

Prepared By:

1201205 - Samuel Dias

1201367 - Carlos Santos

1201416 - Pedro Alves

1211136 - Bruna Costa

1211201 - Henrique Pinto

ESINF SPRINT 2

2022-2023



ÍNDICE

ÍNDICE DE FIGURAS	3
INTRODUÇÃO	4
DESCRIÇÃO	5
DIAGRAMA DE CLASSES	6
MAIN MENU	7
US307	8
DESCRIÇÃO	8
CLASSES DESENVOLVIDAS	8
COMPLEXIDADE	10
US308	11
DESCRIÇÃO	11
CLASSES DESENVOLVIDAS	11
COMPLEXIDADE	16
US309	18
DESCRIÇÃO	18
CLASSES DESENVOLVIDAS	18
COMPLEXIDADE	21
US310	24
DESCRIÇÃO	24
CLASSES DESENVOLVIDAS	24
COMPLEXIDADE	28
US311	31
DESCRIÇÃO	31
CLASSES DESENVOLVIDAS	31
CABAZ	31
CLIENTE	33
PRODUTOR	34
HUB	35
COMPLEXIDADE	37
MELHORIAS	42

ÍNDICE DE FIGURAS

Figura 1 - Diagrama de Classes.....	6
Figura 2 - Menu Inicial	7
Figura 3 - Default Files	8
Figura 4 - LoadFileController.....	8
Figura 5 - BasketLoader: load().....	9
Figura 6 - Complexidade: load().....	10
Figura 7 - ExpeditionListUI	11
Figura 8 - ExpeditionListController.....	12
Figura 9 - ExpeditionListService: getExpeditionList()	14
Figura 10 - ExpeditionListService: setNewStockQuantity()	15
Figura 11 - Complexidade: getExpeditionList()	17
Figura 12 - Complexidade: ExpeditionListController	17
Figura 13 - ExpeditionListToNProducersUI.....	18
Figura 14 - ExpeditionListController.....	19
Figura 15 - ExpeditionListService: getExpeditionList()	21
Figura 16 - Complexidade: ClosestProducersToHubService	21
Figura 17 - Complexidade: ExpeditionListController - getExpeditionList()	22
Figura 18 - Complexidade: ExpeditionListController - getExpeditionListByProducerNumber().....	22
Figura 19 - Complexidade: getExpeditionListService - getExpeditionList()	23
Figura 20 - ExpeditionMinPathUI.....	24
Figura 21 - ExpeditionMinPathController	24
Figura 22 - ExpeditionMinPathService	27
Figura 23 - Complexidade: ExpeditionMinPathService	30
Figura 24 - StatisticsService: cabazStatistics()	33
Figura 25 - StatisticsService: clienteStatistics()	34
Figura 26 - StatisticsService: produtorStatistics()	35
Figura 27 - StatisticsService: hubStatistics()	36
Figura 28 - Complexidade: cabazStatistics()	37
Figura 29 - Complexidade: clienteStatistics().....	38
Figura 30 - Complexidade: produtorStatistics()	40
Figura 31 - Complexidade: hubStatistics()	41

INTRODUÇÃO

O presente relatório apresenta todo o desenvolvimento do SPRINT 2 do Projeto Integrador, no terceiro semestre da Licenciatura em Engenharia Informática no Instituto Superior de Engenharia do Porto, no âmbito da disciplina Estruturas de Informação (ESINF).

O projeto realizado consiste na gestão de uma rede de distribuição de cabazes entre agricultores e clientes. Esta rede gere a distribuição dos produtos dos agricultores de modo a garantir a entrega dos cabazes aos clientes.

Será apresentado o diagrama de classes, com os principais requisitos de domínio e os seus componentes, de modo a contextualizar o projeto e o seu objetivo. Realizar-se-á a análise dos algoritmos de todas as funcionalidades implementadas e da sua complexidade.

Posteriormente, irão ser apresentados possíveis melhoramentos capazes de otimizar a implementação das classes pedidas, garantindo que esta seja a mais eficiente e prática possível.

DESCRIÇÃO

No enunciado do Projeto Integrador, no âmbito da disciplina de Estruturas de Informação (ESINF), pretende-se, com recurso às classes que implementam a interface *Graph*, criar um conjunto de classes e testes que permitam gerir uma rede de distribuição de cabazes entre agricultores e clientes.

Os agricultores/produtores disponibilizam diariamente à rede de distribuição os produtos e respetivas quantidades que têm para vender e os clientes (particulares ou empresas) colocam encomendas (cabazes de produtos agrícolas) à rede de distribuição.

Se a totalidade de um produto disponibilizado por um agricultor para venda, num determinado dia, não for totalmente expedido, fica disponível nos dois dias seguintes, sendo eliminado após esses dois dias. Os cabazes são expedidos num determinado dia, quer sejam totalmente satisfeitos ou não, sendo necessário registar para cada produto a quantidade encomendada, a quantidade entregue e o produtor que forneceu. Cada produto de um cabaz é fornecido por um só produtor, mas um cabaz pode ser fornecido por vários produtores.

A rede gere a distribuição dos produtos dos agricultores de modo a satisfazer os cabazes a serem entregues em *hubs* para posterior levantamento pelos clientes. Um *hub* é localizado numa empresa e cada cliente (particular ou empresa) recolhe as suas encomendas no *hub* mais próximo.

DIAGRAMA DE CLASSES

Para o desenvolvimento do projeto, foi seguida a convenção da arquitetura MVC, dividida em três componentes essenciais: *Model*, *Controller* e *View*, como é possível observar pelo diagrama de classes proposto.

O *Model*, representado pelo *Domain*, é responsável por gerenciar e controlar a forma como os dados se comportam por meio das funções, lógica e regras de negócio estabelecidas.

Utiliza-se a *Store*, de modo a armazenar instâncias de *Domain* e apresentar as funções de *getters/setters* e filtros para obter os resultados pretendidos e o *Service*, para realizar a ligação à *Store* e garantir a implementação das funcionalidades e algoritmos.

O *Controller* é a camada responsável por intermediar as requisições enviadas pelo *View* com as respostas fornecidas pelo *Model*, processando e partilhando os dados que o usuário submeteu.

O *View*, aqui descrito como *UI*, é responsável por garantir a comunicação utilizador-máquina, apresentando as informações de forma visual ao usuário.

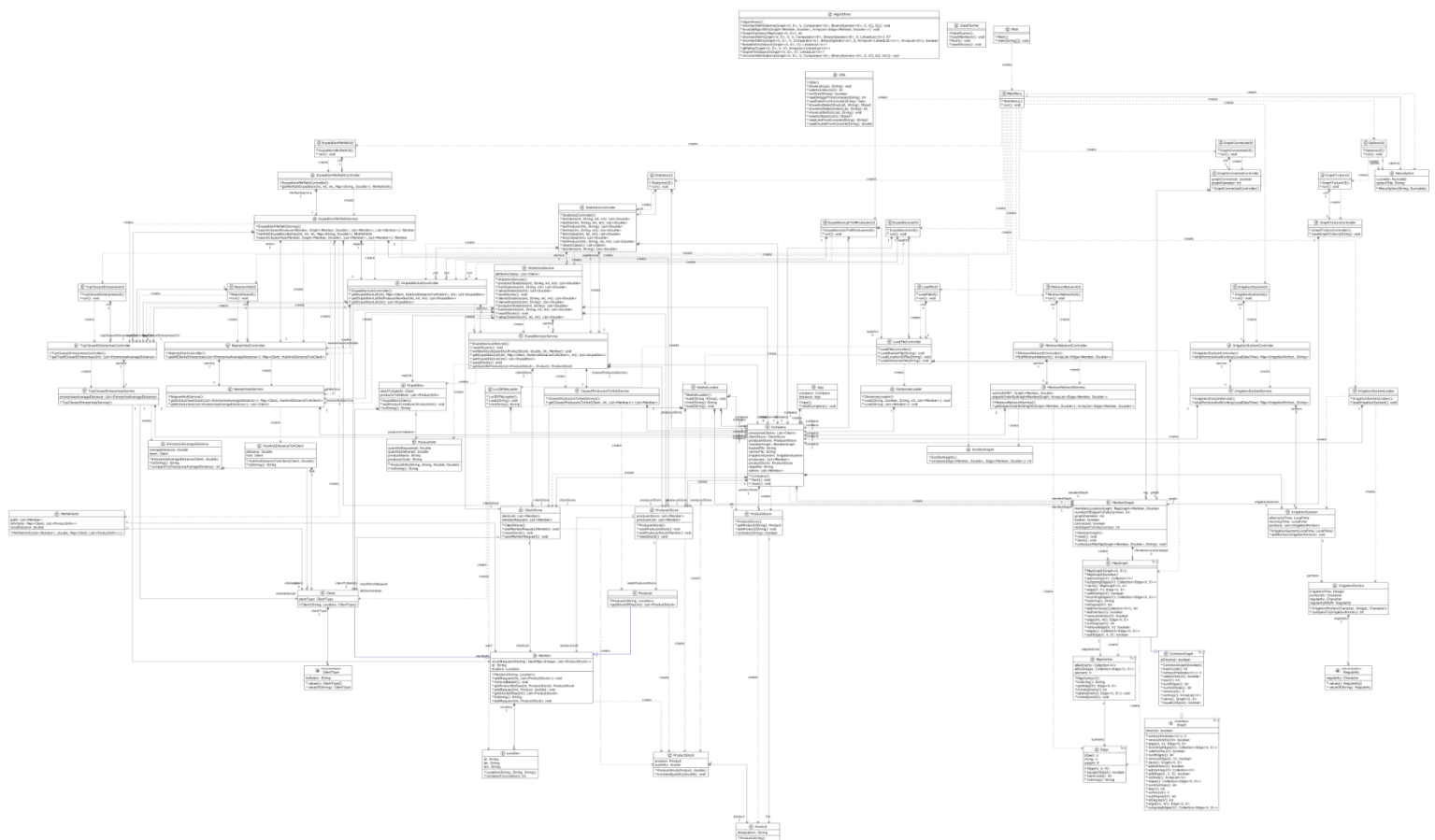
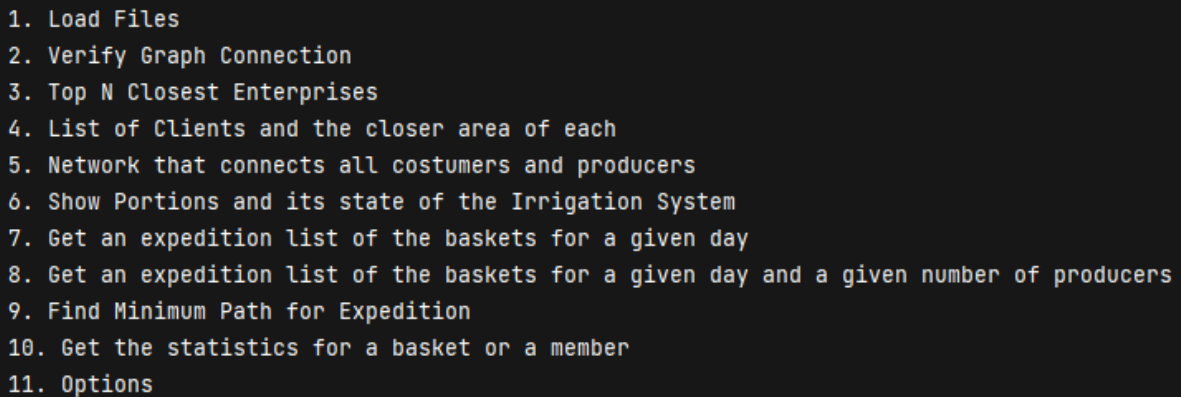


