ANÁLISE DE COMPLEXIDADE

Projeto Integrador – ESINF SPRINT 2

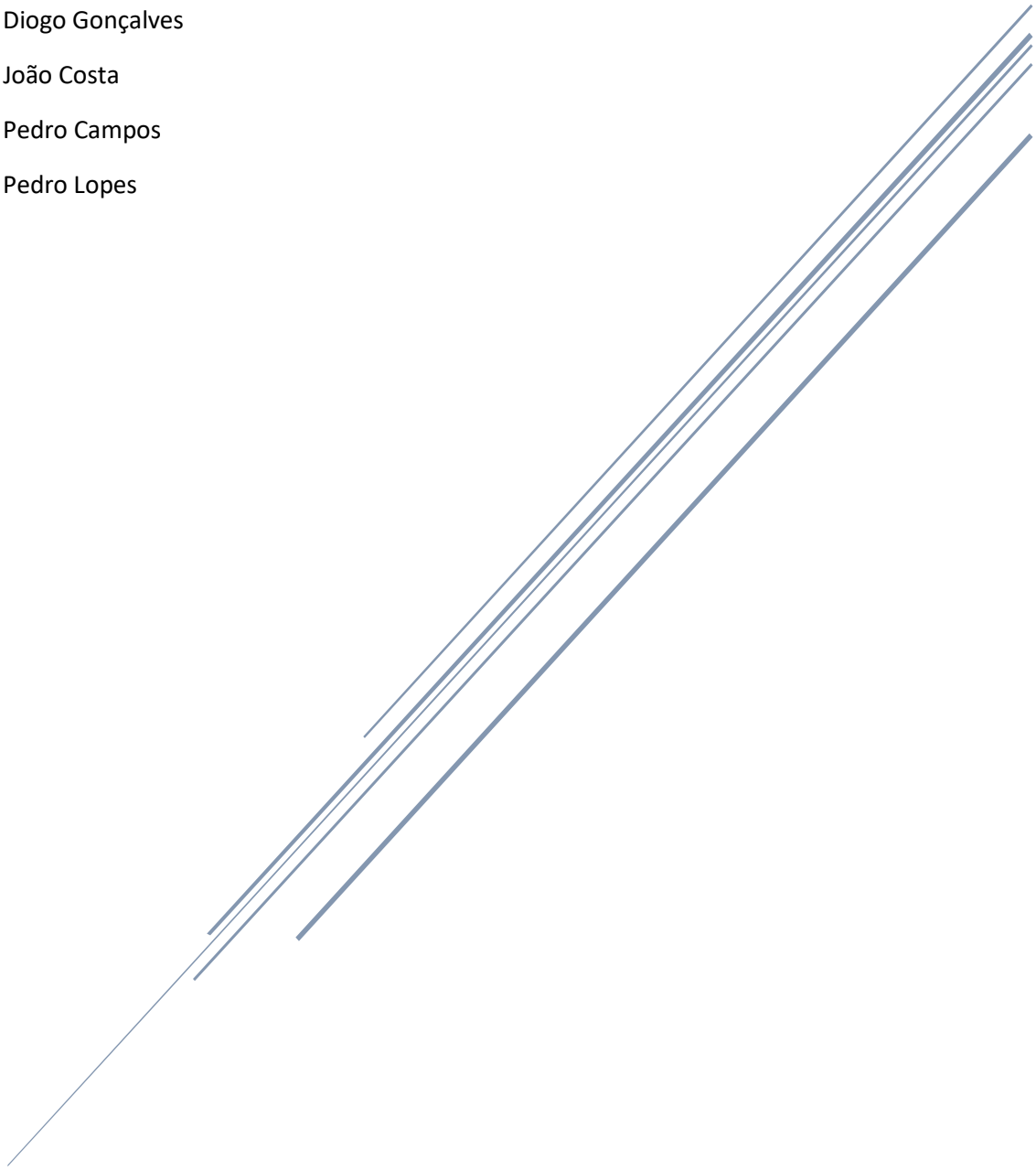
1211514 – Diogo Costa

1211509 – Diogo Gonçalves

1211546 – João Costa

1211511 – Pedro Campos

1211504 – Pedro Lopes



ISEP

Engenharia Informática

Índice

US307 - Importar a lista de cabazes

Resumo:

Para esta user story, começamos por criar novas classes do modelo domínio:

Product – Classe que representa um produto

Basket – Classe que representa um pedido ou produção para um certo dia e que contem uma Lista de Pair de Product e Double, para representar a quantidade

