

Jorge Cunha nº1200618

Fábio Silva nº1201942

Francisco Sampaio nº 1191576

Filipe Morais nº 1190569

João Sousa nº 1190701

**Relatório de Estruturas de**

**Informação:**

**-> Sprint 2**

Índice

**Capa2**

**Índice2**

**Introdução3**

**Exemplo do Map Graph4**

**US307 - Importar a lista de cabazes5**

**US308 - Gerar uma lista de expedição para um determinado dia que forneça os cabazes sem qualquer restrição quanto aos produtores7**

**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 cliente9**

**US310 - Para uma lista de expedição diária gerar o percurso de entrega que minimiza a distância total percorrida11**

**US311 - Para uma lista de expedição calcular estatísticas:13**

# **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.

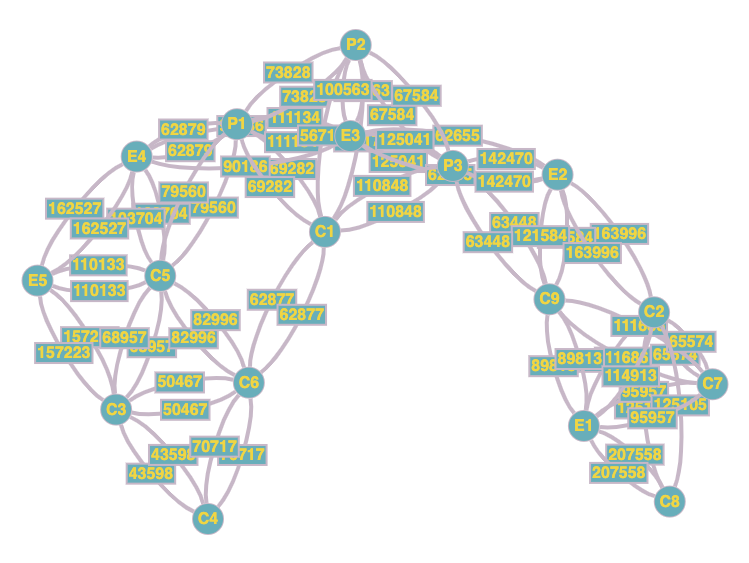
US310: Esta user story pede um percurso de entrega que minimize a distância total percorrida.

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.

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

**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:

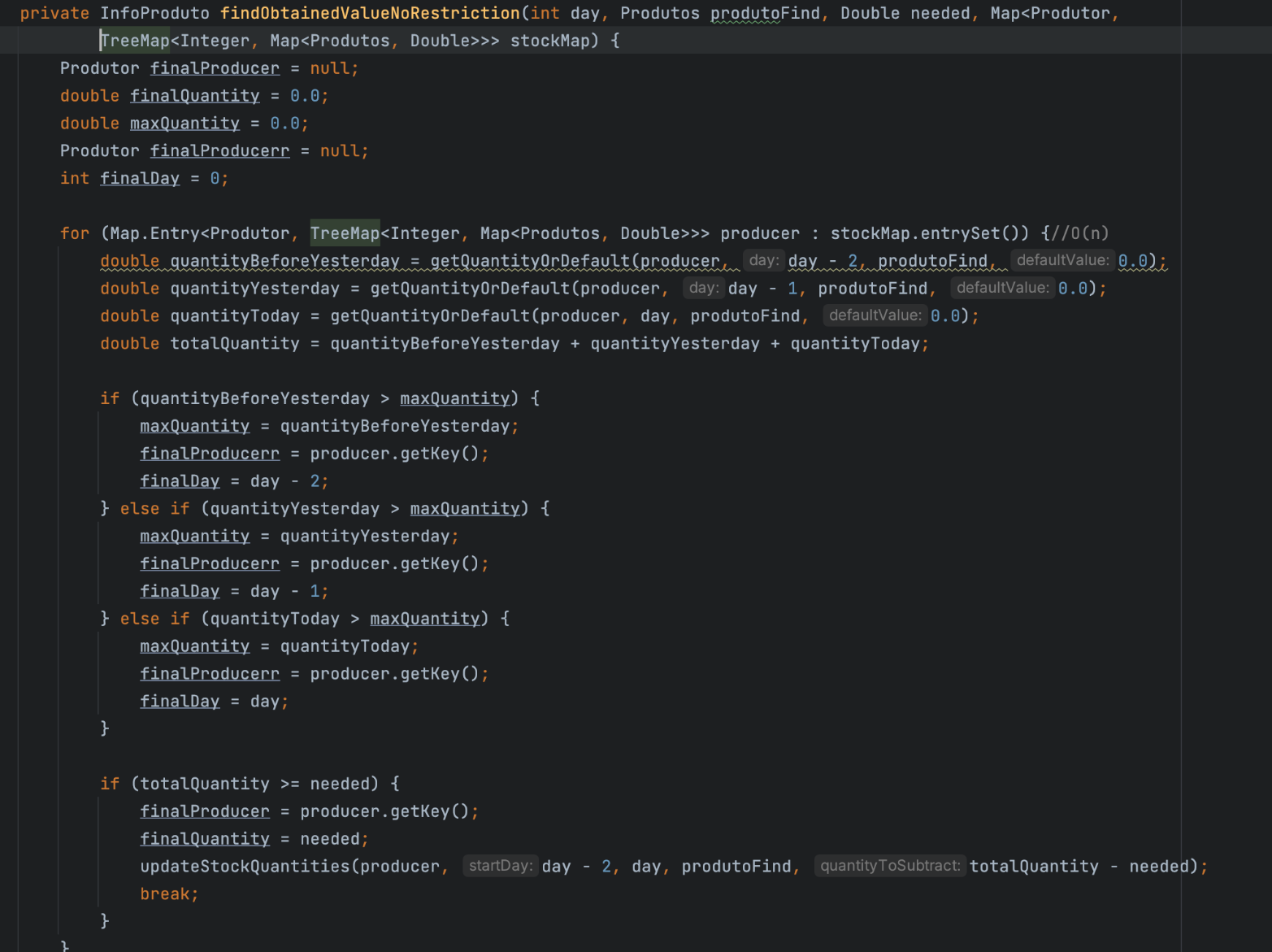
Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Método US308:



