sum, zero: 0f, paths, dists);

        for (LinkedList<User> path : paths) {
            if (path.size() > diâmetro) {
                diâmetro = path.size()-1;
                //pathRes=path;
            }
        }
    }
    //System.out.println(pathRes);
    return diâmetro;
}
```

Análise Complexidade

O método shortestPaths é determinístico e tem complexidade O(V*E).

O método numeroLigacoesMin é também determinístico e tem complexidade $O(n^3)$.

US303

Definir os hubs da rede de distribuição.

```
Map<Company,Float> map = new HashMap<>();
//US303
3 usages  Daniel Braga +1 *
public Map<Company, Float> networkHubs(int n){
    float sum;
    ArrayList<Float> dists = new ArrayList<>();

    for(User u : networkGraph.vertices()){
        sum=0;
        if(u.getClass().equals(Company.class)){
            Algorithms.shortestPaths(networkGraph,u,Float::compare,Float::sum, zero: 0F,new ArrayList<>(),dists);
            for(Float f : dists){
                sum+=f;
            }
            map.put((Company) u,sum/(networkGraph.numVertices()-1));
        }
    }
    map=map.entrySet().stream().sorted(Map.Entry.comparingByValue()).limit(n).collect(Collectors.toMap
            (Map.Entry::getKey, Map.Entry::getValue, (e1, e2)->e1, LinkedHashMap::new));

    for(Company c : map.keySet()){
        c.setHubTrue();
    }

    return map;
}
```

Para definir os hubs da rede de distribuição, é criado um *map* que vai guardar todas as empresas com a sua respetiva distância média a todos os clientes e produtores e iniciada uma variável *sum* que vai acumular o valor da distância média para cada empresa, valor este que está guardado na lista *dists*.

Inicialmente vão ser percorridos todos os utilizadores no grafo e, quando for encontrada uma empresa, será invocado o método *shortestPaths()* que retornará a distância da empresa a todos os outros vértices na lista *dists*. Todos os valores na lista serão somados e guardados no mapa juntamente com a empresa.

Por fim o *map* será cortado para apenas manter as empresas que são relevantes e nessas empresas serão definidos os hubs.

Análise Complexidade

O algoritmo é determinístico e tem complexidade $O(V^3*E)$.

US304

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

```java
/** US304
 * Determine the nearest hub for each client (particular or empresarial)
 * @return clientNearestHub a map where the keys are the clients and the values are the nearest hub
 */
public Map<String, String> getClientsNearestHub (){
    Map<String, String> clientNearestHub = new HashMap<>();

    Map<String, Float> lengthPaths = new HashMap<>();

    for (User client : networkGraph.vertices()){
        for (Company hub : map.keySet()){
            Float lenPath = Algorithms.shortestPath(networkGraph,client,hub,Float::compare,Float::sum, zero: 0F,new LinkedList<>());
            lengthPaths.put(hub.getCodUser(), lenPath);
        }
        Stream<Map.Entry<String, Float>> sorted = lengthPaths.entrySet().stream().sorted(Map.Entry.comparingByValue());
        clientNearestHub.put(client.getCodUser(), sorted.findFirst().get().getKey());
        lengthPaths.clear();
    }
    clientNearestHub=clientNearestHub.entrySet().stream().sorted(Map.Entry.comparingByKey()).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue, (e1, e2)->e1,
                                                                                                                        LinkedHashMap::new));
    return clientNearestHub;
}
```

No método *getClientNearestHub()* em primeiro lugar é declarado o Map *clientNearestHub,* o qual irá ser retornado, tem como *key* o código de cada User, ou seja, cada cliente e os *values* são o código de cada *Company*, hub. Também é declarado o Map *lengthPaths* que tem como chave o código de cada *Company*, hub, e o tamanho do caminho de cada cliente a cada hub. A seguir são percorridos cada cliente(quer seja particular ou empresa), ou seja, cada vértice do grafo e para cada cliente, através do algoritmo de *Dijkstra*, *shortestPat(),* é determinado o tamanho do caminho do cliente a cada hub sendo, em cada iteração do segundo ciclo for ,adicionado ao lenthPath o hub e o tamanho do caminho.

```java
/** Shortest-path between two vertices
 *
 * @param g graph
 * @param vOrig origin vertex
 * @param vDest destination vertex
 * @param ce comparator between elements of type E
 * @param sum sum two elements of type E
 * @param zero neutral element of the sum in elements of type E
 * @param shortPath returns the vertices which make the shortest path
 * @return if vertices exist in the graph and are connected, true, false otherwise
 */
public static <V, E> E shortestPath(Graph<V, E> g, V vOrig, V vDest,
                                    Comparator<E> ce, BinaryOperator<E> sum, E zero,
                                    LinkedList<V> shortPath) {

    if (!g.validVertex(vOrig) || !g.validVertex(vDest)) return null;

    shortPath.clear();
    int numVerts = g.numVertices();
    boolean[]visited = new boolean[numVerts];
    /unchecked/
    V [] pathKeys = (V[]) new Object[numVerts];
    /unchecked/
    E [] dist = (E[]) new Object[numVerts];
    for (int i =0; i< numVerts; i++){
        dist[i]=null;
        pathKeys[i]=null;
    }
    shortestPathDijkstra(g, vOrig, ce, sum, zero, visited, pathKeys, dist);
```

No final deste segundo ciclo, o *map* anterior é ordenado pelos *values* com o auxilio do método *sorted()*, por ordem ascendente, para se puder obter o caminho de menor tamanho. Então, antes da próxima iteração do primeiro ciclo, o código do cliente é adicionado ao *clientNearestHub* assim como o hub cujo tamanho àquele cliente é o menor que corresponde ao primeiro elemento daquele *map*. Também são removidos todos os elementos do *lengthPaths* através do clear() porque as chaves nos *maps* não podem ser repetidas, assim garantimos que para cada cliente a distância a cada hub é

verdadeira. No fim é retornado para cliente o hub mais próximo através do *clientNearestHub,* o qual está ordenado pelas keys devido ao método sorted().

Análise Complexidade

Este algoritmo é determinístico e tem complexidade de O($V^3\ E^2\ log(V)$).

US305

Algoritmo de Kruskall

```
public static <V,E> Graph<V,E> kruskall (Graph<V,E> g) {
    if (!connectComps(g)) {
        return null;
    }

    Graph<V, E> mst = new MapGraph<>( directed: false);

    ArrayList<Edge<V, E>> lstEdges = new ArrayList<>();
    LinkedList<V> connectedVerts;

    V vOrig;
    V vDst;

    for (V vert : g.vertices()) {
        mst.addVertex(vert);
    }

    for (Edge<V, E> edge : g.edges()) {
        lstEdges.add(edge);
    }
```

```
    Comparator<Edge<V, E>> c = new Comparator<Edge<V, E>>() {
        @Override
        public int compare(Edge<V, E> o1, Edge<V, E> o2) {
            if (o1.getWeight().hashCode() < o2.getWeight().hashCode()) return -1;
            else if (o1.getWeight().hashCode() > o2.getWeight().hashCode()) return +1;
            else return 0;
        }
    };

    lstEdges.sort(c);        // in ascending order of weight

    for (Edge<V, E> e : lstEdges) {
        vOrig = e.getVOrig();
        vDst = e.getVDest();
        connectedVerts = DepthFirstSearch(mst, vOrig);
        assert connectedVerts != null;
        if (!connectedVerts.contains(vDst))
            mst.addEdge(vOrig, vDst, e.getWeight());
    }

    return mst;

}
```

Primeiro verificamos se o grafo é conexo. Se não for então o grafo retornado será nulo. Se for vamos então calcular o caminho mínimo.

Começamos por adicionar todos os vértices do grafo passado por parâmetro "g" ao novo grafo (mst). De seguida vamos guardar todos as arestas de "g" num ArrayList para que posteriormente o possamos ordenar por ordem crescente de distâncias. Para o ordenar vamos usar o sort e um compare.

Vamos depois percorrer todos as arestas ordenadas através de um ciclo for. Vamos buscar o vértice de origem e o de destino da aresta. Através do DepthFirstSearch iremos obter todas as ligações que o vértice de origem já tem registada no grafo mst. Se o connectedVerts não contiver ainda o vDest vamos então adicionar a aresta ao grafo mst.

Depois de percorrermos todo o ciclo for teremos o grafo com o caminho de menor custo pronto para retornar.

Análise Complexidade

Este algoritmo é determinístico e tem complexidade de $O(V \times E^2)$ ou $O(n^3)$.

US307

Importar a lista de cabazes

```
//US307
DuarteRamalho +1 *
public void readCabaz(String filename){


    try (BufferedReader input = new BufferedReader(new FileReader(filename))){
        String line = input.readLine();
        String[] header = line.split(",");
        int l = 0;

        while((line = input.readLine()) != null) {
            List<Float> produtos = new ArrayList<>();

            String separator = ",";
            String[] lineFields = line.split(separator);

            for( int i = 2 ; i < lineFields.length;i++){
                produtos.add(Float.parseFloat(lineFields[i]));
            }
            Cabaz cabaz = new Cabaz(Integer.parseInt(lineFields[1]),produtos);

            for(User u : networkGraph.vertices()){
                if (lineFields[0].substring(1,3).equalsIgnoreCase(u.getCodUser())) {
                    u.addCabaz(cabaz);
                }
            }
        }

    } catch (IOException e) {
        throw new RuntimeException(e);
    }

}
```

No método “readCabaz()” começamos por ler a primeira linha do ficheiro inserido, como esta é o cabeçalho ignoramos. De seguida dividimos os valores dos ficheiros pela virgula. A partir da segunda posição a começar do 0 os valores vao ser adicionador a uma lista de “produtos” de seguida comparamos os identificativos de cada “User” e quando for igual adicionamos o cabaz ao respetivo “User”.

Análise da complexidade

Este método é determinístico e a complexidade do código é O(n) onde “n” é o número de linhas do ficheiro “.csv” dado isto o tempo necessário para ler e processar o arquivo aumenta linearmente com o numero de linhas do arquivo, tal como para realizar as operações dos ciclos “for”.

```
86      //US307
        DuarteRamalho +1 *
87      public void readCabaz(String filename){
88
89
90          try (BufferedReader input = new BufferedReader(new FileReader(filename))){
91              String line = input.readLine();
92              String[] header = line.split(",");
93              int l = 0;
94
95              while((line = input.readLine()) != null) {
96                  List<Float> produtos = new ArrayList<>();
97
98                  String separator = ",";
99                  String[] lineFields = line.split(separator);
100
101                 for( int i = 2 ; i < lineFields.length;i++){
102                     produtos.add(Float.parseFloat(lineFields[i]));
103                 }
104                 Cabaz cabaz = new Cabaz(Integer.parseInt(lineFields[1]),produtos);
105
106                 for(User u : networkGraph.vertices()){
107                     if (lineFields[0].substring(1,3).equalsIgnoreCase(u.getCodUser())) {
108                         u.addCabaz(cabaz);
109                     }
110                 }
111             }
112
113         } catch (IOException e) {
114             throw new RuntimeException(e);
115         }
116
117     }
```

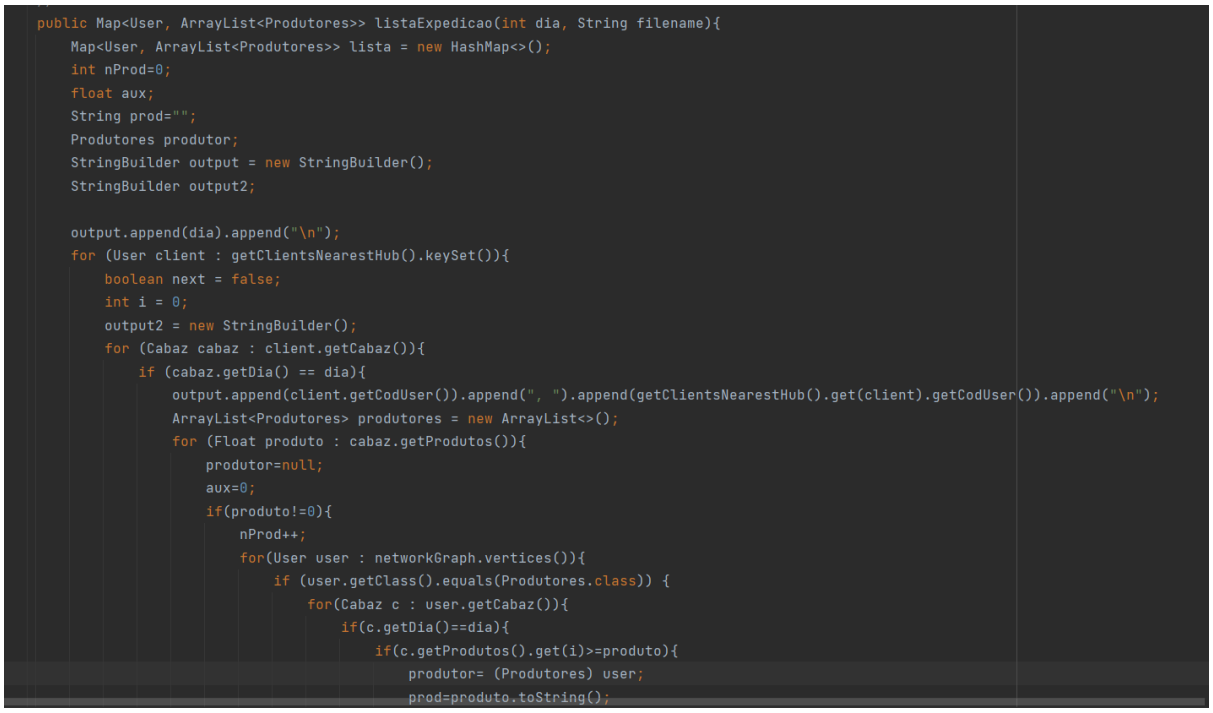
US308

Gerar uma lista de expedição para um determinado dia que forneça os cabazes sem qualquer restrição quanto aos produtores

O algoritmo listaExpedicao() é suposto gerar uma lista de expedição para um determinado dia que forneça os cabazes sem qualquer restrição quanto aos produtores.

Começa por apanhar todos os clientes e para cada um, acede ao cabaz do dia passado como parâmetro, de forma a depois apanhar os produtos do cabaz encomendados.

As quantidades disponíveis são atualizadas com cada expedição para um cliente. No caso de não haver a expedir para um cliente, então é enviada a mensagem “Não há produto para expedir”. E se a quantidade for zero, então o produto correspondente do cabaz do devido cliente não aparece no ficheiro/lista de expedição.



```
public Map<User, ArrayList<Produtores>> listaExpedicao(int dia, String filename){
    Map<User, ArrayList<Produtores>> lista = new HashMap<>();
    int nProd=0;
    float aux;
    String prod="";
    Produtores produtor;
    StringBuilder output = new StringBuilder();
    StringBuilder output2;

    output.append(dia).append("\n");
    for (User client : getClientsNearestHub().keySet()){
        boolean next = false;
        int i = 0;
        output2 = new StringBuilder();
        for (Cabaz cabaz : client.getCabaz()){
            if (cabaz.getDia() == dia){
                output.append(client.getCodUser()).append(", ").append(getClientsNearestHub().get(client).getCodUser()).append("\n");
                ArrayList<Produtores> produtores = new ArrayList<>();
                for (Float produto : cabaz.getProdutos()){
                    produtor=null;
                    aux=0;
                    if(produto!=0){
                        nProd++;
                        for(User user : networkGraph.vertices()){
                            if (user.getClass().equals(Produtores.class)) {
                                for(Cabaz c : user.getCabaz()){
                                    if(c.getDia()==dia){
                                        if(c.getProdutos().get(i)>=produto){
                                            produtor= (Produtores) user;
                                            prod=produto.toString();
```

```
                    for(User user : networkGraph.vertices()){
                        if (user.getClass().equals(Produtores.class)) {
                            for(Cabaz c : user.getCabaz()){
                                if(c.getDia()==dia){
                                    if(c.getProdutos().get(i)>=produto){
                                        produtor= (Produtores) user;
                                        prod=produto.toString();
                                        next=true;
                                        break;
                                    }
                                    if(c.getProdutos().get(i)>aux && c.getProdutos().get(i)<produto){
                                        produtor= (Produtores) user;
                                        prod=c.getProdutos().get(i).toString();
                                        aux=c.getProdutos().get(i);
                                    }
                                }
                            }
                        }
                        if(next) break;
                    }

                    //diminuir ao produtor a quantidade fornecida do produto requesitado pelo cliente
                    if(produtor!=null){
                        for(Cabaz c : produtor.getCabaz()){
                            if(c.getDia()==dia) {
                                c.getProdutos().set(i,c.getProdutos().get(i)-produto);
                            }
                        }
                    }

                    output2.append("\n").append("Produto ").append(i+1).append(", ").append(produto).append(", ");
```

```
                    output2.append("\n").append("Produto ").append(i+1).append(", ").append(produto).append(", ");
                    if(produtor!=null){
                        output2.append(prod).append(", ").append(produtor.getCodUser());
                        produtores.add(produtor);
                    }else{
                        output2.append("Não há produto para expedir");
                        produtores.add(null);
                    }
                }
                i++;
            }
            lista.put(client, produtores);
            output.append(nProd).append(output2).append("\n");

            nProd=0;
        }
    }
}

ficheiroOutput(filename, output.toString());