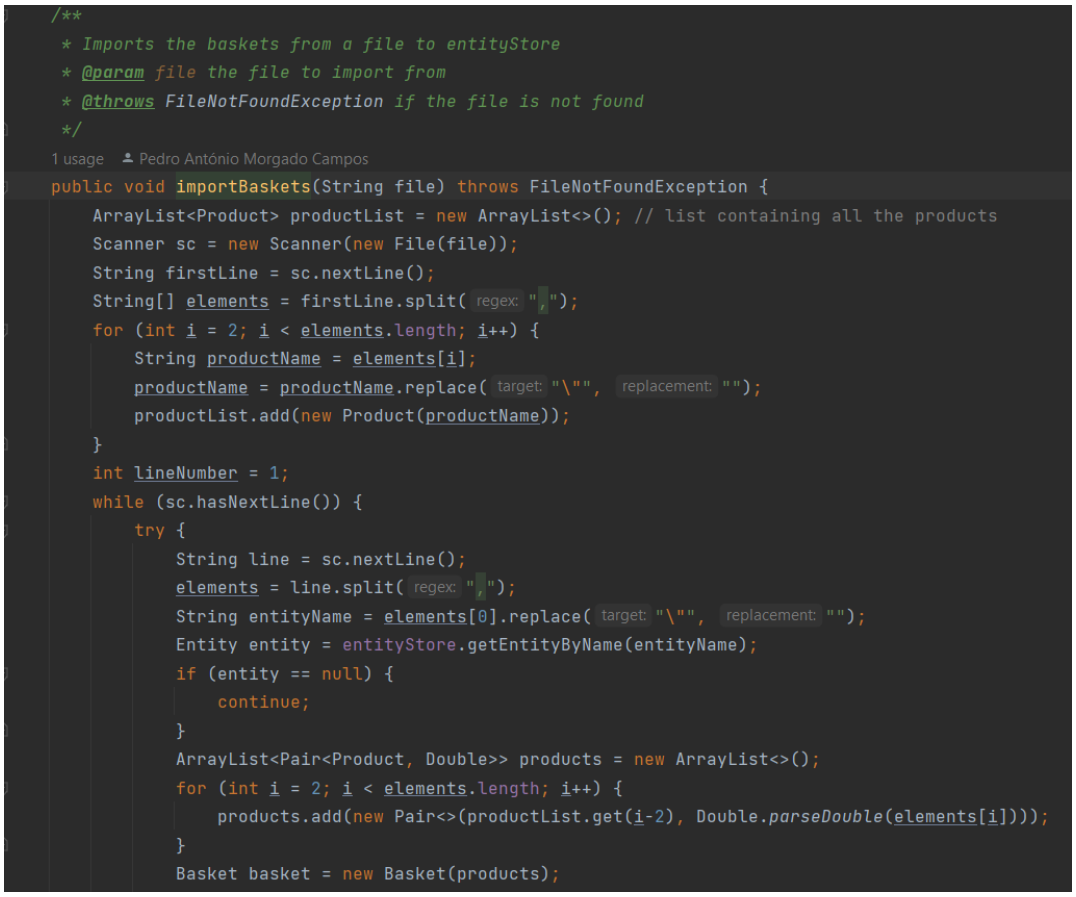
Tambem fizemos algumas alterações nas classes que já tínhamos no modelo domínio.

Para cada entidade também adicionamos um TreeMap<Integer,Basket>, onde a Integer representa o dia e Basket, os pedidos ou produções desse dia, para armazenar todos os cabazes.

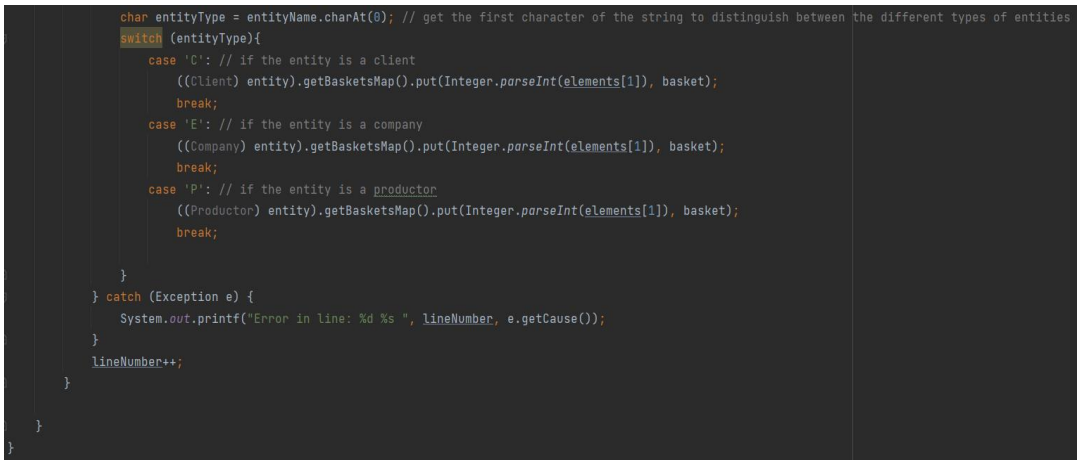
importBaskets

Esta função, que recebe um ficheiro como parâmetro, passa por todas as linhas do ficheiro dos cabazes, deteta a entidade e o dia, e preenche o TreeMap da entidade para esse dia.

- Melhor caso: O(N^2)
- Pior caso: O(N^2)



```java
/**
 * Imports the baskets from a file to entityStore
 * @param file the file to import from
 * @throws FileNotFoundException if the file is not found
 */
1 usage  Pedro António Morgado Campos
public void importBaskets(String file) throws FileNotFoundException {
    ArrayList<Product> productList = new ArrayList<>(); // list containing all the products
    Scanner sc = new Scanner(new File(file));
    String firstLine = sc.nextLine();
    String[] elements = firstLine.split( regex: ",");
    for (int i = 2; i < elements.length; i++) {
        String productName = elements[i];
        productName = productName.replace( target: "\"",  replacement: "");
        productList.add(new Product(productName));
    }
    int lineNumber = 1;
    while (sc.hasNextLine()) {
        try {
            String line = sc.nextLine();
            elements = line.split( regex: ",");
            String entityName = elements[0].replace( target: "\"",  replacement: "");
            Entity entity = entityStore.getEntityByName(entityName);
            if (entity == null) {
                continue;
            }
            ArrayList<Pair<Product, Double>> products = new ArrayList<>();
            for (int i = 2; i < elements.length; i++) {
                products.add(new Pair<>(productList.get(i-2), Double.parseDouble(elements[i])));
            }
            Basket basket = new Basket(products);
```



```java
            char entityType = entityName.charAt(0); // get the first character of the string to distinguish between the different types of entities
            switch (entityType){
                case 'C': // if the entity is a client
                    ((Client) entity).getBasketsMap().put(Integer.parseInt(elements[1]), basket);
                    break;
                case 'E': // if the entity is a company
                    ((Company) entity).getBasketsMap().put(Integer.parseInt(elements[1]), basket);
                    break;
                case 'P': // if the entity is a productor
                    ((Productor) entity).getBasketsMap().put(Integer.parseInt(elements[1]), basket);
                    break;

            }
        } catch (Exception e) {
            System.out.printf("Error in line: %d %s ", lineNumber, e.getCause());
        }
        lineNumber++;
    }

}
}
```

Para gerar listas de expedição:

Para satisfazer estes requisitos, começamos por criar novas classes do modelo domínio:

Expedition – Classe que representa uma expedição para um certo dia. Esta classe contém uma List<Pair<Entity, List<Delivery>>> que representa, para cada cliente, a lista de entregas que esse cliente teve no dia da expedição.

ExpeditionStore – Classe para guardar as expedições.

Tambem fizemos algumas alterações nas classes que já tínhamos no modelo domínio.

Para cada company, se for hub, adicionamos o TreeMap<Integer, List<Pair<Product, List<Pair<Double, Productor>>>>> que representa o stock e o produtor que o forneceu que esta guardado nesse hub, para cada dia, para cada produto, por ordem decrescente de quantidade e o TreeMap<Integer, List<Pair<Product, List<Pair<Double, Entity>>>>> que representa os pedidos feitos por clientes para esse hub para cada dia, para cada produto, por ordem decrescente de quantidade.

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

Resumo:

Para esta user story, como não há restrições quanto aos produtores, assumimos que cada produtor, entrega os seus produtores num "hub global" e todos os clientes vão buscar os seus pedidos a esse hub.

Começamos por importar os pedidos dos clientes, para esse hub, usando a função generateHubOrder para cada cliente.

Após os pedidos serem feitos, entregamos todos os produtos dos produtores nesse hub, usando a função deliverProductsToHub para cada produtor.

Depois, para calcular que produtos é que são entregues a quais produtores, usamos a função calculateDayOrders para o dia pedido.

Esta função começa por adicionar o stock de sobra dos dois dias anteriores, caso exista.

Depois percorre os pedidos para cada produto, e começa por satisfazer os pedidos por ordem decrescente de quantidade de produto ao usar o stock com maior quantidade (sempre o índice 0 da lista de produto), dando sort à lista apos cada fornecimento, para que o maior stock volte para o índice 0.

Desta maneira, garantimos que o maior numero de produto é satisfeito.

deliverProductsToHub

- Melhor caso: O(N^2)
- Pior caso: O(N^3)

```java
public void deliverProductsToHub(Productor productor) {

    TreeMap<Integer, List<Pair<Product, List<Pair<Double, Productor>>>>> stockMap = globalProductStore.getStockMap();

    TreeMap<Integer, Basket> basketsMap = productor.getBasketsMap();
    for (Integer integer : basketsMap.keySet()) {
        Basket basket = basketsMap.get(integer);
        for (Pair<Product, Double> pair : basket.getProducts()) {
            Product product = pair.first();
            Double quantity = pair.second();
            if (!stockMap.containsKey(integer)) {
                stockMap.put(integer, new ArrayList<>());
            }
            List<Pair<Product, List<Pair<Double, Productor>>>> list = stockMap.get(integer);
            boolean found = false;
            for (Pair<Product, List<Pair<Double, Productor>>> pair1 : list) {
                if (pair1.first().equals(product)) {
                    List<Pair<Double, Productor>> list1 = pair1.second();
                    list1.add(new Pair<>(quantity, productor));
                    found = true;
                }
            }
            if (!found) {
                List<Pair<Double, Productor>> list1 = new ArrayList<>();
                list1.add(new Pair<>(quantity, productor));
                list.add(new Pair<>(product, list1));

            }
        }
    }
}
```

generateHubOrder

- Melhor caso: O(N^2)
- Pior caso: O(N^3)

```java
public void generateHubOrder(Entity entity){

    TreeMap<Integer, List<Pair<Product, List<Pair<Double, Entity>>>>> orderMap = globalProductStore.getOrderMap();

    TreeMap<Integer, Basket> basketsMap = entity.getBasketsMap();
    for (Integer integer : basketsMap.keySet()) {
        Basket basket = basketsMap.get(integer);
        for (Pair<Product, Double> pair : basket.getProducts()) {
            Product product = pair.first();
            Double quantity = pair.second();
            if (!orderMap.containsKey(integer)) {
                orderMap.put(integer, new ArrayList<>());
            }
            List<Pair<Product, List<Pair<Double, Entity>>>> list = orderMap.get(integer);
            boolean found = false;
            for (Pair<Product, List<Pair<Double, Entity>>> pair1 : list) {
                if (pair1.first().equals(product)) {
                    List<Pair<Double, Entity>> list1 = pair1.second();
                    list1.add(new Pair<>(quantity, entity));
                    found = true;
                }
            }
            if (!found) {
                List<Pair<Double, Entity>> list1 = new ArrayList<>();
                list1.add(new Pair<>(quantity, entity));
                list.add(new Pair<>(product, list1));

            }
        }
    }
}
```

calculateDayOrders

- Melhor caso: O(N^2)
- Pior caso: O(N^2)

```java
public Expedition calculateDayOrders(int day){
    Expedition expedition = new Expedition(day);

    TreeMap<Integer, List<Pair<Product, List<Pair<Double, Entity>>>>> orderMap = globalProductStore.getOrderMap();
    TreeMap<Integer, List<Pair<Product, List<Pair<Double, Productor>>>>> stockMap = globalProductStore.getStockMap();
    if (orderMap.containsKey(day)) {
        List<Pair<Product, List<Pair<Double, Entity>>>> dayOrders = orderMap.get(day);
        List<Pair<Product, List<Pair<Double, Productor>>>> dayStocks = stockMap.get(day);
        if (day > 2){
            List<Pair<Product, List<Pair<Double, Productor>>>> dayStocksDayBeforeYesterday = stockMap.get(day - 2);
            List<Pair<Product, List<Pair<Double, Productor>>>> dayStocksYesterday = stockMap.get(day - 1);
            for (int y = 0; y < dayStocks.size(); y++) {
                List<Pair<Double, Productor>> additionProducts1 = dayStocksYesterday.get(y).second();
                List<Pair<Double, Productor>> additionProducts2 = dayStocksDayBeforeYesterday.get(y).second();
                List<Pair<Double, Productor>> addedProducts = dayStocks.get(y).second();
                for (int z = 0; z < addedProducts.size(); z++) {
                    Double sum = additionProducts1.get(z).first() + additionProducts2.get(z).first() + addedProducts.get(z).first();
                    addedProducts.get(z).setFirst(sum);
                }


            }

        }
```

```
for (Pair<Product, List<Pair<Double, Entity>>> orders: dayOrders){
    List<Pair<Double, Productor>> products = new ArrayList<>();
    Product product = orders.first();
    for (int j = 0; j < dayStocks.size(); j++) {
        if (dayStocks.get(j).first().equals(product)) {
            products = dayStocks.get(j).second();
        }
    }
    Collections.sort(products, Comparator.comparing(p -> -p.first()));
    List<Pair<Double, Entity>> productOrders = orders.second();
    for (int i = 0; i < productOrders.size(); i++) {
        Pair<Double, Entity> order = productOrders.get(i);
        double quantity = order.first();
        Entity entity = order.second();
        Delivery delivery = new Delivery(entity, day);
        double quantitySupplied = products.get(0).first();

        Productor productor = products.get(0).second();
        if (quantitySupplied > quantity) {
            products.get(0).setFirst(quantitySupplied - quantity);
            quantitySupplied = quantity;
        }
        else if (quantitySupplied <= quantity) {
            products.get(0).setFirst(0.0);
            quantity = quantitySupplied;
        }
        if (quantitySupplied != 0) {
            delivery.addProduct(product, productor, quantity);
            expedition.addDelivery(delivery, entity);
        }
        Collections.sort(products, Comparator.comparing(p -> -p.first()));
```

```
                }
            }

                } else {
            System.out.println("Invalid Day");
        }
        return expedition;
    }


}
```