Uma imagem com texto

Descrição gerada automaticamente

**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:

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Métodos Principal us 309:

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

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

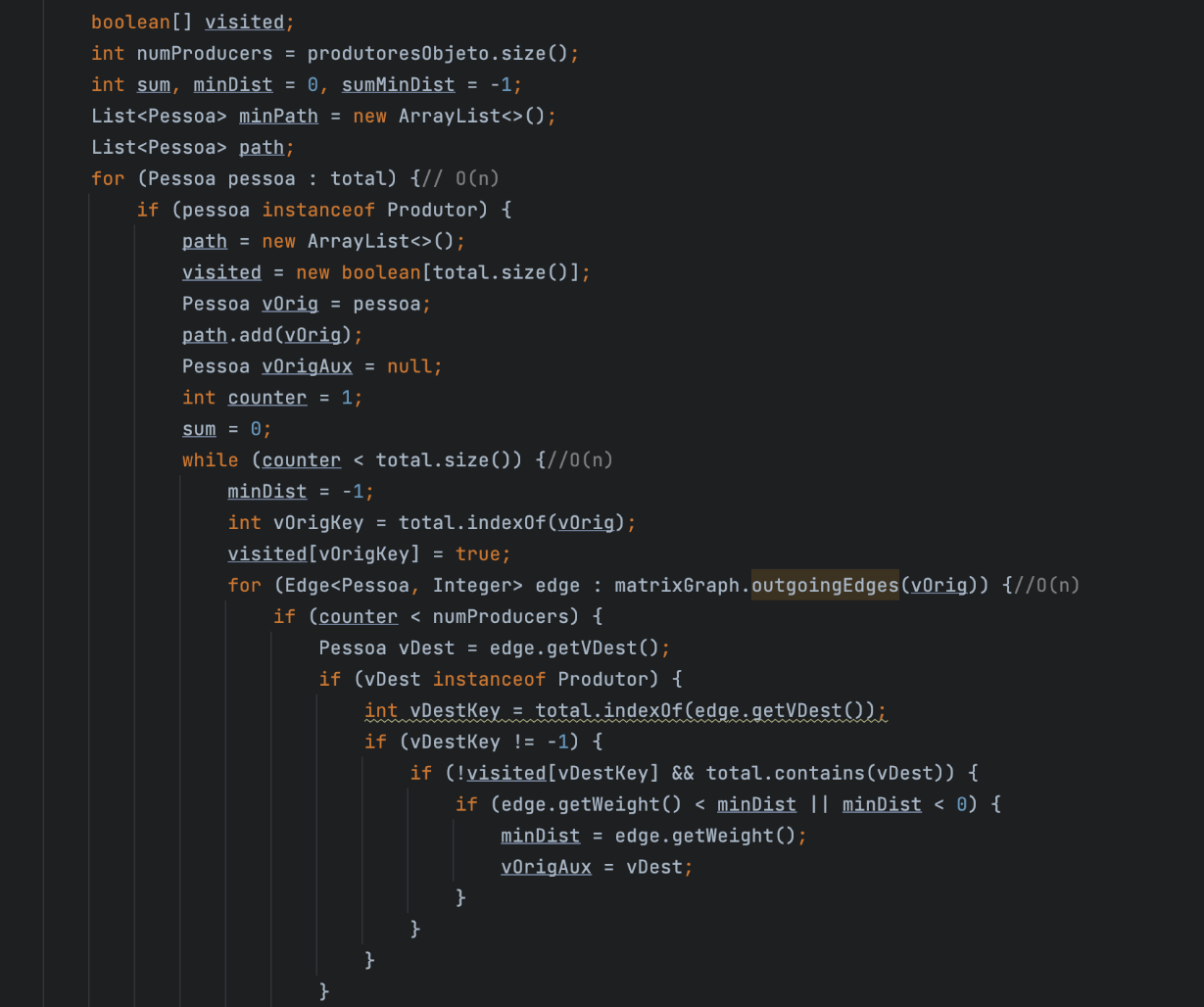
Este método, faz 4 fors todos com complexidade de O(n^2), após os fors usa o algoritmo que cria a matriz com as distâncias mínimas, este método tem complexidade de O(n^3) e por fim para calcular o caminho mínimo que passa em todas as localidades pretendidas para fazer as entregas o código escrito tem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

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente



**Uma imagem com texto

Descrição gerada automaticamente**

**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).

Uma imagem com texto

Descrição gerada automaticamente

Métodos auxiliares:

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente  
Uma imagem com texto

Descrição gerada automaticamente

Métodos US311:

Método Cabaz Statistics:

Uma imagem com texto

Descrição gerada automaticamente

Método Expedition Statistics:

Uma imagem com texto

Descrição gerada automaticamente

Método Produtor Statistics:

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Método Hub Statistics:

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente