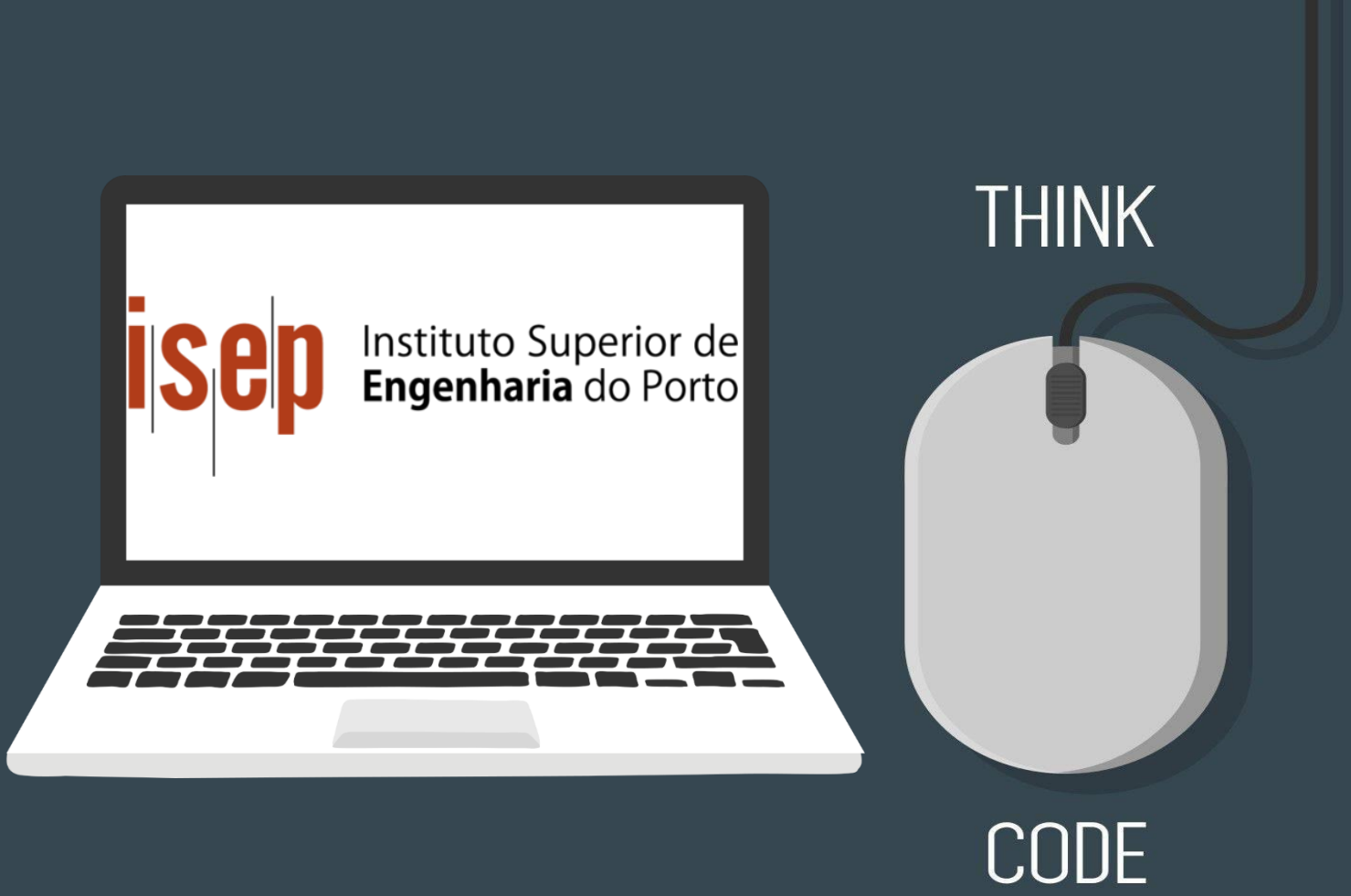


Relatório de Estruturas de Informação:

-> Sprint 2

Jorge Cunha nº1200618
Fábio Silva nº1201942
Francisco Sampaio nº1191576
Filipe Morais nº1190569
João Sousa nº1190701

Índice

Introdução

Alguns pontos específicos que devemos considerar para cada user story são:

US307: Esta user story exige a criação de uma funcionalidade para importar uma lista de cabazes de um arquivo externo.

A complexidade dessa tarefa pode depender do formato do arquivo e da quantidade de dados a serem importados.

US308: Esta user story solicita a geração de uma lista de expedição sem restrições quanto aos produtores. Isso exige a implementação de algoritmo para selecionar os produtos mais adequados para cada cabaz, levando em consideração as quantidades disponíveis e os prazos de validade.

US309: Esta user story é parecida à anterior, mas adiciona a restrição de selecionar apenas os produtores mais próximos do hub de entrega.

Isso exige a implementação de algoritmo de find nearest producer para encontrar os produtores mais próximos e é mais complexo do que a US308.

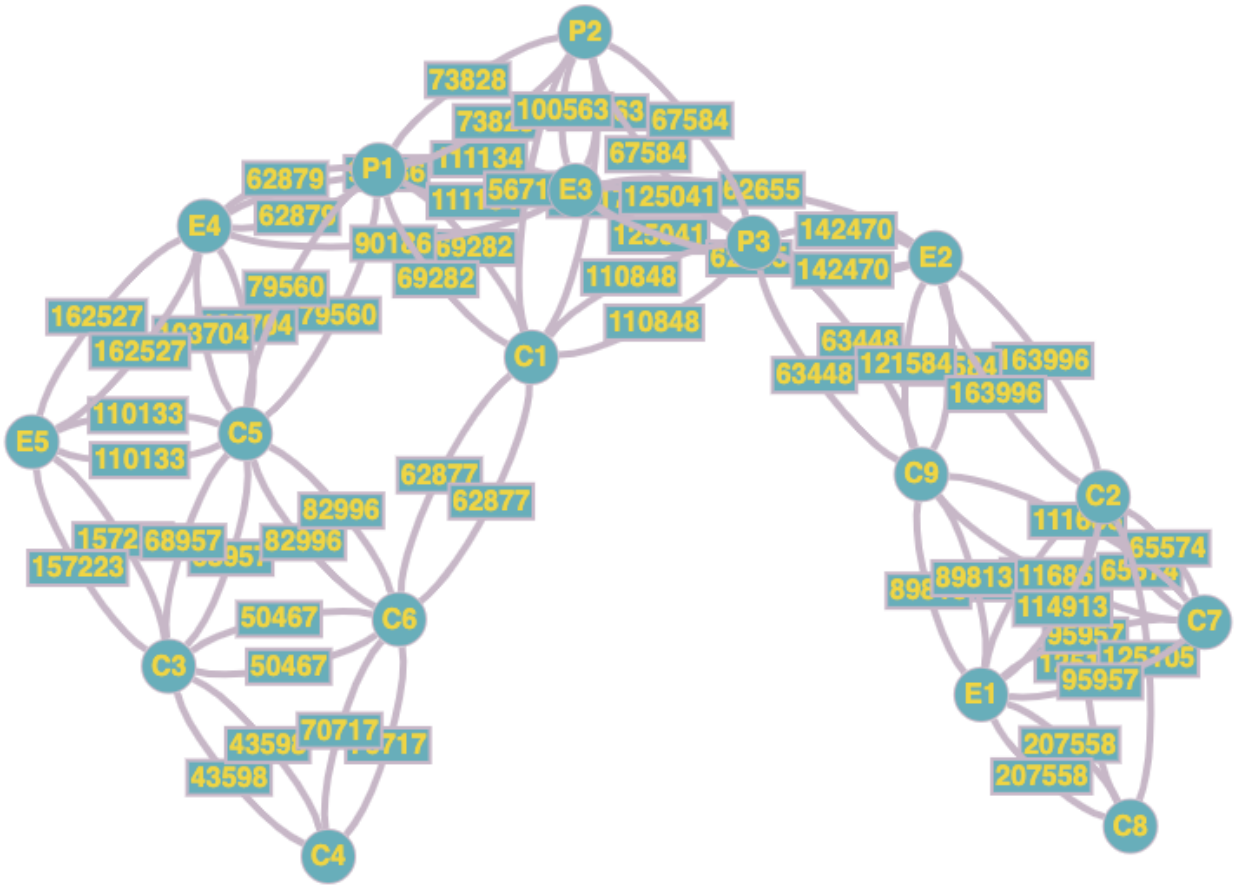
US310: Esta user story pede um percurso de entrega que minimize a distância total percorrida.

Isso exige a implementação de algoritmo para otimizar a rota e irá ficar mais complexo.

US311: Esta user story exige várias estatísticas a partir de uma lista de expedição.

Em resumo, todas as user stories podem ter uma complexidade variável, dependendo do que é exigido e da forma como os componentes envolvidos precisam ser implementados.

Exemplo do Map Graph (Ficheiro Pequeno)



US307 - Importar a lista de cabazes

Linhas 3 e 4: São declaradas as variáveis "pessoaList" e "mapClientes", que são uma lista de objetos "Pessoa" e um mapa de objetos "Pessoa" para árvores de mapas, respectivamente. A complexidade dessas linhas é constante.

Linha 6: É declarada a variável "mapProdutos", que é um mapa de objetos "Produtos" para valores do tipo "Double". Essa variável é inicializada chamando a função "createMapFruta", que percorre o array "values" e cria o mapa. A complexidade dessa linha é O(n), onde n é o tamanho do array "values".

Linhas 8 e 9: São declaradas as variáveis "mapAuxiliar" e "pessoaAuxiliar", que são um mapa e um objeto "Pessoa", respectivamente. A complexidade dessas linhas é constante.

Linha 11: Inicia um loop "for" que percorre a lista "pessoaList". A complexidade dessa linha é O(n), onde n é o tamanho da lista.

Linha 12: Verifica se o ID da pessoa atual do loop é igual ao valor presente no array "values" na posição CLIENTEPRODUTOR_ID. A complexidade dessa linha é constante.

Linha 13: Verifica se o mapa "mapClientes" contém a chave "pessoaAuxiliar". A complexidade dessa linha é O(1) em média, pois os mapas em Java costumam ter uma complexidade de acesso de O(1).

Linhas 14 a 16: Se a chave "pessoaAuxiliar" existir no mapa "mapClientes", são realizadas operações de inserção e atualização em um mapa e uma árvore de mapas. A complexidade dessas linhas é O(log n), onde n é o número de elementos na árvore.

Linhas 17 e 18: Se a chave "pessoaAuxiliar" não existir no mapa "mapClientes", é criado um novo mapa "mapAuxiliar" e inserido no mapa "mapClientes". A complexidade dessas linhas é O(1) em média, pois os mapas em Java costumam ter uma complexidade de inserção de O(1).

Linha 20: Finaliza o loop "for". A complexidade dessa linha é constante.

Linha 22: Finaliza a função "guardarProducao". A complexidade dessa linha é constante.

Em resumo, a complexidade total da função "guardarProducao" é de O(n * log n), pois ela percorre uma lista de tamanho n e realiza operações em uma árvore de mapas de complexidade O(log n). A complexidade da função "createMapFruta" é de O(n), pois ela percorre um array de tamanho n.

```java
public void lerCabaz(String path) throws Exception {
    String data;
    BufferedReader br = new BufferedReader(new FileReader(path));
    String header = br.readLine();
    String[] headerSplit = header.split( regex: ",");

    while ((data = br.readLine()) != null) {
        String[] values = data.split( regex: ",");
        createMap.guardarProducao(values, headerSplit);
    }

}//O(n)
```

```java
private void createStockMap() {
    Map<Pessoa, TreeMap<Integer, Map<Produtos, Double>>> auxMap = mapStore.getStockEcabazMap();
    Map<Produtor, TreeMap<Integer, Map<Produtos, Double>>> stockMap = new HashMap<>();
    Set<Pessoa> pessoasKey = auxMap.keySet();
    Produtor produtorAuxiliar;

    for (Pessoa pessoaAux : pessoasKey) {//O(n)
        if (pessoaAux instanceof Produtor) {
            produtorAuxiliar = (Produtor) pessoaAux;
            stockMap.put(produtorAuxiliar, auxMap.get(pessoaAux));//O(logn)
        }
    }
    mapStore.setstockProdutorMap(stockMap);
}//O(nlogn)
```

```java
private void createCabazMap() {
    Map<Pessoa, TreeMap<Integer, Map<Produtos, Double>>> auxMap = mapStore.getStockEcabazMap();
    Map<Pessoa, TreeMap<Integer, Map<Produtos, Double>>> cabazMap = new HashMap<>();
    Set<Pessoa> pessoasKey = auxMap.keySet();

    for (Pessoa pessoaAux : pessoasKey) {//O(n)
        if (pessoaAux instanceof Cliente || pessoaAux instanceof Empresa) {
            cabazMap.put(pessoaAux, auxMap.get(pessoaAux));//O(logn)
        }
    }
    mapStore.setcabazMap(cabazMap);
}//O(nlogn)
```

US308 - Gerar uma lista de expedição para um determinado dia que forneça os cabazes sem qualquer restrição quanto aos produtores

O método "expedatingForDay" tem uma complexidade O(n*p), onde "n" é o número de clientes. Isso é devido ao loop na linha 18, que itera sobre cada cliente no mapa de clientes.

O método "findObtainedValueNoRestriction" tem uma complexidade O(p), onde "p" é o número de produtores. Isso é devido ao loop na linha 6, que itera sobre cada produtor no mapa de produtores.

O método "updateStockQuantities" tem uma complexidade O(p), onde "p" é o número de produtores. Isso é devido ao loop na linha 3, que itera sobre cada produtor no mapa de produtores.

O método "getQuantityOrDefault" tem uma complexidade O(1), pois ele apenas realiza uma busca em um mapa e retorna um valor.

Método principal US308/309:

```
public List<Cabaz> expedatingForDay(int day, int restriction, int n, Map<Produtor, TreeMap<Integer, Map<Produtos, Double>>> stockMap,
                                    Map<Pessoa, TreeMap<Integer, Map<Produtos, Double>>> clientsMap) {

    Map<Pessoa, Map<Pessoa, Integer>> hubProximoCliente = null;
    Map<Pessoa, Map<Produtor, Integer>> produtoresProximoHub = null;

    if (restriction == 1) {
        hubProximoCliente = expedatingStore.getHubProximoCliente();//O(n^3)
        produtoresProximoHub = expedatingStore.produtoresEachHub( n: 1);//O(n^3)
    }

    List<Cabaz> expedatingList = new ArrayList<>();
    List<InfoProduto> infoProdutoList;
    InfoProduto infoProdutoAuxiliar;
    Produtos produtoCriarObjeto;
    Double valorExpectado;
    Pessoa clientInfo;
```

```
for (Map.Entry<Pessoa, TreeMap<Integer, Map<Produtos, Double>>> entryClient : clientsMap.entrySet()) {//O(n)
    infoProdutoList = new ArrayList<>();
    clientInfo = entryClient.getKey();

    Map<Produtos, Double> clientesCabazez = entryClient.getValue().get(day);
    for (Map.Entry<Produtos, Double> infoProdutoCabaz : clientesCabazez.entrySet()) {//O(n)
        produtoCriarObjeto = infoProdutoCabaz.getKey();
        valorExpectado = infoProdutoCabaz.getValue();

        if (valorExpectado > 0) {
            if (restriction == 0) {
                infoProdutoAuxiliar = findObtainedValueNoRestriction(day, produtoCriarObjeto, valorExpectado, stockMap);//O(n)
            } else {
                Pessoa auxiHub = hubProximoCliente.get(clientInfo).entrySet().iterator().next().getKey();
                Map<Produtor, Integer> auxMap = produtoresProximoHub.get(auxiHub);
                infoProdutoAuxiliar = findObtainedValueRestriction(day, produtoCriarObjeto, valorExpectado, stockMap, auxMap, clientInfo);//O(n)
            }
            if (infoProdutoAuxiliar != null) {
                infoProdutoList.add(infoProdutoAuxiliar);
            } else {
                infoProdutoList.add(new InfoProduto(produtoCriarObjeto.getNomeFruta(), valorExpectado,  received: 0.0,  nomeProdutor: "Produtor Inexistente"));
            }
        }
    }
    expedatingList.add(new Cabaz(clientInfo.getIdPessoa(), day, infoProdutoList));
}
expedatingStore.setExpedatingList(expedatingList);

return expedatingList;
```

Método US308:

```
private InfoProduto findObtainedValueNoRestriction(int day, Produtos produtoFind, Double needed, Map<Produtor,
        TreeMap<Integer, Map<Produtos, Double>>> stockMap) {
    Produtor finalProducer = null;
    double finalQuantity = 0.0;
    double maxQuantity = 0.0;
    Produtor finalProducerr = null;
    int finalDay = 0;

    for (Map.Entry<Produtor, TreeMap<Integer, Map<Produtos, Double>>> producer : stockMap.entrySet()) {//O(n)
        double quantityBeforeYesterday = getQuantityOrDefault(producer, day: day - 2, produtoFind, defaultValue: 0.0);
        double quantityYesterday = getQuantityOrDefault(producer, day: day - 1, produtoFind, defaultValue: 0.0);
        double quantityToday = getQuantityOrDefault(producer, day, produtoFind, defaultValue: 0.0);
        double totalQuantity = quantityBeforeYesterday + quantityYesterday + quantityToday;

        if (quantityBeforeYesterday > maxQuantity) {
            maxQuantity = quantityBeforeYesterday;
            finalProducerr = producer.getKey();
            finalDay = day - 2;
        } else if (quantityYesterday > maxQuantity) {
            maxQuantity = quantityYesterday;
            finalProducerr = producer.getKey();
            finalDay = day - 1;
        } else if (quantityToday > maxQuantity) {
            maxQuantity = quantityToday;
            finalProducerr = producer.getKey();
            finalDay = day;
        }

        if (totalQuantity >= needed) {
            finalProducer = producer.getKey();
            finalQuantity = needed;
            updateStockQuantities(producer, startDay: day - 2, day, produtoFind, quantityToSubtract: totalQuantity - needed);
            break;
        }
    }
```

```
    if (finalProducer != null) {
        return new InfoProduto(produtoFind.getNomeFruta(), needed, finalQuantity, finalProducer.getId());
    }

    if (finalProducerr != null) {
        stockMap.get(finalProducerr).get(finalDay).put(produtoFind, 0.0);//O(logn)
        return new InfoProduto(produtoFind.getNomeFruta(), needed, maxQuantity, finalProducerr.getId());
    }
    return null;
}//O(n)
```

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

O método "expedatingForDay" tem uma complexidade O(n^3), pois getHubProximoCliente e produtoresEachHub tem complexidade de n^3.

O método findObtainedValueRestriction tem complexidade O(N), onde N é o número de produtores no mapa de estoque passado como argumento. Isso é devido ao laço for que itera sobre cada produtor no mapa.

O método "updateStockQuantities" tem uma complexidade O(p), onde "p" é o número de produtores. Isso é devido ao loop na linha 3, que itera sobre cada produtor no mapa de produtores.

O método "getQuantityOrDefault" tem uma complexidade O(1), pois ele apenas realiza uma busca em um mapa e retorna um valor.

Método US309:

Métodos auxiliares 308/309:

```
private void updateStockQuantities(Map.Entry<Produtor, TreeMap<Integer, Map<Produtos, Double>>> producer,
                                   int startDay, int endDay, Produtos produtoFind, double quantityToSubtract) {
    for (int x = startDay; x <= endDay; x++) {//O(k)
        if (producer.getValue().get(x) != null) {
            double currentQuantity = producer.getValue().get(x).get(produtoFind);
            double updatedQuantity = Math.max(currentQuantity - quantityToSubtract, 0);
            producer.getValue().get(x).put(produtoFind, updatedQuantity);
            quantityToSubtract -= currentQuantity;
        }
    }
}//O(1)
```

```
private double getQuantityOrDefault(Map.Entry<Produtor, TreeMap<Integer, Map<Produtos, Double>>> producer,
                                    int day, Produtos produtoFind, double defaultValue) {
    return Optional.ofNullable(producer.getValue().get(day)) Optional<Map<Produtos, Double>>
            .map(m -> m.get(produtoFind)) Optional<Double>
            .orElse(defaultValue);
}//O(1)
```

Métodos Principal us 309:

```
private InfoProduto findObtainedValueRestriction(int day, Produtos produtoFind, Double needed, Map<Produtor,
        TreeMap<Integer, Map<Produtos, Double>>> stockMap, Map<Produtor, Integer> mapRestriction, Pessoa clientInfo) {
    Produtor produtorDoFor = null;
    double finalQuantity = 0.0;
    double maxQuantity = 0.0;
    Produtor finalProducerr = null;
    int finalDay = 0;

    for (Map.Entry<Produtor, TreeMap<Integer, Map<Produtos, Double>>> producer : stockMap.entrySet()) {//O(n)

        if (mapRestriction.get(producer.getKey()) != null) {

            double quantityBeforeYesterday = getQuantityOrDefault(producer, day: day - 2, produtoFind, defaultValue: 0.0);
            double quantityYesterday = getQuantityOrDefault(producer, day: day - 1, produtoFind, defaultValue: 0.0);
            double quantityToday = getQuantityOrDefault(producer, day, produtoFind, defaultValue: 0.0);
            double totalQuantity = quantityBeforeYesterday + quantityYesterday + quantityToday;

            if (quantityBeforeYesterday > maxQuantity) {
                maxQuantity = quantityBeforeYesterday;
                finalProducerr = producer.getKey();
                finalDay = day - 2;
            } else if (quantityYesterday > maxQuantity) {
                maxQuantity = quantityYesterday;
                finalProducerr = producer.getKey();
                finalDay = day - 1;
            } else if (quantityToday > maxQuantity) {
                maxQuantity = quantityToday;
                finalProducerr = producer.getKey();
                finalDay = day;
            }
```

```
            if (totalQuantity >= needed) {
                produtorDoFor = producer.getKey();
                finalQuantity = needed;
                updateStockQuantities(producer, startDay: day - 2, day, produtoFind, quantityToSubtract: totalQuantity - needed);
                break;
            }
        }
    }

    if (produtorDoFor != null) {
        return new InfoProduto(produtoFind.getNomeFruta(), needed, finalQuantity, produtorDoFor.getId());
    }

    if (finalProducerr != null) {
        stockMap.get(finalProducerr).get(finalDay).put(produtoFind, 0.0);
        return new InfoProduto(produtoFind.getNomeFruta(), needed, maxQuantity, finalProducerr.getId());//O(logn)
    }
    return null;
}
```

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

Este método, faz 4 fors todos com complexidade de O(n^2), após os fors usa o algoritmo que cria a matriz com as distâncias mínimas, este método tem complexidade de O(n^3) e por fim para calcular o caminho mínimo que passa em todas as localidades pretendidas para fazer as entregas o código escrito tem também uma complexidade de O(n^3), posto isto o método no total tem uma complexidade de O(n^3).

```
public Map<List<Pessoa>, Integer> expedatingPath(List<Cabaz> auxList, Map<Pessoa, TreeMap<Integer, Map<Produtos, Double>>> clientsMap, int day) {
    Set<Pessoa> auxiliarClientList = clientsMap.keySet(); //Clientes do Dia
    Map<Pessoa, Map<Pessoa, Integer>> hubEachClient = expedatingStore.getHubProximoCliente();

    LinkedList<Pessoa> pessoaLinkedList = new LinkedList<>(); //em forma de objeto os clientes
    //O(n^2)
    for (Cabaz auxiliarPessoa : auxList) {
        for (Pessoa auxiliarClient : auxiliarClientList) {
            if (auxiliarClient.getIdPessoa().equals(auxiliarPessoa.getCliente_Id())) {
                pessoaLinkedList.add(auxiliarClient);
            }
        }
    }

    LinkedList<Pessoa> hubLinkedList = new LinkedList<>();
    //O(n^2)
    for (Pessoa auxiliarPessoa : pessoaLinkedList) {
        Pessoa hub = hubEachClient.get(auxiliarPessoa).entrySet().iterator().next().getKey();
        if (!hubLinkedList.contains(hub)) {
            hubLinkedList.add(hub);
        }
    }

    LinkedList<String> produtores = new LinkedList<>();
    //0(n^2)
    for (Cabaz auxCabaz : auxList) {
        for (InfoProduto infoProduto : auxCabaz.getList()) {
            if (!produtores.contains(infoProduto.getNomeProdutor())) {
                produtores.add(infoProduto.getNomeProdutor());
            }
        }
    }
```

```
    List<Produtor> produtoresList = findMethods.findAllProducers();
    LinkedList<Pessoa> produtoresObjeto = new LinkedList<>();
    //O(n^2)
    for (String aux : produtores) {
        for (Produtor produtor : produtoresList) {
            if (aux.equals(produtor.getId())) {
                produtoresObjeto.add(produtor);
                break;
            }
        }
    }

    LinkedList<Pessoa> total = new LinkedList<>(hubLinkedList);
    total.addAll(produtoresObjeto);

    MapGraph<Pessoa, Integer> graph = mapGraphPessoaStore.getMapGraph();
    MatrixGraph<Pessoa, Integer> matrixGraph = Algorithms.minDistGraph(graph, comparator, binaryOperator);// O(n^3)
```

```
    boolean[] visited;
    int numProducers = produtoresObjeto.size();
    int sum, minDist = 0, sumMinDist = -1;
    List<Pessoa> minPath = new ArrayList<>();
    List<Pessoa> path;
    for (Pessoa pessoa : total) {// O(n)
        if (pessoa instanceof Produtor) {
            path = new ArrayList<>();
            visited = new boolean[total.size()];
            Pessoa vOrig = pessoa;
            path.add(vOrig);
            Pessoa vOrigAux = null;
            int counter = 1;
            sum = 0;
            while (counter < total.size()) {//O(n)
                minDist = -1;
                int vOrigKey = total.indexOf(vOrig);
                visited[vOrigKey] = true;
                for (Edge<Pessoa, Integer> edge : matrixGraph.outgoingEdges(vOrig)) {//O(n)
                    if (counter < numProducers) {
                        Pessoa vDest = edge.getVDest();
                        if (vDest instanceof Produtor) {
                            int vDestKey = total.indexOf(edge.getVDest());
                            if (vDestKey != -1) {
                                if (!visited[vDestKey] && total.contains(vDest)) {
                                    if (edge.getWeight() < minDist || minDist < 0) {
                                        minDist = edge.getWeight();
                                        vOrigAux = vDest;
                                    }
                                }
                            }
                        }
```

```
                    } else if (counter >= numProducers) {
                        Pessoa vDest = edge.getVDest();
                        int vDestKey = total.indexOf(edge.getVDest());
                        if (vDestKey != -1) {
                            if (!visited[vDestKey] && total.contains(vDest)) {
                                if (edge.getWeight() < minDist || minDist < 0) {
                                    minDist = edge.getWeight();
                                    vOrigAux = vDest;
                                }
                            }
                        }
                    }
                }
                sum += minDist;
                vOrig = vOrigAux;
                path.add(vOrig);
                counter++;
            }
            if (sum < sumMinDist || sumMinDist < 0) {
                sumMinDist = sum;
                minPath = path;
            }
        }
    }
    Map<List<Pessoa>, Integer> returnableMap = new HashMap<>();
    returnableMap.put(minPath, minDist);

    return returnableMap;
}//O(n^3)
```

US311 - Para uma lista de expedição calcular estatísticas:

O primeiro for tem complexidade O(n), onde n é o tamanho da lista de cabazes. O segundo for tem complexidade O(m), onde m é o tamanho da lista de produtos em cada cabaz.

Portanto, o método cabazStatistics tem complexidade O(n * m).

O metodo ExpeditionStatistics tem complexidade O(n), pois a lista "statisticsByCabazList" é percorrida apenas uma vez. Em cada iteração, uma lista chamada "produtoresNaoRepetidos" é percorrida, mas essa lista não é dependente do tamanho da lista "statisticsByCabazList", e sim do número de produtores diferentes que aparecem nas estatísticas de cada cesto. Portanto, a complexidade desse código é O(n).

O método produtorStatistics tem complexidade O(n * m), onde n é o número de elementos na lista statisticsbyCabazList e m é o número de elementos na lista produtorListObject. Isso é devido ao loop for que percorre cada elemento da lista statisticsbyCabazList e a chamada aos métodos produtosEsgotados e hubsExistentes, que têm complexidade O(m) e O(p), respectivamente, onde p é o número de elementos na lista listClientObject.

O método hubStatistics tem complexidade O(n^2), pois contém dois loops aninhados que percorrem, respectivamente, listas de tamanho n (statisticsbyCabazList e listClient) e n (listHubs).

O primeiro loop percorre a lista de StatisticsByCabaz e adiciona o nome dos clientes usados a uma lista. O segundo loop percorre essa lista de nomes e a lista de clientes e adiciona cada cliente usado a uma lista de clientes usados.

Depois, há outro loop que percorre a lista de hubs e, para cada hub, conta quantos clientes usados estão associados a ele. Esse loop também tem complexidade O(n), pois percorre uma lista de n hubs. Portanto, a complexidade total é O(n^2).

```java
private List<Pessoa> findClientObjectList(List<Pessoa> listClient, List<String> clientes) {
    List<Pessoa> clientesExistentes = new ArrayList<>();
    for (String clientNome : clientes) {//O(n)
        for (Pessoa client : listClient) {//O(n)
            if (client.getIdPessoa().equals(clientNome)) {
                clientesExistentes.add(client);
                break;
            }
        }
    }
    return clientesExistentes;
}//O(n^2)
```

Métodos auxiliares:

```java
public void printMainStatistics() {
    List<StatisticsByCabaz> statisticsbyCabazList = cabazStatistics();//O(n^2)
    ExpeditionStatistics expeditionStatisticsObject = expeditionStatistics(statisticsbyCabazList);
    List<StatisticsByProdutor> statisticsByProdutorList = produtorStatistics(statisticsbyCabazList, expeditionStatisticsObject);
    List<StatisticsByHub> statisticsByHubList = hubStatistics(statisticsbyCabazList, statisticsByProdutorList);

    for (StatisticsByCabaz cabazprint:statisticsbyCabazList) {//O(n)
        System.out.println(cabazprint);
        System.out.println();

    }
    for (StatisticsByProdutor produtorprint:statisticsByProdutorList) {//O(n)
        System.out.println(produtorprint);
        System.out.println();
    }
    for (StatisticsByHub hubprint:statisticsByHubList) {//O(n)
        System.out.println(hubprint);
        System.out.println();
    }
    System.out.println(expeditionStatisticsObject);//O(1)

}
```

```java
private List<Empresa> hubsExistentes(Map<Pessoa, Map<Pessoa, Integer>> hubClient, List<Pessoa> listClient) {
    List<Empresa> hubs = new ArrayList<>();
    for (Pessoa cliente : listClient) {//O(n)
        Empresa aux = (Empresa) hubClient.get(cliente).entrySet().iterator().next().getKey();
        if (!hubs.contains(aux)) {
            hubs.add(aux);
        }
    }
    return hubs;
}
/O(n)
```

```java
private int produtosEsgotados(Produtor produtor, Map<Produtor, TreeMap<Integer, Map<Produtos, Double>>> stockMap, int dia) {
    Map<Produtos, Double> stockDia = stockMap.get(produtor).get(dia);
    int esgotados = 0;
    for (Map.Entry<Produtos, Double> prr : stockDia.entrySet()) {//O(n)
        if (prr.getValue() == 0) {
            esgotados++;
        }
    }
    return esgotados;
}//O(n)
```

```java
private List<Produtor> findProdutorObjectList(List<Produtor> listProducer, List<String> produtores) {
    List<Produtor> produtoresExistentes = new ArrayList<>();

    for (String produtorNome : produtores) {//O(n)

        for (Produtor produtor : listProducer) {//O(n)
            if (produtor.getIdPessoa().equals(produtorNome)) {
                produtoresExistentes.add(produtor);
                break;
            }
        }
    }
    return produtoresExistentes;
}//O(n^2)
```

Métodos US311:

Método Cabaz Statistics:

```
public List<StatisticsByCabaz> cabazStatistics() {
    List<Cabaz> cabazList = expedatingStore.getExpedatingList();
    int unsatisfiedProduct, parcialSatisfiedProduct, satisfiedProduct;
    List<String> producersByCabaz;
    List<StatisticsByCabaz> statisticsByCabazList = new ArrayList<>();
    for (Cabaz cabaz : cabazList) {//O(n)
        parcialSatisfiedProduct = 0;
        satisfiedProduct = 0;
        unsatisfiedProduct = 0;
        producersByCabaz = new ArrayList<>();
        for (InfoProduto infoProduto : cabaz.getList()) {//O(n*n)
            if (infoProduto.getEstado() == 0) {
                unsatisfiedProduct++;
            } else if (infoProduto.getEstado() == 1) {
                parcialSatisfiedProduct++;
            } else if (infoProduto.getEstado() == 2) {
                satisfiedProduct++;
            }
            if (!producersByCabaz.contains(infoProduto.getNomeProdutor())) {
                producersByCabaz.add(infoProduto.getNomeProdutor());
            }
        }
        statisticsByCabazList.add(new StatisticsByCabaz(satisfiedProduct, parcialSatisfiedProduct, unsatisfiedProduct,
                producersByCabaz.size(), producersByCabaz, cabaz.getCliente_Id(), cabaz.getDia()));
    }
    return statisticsByCabazList;
}//O(n^2)
```

Método Expedition Statistics:

```
public ExpeditionStatistics expeditionStatistics(List<StatisticsByCabaz> statisticsByCabazList) {
    int cabazSatisfeitos = 0;
    int cabazParcialmenteSatisfeito = 0;
    int cabazNaoSatisfeito = 0;
    List<String> produtoresNaoRepetidos = new ArrayList<>();

    for (StatisticsByCabaz statisticsByCabaz : statisticsByCabazList) {//O(n)
        if (statisticsByCabaz.getPercentagemCabazSatisfeito() == 1) {
            cabazSatisfeitos++;
        } else if (statisticsByCabaz.getPercentagemCabazSatisfeito() > 0 && statisticsByCabaz.getPercentagemCabazSatisfeito() < 1) {
            cabazParcialmenteSatisfeito++;
        } else {
            cabazNaoSatisfeito++;
        }
        for (String produtor : statisticsByCabaz.getProducersByCabaz()) {//O(n)
            if (!produtoresNaoRepetidos.contains(produtor)) {
                produtoresNaoRepetidos.add(produtor);
            }
        }
    }

    return new ExpeditionStatistics(statisticsByCabazList.get(0).getDia(),cabazSatisfeitos, cabazParcialmenteSatisfeito,
            cabazNaoSatisfeito, produtoresNaoRepetidos.size(), produtoresNaoRepetidos);
}//O(n^2)
```

Método Produtor Statistics:

```
private List<StatisticsByProdutor> produtorStatistics(List<StatisticsByCabaz> statisticsbyCabazList, ExpeditionStatistics expeditionStatisticsObject) {
    List<StatisticsByProdutor> statisticsByProdutorList = new ArrayList<>();
    List<String> produtores = expeditionStatisticsObject.getProdutoresNaoRepetidos();
    List<String> clientes;
    int cabazTotalmenteFornecido, cabazParcialmenteFornecidos, esgotadoProdutos;
    List<Pessoa> listClient = findMethods.findAllClients();//O(n)
    List<Produtor> listProducer = findMethods.findAllProducers();//O(n)
    Map<Pessoa, Map<Pessoa, Integer>> hubClient = expedatingStore.getHubProximoCliente();//O(n^3)
    Map<Produtor, TreeMap<Integer, Map<Produtos, Double>>> stockMap = mapStore.getstockProdutorMap();
    List<Produtor> produtorListObject = findProdutorObjectList(listProducer, produtores);
    int dia = 0;
    for (Produtor produtor : produtorListObject) {//O(n)
        cabazTotalmenteFornecido = 0;
        cabazParcialmenteFornecidos = 0;
        clientes = new ArrayList<>();

        for (StatisticsByCabaz cabaz : statisticsbyCabazList) {//O(n)
            dia = cabaz.getDia();
            if (cabaz.getProducersByCabaz().contains(produtor.getIdPessoa()) && cabaz.getProducersByCabaz().size() == 1) {
                cabazTotalmenteFornecido++;
                if (!clientes.contains(cabaz.getCliente())) {
                    clientes.add(cabaz.getCliente());
                }
            } else if (cabaz.getProducersByCabaz().contains(produtor.getIdPessoa()) && cabaz.getProducersByCabaz().size() > 1) {
                cabazParcialmenteFornecidos++;
                if (!clientes.contains(cabaz.getCliente())) {
                    clientes.add(cabaz.getCliente());
                }
            }
        }
        List<Pessoa> listClientObject = findClientObjectList(listClient, clientes);//O(n^2)
        esgotadoProdutos = produtosEsgotados(produtor, stockMap, dia);//O(n)
        List<Empresa> hubsFornecidos = hubsExistentes(hubClient, listClientObject);//O(n)

        statisticsByProdutorList.add(new StatisticsByProdutor(produtor,cabazTotalmenteFornecido, cabazParcialmenteFornecidos,
                esgotadoProdutos, hubsFornecidos));

    }

    return statisticsByProdutorList;

}//O(n^3)
```

Método Hub Statistics:

```
private List<StatisticsByHub> hubStatistics(List<StatisticsByCabaz> statisticsbyCabazList, List<StatisticsByProdutor> statisticsByProdutorList) {
    List<StatisticsByHub> statisticsByHubList = new ArrayList<>();
    List<Pessoa> clientsUsed = new ArrayList<>();
    List<Pessoa> listClient = findMethods.findAllClients();//O(n)
    List<Empresa> listHubs = findMethods.findAllHubs();//O(n)
    List<String> listClientName = new ArrayList<>();
    Map<Pessoa, Map<Pessoa, Integer>> hub = expedatingStore.getHubProximoCliente();//O(1)

    for (StatisticsByCabaz cabaz : statisticsbyCabazList) {//O(n)
        if (!listClientName.contains(cabaz.getCliente())) {
            listClientName.add(cabaz.getCliente());
        }
    }

    for (String name : listClientName) {//O(n)
        for (Pessoa pessoa : listClient) {//O(n)
            if (pessoa.getIdPessoa().equals(name)) {
                clientsUsed.add(pessoa);
                break;
            }
        }
    }
```

```
    int pessoas;
    for (Pessoa hubss : listHubs) {//O(n)
        pessoas = 0;
        for (Pessoa pessoaAuxiliar : clientsUsed) {//O(n)
            if (hub.get(pessoaAuxiliar).entrySet().iterator().next().getKey().equals(hubss)) {
                pessoas++;
            }
        }

        statisticsByHubList.add(new StatisticsByHub(hubss, pessoas));
    }

    return statisticsByHubList;
}//O(n^2)
```