return lista;
}
```

Análise da complexidade

O algoritmo listaExpedicao() tem a complexidade de O(n^2) e é determinístico.

US309

Gerar uma lista de expedição para um determinado dia que forneça apenas com os N produtores agrícolas mais próximos do hub de entrega do cliente

Critério de Aceitação: A lista de expedição deve indicar para cada cliente/cabaz, os produtos, quantidade encomendada/expedida e o produtor que forneceu o produto.

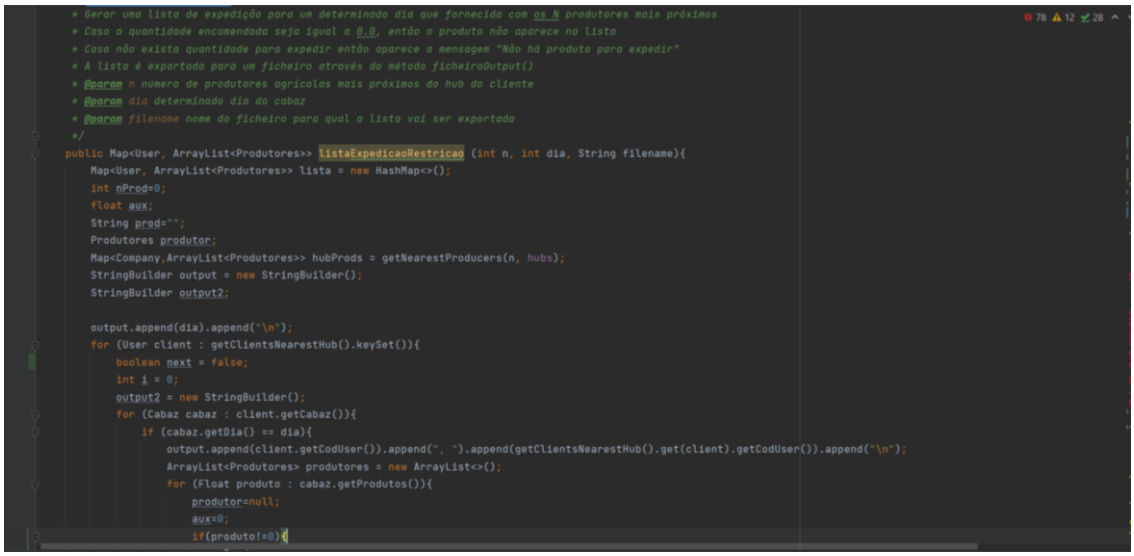
Para esta funcionalidade foram implementados dois algoritmos – getNearestProducers e listaExpedicaoRestricao.

O método getNearestProducers tem como objetivo determinar os N produtores agrícolas mais próximos dos hub de cada cliente. Para isso são percorridos os hubs existentes e para cada hub é determinada a distância a cada produtor através do shortestPath. A cada iteração do hub os produtores e as distâncias ao hub são guardados num map auxiliar, lengthPaths, na forma key, value, respetivamente. Depois de determinadas todas as distâncias àquele hub, então o map auxiliar é ordenado pelos N produtores com menor distância, ou seja, ordenado por values, onde através de um ciclo se guarda esses N produtores num arrayList denominado produtores. Após esta etapa é colocado no map final o hub iterado e o arrayList no formato key, value, respetivamente. Por fim o map auxiliar é inicializado através do clear de forma a receber as novas distâncias para o próximo hub.

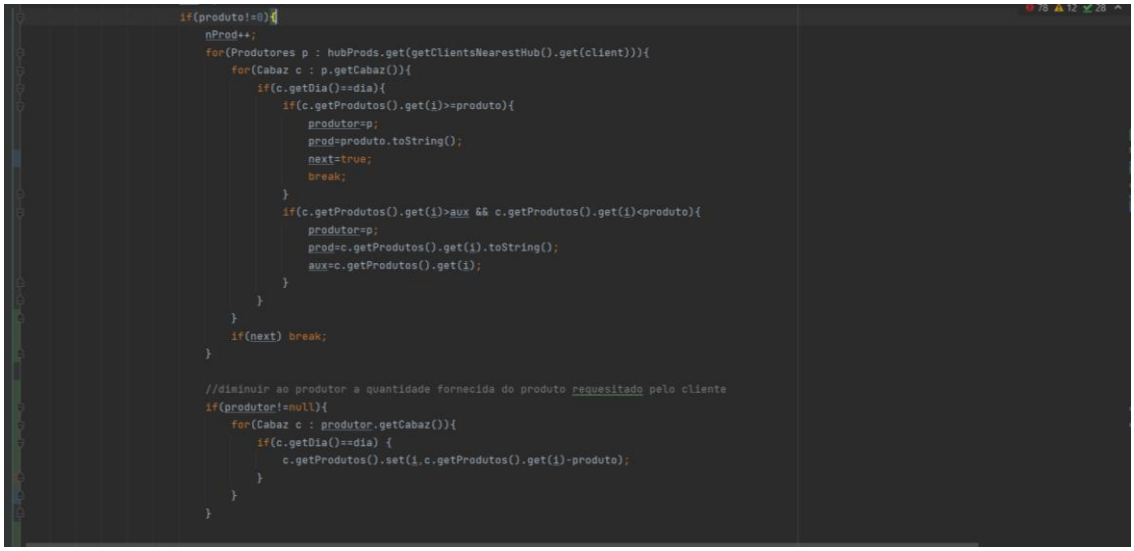
```java
/**
 * Percorrer todos os hubs existentes - Map<Company, Float> hubs
 * Invocar o método shortestPath() para ver as distâncias de cada produtor aos hub existentes
 * Ordenar map auxiliar por distâncias, de forma ascendente
 * Guardar no arrayList "produtores" os n primeiros produtores
 * @param n número de produtores agrícolas mais próximos do hub do cliente
 * @return hubProds map em que a chave são os hubs e os valores correspondentes um arrayList com os produtores mais próximos
 */
public Map<Company,ArrayList<Produtores>> getNearestProducers (int n, Map<Company, Float> hubs){
    ArrayList<Produtores> produtores;
    Map<User, Float> lengthPaths = new HashMap<>();
    Map<Company,ArrayList<Produtores>> hubProds = new HashMap<>();

    for (Company hub : hubs.keySet()){
        for (User produtor : networkGraph.vertices()) {
            if (produtor.getClass().equals(Produtores.class)) {
                Float lenPath = Algorithms.shortestPath(networkGraph, hub, produtor, Float::compare, Float::sum, 0F, new LinkedList<>());
                lengthPaths.put(produtor, lenPath);
            }
        }
        lengthPaths = lengthPaths.entrySet().stream().sorted(Map.Entry.comparingByValue()).limit(n).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,
                                                                                                                             (e1, e2)->e1, LinkedHashMap::new));

        produtores = new ArrayList<>();
        for(User prod : lengthPaths.keySet()){
            produtores.add((Produtores) prod);
        }
        hubProds.put(hub, produtores);
        lengthPaths.clear();
    }
    return hubProds;
}
```

O método listaExpedicaoRestricao tem como objetivo criar a lista de expedição que indica para cada cliente/cabaz, os produtos, quantidade encomendada/expedida, o produtor que forneceu o produto e o hub do cliente para um determinado dia. Para isso é necessário percorrer todos os clientes e para cada um, para o dia passado por parâmetro, aceder ao cabaz correspondente, depois são percorridos os produtos daquele cabaz, ou seja, quantidade encomendada de cada produto, assim como os produtos do cabaz dos N produtores mais próximos, ou seja, quantidades disponíveis

para cada produto. As quantidades disponíveis vão sendo atualizadas à medida que vão sendo expedidas para os clientes, caso não haja quantidade a expedir para um cliente, então é enviada a mensagem “Não há produto para expedir”. Se a quantidade encomendada de um produto for zero, então esse produto do cabaz do cliente não aparece no ficheiro/lista de expedição. O algoritmo chama o método ficheiroOutput para escrever a lista para um ficheiro. Por último, o método retorna um map com users para lista de produtores, que é utilizado na US310.



```
    * Gerar uma lista de expedição para um determinado dia que fornecida com os N produtores mais próximos
    * Caso a quantidade encomendada seja igual a 0.0, então o produto não aparece na lista
    * Caso não exista quantidade para expedir então aparece a mensagem "Não há produto para expedir"
    * A lista é exportada para um ficheiro através do método ficheiroOutput()
    * @param n número de produtores agrícolas mais próximos do hub do cliente
    * @param dia determinado dia do cabaz
    * @param filename nome do ficheiro para qual a lista vai ser exportada
    */
    public Map<User, ArrayList<Produtores>> listaExpedicaoRestricao (int n, int dia, String filename){
        Map<User, ArrayList<Produtores>> lista = new HashMap<>();
        int nProd=0;
        float aux;
        String prod="";
        Produtores produtor;
        Map<Company,ArrayList<Produtores>> hubProds = getNearestProducers(n, hubs);
        StringBuilder output = new StringBuilder();
        StringBuilder output2;

        output.append(dia).append("\n");
        for (User client : getClientsNearestHub().keySet()){
            boolean next = false;
            int i = 0;
            output2 = new StringBuilder();
            for (Cabaz cabaz : client.getCabaz()){
                if (cabaz.getDia() == dia){
                    output.append(client.getCodUser()).append(", ").append(getClientsNearestHub().get(client).getCodUser()).append("\n");
                    ArrayList<Produtores> produtores = new ArrayList<>();
                    for (Float produto : cabaz.getProdutos()){
                        produtor=null;
                        aux=0;
                        if(produto!=0){
```



```
                if(produto!=0){
                    nProd++;
                    for(Produtores p : hubProds.get(getClientsNearestHub().get(client))){
                        for(Cabaz c : p.getCabaz()){
                            if(c.getDia()==dia){
                                if(c.getProdutos().get(i)>=produto){
                                    produtor=p;
                                    prod=produto.toString();
                                    next=true;
                                    break;
                                }
                                if(c.getProdutos().get(i)>aux && c.getProdutos().get(i)<produto){
                                    produtor=p;
                                    prod=c.getProdutos().get(i).toString();
                                    aux=c.getProdutos().get(i);
                                }
                            }
                        }
                        if(next) break;
                    }

                    //diminuir ao produtor a quantidade fornecida do produto requesitado pelo cliente
                    if(produtor!=null){
                        for(Cabaz c : produtor.getCabaz()){
                            if(c.getDia()==dia) {
                                c.getProdutos().set(i,c.getProdutos().get(i)-produto);
                            }
                        }
                    }
```

```
                    output2.append("\n").append("Produto ").append(i+1).append(",").append(produto).append(", ");
                    if(produtor!=null){
                        output2.append(prod).append(", ").append(produtor.getCodUser());
                        produtores.add(produtor);
                    }else{
                        output2.append("Não há produto para expedir");
                        produtores.add(null);
                    }
                }
                i++;
            }
            lista.put(client, produtores);
            output.append(nProd).append(output2).append("\n");

            nProd=0;
        }
    }
}

ficheiroOutput(filename, output.toString());

return lista;
}
```

Análise da complexidade

O algoritmo getNearestProducers é determinístico e tem complexidade $O(n^2\ log(n))$.

O algoritmo listaExpedicaoRestricao é determinístico e tem complexidade $O(n^2)$.

US310

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

Critério de Aceitação: Indicar os pontos de passagem do percurso (produtores e hubs), cabazes entregues em cada hub, distância entre todos os pontos do percurso e a distância total.

```
//US310 - Para uma lista de expedição diária gerar o percurso de entrega que minimiza a distância total percorrida.

public <V,E> Graph<User, Float> percursoEntregaMin (Map<User, ArrayList<Produtores>> lista) {

    Graph<User, Float> grafoPercursoMin = new MapGraph<>( directed: true);
    Graph<User, Float> grafoAuxiliar = networkGraph.clone();

    float distanciaTotal = 0;

    List<User> lp = new ArrayList<>();
    List<User> lh = new ArrayList<>();

    for (User u : lista.keySet()) {
        grafoAuxiliar.removeVertex(u);
        lh.add(u);
        ArrayList<Produtores> listaProdutores = lista.get(u);
        lp.addAll(listaProdutores);
    }

    boolean[] visited = new boolean[networkGraph.numVertices()];

    int vKey;
```

Para implementar o algoritmo da US310 começou-se por obter a lista de expedição diária, gerada através da US308 ou da US309 (essa lista, que é um map, diz para cada hub os produtores que lhe fornecem produtos).

De seguida criou-se um grafo temporário, do qual se removeu os hubs, pois só serão percorridos posteriormente. Através do shortestPath da classe Algorithms calcula-se o caminho mínimo entre todos os produtores que vão fornecer produtos.

Posteriormente é encontrado o hub mais próximo do último produtor visitado. Por fim calcula-se o caminho mínimo entre todos os hubs, da mesma forma que foi calculado para os produtores, a começar no hub mais próximo do último produtor visitado.

É retornado o grafo que contém o caminho mínimo de expedição.

Nota: neste método temos um array de vértices visitados (do tipo boolean) de modo a garantir que cada vértice só é visitado uma vez.

Análise da complexidade

Este método é não determinístico. No melhor caso tem complexidade $O(n^3)$ e no pior caso tem complexidade $O(n^4)$.

US311

Para uma lista de expedição calcular estatísticas

Inicialmente vão ser iniciadas variáveis com o número de produtos totalmente satisfeitos(*nProdS*), o número de produtos parcialmente satisfeitos(*nProdPS*), o número de produtos não satisfeitos(*nProdNS*), a quantidade de produto requisitada(*qProdReq*) a quantidade de produto fornecida(*qProdFor*) (necessário para o cálculo da percentagem de cabaz satisfeita). É iniciada também uma array list *produtores* que vai registar todos os produtores que participaram em cada cabaz.

De seguida o ficheiro com a lista de expedição vai começar a ser lido e vai ser feito um ciclo *for* para cada cabaz que vai percorrer todos os produtos (requisitados e fornecidos) dentro deste e incrementar as variáveis inicialmente criadas. De seguida vai registar no ficheiro com as estatísticas as informações calculadas.

Antes de passar para o próximo cabaz as variáveis e a lista são limpas para poder registar novos valores.

Após o registo das estatísticas por cabaz/cliente é invocado o método para registar as estatísticas por Produtor/Hub.

Método *estatisticasCabaz*:

```java
public void estatisticasCabaz(String inputFile, String outputFile){
    int nProdS=0, nProdPS=0, nProdNS=0;
    float qProdReq=0, qProdFor=0;
    ArrayList<String> produtores = new ArrayList<>();
    StringBuilder output= new StringBuilder();

    output.append("--------------Estatísticas por cabaz/cliente--------------").append("\n").append("\n");

    try (BufferedReader input = new BufferedReader(new FileReader(inputFile))){

        String line=input.readLine();
        String separator = ",";
        String[] lineFields = line.split(separator);
        output.append("Dia ").append(lineFields[0]).append("\n").append("\n");

        while((line = input.readLine()) != null){
            lineFields = line.split(separator);
            StringBuilder output2 = new StringBuilder();
            output2.append("Cabaz -> ").append("Cliente: ").append(lineFields[0]).append("; Hub: ").append(lineFields[1]);

            line=input.readLine();
            lineFields = line.split(separator);
            int lenght = Integer.parseInt(lineFields[0]);

            if(lenght!=0){
                for(int i=0;i<lenght;i++){
                    line=input.readLine();
                    lineFields = line.split(separator);
```

```java
                    if(lineFields.length==4){
                        if(Float.parseFloat(lineFields[1])>Float.parseFloat(lineFields[2]) && Float.parseFloat(lineFields[2])!=0){
                            nProdPS++;
                        }else if(Float.parseFloat(lineFields[1])==Float.parseFloat(lineFields[2])){
                            nProdS++;
                        }
                        if(!produtores.contains(lineFields[3])){
                            produtores.add(lineFields[3]);
                        }
                        qProdFor+=Float.parseFloat(lineFields[2]);
                    }else{nProdNS++;}
                    qProdReq+=Float.parseFloat(lineFields[1]);

                }

                output.append(output2).append("\n").append("nº de produtos totalmente satisfeitos: ").append(nProdS).append("\n")
                        .append("nº de produtos parcialmente satisfeitos: ").append(nProdPS).append("\n").append("nº de produtos não satisfeitos: ")
                        .append(nProdNS).append("\n").append("percentagem total do cabaz satisfeito: ").append((qProdFor/qProdReq)*100).append("%")
                        .append("\n").append("nº de produtores que forneceram o cabaz: ").append(produtores.size()).append("\n").append("\n");
            }
```

```java
            nProdS=0;
            nProdPS=0;
            nProdNS=0;
            qProdFor=0;
            qProdReq=0;
            produtores.clear();
        }
        estatisticasProdutorHub(inputFile, outputFile, output);

    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Iniciando o método das estatísticas por Produtor/Hub vão ser criadas as strings *cliente* e *hub* que vão registar o cliente e o hub de cada cabaz presente no ficheiro e algumas array lists, estas são prodCliente (vai registar cada produtor juntamente com uma lista dos clientes a quem forneceram algum produto), prodHub(vai registar cada produtor juntamente com uma lista dos hubs onde forneceram algum produto), prodNC(vai registar cada produtor juntamente com um par de ints em que a chave vai guardar nº de cabazes fornecidos totalmente e o valor o nº de cabazes fornecidos parcialmente), hubsLista(vai registar todos os hubs presentes na lista de expedição), hubProd(vai registar cada hub juntamente com uma lista dos produtores que forneceram algum produto neste) e hubCliente(vai registar cada hub juntamente com uma lista dos clientes que foram buscar algum produto a este).

Após a criação das listas é iniciada a leitura do ficheiro. A leitura vai ser dividida por cabaz, para cada cabaz vai ser adicionado o hub à lista *hubsLista*, e de seguida os produtores que participaram no cabaz serão adicionados à lista *prodCabaz* e será feita uma verificação para o caso de um produtor ter fornecido 100% do cabaz (*cabazFT*) . De seguida o cliente que requisitou o cabaz vai ser adicionado às listas *hubCliente* e *prodCliente*, os produtores vão ser adicionados à lista *hubProd* e à lista *prodNC* e vai ser incrementado o nº de cabazes fornecidos totalmente ou o nº de cabazes fornecidos parcialmente, e o hub vai ser adicionado à lista *prodCabaz*. Antes de passar para a leitura do cabaz seguinte a lista *prodCabaz* vai ser limpa.

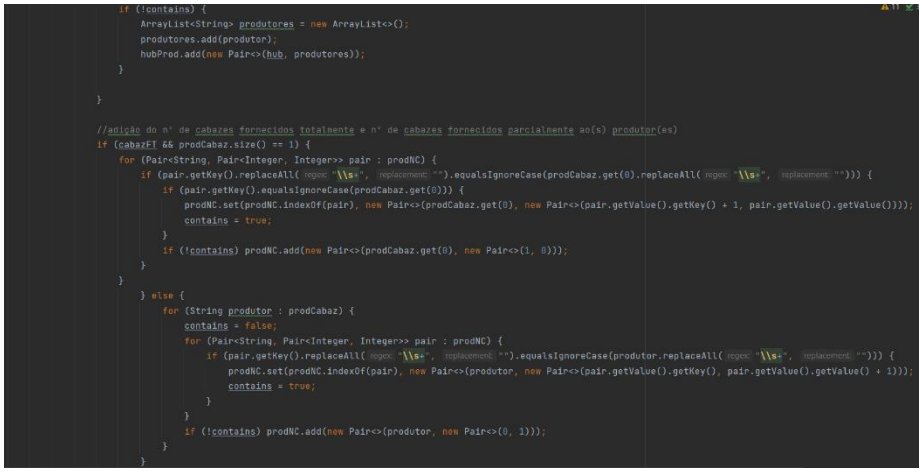
Depois da leitura de todos os cabazes, vão ser escritas para o ficheiro as informações todas que foram registadas juntamente com o nº de produtos totalmente esgotados de cada fornecedor.

Método *estatisticasProdutorHub*:

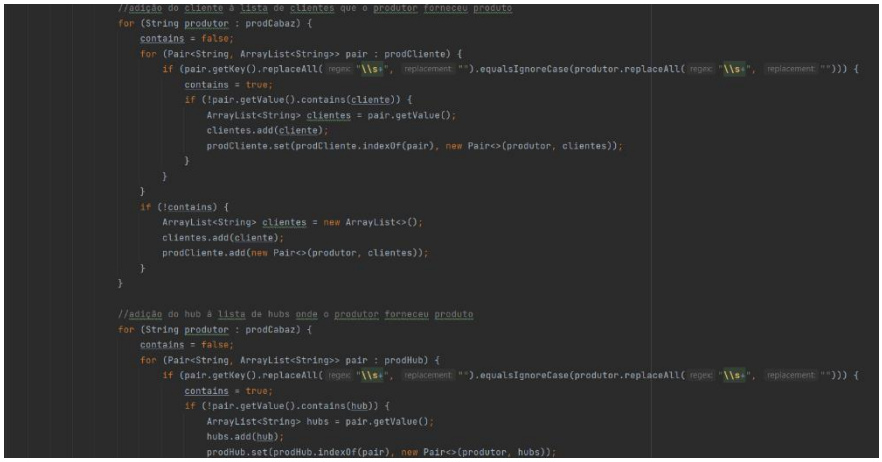
```
public void estatisticasProdutorHub(String inputFile, String outputFile, StringBuilder output) {
    String line, cliente, hub;

    ArrayList<Pair<String, ArrayList<String>>> prodCliente = new ArrayList<>(); //vai registar cada produtor juntamente com uma lista dos clientes a quem
                                                                                 // forneceram algum produto
    ArrayList<Pair<String, ArrayList<String>>> prodHub = new ArrayList<>();     //vai registar cada produtor juntamente com uma lista dos hubs onde forneceram algum produto
    ArrayList<Pair<String, Pair<Integer, Integer>>> prodNC = new ArrayList<>(); //vai registar cada produtor juntamente com um par de ints em que a chave vai guardar
                                                                                 //nº de cabazes fornecidos totalmente e o valor o nº de cabazes fornecidos parcialmente
    ArrayList<String> hubsLista = new ArrayList<>();                            //vai registar todos os hubs presentes na lista de expedição
    ArrayList<Pair<String, ArrayList<String>>> hubProd = new ArrayList<>();     //vai registar cada hub juntamente com uma lista dos produtores
                                                                                 // que forneceram algum produto neste
    ArrayList<Pair<String, ArrayList<String>>> hubCliente = new ArrayList<>();  //vai registar cada hub juntamente com uma lista dos clientes que foram
                                                                                 // buscar algum produto a este

    output.append("-------------Estatísticas por produtor-------------").append("\n").append("\n");

    try (BufferedReader input = new BufferedReader(new FileReader(inputFile))) {
        line = input.readLine();
        String separator = " ";
        String[] lineFields = line.split(separator);
        int dia = Integer.parseInt(lineFields[0]);

        while ((line = input.readLine()) != null) {
            lineFields = line.split(separator);
            ArrayList<String> prodCabaz = new ArrayList<>();

            cliente = lineFields[0];
            hub = lineFields[1];
```

```
            boolean contains = false;
            //adição do hub à lista de hubs
            if (!hubsLista.contains(hub)) {
                hubsLista.add(hub);
            }


            line = input.readLine();
            lineFields = line.split(separator);
            int lenght = Integer.parseInt(lineFields[0]);

            boolean cabazFT = true;
            if (lenght != 0) {
                for (int i = 0; i < lenght; i++) {
                    line = input.readLine();
                    lineFields = line.split(separator);

                    if (lineFields.length == 4) {

                        if (Float.parseFloat(lineFields[1]) != Float.parseFloat(lineFields[2])) {
                            cabazFT = false;
                        }

                        //adição do produtor à lista de produtores do cabaz
                        if (!prodCabaz.contains(lineFields[3])) {
                            prodCabaz.add(lineFields[3]);
                        }
                    }
                }
```

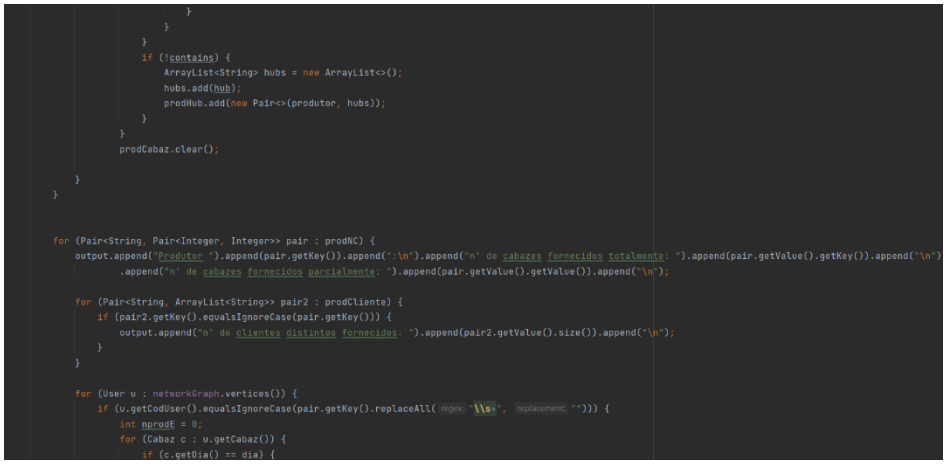
```
//adição do cliente à lista de clientes do hub
for (Pair<String, ArrayList<String>> pair : hubCliente) {
    if (pair.getKey().replaceAll( regex: "\\s+", replacement: "").equalsIgnoreCase(hub.replaceAll( regex: "\\s+", replacement: ""))) {
        contains = true;
        if (!pair.getValue().contains(cliente)) {
            ArrayList<String> clientes = pair.getValue();
            clientes.add(cliente);
            hubCliente.set(hubCliente.indexOf(pair), new Pair<>(hub, clientes));
        }
    }
}
if (!contains) {
    ArrayList<String> clientes = new ArrayList<>();
    clientes.add(cliente);
    hubCliente.add(new Pair<>(hub, clientes));
}

//adição dos produtores à lista de produtores que forneceram produto no hub
for (String produtor : prodCabaz) {
    contains = false;
    for (Pair<String, ArrayList<String>> pair : hubProd) {
        if (pair.getKey().replaceAll( regex: "\\s+", replacement: "").equalsIgnoreCase(hub.replaceAll( regex: "\\s+", replacement: ""))) {
            contains = true;
            if (!pair.getValue().contains(produtor)) {
                ArrayList<String> produtores = pair.getValue();
                produtores.add(produtor);
                hubProd.set(hubProd.indexOf(pair), new Pair<>(hub, produtores));
            }
        }
    }
```



```
    if (!contains) {
        ArrayList<String> produtores = new ArrayList<>();
        produtores.add(produtor);
        hubProd.add(new Pair<>(hub, produtores));
    }

}

//adição do nº de cabazes fornecidos totalmente e nº de cabazes fornecidos parcialmente ao(s) produtor(es)
if (cabazFT && prodCabaz.size() == 1) {
    for (Pair<String, Pair<Integer, Integer>> pair : prodNC) {
        if (pair.getKey().replaceAll( regex: "\\s+", replacement: "").equalsIgnoreCase(prodCabaz.get(0).replaceAll( regex: "\\s+", replacement: ""))) {
            if (pair.getKey().equalsIgnoreCase(prodCabaz.get(0))) {
                prodNC.set(prodNC.indexOf(pair), new Pair<>(prodCabaz.get(0), new Pair<>(pair.getValue().getKey() + 1, pair.getValue().getValue())));
                contains = true;
            }
            if (!contains) prodNC.add(new Pair<>(prodCabaz.get(0), new Pair<>(1, 0)));
        }
    }
    } else {
        for (String produtor : prodCabaz) {
            contains = false;
            for (Pair<String, Pair<Integer, Integer>> pair : prodNC) {
                if (pair.getKey().replaceAll( regex: "\\s+", replacement: "").equalsIgnoreCase(produtor.replaceAll( regex: "\\s+", replacement: ""))) {
                    prodNC.set(prodNC.indexOf(pair), new Pair<>(produtor, new Pair<>(pair.getValue().getKey(), pair.getValue().getValue() + 1)));
                    contains = true;
                }
            }
            if (!contains) prodNC.add(new Pair<>(produtor, new Pair<>(0, 1)));
        }
    }
```

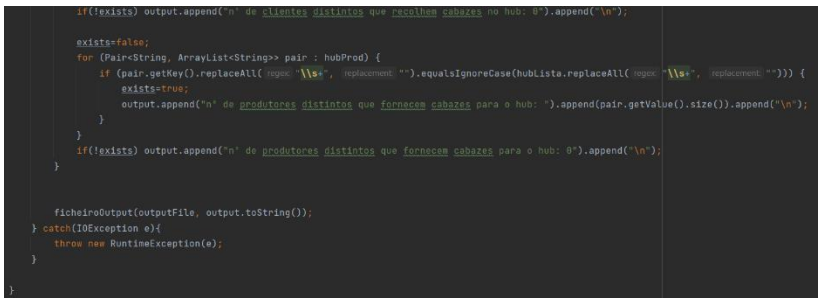


```
//adição do cliente à lista de clientes que o produtor forneceu produto
for (String produtor : prodCabaz) {
    contains = false;
    for (Pair<String, ArrayList<String>> pair : prodCliente) {
        if (pair.getKey().replaceAll( regex: "\\s+", replacement: "").equalsIgnoreCase(produtor.replaceAll( regex: "\\s+", replacement: ""))) {
            contains = true;
            if (!pair.getValue().contains(cliente)) {
                ArrayList<String> clientes = pair.getValue();
                clientes.add(cliente);
                prodCliente.set(prodCliente.indexOf(pair), new Pair<>(produtor, clientes));
            }
        }
    }
    if (!contains) {
        ArrayList<String> clientes = new ArrayList<>();
        clientes.add(cliente);
        prodCliente.add(new Pair<>(produtor, clientes));
    }
}

//adição do hub à lista de hubs onde o produtor forneceu produto
for (String produtor : prodCabaz) {
    contains = false;
    for (Pair<String, ArrayList<String>> pair : prodHub) {
        if (pair.getKey().replaceAll( regex: "\\s+", replacement: "").equalsIgnoreCase(produtor.replaceAll( regex: "\\s+", replacement: ""))) {
            contains = true;
            if (!pair.getValue().contains(hub)) {
                ArrayList<String> hubs = pair.getValue();
                hubs.add(hub);
                prodHub.set(prodHub.indexOf(pair), new Pair<>(produtor, hubs));
```



```
                        }
                    }
                }
                if (!contains) {
                    ArrayList<String> hubs = new ArrayList<>();
                    hubs.add(hub);
                    prodHub.add(new Pair<>(produtor, hubs));
                }
            }
            prodCabaz.clear();

        }
    }


    for (Pair<String, Pair<Integer, Integer>> pair : prodNC) {
        output.append("Produtor ").append(pair.getKey()).append(":\n").append("nº de cabazes fornecidos totalmente: ").append(pair.getValue().getKey()).append("\n")
                .append("nº de cabazes fornecidos parcialmente: ").append(pair.getValue().getValue()).append("\n");

        for (Pair<String, ArrayList<String>> pair2 : prodCliente) {
            if (pair2.getKey().equalsIgnoreCase(pair.getKey())) {
                output.append("nº de clientes distintos fornecidos: ").append(pair2.getValue().size()).append("\n");
            }
        }

        for (User u : networkGraph.vertices()) {
            if (u.getCodUser().equalsIgnoreCase(pair.getKey().replaceAll( regex: "\\s+",  replacement: ""))) {
                int nprodE = 0;
                for (Cabaz c : u.getCabaz()) {
                    if (c.getDia() == dia) {
```

```
                        for (Float f : c.getProdutos()) {
                            if (f == 0) nprodE++;
                        }
                    }
                }
                output.append("nº de produtos totalmente esgotados: ").append(nprodE).append("\n");
            }
        }


        for (Pair<String, ArrayList<String>> pair3 : prodHub) {
            if (pair3.getKey().equalsIgnoreCase(pair.getKey())) {
                output.append("nº de hubs fornecidos: ").append(pair3.getValue().size()).append("\n").append("\n");
            }
        }
    }

    output.append("--------------Estatísticas por hub--------------").append("\n").append("\n");

    for (String hubLista : hubsLista) {
        boolean exists=false;
        output.append("\n").append("Hub ").append(hubLista).append("\n");

        for (Pair<String, ArrayList<String>> pair : hubCliente) {
            if (pair.getKey().replaceAll( regex: "\\s+",  replacement: "").equalsIgnoreCase(hubLista.replaceAll( regex: "\\s+",  replacement: ""))) {
                exists=true;
                output.append("nº de clientes distintos que recolhem cabazes no hub: ").append(pair.getValue().size()).append("\n");
            }
        }
```



```
        if(!exists) output.append("nº de clientes distintos que recolhem cabazes no hub: 0").append("\n");

        exists=false;
        for (Pair<String, ArrayList<String>> pair : hubProd) {
            if (pair.getKey().replaceAll( regex: "\\s+",  replacement: "").equalsIgnoreCase(hubLista.replaceAll( regex: "\\s+",  replacement: ""))) {
                exists=true;
                output.append("nº de produtores distintos que fornecem cabazes para o hub: ").append(pair.getValue().size()).append("\n");
            }
        }
        if(!exists) output.append("nº de produtores distintos que fornecem cabazes para o hub: 0").append("\n");
    }


    ficheiroOutput(outputFile, output.toString());
} catch(IOException e){
    throw new RuntimeException(e);
}
}
```

Análise da complexidade

A complexidade será de O(n)onde n será o número de linhas no ficheiro.