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

Resumo:

Para esta user story, como temos a restrição que só podemos usar os N produtores mais próximos do hub de entrega, já não vamos poder usar um “hub global”, ou seja cada cliente faz o seu pedido no seu hub mais próximo e os produtores vão fornecer aos hubs, tendo em conta a quantidade de produto que precisam, de maneira a garantir o mínimo produto desperdiçado.

Começamos por importar os pedidos dos clientes, para o hub mais próximo, usando a função generateHubOrder para cada cliente.

Após os pedidos serem feitos, para cada hub, vamos buscar os N produtores mais próximos e preencher os stocks com os produtos desses produtores, ao usar a função deliverProductsToHub, para cada produtor mais próximo desse hub.

Esta função vai garantir que um produtor apenas vai entregar a quantidade de produto necessária para satisfazer os pedidos, para poder entregar o que sobra noutros hubs.

Depois, para calcular que produtos é que são entregues a quais produtores, usamos a função calculateDayOrders para o dia pedido, que vai ser igual à user story anterior.

deliverProductsToHub

- Melhor caso: O(N^2)
- Pior caso: O(N^3)

```
public void deliverProductsToHub(Productor productor, Company hub) {


    TreeMap<Integer, List<Pair<Product, List<Pair<Double, Productor>>>>> stockMap = hub.getStockMap();
    TreeMap<Integer, List<Pair<Product, Double>>> wantedProductsMap = hub.getWantedProductsMap();
    TreeMap<Integer, Basket> basketsMap = productor.getBasketsMap();
    for (Integer integer : basketsMap.keySet()) {
        Basket basket = basketsMap.get(integer);
        for (Pair<Product, Double> pair : basket.getProducts()) {
            Product product = pair.first();
            Double quantity = pair.second();

                if (!stockMap.containsKey(integer)) {
                    stockMap.put(integer, new ArrayList<>());
                }
                List<Pair<Product, Double>> wantedProducts = wantedProductsMap.get(integer);
                if (wantedProducts != null) {
                    for (Pair<Product, Double> wantedProduct : wantedProducts) {
                        if (wantedProduct.first().equals(product)) {
                            if (wantedProduct.second() >= quantity) {
                                wantedProduct.setSecond(wantedProduct.second() - quantity);
                            } else if (wantedProduct.second() == 0) {
                                quantity = 0.0;
                            } else {
                                quantity = quantity - wantedProduct.second();
                                wantedProduct.setSecond(0.0);
                            }
                        }
                    }
```

```
List<Pair<Product, List<Pair<Double, Productor>>>> list = stockMap.get(integer);
boolean found = false;
for (Pair<Product, List<Pair<Double, Productor>>> pair1 : list) {
    if (pair1.first().equals(product)) {
        List<Pair<Double, Productor>> list1 = pair1.second();
        list1.add(new Pair<>(quantity, productor));
        found = true;
    }
}
if (!found) {
    List<Pair<Double, Productor>> list1 = new ArrayList<>();
    list1.add(new Pair<>(quantity, productor));
    list.add(new Pair<>(product, list1));
}
ArrayList<Pair<Product, Double>> basketList = productor.getBasketsMap().get(integer).getProducts();
for (Pair<Product, Double> pair1 : basketList) {
    if (pair1.first().equals(product)) {
        pair1.setSecond(pair.second() - quantity);
    }
}
HashMap<Company, HashMap<Product, Double>> originalHubStockList = expeditionStore.getOriginalHubStockList();
```

```
if (!originalHubStockList.containsKey(hub)) {
    originalHubStockList.put(hub, new HashMap<>());
}
HashMap<Product, Double> productDoubleHashMap = originalHubStockList.get(hub);
if (!productDoubleHashMap.containsKey(product)) {
    if (quantity != 0) {
        productDoubleHashMap.put(product, quantity);
    }
} else {
    if (quantity != 0) {
        productDoubleHashMap.put(product, productDoubleHashMap.get(product) + quantity);
    }
}
```

generateHubOrder

- Melhor caso: O(N^2)
- Pior caso: O(N^3)

```
public void generateHubOrder(Entity entity){
    Pair<Company, Integer> nearestHub = findNearestHubController.getNearestDistributionHub(entity);

    Company hub = nearestHub.first();
    TreeMap<Integer, List<Pair<Product, List<Pair<Double, Entity>>>>> orderMap = hub.getOrderMap();
    TreeMap<Integer, List<Pair<Product, Double>>> wantedProductsMap = hub.getWantedProductsMap();


    TreeMap<Integer, Basket> basketsMap = entity.getBasketsMap();
    for (Integer integer : basketsMap.keySet()) {
        Basket basket = basketsMap.get(integer);
        for (Pair<Product, Double> pair : basket.getProducts()) {
            Product product = pair.first();
            Double quantity = pair.second();
            if (!orderMap.containsKey(integer)) {
                orderMap.put(integer, new ArrayList<>());
            }
            List<Pair<Product, List<Pair<Double, Entity>>>> list = orderMap.get(integer);
            boolean found = false;
            for (Pair<Product, List<Pair<Double, Entity>>> pair1 : list) {
                if (pair1.first().equals(product)) {
                    List<Pair<Double, Entity>> list1 = pair1.second();
                    list1.add(new Pair<>(quantity, entity));
                    found = true;
                }
            }
            if (!found) {
                List<Pair<Double, Entity>> list1 = new ArrayList<>();
                list1.add(new Pair<>(quantity, entity));
                list.add(new Pair<>(product, list1));
```

```java
if (!wantedProductsMap.containsKey(integer)) {
    wantedProductsMap.put(integer, new ArrayList<>());
}
List<Pair<Product, Double>> list1 = wantedProductsMap.get(integer);
boolean found1 = false;
for (Pair<Product, Double> pair1 : list1) {
    if (pair1.first().equals(product)) {
        pair1.setSecond(pair1.second() + quantity);
        found1 = true;
    }
}
if (!found1) {
    list1.add(new Pair<>(product, quantity));
}
```

calculateDayOrders

- Melhor caso: O(N^2)
- Pior caso: O(N^2)

```java
public Expedition calculateDayOrders(int day){
    Expedition expedition = new Expedition(day);
    List<Company> companies = entityStore.getCompanyList();

    for (Company company1 : companies){
        if (company1.isDistributionHub()){
            TreeMap<Integer, List<Pair<Product, List<Pair<Double, Entity>>>>> orderMap = company1.getOrderMap();
            TreeMap<Integer, List<Pair<Product, List<Pair<Double, Productor>>>>> stockMap = company1.getStockMap();
            //System.out.println("Hub: " + company1.getName());
        }
    }

    for (Company company : companies) {
        if (company.isDistributionHub()){
            TreeMap<Integer, List<Pair<Product, List<Pair<Double, Entity>>>>> orderMap = company.getOrderMap();
            TreeMap<Integer, List<Pair<Product, List<Pair<Double, Productor>>>>> stockMap = company.getStockMap();
            if (company.getStockMap().size() != 0){
                if (orderMap.containsKey(day)) {
                    List<Pair<Product, List<Pair<Double, Entity>>>> dayOrders = orderMap.get(day);
                    List<Pair<Product, List<Pair<Double, Productor>>>> dayStocks = stockMap.get(day);
                    // add stock of previous 2 days
                    if (day > 2){
                        List<Pair<Product, List<Pair<Double, Productor>>>> dayStocksDayBeforeYesterday = stockMap.get(day - 2);
                        List<Pair<Product, List<Pair<Double, Productor>>>> dayStocksYesterday = stockMap.get(day - 1);
                        for (int y = 0; y < dayStocks.size(); y++) {
                            List<Pair<Double, Productor>> additionProducts1 = dayStocksYesterday.get(y).second();
                            List<Pair<Double, Productor>> additionProducts2 = dayStocksDayBeforeYesterday.get(y).second();
                            List<Pair<Double, Productor>> addedProducts = dayStocks.get(y).second();
                            for (int z = 0; z < addedProducts.size(); z++) {
                                Double sum = additionProducts1.get(z).first() + additionProducts2.get(z).first() + addedProducts.get(z).first();
                                addedProducts.get(z).setFirst(sum);
```

```
        }
    }
}

for (Pair<Product, List<Pair<Double, Entity>>> orders: dayOrders){
    List<Pair<Double, Productor>> products = new ArrayList<>();
    Product product = orders.first();
    for (int j = 0; j < dayStocks.size(); j++) {
        if (dayStocks.get(j).first().equals(product)) {
            products = dayStocks.get(j).second();
        }
    }
    Collections.sort(products, Comparator.comparing(p -> -p.first()));
    List<Pair<Double, Entity>> productOrders = orders.second();
    for (int i = 0; i < productOrders.size(); i++) {
        Pair<Double, Entity> order = productOrders.get(i);
        double quantity = order.first();
        Entity entity = order.second();
        Delivery delivery = new Delivery(entity, day);
        double quantitySupplied = products.get(0).first();

        Productor productor = products.get(0).second();
        if (quantitySupplied > quantity) {
            products.get(0).setFirst(quantitySupplied - quantity);
            quantitySupplied = quantity;
        }
        else if (quantitySupplied <= quantity) {
            products.get(0).setFirst(0.0);
            quantity = quantitySupplied;
        }
        if (quantitySupplied != 0) {
```

```
                        if (quantitySupplied != 0) {
                            delivery.addProduct(product, productor, quantity);
                            expedition.addDelivery(delivery, entity);
                        }
                        Collections.sort(products, Comparator.comparing(p -> -p.first()));

                    }
                }
            }
        }

    }
    return expedition;
}
```

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

Resumo:

Para esta user story, começamos por obter todos os produtores usados na entrega e obtivemos o menor caminho que passe por eles todos, ao usar a função calculateProductorsMinimumRoute, e depois começamos no ultimo produtor dessa lista e fazemos o menor caminho que passe por todos os hubs usados na expedição, usando a função calculateHubsMinimumRoute. Depois unimos esses dois caminhos.

calculateHubsMinimumRoute

- Melhor caso: O(N^2 * N Log(N))
- Pior caso: O(N^2 * N Log(N))

```
public Pair<List<Integer>, Pair<List<LinkedList<Entity>>, Integer>> calculateHubsMinimumRoute(Productor startPoint, Expedition expedition) {
    HashMap<Company, HashMap<Product, Double>> originalHubStockList = expeditionStore.getOriginalHubStockList();
    List<Company> companyList = entityStore.getCompanyList();
    List<Company> hubList = new ArrayList<>();
    for (Company company : companyList) {
        if (company.isDistributionHub()) {
            boolean hasStock = false;
            for (Product product : originalHubStockList.get(company).keySet()) {
                if (originalHubStockList.get(company).get(product) > 0) {
                    hasStock = true;
                }
            }
            if (hasStock) {
                hubList.add(company);
            }
        }
    }
    int minDistance = Integer.MAX_VALUE;
    LinkedList<Entity> minPath = new LinkedList<>();
    for (int i = 0; i < hubList.size(); i++) {
        LinkedList<Entity> path = new LinkedList<>();
        int distance = Algorithms.shortestPath(entityStore.getEntitiesGraph(), startPoint, hubList.get(i), Integer::compareTo, Integer::sum, zero: 0, path);
        if (distance < minDistance){
            minDistance = distance;
            minPath = path;
        }
    }
```

```
    Company firstHub = (Company) minPath.getLast();
    List<LinkedList<Entity>> hubRoutes = new ArrayList<>();
    List<Integer> distances = new ArrayList<>();
    distances.add(minDistance);
    hubRoutes.add(minPath);
    int totalDistance = 0;
    List<Company> hubListCopy = new ArrayList<>(hubList);
    hubListCopy.remove(firstHub);
    Pair<Company, Pair<LinkedList<Entity>, Integer>> hubRoute = calculateNextHub(firstHub, hubListCopy);
    hubRoutes.add(hubRoute.second().first());
    totalDistance += hubRoute.second().second();
    distances.add(hubRoute.second().second());
    Company nextHub = hubRoute.first();
    hubListCopy.remove(nextHub);
    do {
        hubRoute = calculateNextHub(nextHub, hubListCopy);
        distances.add(hubRoute.second().second());
        hubRoutes.add(hubRoute.second().first());
        totalDistance += hubRoute.second().second();
        nextHub = hubRoute.first();
        hubListCopy.remove(nextHub);
    } while (hubListCopy.size() > 0);

    return new Pair<>(distances, new Pair<>(hubRoutes, totalDistance));
}
```

calculateNextHub

- Melhor caso: O(N * Log(N))
- Pior caso: O(N * Log(N))

```
public Pair<Company, Pair<LinkedList<Entity>, Integer>> calculateNextHub(Company hub, List<Company> hubs){
    int minDistance = Integer.MAX_VALUE;
    LinkedList<Entity> minPath = new LinkedList<>();
    for (int i = 0; i < hubs.size(); i++) {
        LinkedList<Entity> path = new LinkedList<>();
        int distance = Algorithms.shortestPath(entityStore.getEntitiesGraph(), hub, hubs.get(i), Integer::compareTo, Integer::sum, zero: 0, path);
        if (distance < minDistance){
            minDistance = distance;
            minPath = path;
        }
    }
    Company returnHub = (Company) minPath.getLast();
    return new Pair<>(returnHub, new Pair<>(minPath, minDistance));
}
```

calculateProductorsMinimumRoute

- Melhor caso: O(N^2 * N Log(N))
- Pior caso: O(N^2 * N Log(N))

```
public Pair<List<Integer>, Pair<List<LinkedList<Entity>>, Integer>> calculateProductorsMinimumRoute(){
    List<LinkedList<Entity>> productorsRoutes = new ArrayList<>();
    List<Integer> distances = new ArrayList<>();
    int totalDistance = 0;
    List<Productor> productors = entityStore.getProductorsList();
    List<Productor> productorsCopy = new ArrayList<>(productors);
    Productor firstProductor = productors.get(0);
    productorsCopy.remove(firstProductor);
    Pair<Productor, Pair<LinkedList<Entity>, Integer>> productorRoute = calculateNextProductor(firstProductor, productorsCopy);
    productorsRoutes.add(productorRoute.second().first());
    totalDistance += productorRoute.second().second();
    distances.add(productorRoute.second().second());
    Productor nextProductor = productorRoute.first();
    productorsCopy.remove(nextProductor);
    do {
        productorRoute = calculateNextProductor(nextProductor, productorsCopy);
        distances.add(productorRoute.second().second());
        productorsRoutes.add(productorRoute.second().first());
        totalDistance += productorRoute.second().second();
        nextProductor = productorRoute.first();
        productorsCopy.remove(nextProductor);


    } while (productorsCopy.size() > 0);

    return new Pair<>(distances, new Pair<>(productorsRoutes, totalDistance));
}
```

calculateNextProductor

- Melhor caso: O(N * Log(N))
- Pior caso: O(N * Log(N))

```
public Pair<Productor, Pair<LinkedList<Entity>, Integer>> calculateNextProductor(Productor productor, List<Productor> productors){
    int minDistance = Integer.MAX_VALUE;
    LinkedList<Entity> minPath = new LinkedList<>();
    for (int i = 0; i < productors.size(); i++) {
        LinkedList<Entity> path = new LinkedList<>();
        int distance = Algorithms.shortestPath(entityStore.getEntitiesGraph(), productor, productors.get(i), Integer::compareTo, Integer::sum, zero: 0, path);
        if (distance < minDistance){
            minDistance = distance;
            minPath = path;
        }
    }
    Productor returnProductor = (Productor) minPath.getLast();
    return new Pair<>(returnProductor, new Pair<>(minPath, minDistance));

}
```

US311 - Para uma lista de expedição calcular estatísticas

Resumo:

Estatisticas por cabaz:

Usamos a função basketDayStats para o dia pedido. Esta função vai percorrer todas as entregas para esse dia e comparar aos pedidos dos clientes, de maneira a saber quais pedidos foram

completamente satisfeitos, percentagem total do cabaz satisfeito e nº de produtores que forneceram o cabaz.

Estatisticas por cliente:

Usamos a função clientStats, que vai percorrer todos os dias e para cada dia, comparar as entregas aos pedidos, de maneira a saber o numero de cabazes que foram completamente satisfeitos, numero de cabazes parcialmente satisfeitos e numero de fornecedores distintos que forneceram todos os seus cabazes

Estatisticas por produtor:

Usamos a função productorStats, que vai, por produtor, comparar o stock antes das entregas e o stock depois das entregas, de maneira a perceber o numero de cabazes fornecidos totalmente, numero de cabazes fornecidos parcialmente, numero de clientes distintos fornecidos ,numero de produtos totalmente esgotados e numero de hubs fornecidos

Estatisticas por hub:

Usamos a função hubStats, que vai, por hub, percorrer a lista de pedidos, para saber o numero de clientes distintos que recolhem cabazes em cada hub e percorrer a lista de stock para saber o numero de produtores distintos que fornecem cabazes para o hub.

basketDayStats

- Melhor caso: O(N^3)
- Pior caso: O(N^4)

```
public void basketDayStats(Expedition expedition, int day) {

    List<Pair<Entity, List<Delivery>>> dayDeliveries = expedition.getDayDeliveries();
    System.out.println();
    for (Pair<Entity,List<Delivery>> clientDeliveries : dayDeliveries) {
        double totalProducts = 0;
        double totalProductsSupplied = 0;
        int satisfiedProducts = 0;
        int partiallySatisfiedProducts = 0;
        int unsatisfiedProducts = 0;
        double percentageSatisfiedProducts = 0;
        List<Productor> productors = new ArrayList<>();
        System.out.println("Client:" + clientDeliveries.first().getName());
        System.out.println();
        for (Delivery delivery : clientDeliveries.second()) {
            ArrayList<Pair<Product, Pair<Productor, Double>>> products = delivery.getProductProductorAndQuantityDeliveredList();
            for (Pair<Product, Pair<Productor, Double>> productProductorAndQuantityDelivered : products) {
                double askedProduct = 0;
                ArrayList<Pair<Product, Double>> list = clientDeliveries.first().getBasketsMap().get(day).getProducts();
                for (Pair<Product, Double> productDoublePair : list) {
                    if (productDoublePair.first().getName().equals(productProductorAndQuantityDelivered.first().getName())) {
                        askedProduct = productDoublePair.second();

                    }
                }
```

```
    if (askedProduct == productProductorAndQuantityDelivered.second().second()) {
        satisfiedProducts++;
    } else  {
        partiallySatisfiedProducts++;
    }

    if (!productors.contains(productProductorAndQuantityDelivered.second().first())) {
        productors.add(productProductorAndQuantityDelivered.second().first());
    }
    totalProductsSupplied = totalProductsSupplied + productProductorAndQuantityDelivered.second().second();
    //System.out.println("Productor " + productProductorAndQuantityDelivered.second().first() + " delivered pr
}
```

```
        }
        ArrayList<Pair<Product, Double>> list1 = clientDeliveries.first().getBasketsMap().get(day).getProducts();
        for (Pair<Product, Double> productDoublePair : list1) {
            totalProducts = totalProducts + productDoublePair.second();
            if (productDoublePair.second() == 0) {
                satisfiedProducts++;
            }
        }
        unsatisfiedProducts = list1.size() - (satisfiedProducts + partiallySatisfiedProducts);
        percentageSatisfiedProducts = totalProductsSupplied / totalProducts * 100;
        percentageSatisfiedProducts = (double)Math.round(percentageSatisfiedProducts * 100d) / 100d; // round
        System.out.println("Amount of producers that contributed to this basket: " + productors.size());
        System.out.println("Amount of satisfied products: " + satisfiedProducts);
        System.out.println("Amount of partially satisfied products: " + partiallySatisfiedProducts);
        System.out.println("Amount of unsatisfied products: " + unsatisfiedProducts);
        System.out.println("Percentage of satisfied products: " + percentageSatisfiedProducts + "% (" + totalProductsSupplied + "/" + totalProducts + ")");
        System.out.println();
        System.out.println();

        productors.clear();
        System.out.println();

    }
}
```

clientStats

- Melhor caso: O(N^4)
- Pior caso: O(N^5)

```
public void clientStats(TreeMap<Integer, Expedition> expeditions) {
    HashMap<Entity, Pair<Pair<Integer, Integer>, List<Productor>>> clients = new HashMap<>(); // cliente, cabazes satisfeitos, cabazes parciais, lista de fornecedores
    for (Expedition expedition : expeditions.values()) {
        List<Pair<Entity, List<Delivery>>> dayDeliveries = expedition.getDayDeliveries();
        //System.out.println();
        for (Pair<Entity,List<Delivery>> clientDeliveries : dayDeliveries) {
            if (!clients.containsKey(clientDeliveries.first())) {
                clients.put(clientDeliveries.first(), new Pair<>(new Pair<>(0,0), new ArrayList<>()));
            }
            double totalProducts = 0;
            double totalProductsSupplied = 0;
            int satisfiedProducts = 0;
            int partiallySatisfiedProducts = 0;
            int unsatisfiedProducts = 0;
            double percentageSatisfiedProducts = 0;
            List<Productor> productors = new ArrayList<>();
            //System.out.println("Client:" + clientDeliveries.first().getName());
            //System.out.println();
            for (Delivery delivery : clientDeliveries.second()) {
                ArrayList<Pair<Product, Pair<Productor, Double>>> products = delivery.getProductProductorAndQuantityDeliveredList();
                for (Pair<Product, Pair<Productor, Double>> productProductorAndQuantityDelivered : products) {
                    double askedProduct = 0;
                    ArrayList<Pair<Product, Double>> list = clientDeliveries.first().getBasketsMap().get(expedition.getDayOffset()).getProducts();
                    for (Pair<Product, Double> productDoublePair : list) {
                        if (productDoublePair.first().getName().equals(productProductorAndQuantityDelivered.first().getName())) {
                            askedProduct = productDoublePair.second();
                        }
                    }
```

```
        if (askedProduct == productProductorAndQuantityDelivered.second().second()) {
            satisfiedProducts++;
        } else  {
            partiallySatisfiedProducts++;
        }

        if (!clients.get(clientDeliveries.first()).second().contains(productProductorAndQuantityDelivered.second().first())) {
            clients.get(clientDeliveries.first()).second().add(productProductorAndQuantityDelivered.second().first());
        }
        totalProductsSupplied = totalProductsSupplied + productProductorAndQuantityDelivered.second().second();
        //System.out.println("Productor " + productProductorAndQuantityDelivered.second().first() + " delivered product " + produ
    }




}
ArrayList<Pair<Product, Double>> list1 = clientDeliveries.first().getBasketsMap().get(expedition.getDayOffset()).getProducts();
for (Pair<Product, Double> productDoublePair : list1) {
    totalProducts = totalProducts + productDoublePair.second();
    if (productDoublePair.second() == 0) {
        satisfiedProducts++;
    }
}
unsatisfiedProducts = list1.size() - (satisfiedProducts + partiallySatisfiedProducts);
percentageSatisfiedProducts = totalProductsSupplied / totalProducts * 100;
```

```
            if (percentageSatisfiedProducts == 100) {
                clients.get(clientDeliveries.first()).first().setFirst(clients.get(clientDeliveries.first()).first().first() + 1);
            } else if (percentageSatisfiedProducts > 0) {
                clients.get(clientDeliveries.first()).first().setSecond(clients.get(clientDeliveries.first()).first().second() + 1);
            }



            productors.clear();


        }
    }
    for (Entity client : clients.keySet()) {
        System.out.println("Client: " + client.getName());
        System.out.println("Amount of satisfied baskets: " + clients.get(client).first().first());
        System.out.println("Amount of partially satisfied baskets: " + clients.get(client).first().second());
        System.out.println("Amount of producers that contributed to this client: " + clients.get(client).second().size());
        System.out.println();
    }
}
```