Figura 1 - Diagrama de Classes

MAIN MENU

De modo a organizar o conjunto de classes desenvolvidas para o projeto, foi criado um menu inicial onde o utilizador consegue selecionar todas as principais funcionalidades, cuja ordem corresponde diretamente à ordem dos exercícios propostos. Todos os exercícios dependem que a primeira funcionalidade seja executada.



```
1. Load Files
2. Verify Graph Connection
3. Top N Closest Enterprises
4. List of Clients and the closer area of each
5. Network that connects all costumers and producers
6. Show Portions and its state of the Irrigation System
7. Get an expedition list of the baskets for a given day
8. Get an expedition list of the baskets for a given day and a given number of producers
9. Find Minimum Path for Expedition
10. Get the statistics for a basket or a member
11. Options
```

Figura 2 - Menu Inicial

1. Load Files – constrói a rede de distribuição de cabazes a partir da informação fornecida em ficheiros.
2. Verify Graph Connection – verifica se o grafo carregado é conexo e devolve o número mínimo de ligações.
3. Top N Closest Enterprises – encontra as N empresas mais próximas de todos os pontos da rede.
4. List of Clientes and the closer area of each – determina o hub mais próximo para cada cliente.
5. Network that connects all costumers and producers – determina a rede que conecta todos os clientes e produtores agrícolas com uma distância total mínima.
6. Show Portions and its state of the Irrigation System – esta funcionalidade não está abrangida pela disciplina de ESINF.
7. Get an Expedition list of the baskets for a given day – gera uma lista de expedição para um determinado dia que forneça os cabazes, sem qualquer restrição quanto aos produtores.
8. Get an Expedition list of the baskets for a given day and a given number of producers – gera uma lista de expedição para um determinado dia que forneça apenas com os N produtores agrícolas mais próximos do *hub* de entrega do cliente.
9. Find Minimum Path for Expedition – gera o percurso de entrega que minimiza a distância total percorrida.
10. Get the Statistics for a Basket or a Member – para uma lista de expedição, calcula as estatísticas por cabaz, por cliente, por produtor e por *hub*.
11. Options – permite guardar o grafo atual em um ficheiro json. 0. Exit – permite a saída do programa.

US307

DESCRIÇÃO

Importar uma lista de cabazes.

CLASSES DESENVOLVIDAS

Esta funcionalidade inicia-se após o utilizador seleccionar a primeira opção do Menu Inicial. Nesta fase da implementação, foi carregado um novo tipo distinto de ficheiro, que contém um determinado dia, os três tipos de membros (código cliente: C, código empresa: E, código produtor: P), os pedidos para um cabaz (pedidos pelos dois primeiros membros) e a quantidade de produtos disponíveis (pelos produtores).

```
1 usage
private static final String DEFAULT_LOCATIONS_BIG_FILE = "src/main/resources/Big/clientes-produtores_big.csv";
1 usage
private static final String DEFAULT_DISTANCES_BIG_FILE = "src/main/resources/Big/distancias_big.csv";
1 usage
private static final String DEFAULT_CABAZES_BIG_FILE = "src/main/resources/Big/cabazes_big.csv";

1 usage
private static final String DEFAULT_LOCATIONS_SMALL_FILE = "src/main/resources/Small/clientes-produtores_small.csv";
1 usage
private static final String DEFAULT_DISTANCES_SMALL_FILE = "src/main/resources/Small/distancias_small.csv";
1 usage
private static final String DEFAULT_CABAZES_SMALL_FILE = "src/main/resources/Small/cabazes_small.csv";
```

Figura 3 - Default Files

De modo a garantir uma facilidade na leitura do ficheiro, esta será automática. Ou seja, o caminho que remete à localização do ficheiro já está presente no código e o utilizador não interage diretamente com a máquina, a não ser que pretenda utilizar um ficheiro distinto.

```
public void LoadBasketFile(String filepath) throws InstanceNotFoundException, FileNotFoundException {
    BasketLoader loader = new BasketLoader();
    loader.load(filepath);
}
```

Figura 4 - LoadFileController

O *LoadFileController* contém agora a função *LoadBasketFile()*, que importa o ficheiro pedido, a partir do método *load* na classe *BasketLoader*. Este método guarda os baskets dentro de cada membro do grafo, para otimização.


```

public void load(String filepath, String separator) throws IOException, InstanceNotFoundException {
    if (!memberGraph.isLoaded())
        throw new InstanceNotFoundException("Graph not loaded");

    BufferedReader reader = new BufferedReader(new FileReader(filepath));

    List<String> header = Arrays.asList(reader.readLine().split(separator)).stream().map(s -> trim(s)).toList();
    for (int i = ProductColumnShift; i < header.size(); i++) {
        try {
            ps.addProduct(header.get(i));
        } catch (Exception ignored) {
        }
    }

    int nt = 0, n = 0;
    String lineNotSeparated;
    while ((lineNotSeparated = reader.readLine()) != null) {
        String[] line = lineNotSeparated.split(separator);
        String actor = trim(line[0]);
        int day = Integer.parseInt(trim(line[1]));
        // Find graphMember
        Member member = memberGraph.getMembersLocationGraph().vertices().stream()
            .filter(m -> m.getId().equals(actor)).findFirst().orElse(other: null);
        List<ProductStock> pl = new ArrayList<>();
        for (int i = ProductColumnShift; i < header.size(); i++) {
            pl.add(new ProductStock(ps.getProduct(header.get(i)), Double.parseDouble(line[i])));
        }
        if (member != null) {
            member.addRequest(day, pl);
            if (member instanceof Client) {
                clientStore.addMemberRequest(member);
            }
            if (member instanceof Producer) {
                producerStore.addProducerStock(member);
            }
        } else {
            n--;
        }
        nt++;
        n++;
    }
    System.out.println("\nLoaded " + n + " out of " + nt + " baskets\n");

    this.company.setBasketFile(filepath);
}

```

Figura 5 - BasketLoader: load()

COMPLEXIDADE

```

public void load(String filepath, String separator) throws IOException, InstanceNotFoundException {
    if (!memberGraph.isLoaded())
        throw new InstanceNotFoundException("Graph not loaded");

    BufferedReader reader = new BufferedReader(new FileReader(filepath));

    List<String> header = Arrays.asList(reader.readLine().split(separator)).stream().map(s -> trim(s)).toList();
    for (int i = ProductColumnShift; i < header.size(); i++) {
        try {
            ps.addProduct(header.get(i));
        } catch (Exception ignored) {}
    }

    int nt = 0, n = 0;
    String lineNotSeparated;
    while ((lineNotSeparated = reader.readLine()) != null) {
        String[] line = lineNotSeparated.split(separator);
        String actor = trim(line[0]);
        int day = Integer.parseInt(trim(line[1]));
        // Find graphMember
        Member member = memberGraph.getMembersLocationGraph().vertices().stream()
            .filter(m -> m.getId().equals(actor)).findFirst().orElse(null);
        List<ProductStock> pl = new ArrayList<>();
        for (int i = ProductColumnShift; i < header.size(); i++) {
            pl.add(new ProductStock(ps.getProduct(header.get(i)), Double.parseDouble(line[i])));
        }
        if (member != null) {
            member.addRequest(day, pl);
            if (member instanceof Client) {
                clientStore.addMemberRequest(member);
            }
            if (member instanceof Producer) {
                producerStore.addProducerStock(member);
            }
        } else {
            n--;
        }
        nt++;
        n++;
    }

    System.out.println("\nLoaded " + n + " out of " + nt + " baskets\n");

    this.company.setBasketFile(filepath);
}

```

$O(n)$

$O(n)$

$O(n)$

Final: $O(3n)$

Figura 6 - Complexidade: load()

A complexidade da classe *BasketLoader* é $O(3n)$, o que pode ser simplificado para $O(n)$. Assim sendo, a complexidade final da funcionalidade é $O(n)$.

US308

DESCRIÇÃO

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

CLASSES DESENVOLVIDAS

Esta funcionalidade inicia-se após o utilizador seleccionar a sétima opção do Menu Inicial. A classe *ExpeditionListUI* irá solicitar o dia a ser exibido.

```
brunacosta01 +2
@Override
public void run() {
    try {
        System.out.println("Type the day you want to check the expedition list: ");
        int day = Integer.parseInt(sc.nextLine());

        if (day < 1) {
            System.out.println("Day must be a positive number");
        } else {
            try {
                List<Expedition> result = ctrl.getExpeditionList(day);

                printResult(result);
            } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
            }
        }
    }

    } catch (NumberFormatException nfe) {
        System.out.println("\nDay must be a number\n");
    } catch (Exception e) {
        System.out.println("\n"+e.getMessage()+"\n");
    }
}

1 usage brunacosta01 +1
private void printResult(List<Expedition> result) {
    for (Expedition expedition : result)
        System.out.println(expedition);
}
}
```

Figura 7 - ExpeditionListUI

A classe *ExpeditionListController* contém apenas um método referente a esta funcionalidade. A função *getExpeditionList()* itera os dados obtidos na classe *ExpeditionListService*.

```
public class ExpeditionListController {

    7 usages
    private final ExpeditionListService service;
    2 usages
    private final NearestHubController nearestHubController;
    2 usages
    private final TopClosestEnterprisesController topClosestEnterprisesController;

    4 usages  🧑 brunacosta01 +1
    public ExpeditionListController() {
        this.service = new ExpeditionListService();
        this.nearestHubController = new NearestHubController();
        this.topClosestEnterprisesController = new TopClosestEnterprisesController();
    }

    6 usages  🧑 brunacosta01 +2
    public List<Expedition> getExpeditionList(int day) {
        List<Expedition> result = new ArrayList<>();
        service.saveStocks();

        for (int i = 1; i <= day; i++) {
            result = service.getExpeditionList(i);
        }

        service.resetStocks();

        if(result.size() == 0) {
            throw new IllegalArgumentException("No expeditions found for day " + day);
        }

        return result;
    }
}
```

Figura 8 - ExpeditionListController

Agora, no *Service*, a função *getExpeditionList()* cria uma instância de *Member* chamada de *maxProducer* igualado a *NULL*, uma variável *double* denominada de *maxQuantity* igualada a zero e um *boolean* *isSatisfy* como falso. Agora, começa-se por iterar os membros, criando uma lista com todos os pedidos do cliente no dia escolhido anteriormente. De seguida, itera-se pelo tamanho da lista criada, iterando, agora, os produtores e criando uma lista com todo o estoque de produtos do produtor.

Caso o produtor contenha quantidade suficiente para satisfazer tal pedido, o produto, o *ID* do produtor, a quantidade pedida e a quantidade fornecida (que, neste caso, são iguais) serão adicionados à uma lista de expedições que mais tarde será retornada, transformando

o *boolean* agora para verdadeiro. Se isto não acontecer e, caso o produtor contenha quantidade superior à quantidade máxima criada anteriormente, esta quantidade passa a ser a quantidade de produtos disponíveis e o produtor passa a ser este.

```
public List<Expedition> getExpeditionList (int day) {
    List<Expedition> expeditionList = new ArrayList<>();
    List<Member> requestsList = clientStore.getMemberRequest();
    List<Member> producersList = producerStore.getProducerStore();
    Member maxProducer = null;
    ProductStock maxProductStock = null;
    double maxQuantity = 0;
    boolean isSatisfy = false;

    for (Member member : requestsList) {
        List<ProductStock> listOfRequest = member.getStockOfDay(day);

        if(listOfRequest == null)
            continue;

        Expedition expedition = new Expedition((Client) member);

        for (int i = 0; i < listOfRequest.size(); i++) {
            ProductStock product = listOfRequest.get(i);

            if (product.getQuantity() != 0) {
                for (Member producer : producersList) {
                    List<ProductStock> stockProducer = producer.getStockOfDay(day);
                    ProductStock productStock = getSpecificProduct(stockProducer, product.getProduct());

                    if (product.getQuantity() != 0) {
                        if (productStock.getQuantity() >= product.getQuantity()) {
                            ProductInfo productInfo = new ProductInfo(
                                product.getProduct().getDesignation(),
                                producer.getId(),
                                product.getQuantity(),
                                product.getQuantity()
                            );
                            expedition.getProductsToDeliver().add(productInfo);
                            setNewStockQuantity(productStock, product.getQuantity(), day, producer);
                            product.setQuantity(0);
                            isSatisfy = true;
                        } else {
                            if (productStock.getQuantity() > maxQuantity) {
                                maxQuantity = productStock.getQuantity();
                                maxProducer = producer;
                                maxProductStock = productStock;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Ao verificar que cliente não está satisfeito, ou seja, nenhum produtor consegue fornecer totalmente o produto pedido, o produto máximo, o ID do produtor máximo, a

quantidade pedida e a quantidade fornecida por este produtor (verificando sempre que a quantidade pedida é superior à quantidade fornecida) irão ser adicionados à lista de expedições.

Por fim, caso o produtor continue a *NULL*, ou seja, não existe nenhum produtor com algum estoque daquele produto, este produto do cliente não será fornecido. Ao chegar ao final de todas as iterações, será retornada a lista de expedições.

```
if (product.getQuantity() != 0) {
    if (!isSatisfy && maxQuantity != 0) {
        ProductInfo productInfo = new ProductInfo(
            product.getProduct().getDesignation(),
            maxProducer.getId(),
            product.getQuantity(),
            maxQuantity
        );
        expedition.getProductsToDeliver().add(productInfo);
        setNewStockQuantity(maxProductStock, maxQuantity, day, maxProducer);
        product.setQuantity(product.getQuantity() - maxQuantity);
        product.setQuantity(0);
    }
    if (!isSatisfy && maxProducer == null) {
        ProductInfo productInfo = new ProductInfo(
            product.getProduct().getDesignation(),
            producerCode: "No Producer",
            product.getQuantity(),
            quantityDelivered: 0.0
        );
        expedition.getProductsToDeliver().add(productInfo);
        product.setQuantity(0);
    }
}
maxProducer = null;
maxQuantity = 0;
isSatisfy = false;
}
expeditionList.add(expedition);
}

return expeditionList;
}
```

Figura 9 - ExpeditionListService: getExpeditionList()

Durante as iterações, é invocada a função *setNewStockQuantity()*, já que os produtos do estoque podem ficar disponíveis durante três dias (o dia atual e os próximos dois dias). Assim, caso o dia seja igual a dois, é necessário atualizar o estoque, adicionado os produtos do primeiro dia. Se o dia for igual ou superior a três, os dois dias anteriores serão realizados e atualizados antes.


```
public void setNewStockQuantity(ProductStock product, double quantity, int day, Member producer){
    if (day == 1) {
        if(product.getQuantity() - quantity < 0) {
            product.setQuantity(0);
        } else {
            product.setQuantity(product.getQuantity() - quantity);
        }
    }

    if (day == 2) {
        ProductStock productDayOne = producer.getProductByDay(day: day - 1, product);
        ProductStock productDayTwo = producer.getProductByDay(day, product);

        if (productDayOne.getQuantity() - quantity < 0) {
            quantity = quantity - productDayOne.getQuantity();
            productDayOne.setQuantity(0);
            productDayTwo.setQuantity(productDayTwo.getQuantity() - quantity);
        } else {
            quantity = productDayOne.getQuantity() - quantity;
            productDayOne.setQuantity(quantity);
        }
    }

    if (day >= 3) {
        ProductStock productDayOne = producer.getProductByDay(day: day - 2, product);
        ProductStock productDayTwo = producer.getProductByDay(day: day - 1, product);
        ProductStock productDayThree = producer.getProductByDay(day, product);

        if (productDayOne.getQuantity() - quantity < 0) {
            quantity = quantity - productDayOne.getQuantity();

            productDayOne.setQuantity(0);

            if (productDayTwo.getQuantity() - quantity < 0) {
                quantity = quantity - productDayTwo.getQuantity();
                productDayTwo.setQuantity(0);

                if (productDayThree.getQuantity() - quantity < 0) {
                    productDayThree.setQuantity(0);
                } else {
                    productDayThree.setQuantity(productDayThree.getQuantity() - quantity);
                }
            } else {
                quantity = productDayTwo.getQuantity() - quantity;
                productDayTwo.setQuantity(quantity);
            }
        } else {
            quantity = productDayOne.getQuantity() - quantity;
            productDayOne.setQuantity(quantity);
        }
    }
}
```

Figura 10 - ExpeditionListService: setNewStockQuantity()

COMPLEXIDADE

```

public List<Expedition> getExpeditionList (int day) {
    List<Expedition> expeditionList = new ArrayList<>();
    List<Member> requestsList = clientStore.getMemberRequest();
    List<Member> producersList = producerStore.getProducerStore();
    Member maxProducer = null;
    ProductStock maxProductStock = null;    O(1)
    double maxQuantity = 0;
    boolean isSatisfy = false;

    for (Member member : requestsList) {    O(n)
        List<ProductStock> listOfRequest = member.getStockOfDay(day);

        if(listOfRequest == null)            O(1) * O(n) = O(n)
            continue;

        Expedition expedition = new Expedition((Client) member);

        for (int i = 0; i < listOfRequest.size(); i++) {    O(n) * O(n) = O(n^2)
            ProductStock product = listOfRequest.get(i);    O(n^2) * O(n) = O(n^3)

            if (product.getQuantity() != 0) {
                for (Member producer : producersList) {
                    List<ProductStock> stockProducer = producer.getStockOfDay(day);
                    ProductStock productStock = getSpecificProduct(stockProducer, product.getProduct());

                    if (product.getQuantity() != 0) {
                        if (productStock.getQuantity() >= product.getQuantity()) {
                            ProductInfo productInfo = new ProductInfo(
                                product.getProduct().getDesignation(),
                                producer.getId(),
                                product.getQuantity(),
                                product.getQuantity()
                            );
                            expedition.getProductsToDeliver().add(productInfo);
                            setNewStockQuantity(productStock, product.getQuantity(), day, producer);
                            product.setQuantity(0);
                            isSatisfy = true;
                        } else {
                            if (productStock.getQuantity() > maxQuantity) {
                                maxQuantity = productStock.getQuantity();
                                maxProducer = producer;
                                maxProductStock = productStock;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

$O(1) * O(n^3) = O(n^3)$

```

    }
    if (product.getQuantity() != 0) {
        if (!isSatisfy && maxQuantity != 0) {
            ProductInfo productInfo = new ProductInfo(
                product.getProduct().getDesignation(),
                maxProducer.getId(),
                product.getQuantity(),
                maxQuantity
            );
            expedition.getProductsToDeliver().add(productInfo);
            setNewStockQuantity(maxProductStock, maxQuantity, day, maxProducer);
            product.setQuantity(product.getQuantity() - maxQuantity);
            product.setQuantity(0);
        }
        if (!isSatisfy && maxProducer == null) {
            ProductInfo productInfo = new ProductInfo(
                product.getProduct().getDesignation(),
                producerCode: "No Producer",
                product.getQuantity(),
                quantityDelivered: 0.0
            );
            expedition.getProductsToDeliver().add(productInfo);
            product.setQuantity(0);
        }
    }
    maxProducer = null;
    maxQuantity = 0;
    isSatisfy = false;
    expeditionList.add(expedition);
}
return expeditionList;

```

$O(1) * O(n^2) = O(n)$

$O(1) * O(n) = O(n)$

$O(1)$

Final: $O(n^3)$

Figura 11 - Complexidade: `getExpeditionList()`

```

public List<Expedition> getExpeditionList(int day) {
    List<Expedition> result = new ArrayList<>();
    service.saveStocks();
    for (int i = 1; i <= day; i++) {
        result = service.getExpeditionList(i);
    }
    service.resetStocks();
    if(result.size() == 0) {
        throw new IllegalArgumentException("No expeditions found for day " + day);
    }
    return result;
}

```

$O(1)$

$O(n^3)$ (this is only used in order to not change import values)

$O(n)$

$O(n) * O(n^3) = O(n^4)$

$O(1)$

$O(1)$

Final: $O(n^4)$

Figura 12 - Complexidade: `ExpeditionListController`

A complexidade das classes *ExpeditionListService* e *ExpeditionListController* é $O(n^3)$ e $O(n^4)$, respetivamente. Assim sendo, a complexidade final da funcionalidade é $O(n^4)$.

US309

DESCRIÇÃO

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

CLASSES DESENVOLVIDAS

Esta funcionalidade inicia-se após o utilizador seleccionar a oitava opção do Menu Inicial. A classe *ExpeditionListToNProducersUI* irá solicitar o dia a ser exibido, o número de hubs e o número de produtores a serem considerados.

```
@Override
public void run() {
    try {
        System.out.println("Type the day you want to check the expedition list: ");
        int day = askNumber("Day");

        System.out.println("Type the number of hubs you want to define: ");
        int numberOfHubsDefined = askNumber("Number of hubs");

        System.out.println("Type the number of producers that will distribute: ");
        int numberOfProducers = askNumber("Number of producers");

        if (day < 1) {
            throw new IllegalArgumentException("Day must be a positive number");
        }

        List<Expedition> result = ctrl.getExpeditionListByProducerNumber(day, numberOfHubsDefined, numberOfProducers);

        printResult(result);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

Figura 13 - *ExpeditionListToNProducersUI*

Utilizou-se o mesmo Controller do usado anteriormente, adicionando duas funções extras: *getExpeditionList()*, que itera os dados obtidos na *ExpeditionListService* e *getExpeditionListByProducerNumber()*, que retorna os resultados considerando os N produtores solicitados.

```
1 usage Henrique +1
public List<Expedition> getExpeditionList(int day, Map<Client, HubAndDistanceToAClient> clientAndItsClosestHub, int numberOfProducers) {
    List<Expedition> result = new ArrayList<>();
    service.saveStocks();

    for (int i = 1; i <= day; i++) {
        //TODO getExpeditionList with number and clientClosestToHub
        result = service.getExpeditionList(i, clientAndItsClosestHub, numberOfProducers);
    }

    service.resetStocks();

    if(result.size() == 0) {
        throw new IllegalArgumentException("No expeditions found for day " + day);
    }

    return result;
}

2 usages Henrique +1
public List<Expedition> getExpeditionListByProducerNumber(int day, int numberOfHubsDefined, int numberOfProducers) {
    List<EnterpriseAverageDistance> topNClosestEnterprises = topClosestEnterprisesController.getTopNClosestEnterprises(numberOfHubsDefined);
    Map<Client, HubAndDistanceToAClient> clientAndItsClosestHub = nearestHubController.getAllClientsEnterprises(topNClosestEnterprises);
    List<Expedition> result = getExpeditionList(day, clientAndItsClosestHub, numberOfProducers);

    return result;
}
```

Figura 14 - ExpeditionListController

Agora, no *ExpeditionListService*, foi adicionada a função *getExpeditionList()*, que começa por verificar se o número de produtores inseridos consta nos dados da aplicação, ou seja, se o número é menor ou igual ao número de produtores existentes na *Company* e se este valor é um número positivo. É criada uma instância de *Member* chamada de *maxProducer* igualado a *NULL*, uma variável *double* denominada de *maxQuantity* igualada a zero e um *boolean* *isSatisfy* como falso.

Agora, começa-se por iterar os membros, criando uma lista com todos os pedidos do cliente no dia escolhido anteriormente e será obtido o *hub* mais próximo do tal cliente. Se este *hub* for *NULL*, considera-se que o *hub* mais próximo do cliente é o próprio cliente. De seguida, é obtido o *hub* dos produtores mais próximos.

Agora, itera-se pelo tamanho da lista criada, iterando, agora, os produtores e criando uma lista com todo o estoque de produtos do produtor.

Caso o produtor contenha quantidade suficiente para satisfazer tal pedido, o produto, o *ID* do produtor, a quantidade pedida e a quantidade fornecida (que, neste caso, são iguais) serão adicionados à uma lista de expedições que mais tarde será retornada, transformando o *boolean* agora para verdadeiro. Se isto não acontecer e, caso o produtor contenha quantidade superior à quantidade máxima criada anteriormente, esta quantidade passa a ser a quantidade de produtos disponíveis e o produtor passa a ser este.

```

for (Member member : requestsList) {
    List<ProductStock> listOfRequest = member.getStockOfDay(day);

    if(listOfRequest == null)
        continue;

    Expedition expedition = new Expedition((Client) member);

    HubAndDistanceToAClient hubAndDistanceToThisClient = clientAndItsClosestHub.get(member);

    Client closestHub;

    if(hubAndDistanceToThisClient == null)
        closestHub = (Client) member;
    else
        closestHub = hubAndDistanceToThisClient.getHub();

    producersList = closestProducersService.getClosestProducersToHub(closestHub, numberOfProducers, producersStock);

    for (int i = 0; i < listOfRequest.size(); i++) {
        ProductStock product = listOfRequest.get(i);

        if (product.getQuantity() != 0) {

            for (Member producer : producersList) {
                List<ProductStock> stockProducer = producer.getStockOfDay(day);
                ProductStock productStock = getSpecificProduct(stockProducer, product.getProduct());

                if (product.getQuantity() != 0) {
                    if (productStock.getQuantity() >= product.getQuantity()) {
                        ProductInfo productInfo = new ProductInfo(product.getProduct().getDesignation(), producer.getId(), product.getQuantity(), product.getQuantity());
                        expedition.getProductsToDeliver().add(productInfo);
                        setNewStockQuantity(productStock, product.getQuantity(), day, producer);
                        product.setQuantity(0);
                        isSatisfy = true;
                    } else {
                        if (productStock.getQuantity() > maxQuantity) {
                            maxQuantity = productStock.getQuantity();
                            maxProducer = producer;
                            maxProductStock = productStock;
                        }
                    }
                }
            }
        }
    }
}

```

Ao verificar que cliente não está satisfeito, ou seja, nenhum produtor consegue fornecer totalmente o produto pedido, o produto máximo, o ID do produtor máximo, a quantidade pedida e a quantidade fornecida por este produtor (verificando sempre que a quantidade pedida é superior à quantidade fornecida) irão ser adicionados à lista de expedições.

Por fim, caso o produtor continue a *NULL*, ou seja, não existe nenhum produtor com algum estoque daquele produto, este produto do cliente não será fornecido. Ao chegar ao final de todas as iterações, será retornada a lista de expedições.


```

    if (product.getQuantity() != 0) {
        if (!isSatisfy && maxQuantity != 0) {
            ProductInfo productInfo = new ProductInfo(product.getProduct().getDesignation(), maxProducer.getId(), product.getQuantity(), maxQuantity);
            expedition.getProductToDeliver().add(productInfo);
            setNewStockQuantity(maxProductStock, maxQuantity, day, maxProducer);
            product.setQuantity(product.getQuantity() - maxQuantity);
            product.setQuantity(0);
        }
        if (!isSatisfy && maxProducer == null) {
            ProductInfo productInfo = new ProductInfo(product.getProduct().getDesignation(), producerCode, "No Producer", product.getQuantity(), quantityDelivered: 0.0);
            expedition.getProductToDeliver().add(productInfo);
            product.setQuantity(0);
        }
    }
    maxProducer = null;
    maxQuantity = 0;
    isSatisfy = false;
}
expeditionList.add(expedition);
}
return expeditionList;
}

```

Figura 15 - ExpeditionListService: getExpeditionList()

COMPLEXIDADE

```

public List<Member> getClosestProducersToHub(Client hub, int numberOfProducers, List<Member> producers) {
    Map<Member, Double> closestProducersToHub = new HashMap<>();
    ArrayList<LinkedList<Member>> shortPaths = new ArrayList<>();
    ArrayList<Double> distances = new ArrayList<>();    O(1)

    Algorithms.shortestPaths(
        this.company.getMemberGraph().getMembersLocationGraph(),
        hub,
        Double::compare,
        Double::sum,
        zero: 0.0,
        shortPaths,
        distances    O(n log(n))
    );

    for(Member producer : producers) {
        closestProducersToHub.put(
            producer,
            distances.get(this.company.getMemberGraph().getMembersLocationGraph().key(producer))
        );
    }    O(n)

    Stream<Map.Entry<Member, Double>> producerStream = closestProducersToHub.entrySet().stream(); O(1)
    producerStream = producerStream.sorted(Map.Entry.comparingByValue()); O(n log(n))

    List<Member> closestProducers = new ArrayList<>();    O(1)

    for(Map.Entry<Member, Double> entry : producerStream.collect(Collectors.toList())) {
        closestProducers.add(entry.getKey());    O(n)
    }

    return closestProducers.subList(0, numberOfProducers); O(1)
}

```

Final: O(n log(n))

Figura 16 - Complexidade: ClosestProducersToHubService

```

public List<Expedition> getExpeditionList(int day, Map<Client, HubAndDistanceToAClient> clientAndItsClosestHub, int numberOfProducers) {
    List<Expedition> result = new ArrayList<>(); O(1)
    service.saveStocks(); O(n^3) (used in order to not change imported values with the calculus)

    for (int i = 1; i <= day; i++) {
        result = service.getExpeditionList(i, clientAndItsClosestHub, numberOfProducers); O(n) * O(n^3) = O(n^4)
    }

    service.resetStocks(); O(n) (used in order to not change imported values with the calculus)

    if(result.size() == 0) {
        throw new IllegalArgumentException("No expeditions found for day " + day); O(1)
    }

    return result; O(1)
}

```

Final: $O(n^4)$

Figura 17 - Complexidade: ExpeditionListController - getExpeditionList()

```

public List<Expedition> getExpeditionListByProducerNumber(int day, int numberOfHubsDefined, int numberOfProducers) {
    List<EnterpriseAverageDistance> topNClosestEnterprises = topClosestEnterprisesController.getTopNClosestEnterprises(numberOfHubsDefined);
    Map<Client, HubAndDistanceToAClient> clientAndItsClosestHub = nearestHubController.getAllClientsEnterprises(topNClosestEnterprises);
    List<Expedition> result = getExpeditionList(day, clientAndItsClosestHub, numberOfProducers);
    return result; O(1)
}

```

$O(n^2 \log(n))$
 $O(n^2 \log(n))$
 $O(n^4)$
 Final: $O(n^4)$

Figura 18 - Complexidade: ExpeditionListController - getExpeditionListByProducerNumber()

```

public List<Expedition> getExpeditionList (int day, Map<Client, HubAndDistanceToAClient> clientAndItsClosestHub, int numberOfProducers) {
    List<Member> producersStock = producerStore.getProducerStore();

    if(numberOfProducers > producersStock.size()) {
        throw new IllegalArgumentException("Number of producers input is greater than the number of producers in the company\nTry again");
    } else if(numberOfProducers <= 0) {
        throw new IllegalArgumentException("Number of producers input has to be positive\nTry again");
    }

    List<Expedition> expeditionList = new ArrayList<>();

    List<Member> requestsList = clientStore.getMemberRequest();
    List<Member> producersList = null;

    Member maxProducer = null;
    ProductStock maxProductStock = null;
    double maxQuantity = 0;
    boolean isSatisfy = false;

    for (Member member : requestsList) { O(n)
        List<ProductStock> listOfRequest = member.getStockOfDay(day);

        if(listOfRequest == null)
            continue;

        Expedition expedition = new Expedition((Client) member);

        HubAndDistanceToAClient hubAndDistanceToThisClient = clientAndItsClosestHub.get(member);
        Client closestHub;

        if(hubAndDistanceToThisClient == null)
            closestHub = (Client) member;
        else
            closestHub = hubAndDistanceToThisClient.getHub();

        producersList = closestProducersService.getClosestProducersToHub(closestHub, numberOfProducers, producersStock);
    }
}

```

$O(1)$
 $O(n) * O(1) = O(n)$
 $O(n) * O(n \log(n)) = O(n^2 \log(n))$

The image shows a C# code snippet with several complexity annotations. The code is as follows:

```

for (int i = 0; i < listOfRequest.size(); i++) {
    ProductStock product = listOfRequest.get(i);

    if (product.getQuantity() != 0) {
        for (Member producer : producersList) {
            List<ProductStock> stockProducer = producer.getStockOfDay(day);
            ProductStock productStock = getSpecificProduct(stockProducer, product.getProduct());

            if (product.getQuantity() != 0) {
                if (productStock.getQuantity() >= product.getQuantity()) {
                    ProductInfo productInfo = new ProductInfo(product.getProduct().getDesignation(), producer.getId(), product.getQuantity(), product.getQuantity());
                    expedition.getProductsToDeliver().add(productInfo);
                    setNewStockQuantity(productStock, product.getQuantity(), day, producer);
                    product.setQuantity(0);
                    isSatisfy = true;
                } else {
                    if (productStock.getQuantity() > maxQuantity) {
                        maxQuantity = productStock.getQuantity();
                        maxProducer = producer;
                        maxProductStock = productStock;
                    }
                }
            }
        }

        if (product.getQuantity() != 0) {
            if (!isSatisfy && maxQuantity != 0) {
                ProductInfo productInfo = new ProductInfo(product.getProduct().getDesignation(), maxProducer.getId(), product.getQuantity(), maxQuantity);
                expedition.getProductsToDeliver().add(productInfo);
                setNewStockQuantity(maxProductStock, maxQuantity, day, maxProducer);
                product.setQuantity(product.getQuantity() - maxQuantity);
                product.setQuantity(0);
            }

            if (!isSatisfy && maxProducer == null) {
                ProductInfo productInfo = new ProductInfo(product.getProduct().getDesignation(), producerCode: "No Producer", product.getQuantity(), quantityDelivered: 0.0);
                expedition.getProductsToDeliver().add(productInfo);
                product.setQuantity(0);
            }

            maxProducer = null;
            maxQuantity = 0;
            isSatisfy = false;
        }

        expeditionList.add(expedition);
    }
}

return expeditionList;

```

Complexity annotations in the image:

- $O(n) * O(n) = O(n^2)$ (next to the first loop)
- $O(n^2) * O(1) = O(n^2)$ (next to the if condition)
- $O(n) * O(n^2) = O(n^3)$ (next to the inner loop)
- $O(1) * O(n^3) = O(n^3)$ (next to the inner loop body)
- $O(1) * O(n^2) = O(n^2)$ (next to the if condition inside the inner loop)
- $O(1) * O(n) = O(n)$ (next to the expeditionList.add line)
- Final: $O(n^3)$ (at the bottom right)

Figura 19 - Complexidade: *getExpeditionListService - getExpeditionList()*

A complexidade das classes *ExpeditionListController* e *ExpeditionListService* é $O(n^4)$ e $O(n^3)$, respetivamente. Assim sendo, a complexidade final da funcionalidade é $O(n^4)$.

US310

DESCRIÇÃO

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

CLASSES DESENVOLVIDAS

Esta funcionalidade inicia-se após o utilizador selecionar a nona opção do Menu Inicial. A classe *ExpeditionMinPathUI* solicitará o dia da expedição, o número de *hubs* e o número de produtores. Caso o utilizador pretenda selecionar todos os produtores, basta escolher o número “-1”.

```
public class ExpeditionMinPathUI implements Runnable {

    1 usage
    ExpeditionMinPathController ctrl = new ExpeditionMinPathController();

    1201367_CarlosSantos +1
    @Override
    public void run() {
        try {
            int day = Utils.readIntegerFromConsole( prompt: "Day of the expedition: ");
            int nhubs = Utils.readIntegerFromConsole( prompt: "Number of Hubs to consider: ");
            int nprod = Utils.readIntegerFromConsole( prompt: "Number of Producers to consider (-1 for all): ");

            Map<String, Double> originAndDestiny = new HashMap<>();

            MinPathInfo minPathInfo = ctrl.getMinPathExpedition(day, nhubs, nprod, originAndDestiny);

            printMinPathInfo(minPathInfo, originAndDestiny);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Figura 20 - ExpeditionMinPathUI

A UI então envia toda a informação para o *MinPathExpeditionController*. Este contém a função *getMinPathExpedition()*, que retorna uma *MinPathInfo* a partir do *MinPathExpeditionService*.

```
public class ExpeditionMinPathController {

    1 usage
    private ExpeditionMinPathService minPathService = new ExpeditionMinPathService();

    2 usages 1 Henrique +1
    public MinPathInfo getMinPathExpedition(int day, int numberOfHubs, int numberOfProducers, Map<String, Double> originAndDestiny) {
        return minPathService.minPathExpeditionByDay(day, numberOfHubs, numberOfProducers, originAndDestiny);
    }
}
```

Figura 21 - ExpeditionMinPathController

Segue-se a função *minPathExpeditionByDay* da classe *ExpeditionMinPathService*. Esta função calcula o caminho mínimo para entrega dos cabazes num dia. Em primeiro lugar, procura-se definir o primeiro produtor do caminho. Em cada iteração até ao número de

produtores definidos, é utilizada uma *BreadthFirstSearch* capaz de encontrar o produtor mais próximo. Assim, é utilizado o algoritmo de *Dijkstra* para calcular o melhor caminho até ele.

Agora, itera-se pelo número de hubs, voltando a ser utilizada uma *BreadthFirstSearch* para encontrar o hub mais próximo e o algoritmo de *Dijkstra* para calcular o melhor caminho. No final, a função retorna uma instância da classe *MinPathInfo*.

```
public MinPathInfo minPathExpeditionByDay(int day, int numberOfHubs, int numberOfProducers, Map<String, Double> originAndDestiny) {

    List<Expedition> expeditions = numberOfProducers==1?expCtrl.getExpeditionList(day):expCtrl.getExpeditionListByProducerNumber(day, numberOfHubs, numberOfProducers);
    List<Member> producersInvolved = getProducersOfExpeditions(expeditions);
    List<Member> hubsInvolved = getHubsInvolved(numberOfHubs);

    Double distance = Double.valueOf(0.0);

    List<Map.Entry<String, Double>> originAndDestinyList = new ArrayList<>();

    // Choose Starting Producer
    Member p1 = producersInvolved.get(0);

    LinkedList<Member> path = new LinkedList<>();
    path.add(p1);

    List<Member> passed = new ArrayList<>();
    passed.add(p1);

    /**
     * Find the closest Producers and the paths between them until all producer have been covered
     */
    List<Member> piClone = new ArrayList<>(producersInvolved);
    while (piClone.size()>0) {

        Member closest = searchClosestProducer(path.getLast(), memberGraph.getMembersLocationGraph(), producersInvolved, passed);

        LinkedList<Member> pathToClosest = new LinkedList<>();

        Double dist = Algorithms.shortestPath(
            memberGraph.getMembersLocationGraph(),
            path.getLast(),
            closest,
            Double::compare,
            Double::sum,
            Double.sum(0.0, 0.0),
            pathToClosest
        );

        if(dist != null)
            originAndDestiny.put(path.getLast().getId() + "-" + closest.getId(), dist);

        distance = distance + (dist==null?0:dist);
    }
}
```

```
piClone.remove(path.getLast());

// Add pathToClosest to path
for (int j = 0; j < pathToClosest.size(); j++) {
    if (!(j==0 && pathToClosest.get(j).equals(path.getLast()))) {
        path.add(pathToClosest.get(j));
    }
}

}

List<Member> hubsClone = new ArrayList<>(hubsInvolved);
passed = new ArrayList<>();
while (hubsClone.size()>0) {

    Member closestHub = searchClosestHub(path.getLast(), memberGraph.getMembersLocationGraph(), hubsInvolved, passed);

    LinkedList<Member> pathToClosest = new LinkedList<>();

    Double dist = Algorithms.shortestPath(
        memberGraph.getMembersLocationGraph(),
        path.getLast(),
        closestHub,
        Double::compare,
        Double::sum,
        Double.sum( a: 0, b: 0),
        pathToClosest
    );

    if(dist != null)
        originAndDestiny.put(path.getLast().getId() + "-" + closestHub.getId(), dist);

    distance = distance + (dist==null?0:dist);

    hubsClone.remove(closestHub);

    // Add pathToClosest to path
    for (int j = 0; j < pathToClosest.size(); j++) {
        if (!(j==0 && pathToClosest.get(j).equals(path.getLast()))) {
            path.add(pathToClosest.get(j));
        }
    }
}
```



```
}

Map<Client, HubAndDistanceToAClient> map = nhCtrl.getAllClientsEnterprises(tceCtrl.getTopNClosestEnterprises(numberOfHubs));

Map<Client, List<ProductInfo>> hubAndTheProductsItOwns = new HashMap<>();

for (Expedition e : expeditions) {
    Client hub;

    if(map.get(e.getClientToSatisfy()) == null)
        hub = e.getClientToSatisfy();
    else
        hub = map.get(e.getClientToSatisfy()).getHub();

    List<ProductInfo> products = e.getProductsToDeliver();

    if(!hubAndTheProductsItOwns.containsKey(hub)){

        //filter all products that have no producer
        List<ProductInfo> productsWithProducer = products
            .stream().filter(
                p -> !p.getProducerCode().equals("No Producer")
            ).collect(Collectors.toList());

        hubAndTheProductsItOwns.put(hub, new ArrayList<>(productsWithProducer));
    }else{
        List<ProductInfo> containsKey = hubAndTheProductsItOwns.get(hub);

        //filter all products that have no producer
        List<ProductInfo> productsWithProducer = products
            .stream().filter(
                p -> !p.getProducerCode().equals("No Producer")
            ).collect(Collectors.toList());

        containsKey.addAll(productsWithProducer);
    }
}

return new MinPathInfo(path, distance, hubAndTheProductsItOwns);
}
```

Figura 22 – ExpeditionMinPathService

COMPLEXIDADE

```

public MinPathInfo minPathExpeditionByDay(int day, int numberOfHubs, int numberOfProducers, Map<String, Double> originAndDestiny) {

    List<Expedition> expeditions = numberOfProducers==1?expCtrl.getExpeditionList(day):expCtrl.getExpeditionListByProducerNumber(day, numberOfHubs, numberOfProducers);
    List<Member> producersInvolved = getProducersOfExpeditions(expeditions);
    List<Member> hubsInvolved = getHubsInvolved(numberOfHubs);

    Double distance = Double.valueOf( d, 0.0);

    List<Map.Entry<String, Double>> originAndDestinyList = new ArrayList<>();

    // Choose Starting Producer
    Member p1 = producersInvolved.get(0);

    LinkedList<Member> path = new LinkedList<>();
    path.add(p1);

    List<Member> passed = new ArrayList<>();
    passed.add(p1);

    /**
     * Find the closest Producers and the paths between them until all producer have been covered
     */
    List<Member> piClone = new ArrayList<>(producersInvolved);
    while (piClone.size()>0) {

        Member closest = searchClosestProducer(path.getLast(), memberGraph.getMembersLocationGraph(), producersInvolved, passed);

        LinkedList<Member> pathToClosest = new LinkedList<>();

        Double dist = Algorithms.shortestPath(
            memberGraph.getMembersLocationGraph(),
            path.getLast(),
            closest,
            Double::compare,
            Double::sum,
            Double.sum( a, 0, b, 0),
            pathToClosest
        );

        if(dist != null)
            originAndDestiny.put(path.getLast().getId() + "-" + closest.getId(), dist);

        distance = distance + (dist==null?0:dist);
    }
}

```

$O(n^4)$

$O(n)$

$O(n)*O(n)$

```

    piClone.remove(path.getLast());

    // Add pathToClosest to path
    for (int j = 0; j < pathToClosest.size(); j++) {
        if (!(j==0 && pathToClosest.get(j).equals(path.getLast()))) {
            path.add(pathToClosest.get(j));
        }
    }
}

```

$O(n)$

```

List<Member> hubsClone = new ArrayList<>(hubsInvolved);
passed = new ArrayList<>();
while (hubsClone.size()>0) {

    Member closestHub = searchClosestHub(path.getLast(), memberGraph.getMembersLocationGraph(), hubsInvolved, passed);

    LinkedList<Member> pathToClosest = new LinkedList<>();

    Double dist = Algorithms.shortestPath(
        memberGraph.getMembersLocationGraph(),
        path.getLast(),
        closestHub,
        Double::compare,
        Double::sum,
        Double.sum(a: 0, b: 0),
        pathToClosest
    );

    if(dist != null)
        originAndDestiny.put(path.getLast().getId() + "-" + closestHub.getId(), dist);

    distance = distance + (dist==null?0:dist);

    hubsClone.remove(closestHub);

    // Add pathToClosest to path
    for (int j = 0; j < pathToClosest.size(); j++) {
        if (!(j==0 && pathToClosest.get(j).equals(path.getLast()))) {
            path.add(pathToClosest.get(j));
        }
    }
}

```

$O(n)$

$* O(n)O(n)$

$O(n)*O(n)$

```

}

Map<Client, HubAndDistanceToAClient> map = nhCtrl.getAllClientsEnterprises(tceCtrl.getTopNClosestEnterprises(numberOfHubs));

Map<Client, List<ProductInfo>> hubAndTheProductsItOwns = new HashMap<>();

for (Expedition e : expeditions) {
    Client hub;

    if(map.get(e.getClientToSatisfy()) == null)
        hub = e.getClientToSatisfy();
    else
        hub = map.get(e.getClientToSatisfy()).getHub();

    List<ProductInfo> products = e.getProductsToDeliver();

    if(!hubAndTheProductsItOwns.containsKey(hub)){
        //filter all products that have no producer
        List<ProductInfo> productsWithProducer = products
            .stream().filter(
                p -> !p.getProducerCode().equals("No Producer")
            ).collect(Collectors.toList());

        hubAndTheProductsItOwns.put(hub, new ArrayList<>(productsWithProducer));
    }else{
        List<ProductInfo> containsKey = hubAndTheProductsItOwns.get(hub);

        //filter all products that have no producer
        List<ProductInfo> productsWithProducer = products
            .stream().filter(
                p -> !p.getProducerCode().equals("No Producer")
            ).collect(Collectors.toList());

        containsKey.addAll(productsWithProducer);
    }
}

return new MinPathInfo(path, distance, hubAndTheProductsItOwns);
}

```

$O(n)*O(n)$

$O(n)$

$O(n)*O(n)$

Figura 23 - Complexidade: ExpeditionMinPathService

A complexidade da classe *ExpeditionMinPathService* é $O(n^4)$. Assim sendo, a complexidade final da funcionalidade é $O(n^4)$.

US311

DESCRIÇÃO

Para uma lista de expedição calcular estatísticas por cabaz, por cliente, por produtor e por *hub*.

CLASSES DESENVOLVIDAS

Esta funcionalidade inicia-se após o utilizador seleccionar a décima opção do Menu Inicial. A classe *StatisticsUI* irá solicitar o tipo de expedição (cabaz, cliente, produtor, *hub*).

É de notar que, em todas estas opções, é possível aceder a ambas as listas de expedição desenvolvidas nas funcionalidades 308 e 309, o que será questionado mais tarde.

CABAZ

Se a opção escolhida for a 1 (cabaz), o programa irá perguntar de seguida ao utilizador qual é o dia definido para calcular as estatísticas de todos os cabazes cujo dia seja igual e coloca a pergunta relativa ao uso de números máximos de empresas e produtores ou não.

Caso o utilizador pretenda escolher um número máximo, a lista de expedição associada será a realizada na US309. Como tal, é necessário saber o número de *hubs* e *produtores* máximos a serem definidos e, seguidamente, vai buscar a informação necessária ao *StatisticsController*.

O método *listsCabaz()* tem um alto controlo de exceções. Caso passe em todas as verificações, o *StatisticsService* irá ser associado à classe *ExpeditionListService* para ter acesso à lista de expedição e é gerada uma outra lista de clientes para guardar o nome de todos os clientes ordenadamente.

Dentro desta lista, cada iteração tem todos os seus elementos inicializados. Neste caso são necessários o total de produtos que foram entregues completos, o total de produtos que foram entregues parcialmente, o total de produtos que não forma entregues de todo, a percentagem de satisfação, e o número de produtores diferentes que entregaram a este. Os valores *totalQuantityRequested* e *totalQuantityDelivered* serão usadas para o cálculo da percentagem.

Para o cálculo das 3 primeiras estatísticas, é preciso analisar produto a produto e verificar se a quantidade recebida é igual à quantidade pedida, igual a 0 ou inferior à quantidade pedida. No primeiro caso, a variável *totalProdCompleto* aumenta, no segundo caso, a variável *prodNulo* aumenta e último caso é a vez da variável *parcialProd* aumentar. É também verificado se o produtor já está na lista e adicionando sempre novos produtores. Os valores para o cálculo da percentagem de satisfação também são consistentemente alterados.

```
for (ProductInfo prod : info) {  
    if (Objects.equals(prod.getQuantityRequested(), prod.getQuantityDelivered())) {  
        totalProdCompleto++;  
    }  
    if (prod.getQuantityDelivered() == 0) {  
        prodNulo++;  
    }  
    if (prod.getQuantityDelivered() < prod.getQuantityRequested() && prod.getQuantityDelivered() != 0) {  
        parcialProd++;  
    }  
    producers.removeIf(s -> s.equals(prod.getProducerCode()));  
    producers.add(prod.getProducerCode());  
    totalQuantityDelivered += prod.getQuantityDelivered();  
    totalQuantityRequested += prod.getQuantityRequested();  
}
```

Por fim, é adicionado em cada iteração todas as informações por esta ordem. Todos os valores serão iterativamente demonstrados em *StatisticsUI*, visto que a ordem de adicionar clientes é igual à ordem das suas estatísticas. No final do ciclo, é retornada a lista cujo tamanho é igual a 5 vezes o tamanho de *allClientsCabaz*.

```
statistics.add(totalProdCompleto);  
statistics.add(parcialProd);  
statistics.add(prodNulo);  
double percentSatisfeito = totalQuantityDelivered * 100 / totalQuantityRequested ;  
statistics.add(percentSatisfeito);  
statistics.add((double) producers.size());  
allClientsCabaz.add(e.getClientToSatisfy());  
}  
return statistics;
```

No caso de não serem pretendidos números máximos para produtores, serão utilizados os métodos desenvolvidos na funcionalidade US308. A classe *StatisticsController* tem bastante menos exceções devido a ter só um atributo.

O método *cabazStatistics()* presente no *Service* funciona duma maneira idêntica ao descrito anteriormente, a única diferença é mesmo o uso da lista de expedição gerada duma maneira diferente.


```

public List<Double> cabazStatistics(int day){
    List<Double> statistics = new ArrayList<>();
    explist = service.getExpeditionList(day);
    for(Expedition e: explist){
        List<ProductInfo> info = e.getProductsToDeliver();
        double totalProdCompleto = 0;
        double prodNulo = 0;
        double parcialProd = 0;
        double totalQuantityRequested = 0;
        double totalQuantityDelivered = 0;
        List<String> producers = new ArrayList<>();
        for (ProductInfo prod : info) {
            if (Objects.equals(prod.getQuantityRequested(), prod.getQuantityDelivered())) {
                totalProdCompleto++;
            }
            if (prod.getQuantityDelivered() == 0) {
                prodNulo++;
            }
            if (prod.getQuantityDelivered() < prod.getQuantityRequested() && prod.getQuantityDelivered() != 0) {
                parcialProd++;
            }
            producers.removeIf(s -> s.equals(prod.getProducerCode()));
            producers.add(prod.getProducerCode());
            totalQuantityDelivered += prod.getQuantityDelivered();
            totalQuantityRequested += prod.getQuantityRequested();
        }
        statistics.add(totalProdCompleto);
        statistics.add(parcialProd);
        statistics.add(prodNulo);
        double percentSatisfeito = totalQuantityDelivered * 100 / totalQuantityRequested ;
        statistics.add(percentSatisfeito);
        statistics.add((double) producers.size());

        if(!allClientsCabaz.contains(e.getClientToSatisfy())){
            allClientsCabaz.add(e.getClientToSatisfy());
        }
    }
    return statistics;
}

```

Figura 24 - StatisticsService: cabazStatistics()

CLIENTE

Se a opção escolhida pelo utilizador for a número 2 (cliente), é necessário saber o ID do cliente a ser analisado, o número máximo de dias a analisar e se o utilizador quer definir um número máximo de *hubs* e produtores. No caso de os querer definir, será novamente invocada a lista de expedição utilizada na segunda funcionalidade (US309). O *Controller*, mais uma vez, é responsável pela verificação dos dados inseridos e na confirmação se existe ou não aquele cliente.

Acessando o *Service*, as seguintes variáveis serão inicializadas: total de cabazes completos, total de cabazes parciais e o número de produtores distintos. Visto que o número de dias definido é um máximo, são analisadas todas as listas de expedição dentro desse intervalo de dias e é verificado se a lista de expedição não é nula.

Para cada expedição desta lista é verificado se o nome do cliente é o mesmo e só depois serão realizados todos os cálculos. Dentro da expedição são contados os valores em que a quantidade entregue e a quantidade pedida são diferentes, aumentando o contador. Também é calculado se os valores deixados são todos 0. No caso de os contadores serem nulos, este cabaz está completo, se houver algum produto que seja tenha sido minimamente entregue, o cabaz é entregue parcialmente. Noutro caso, é nulo.

```

for (Expedition e : explist) {
    if (e.getClientToSatisfy().getId().equals(clientName)) {
        int count = 0;
        int count2 = 0;
        for (ProductInfo prod : e.getProductsToDeliver()) {
            if (prod.getQuantityDelivered() == 0) {
                count++;
            }
            if (prod.getQuantityDelivered() != 0 && prod.getQuantityDelivered() < prod.getQuantityRequested()) {
                count2++;
            }
            if (!producers.contains(prod.getProducerCode()) && prod.getProducerCode() != "No Producer") {
                producers.add(prod.getProducerCode());
            }
        }
        if (count != e.getProductsToDeliver().size()) {
            if (count == 0 && count2 == 0) {
                totalCabazesComp++;
            } else if (count2 > 0) {
                totalCabazesParciais++;
            }
        }
        break;
    }
}
}

```

Figura 25 - *StatisticsService: clienteStatistics()*

Caso o utilizador não pretenda definir tais valores máximos, ou seja, selecione a opção 2, a funcionalidade invocada será a US308. Dentro do *StatisticsController* são analisados igualmente se o Cliente existe e se o dia está corretamente introduzido.

No *Service*, o método é executado da mesma forma, apenas com dados distintos.

PRODUTOR

Quando a opção 3 é selecionada (produtor), serão perguntados valores semelhantes aos do cliente: o id do produtor, o dia máximo a ser definido e se o cliente quer definir um número máximo de *hubs* e produtores, cuja resposta invoca a US309 ou US308, respetivamente.

É de notar que como um dos dados necessários às estatísticas deste utilizador são os produtos sem stock, é necessário usar o método *saveStocks()* já anteriormente usado nas US308 e US309 no início destes métodos e o método *resetStocks()* antes do fim dele. É inicialmente confirmado se o produtor realmente existe no sistema:

Os valores necessários são o total de cabazes completos, o total de cabazes parciais, o número de produtos sem stock, o número de clientes distinto e o número de hubs distintos. Para um desenvolvimento mais fácil, o produtor é logo definido para acesso de dados.

```

public List<Double> produtorStatistics(int maxDay, String producerName, int maxHub, int maxProd){

    double totalCabazesComp = 0;
    double totalCabazesParciais = 0;
    double countStock = 0;

    List<String> clients = new ArrayList<>();
    List<String> hubs = new ArrayList<>();
    List<Double> statistics = new ArrayList<>();

    List<Member> producers = comp.getProducerStore().getProducerStore();
    Producer producer = null;
    for(Member m : producers){
        if (m.getId().equals(producerName)){
            producer = (Producer) m;
            break;
        }
    }
}

```

Desde que o produtor não seja nulo, vai iniciar uma iteração de todos os dias até ao dia máximo definido e também verifica sempre se a lista não é nula. Dentro de cada expedição é necessário ver se o produtor que entrega determinado produto é igual ao definido nessa informação do produto. Caso seja um cliente, é adicionado à lista de clientes e caso seja uma empresa, é adicionado à lista de empresas. É inicializado um contador para verificar se este cabaz é completo ou não e outra para ver se o cabaz não é nulo, aumentando o total de cabazes completos/parciais, respetivamente.

```

for (Expedition e : explist) {
    int count = 0;
    int count2 = 0;
    for (ProductInfo prod : e.getProductsToDeliver()) {
        if (prod.getProducerCode().equals(producerName)) {
            if (String.valueOf(e.getClientToSatisfy().getId().charAt(0)).equals("C")) {
                clients.removeIf(c -> c.equals(e.getClientToSatisfy().getId()));
                clients.add(e.getClientToSatisfy().getId());
            }
            if (String.valueOf(e.getClientToSatisfy().getId().charAt(0)).equals("E")) {
                hubs.removeIf(c -> c.equals(e.getClientToSatisfy().getId()));
                hubs.add(e.getClientToSatisfy().getId());
            }
            if (prod.getQuantityDelivered() == prod.getQuantityRequested()) {
                count++;
            } else
            if (prod.getQuantityDelivered() > 0) {
                count2++;
            }
        }
    }
    if (count == e.getProductsToDeliver().size() && !e.getProductsToDeliver().isEmpty()) {
        totalCabazesComp++;
    } else if (count2 > 0) {
        totalCabazesParciais++;
    }
}
}

```

Figura 26 - StatisticsService: produtorStatistics()

Ao serem utilizados os dados da funcionalidade US308, todos os métodos utilizados no Service realizam a mesma função, apenas com a lista de expedição distinta.

HUB

No caso de a opção ser a opção 4 (*hub*), serão necessários o id do *hub* a ser avaliado, o dia máximo a ser avaliado, o número máximo de *hubs* a ser definido (visto que usando

qualquer lista de expedição é necessário este número e se o utilizador quer definir um número máximo de produtores ou não.

No caso de querer, ou seja, garantindo a invocação da funcionalidade US309, o *Controller* começa por verificar se a empresa a ser avaliada está dentro do sistema e caso não esteja, o contador está a 0 e a exceção é lançada.

Ao chegar ao método na classe *StatisticsService*, são inicializados um dos valores pedidos para a estatística (total de clientes), a lista para ver as várias distâncias e o *hub* definido é procurado.

```
public List<Double> hubStatistics(int nEnterprises, String enterpriseID, int maxDay, int nProducers){
    List<Double> statistics = new ArrayList<>();
    double totalClients = 0;
    List<EnterpriseAverageDistance> enterprises = topClosestEnterprisesCtrl.getTopNClosestEnterprises(nEnterprises);
    Client c = null;
    for(Client e : comp.getEnterpriseClients()){
        if(e.getId().equals(enterpriseID)){
            c = e;
            break;
        }
    }
}
```

No caso do *hub* não ser nulo, é gerado anteriormente gerado na funcionalidade US304 que contém vários conjuntos de *Cliente* e *HubAndDistanceToAClient* (que contém o *hub* mais próximo e a distância ao cliente). São iterados os vários valores que correspondem aos *HubAndDistanceToAClient* e, caso o *hub* seja igual ao definido, o total de clientes aumenta. O valor é logo adicionado às estatísticas.

Dentro deste caso ainda, é inicializado um ciclo que vê todas as listas de expedição dentro do máximo de dias definido e procura pela expedição que esteja associada à empresa pedida. São lidos todos os produtores e, caso o produtor não seja nulo ou não esteja na lista anteriormente inicializada, o seu código é adicionado. Como o necessário é o número de produtores, apenas o *size()* da lista é adicionado à lista final e esta é retornada.

```
for (int i = 1; i < maxDay + 1; i++) {
    expList = service.getExpeditionList(i, clientHubAndDistanceToClientMap, nProducers);
    if (expList == null) {
        break;
    }
    for (Expedition e : expList) {
        if (e.getClientToSatisfy().getId().equals(enterpriseID)) {
            for (ProductInfo prod : e.getProductsToDeliver()) {
                if (!producers.contains(prod.getProducerCode()) && prod.getProducerCode() != "No Producer") {
                    producers.add(prod.getProducerCode());
                }
            }
        }
    }
    statistics.add((double) producers.size());
}
return statistics;
```

Figura 27 - *StatisticsService: hubStatistics()*

No caso de não serem definido um máximo de produtores, outro método no *StatisticsController* é inicializado, verificando os mesmos dados. O total de clientes é calculado exatamente da mesma forma e não sofre qualquer alteração entre ter ou não máximo de produtores definidos.

COMPLEXIDADE

```

public List<Double> cabazStatistics(int day, int nHubs, int nProducers){
    List<Double> statistics = new ArrayList<>();
    expList = service.getExpeditionList(day, nearestHubService.getEnterprisesCloser(topClosestEnterprisesCtrl.getTopNClosestEnterprises(nHubs)), nProducers);
    for(Expedition e: expList){
        List<ProductInfo> info = e.getProductsToDeliver();
        double totalProdCompleto = 0;
        double prodNulo = 0;
        double parcialProd = 0;
        double totalQuantityRequested = 0;
        double totalQuantityDelivered = 0;
        List<String> producers = new ArrayList<>();
        for (ProductInfo prod : info) {
            if (Objects.equals(prod.getQuantityRequested(), prod.getQuantityDelivered())) {
                totalProdCompleto++;
            }
            if (prod.getQuantityDelivered() == 0) {
                prodNulo++;
            }
            if (prod.getQuantityDelivered() < prod.getQuantityRequested() && prod.getQuantityDelivered() != 0) {
                parcialProd++;
            }
            producers.removeIf(s -> s.equals(prod.getProducerCode()));
            producers.add(prod.getProducerCode());
            totalQuantityDelivered += prod.getQuantityDelivered();
            totalQuantityRequested += prod.getQuantityRequested();
        }
        statistics.add(totalProdCompleto);
        statistics.add(parcialProd);
        statistics.add(prodNulo);
        double percentSatisfeito = totalQuantityDelivered * 100 / totalQuantityRequested ;
        statistics.add(percentSatisfeito);
        statistics.add((double) producers.size());
        allClientsCabaz.add(e.getClientToSatisfy());
    }
    return statistics;
}

```

$O(n)$

$O(n^2)$

$O(n^4)$ ↑

Final : $O(n^4)$

```

public List<Double> cabazStatistics(int day){
    List<Double> statistics = new ArrayList<>();
    expList = service.getExpeditionList(day);  $O(n^3)$ 
    for(Expedition e: expList){
        List<ProductInfo> info = e.getProductsToDeliver();
        double totalProdCompleto = 0;
        double prodNulo = 0;
        double parcialProd = 0;
        double totalQuantityRequested = 0;
        double totalQuantityDelivered = 0;
        List<String> producers = new ArrayList<>();
        for (ProductInfo prod : info) {
            if (Objects.equals(prod.getQuantityRequested(), prod.getQuantityDelivered())) {
                totalProdCompleto++;
            }
            if (prod.getQuantityDelivered() == 0) {
                prodNulo++;
            }
            if (prod.getQuantityDelivered() < prod.getQuantityRequested() && prod.getQuantityDelivered() != 0) {
                parcialProd++;
            }
            producers.removeIf(s -> s.equals(prod.getProducerCode()));
            producers.add(prod.getProducerCode());
            totalQuantityDelivered += prod.getQuantityDelivered();
            totalQuantityRequested += prod.getQuantityRequested();
        }
        statistics.add(totalProdCompleto);
        statistics.add(parcialProd);
        statistics.add(prodNulo);
        double percentSatisfeito = totalQuantityDelivered * 100 / totalQuantityRequested ;
        statistics.add(percentSatisfeito);
        statistics.add((double) producers.size());

        if(!allClientsCabaz.contains(e.getClientToSatisfy())){
            allClientsCabaz.add(e.getClientToSatisfy());
        }
    }
    return statistics;
}

```

$O(n)$

$O(n^2)$

Final : $O(n^3)$

Figura 28 - Complexidade: cabazStatistics()

```

public List<Double> clienteStatistics(int maxDay, String clientName, int nHubs, int nProducers){
    double totalCabazesComp = 0;
    double totalCabazesParciais = 0;
    List<String> producers = new ArrayList<>();
    List<Double> statistics = new ArrayList<>();
    for(int i = 1; i < maxDay+1; i++) {
        expList = service.getExpeditionList(i, nearestHubService.getEnterprisesCloser(topClosestEnterprisesCtrl.getTopNClosestEnterprises(nHubs)), nProducers);
        if(expList==null){
            break;
        }
        for (Expedition e : expList) {
            if (e.getClientToSatisfy().getId().equals(clientName)) {
                int count = 0;
                int count2 = 0;
                for(ProductInfo prod : e.getProductsToDeliver()){
                    if(prod.getQuantityDelivered()==0){
                        count++;
                    }
                    if(prod.getQuantityDelivered()!=0 && prod.getQuantityDelivered()<prod.getQuantityRequested()){
                        count2++;
                    }
                    if(!producers.contains(prod.getProducerCode()) && prod.getProducerCode()!="No Producer"){
                        producers.add(prod.getProducerCode());
                    }
                }
                if(count!=e.getProductsToDeliver().size()){
                    if(count==0 && count2 ==0){
                        totalCabazesComp++;
                    }else if(count2>0){
                        totalCabazesParciais++;
                    }
                }
            }
        }
        statistics.add(totalCabazesComp);
        statistics.add(totalCabazesParciais);
        statistics.add((double)producers.size());
    }
    return statistics;
}

```

Final : $O(n^4)$

Ativar o Windows

```

public List<Double> clienteStatistics(int maxDay, String clientName){
    double totalCabazesComp = 0;
    double totalCabazesParciais = 0;
    List<String> producers = new ArrayList<>();
    List<Double> statistics = new ArrayList<>();
    for(int i = 1; i < maxDay+1; i++) {
        expList = service.getExpeditionList(i);
        if(expList==null){
            break;
        }
        for (Expedition e : expList) {
            if (e.getClientToSatisfy().getId().equals(clientName)) {
                int count = 0;
                int count2 = 0;
                for(ProductInfo prod : e.getProductsToDeliver()){
                    if(prod.getQuantityDelivered()==0){
                        count++;
                    }
                    if(prod.getQuantityDelivered()!=0 && prod.getQuantityDelivered()<prod.getQuantityRequested()){
                        count2++;
                    }
                    if(!producers.contains(prod.getProducerCode()) && prod.getProducerCode()!="No Producer"){
                        producers.add(prod.getProducerCode());
                    }
                }
                if(count!=e.getProductsToDeliver().size()){
                    if(count==0 && count2 ==0){
                        totalCabazesComp++;
                    }else if(count2>0){
                        totalCabazesParciais++;
                    }
                }
            }
        }
        statistics.add(totalCabazesComp);
        statistics.add(totalCabazesParciais);
        statistics.add((double)producers.size());
    }
    return statistics;
}

```

Final : $O(n^3)$

Figura 29 - Complexidade: clienteStatistics()

```

public List<Double> produtorStatistics(int maxDay, String producerName, int maxHub, int maxProd){
    double totalCabazesComp = 0;
    double totalCabazesParciais = 0;
    double countStock = 0;
    List<String> clients = new ArrayList<>();
    List<String> hubs = new ArrayList<>();
    List<Double> statistics = new ArrayList<>();
    List<Member> producers = comp.getProducerStore().getProducerStore();
    Producer producer = null;
    for(Member m : producers){
        if (m.getId().equals(producerName)){
            producer = (Producer) m;
            break;
        }
    }
    if(producer!=null){
        for (int i = 1; i < maxDay + 1; i++) {
            explist = service.getExpeditionList(i, nearestHubService.getEnterprisesCloser(topClosestEnterprisesCtrl.getTopNClosestEnterprises(maxHub)), maxProd);
            if (explist == null) {
                break;
            }
        }
    }
    for (Expedition e : explist) {
        int count = 0;
        int count2 = 0;
        for (ProductInfo prod : e.getProductsToDeliver()) {
            if (prod.getProducerCode().equals(producerName)) {
                if (String.valueOf(e.getClientToSatisfy().getId().charAt(0)).equals("C")) {
                    clients.removeIf(c -> c.equals(e.getClientToSatisfy().getId()));
                    clients.add(e.getClientToSatisfy().getId());
                }
                if (String.valueOf(e.getClientToSatisfy().getId().charAt(0)).equals("E")) {
                    hubs.removeIf(c -> c.equals(e.getClientToSatisfy().getId()));
                    hubs.add(e.getClientToSatisfy().getId());
                }
                if (prod.getQuantityDelivered() == prod.getQuantityRequested()) {
                    count++;
                } else {
                    if (prod.getQuantityDelivered() > 0) {
                        count2++;
                    }
                }
            }
        }
        if (count == e.getProductsToDeliver().size() && !e.getProductsToDeliver().isEmpty()) {
            totalCabazesComp++;
        } else if (count2 > 0) {
            totalCabazesParciais++;
        }
    }
    statistics.add(totalCabazesComp); // Ponto 1
    statistics.add(totalCabazesParciais); // Ponto 2
    statistics.add((double) clients.size()); // Ponto 3
    for (ProductStock stock : producer.getStockOfDay(maxDay)) {
        if (stock.getQuantity() <= 0) {
            countStock++;
        }
    } // Ponto 4
    statistics.add(countStock);
    statistics.add((double) hubs.size()); // Ponto 5
    return statistics;
}

```

Complexity Analysis:

- $O(1)$: Initialization of variables and lists.
- $O(n)$: Finding the producer in the list of producers.
- $O(n^4)$: The main loop over days, expeditions, products, and clients/hubs.
- $O(n^3)$: The inner loop over products and clients/hubs.
- $O(n^2)$: The loop over expeditions.
- $O(n)$: The loop over products.
- $O(n)$: The loop over clients/hubs.
- Final: $O(n^4)$**

```

public List<Double> produtorStatistics(int maxDay, String producerName){
    double totalCabazesComp = 0;
    double totalCabazesParciais = 0;
    double countStock = 0;

    List<String> clients = new ArrayList<>();
    List<String> hubs = new ArrayList<>();
    List<Double> statistics = new ArrayList<>();

    List<Member> producers = comp.getProducerStore().getProducerStore();
    Producer producer = null;
    for(Member m : producers){
        if (m.getId().equals(producerName)){
            producer = (Producer) m;
            break;
        }
    }

    if(producer!=null) {
        for (int i = 1; i < maxDay + 1; i++) {
            explist = service.getExpeditionList(i);
            if (explist == null) {
                break;
            }

            for (Expedition e : explist) {
                int count = 0;
                int count2 = 0;
                for (ProductInfo prod : e.getProductsToDeliver()) {
                    if (prod.getProducerCode().equals(producerName)) {
                        if (String.valueOf(e.getClientToSatisfy().getId().charAt(0)).equals("C")) {
                            clients.removeIf(c -> c.equals(e.getClientToSatisfy().getId()));
                            clients.add(e.getClientToSatisfy().getId());
                        }
                        if (String.valueOf(e.getClientToSatisfy().getId().charAt(0)).equals("E")) {
                            hubs.removeIf(c -> c.equals(e.getClientToSatisfy().getId()));
                            hubs.add(e.getClientToSatisfy().getId());
                        }
                        if (prod.getQuantityDelivered() == prod.getQuantityRequested()) {
                            count++;
                        } else if (prod.getQuantityDelivered() > 0) {
                            count2++;
                        }
                    }
                }
                if (count == e.getProductsToDeliver().size() && !e.getProductsToDeliver().isEmpty()) {
                    totalCabazesComp++;
                } else if (count2 > 0) {
                    totalCabazesParciais++;
                }
            }
        }

        statistics.add(totalCabazesComp); // Ponto 1
        statistics.add(totalCabazesParciais); // Ponto 2
        statistics.add((double) clients.size()); // Ponto 3

        for (ProductStock stock : producer.getStockOfDay(maxDay)) {
            if (stock.getQuantity() <= 0) {
                countStock++;
            }
        } // Ponto 4

        statistics.add(countStock);
        statistics.add((double) hubs.size()); // Ponto 5
        return statistics;
    }
}

```

Complexity Analysis:

- $O(1)$** : Initialization of variables and lists.
- $O(n)$** : Retrieving the producer from the store.
- $O(n^3)$** : The main loop over days, expeditions, and products.
- $O(n^2)$** : The inner loop over products for each expedition.
- $O(n)$** : The loop over the producer's stock.
- Final: $O(n^3)$** : The overall complexity of the method.

Figura 30 - Complexidade: produtorStatistics()

A complexidade dos métodos utilizando a funcionalidade US309 é $O(n^4)$ e com a funcionalidade US308 é $O(n^3)$ exceto no *hubStatistics()*, que passa a ser $O(n^3 \log(n))$. Assim sendo, em geral, a complexidade máxima desta funcionalidade é $O(n^4)$.


```

public List<Double> hubStatistics(int nEnterprises, String enterpriseID, int maxDay, int nProducers){
    List<Double> statistics = new ArrayList<>();
    double totalClients = 0;
    List<EnterpriseAverageDistance> enterprises = topClosestEnterprisesCtrl.getTopNClosestEnterprises(nEnterprises);
    Client e = null;
    for(Client e : comp.getEnterpriseClients()){
        if(e.getId().equals(enterpriseID)){
            c = e;
            break;
        }
    }

    if(c != null) {
        Map<Client, HubAndDistanceToAClient> clientHubAndDistanceToAClientMap = nearestHubService.getEnterprisesCloser(enterprises);
        for (HubAndDistanceToAClient v : clientHubAndDistanceToAClientMap.values()) {
            if (v.getHub().equals(c)) {
                totalClients++;
            }
        }
        statistics.add(totalClients);
        List<String> producers = new ArrayList<>();
        for (int i = 1; i < maxDay + 1; i++) {
            expList = service.getExpeditionList(i, clientHubAndDistanceToAClientMap, nProducers);
            if (expList == null) {
                break;
            }
            for (Expedition e : expList) {
                if (e.getClientToSatisfy().getId().equals(enterpriseID)) {
                    for (ProductInfo prod : e.getProductsToDeliver()) {
                        if (!producers.contains(prod.getProducerCode()) && prod.getProducerCode() != "No Producer") {
                            producers.add(prod.getProducerCode());
                        }
                    }
                }
            }
        }
        statistics.add((double) producers.size());
    }
    return statistics;
}

```

Complexity analysis for the first version:

- $O(1)$ for `statistics = new ArrayList<>()`
- $O(1)$ for `totalClients = 0`
- $O(nE(\log n))$ for `enterprises = topClosestEnterprisesCtrl.getTopNClosestEnterprises(nEnterprises)`
- $O(1)$ for `Client e = null`
- $O(n)$ for the first `for` loop (finding the client)
- $O(n^3(\log n))$ for the nested loops (finding closest hubs and counting clients)
- $O(n)$ for the `statistics.add(totalClients)`
- $O(n^4)$ for the `for` loop (iterating over days)
- $O(n^2)$ for the `for` loop (iterating over expeditions)
- $O(n^3)$ for the `for` loop (iterating over products)
- Final : $O(n^4)$**

```

public List<Double> hubStatistics(int nEnterprises, String enterpriseID, int maxDay){
    List<Double> statistics = new ArrayList<>();
    double totalClients = 0;
    List<EnterpriseAverageDistance> enterprises = topClosestEnterprisesCtrl.getTopNClosestEnterprises(nEnterprises);
    Client e = null;
    for(Client e : comp.getEnterpriseClients()){
        if(e.getId().equals(enterpriseID)){
            c = e;
            break;
        }
    }

    if(c != null) {
        Map<Client, HubAndDistanceToAClient> clientHubAndDistanceToAClientMap = nearestHubService.getEnterprisesCloser(enterprises);
        for (HubAndDistanceToAClient v : clientHubAndDistanceToAClientMap.values()) {
            if (v.getHub().equals(c)) {
                totalClients++;
            }
        }
        statistics.add(totalClients);
        List<String> producers = new ArrayList<>();
        for (int i = 1; i < maxDay + 1; i++) {
            expList = service.getExpeditionList(i);
            if (expList == null) {
                break;
            }
            for (Expedition e : expList) {
                if (e.getClientToSatisfy().getId().equals(enterpriseID)) {
                    for (ProductInfo prod : e.getProductsToDeliver()) {
                        if (!producers.contains(prod.getProducerCode()) && prod.getProducerCode() != "No Producer") {
                            producers.add(prod.getProducerCode());
                        }
                    }
                }
            }
        }
        statistics.add((double) producers.size());
    }
    return statistics;
}

```

Complexity analysis for the second version:

- $O(1)$ for `statistics = new ArrayList<>()`
- $O(1)$ for `totalClients = 0`
- $O(nE \log(n))$ for `enterprises = topClosestEnterprisesCtrl.getTopNClosestEnterprises(nEnterprises)`
- $O(1)$ for `Client e = null`
- $O(n)$ for the first `for` loop (finding the client)
- $O(n^3 \log(n))$ for the nested loops (finding closest hubs and counting clients)
- $O(1)$ for `statistics.add(totalClients)`
- $O(n^3)$ for the `for` loop (iterating over days)
- $O(n)$ for the `for` loop (iterating over expeditions)
- $O(n^2)$ for the `for` loop (iterating over products)
- $O(n^3)$ for the `for` loop (iterating over products)
- Final : $O(n^3 \log(n))$**

Figura 31 - Complexidade: `hubStatistics()`

MELHORIAS

Após a análise de todo o projeto e da complexidade de todos os algoritmos das funcionalidades implementadas, consegue-se chegar à conclusão de que existem alguns melhoramentos possíveis para aprimorar o trabalho realizado.

As principais mudanças a serem apontadas consistem numa tentativa de reduzir a complexidade das funcionalidades desenvolvidas, permitindo uma melhor otimização das mesmas. Um caso específico seria o melhoramento da funcionalidade US304, uma vez que o tempo de carregamento utilizando o maior ficheiro, em comparação com o restante trabalho é ligeiramente elevado.

No entanto, em suma, conclui-se que o grupo conseguiu corresponder positivamente aos desafios propostos, apresentando todas as funcionalidades executadas de uma boa forma e organizadas, apresentando também, na sua maioria, um bom tempo de execução e todos os resultados em conformidade com o pedido.