productorStats

- Melhor caso: O(N^3)
- Pior caso: O(N^4)

```java
public void productorStats(TreeMap<Integer, Expedition> expeditions) {

    List<Productor> productorList = entityStore.getProductorsList();
    List<Company> companyList = entityStore.getCompanyList();
    List<Pair<Company, List<Productor>>> companyProductorList = new ArrayList<>();

    for (Company company : companyList) {
        if (company.isDistributionHub()) {
            List<Productor> distinctProductors = new ArrayList<>();
            TreeMap<Integer, List<Pair<Product, List<Pair<Double, Productor>>>>> stockMap = company.getStockMap();
            for (Integer day : stockMap.keySet()) {
                List<Pair<Product, List<Pair<Double, Productor>>>> stocks = stockMap.get(day);
                for (Pair<Product, List<Pair<Double, Productor>>> stock : stocks) {
                    for (Pair<Double, Productor> productorDoublePair : stock.second()) {
                        if (!distinctProductors.contains(productorDoublePair.second()) && productorDoublePair.first() != 0) {
                            distinctProductors.add(productorDoublePair.second());
                        }
                    }
                }

            }
            companyProductorList.add(new Pair<>(company, distinctProductors));
        }
    }
```

```java
for (Productor productor : productorList){
    int suppliedHubs = 0;
    for (Pair<Company, List<Productor>> companyListPair : companyProductorList) {
        if (companyListPair.second().contains(productor)) {
            suppliedHubs++;
        }
    }
    int totallyDelivered = 0;
    int partiallyDelivered = 0;
    int soldOutProducts = 0;
    TreeMap<Integer, Basket> originalBasketsMap = productor.getOriginalBasketsMap();
    TreeMap<Integer, Basket> basketsMap = productor.getBasketsMap();
    for (Integer day : basketsMap.keySet()) {
        boolean totallyDeliveredBool = true;
        boolean partiallyDeliveredBool = false;
        Basket basket = basketsMap.get(day);
        Basket originalBasket = originalBasketsMap.get(day);
        List<Pair<Product, Double>> products = basket.getProducts();
        List<Pair<Product, Double>> originalProducts = originalBasket.getProducts();
        for (int i = 0; i < products.size(); i++) {
            if (products.get(i).second() != 0){
                products.get(i).first().setSoldOut(false);
                totallyDeliveredBool = false;
            }
            if (products.get(i).second() != originalProducts.get(i).second()) {
                partiallyDeliveredBool = true;
            }
        }
```

```
            if (totallyDeliveredBool) {
                totallyDelivered++;
            } else if (partiallyDeliveredBool) {
                partiallyDelivered++;
            }
        }

        TreeMap<Integer, Basket> basketsMap1 = productor.getBasketsMap();
        Basket basket = basketsMap1.get(1);
        List<Pair<Product, Double>> products = basket.getProducts();
        for (Pair<Product, Double> productDoublePair : products) {
            if (productDoublePair.first().isSoldOut()) {
                soldOutProducts++;
            }
            productDoublePair.first().setSoldOut(true);
        }
        System.out.println("Productor: " + productor.getName());
        System.out.println("Amount of totally delivered baskets: " + totallyDelivered);
        System.out.println("Amount of partially delivered baskets: " + partiallyDelivered);
        System.out.println("Amount of sold out products: " + soldOutProducts);
        System.out.println("Amount of hubs that received products from this productor: " + suppliedHubs);
        System.out.println();

    }

}
```

hubStats

- Melhor caso: O(N^3)
- Pior caso: O(N^4)

```
public void hubStats(TreeMap<Integer, Expedition> expeditions) {
    List<Company> companyList = entityStore.getCompanyList();

    for (Company company : companyList) {
        if (company.isDistributionHub()){
            List<Productor> distinctProductors = new ArrayList<>();
            TreeMap<Integer, List<Pair<Product, List<Pair<Double, Entity>>>>> orderMap = company.getOrderMap();
            TreeMap<Integer, List<Pair<Product, List<Pair<Double, Productor>>>>> stockMap = company.getStockMap();

            for (Integer day : orderMap.keySet()) {
                List<Pair<Product, List<Pair<Double, Productor>>>> stocks = stockMap.get(day);
                for (Pair<Product, List<Pair<Double, Productor>>> stock : stocks) {
                    for (Pair<Double, Productor> productorDoublePair : stock.second()) {
                        if (!distinctProductors.contains(productorDoublePair.second()) && productorDoublePair.first() != 0) {
                            distinctProductors.add(productorDoublePair.second());
                        }
                    }
                }

            }
```

```
            System.out.println("Hub: " + company.getName());
            if (orderMap.size() == 0) {
                System.out.println("Zero clients pickup orders at this hub");
            } else System.out.println("Amount of clients that pickup baskets at this hub: " + orderMap.get(1).get(0).second().size());

            if (distinctProductors.size() != 0){
                System.out.println("Amount of producers that deliver baskets to this hub: " + distinctProductors.size());
            } else {
                System.out.println("Zero producers deliver baskets to this hub");
            }

            System.out.println();
            System.out.println();
        }

    }
}
```