# RELATÓRIO DE ESINF

*Análise de complexidade*


*Trabalho realizado por:*

*Beatriz Neves 1211512*

*Clarisse Sousa 1211434*

*Cláudio Coelho 1211435*

*Martim Botelho 1211523*

*Filipe Duarte 1210959*

*João Castro 1210816*


*Projeto Integrador 1º Semestre, 2º Sprint*

*06/01/2023*

# US308

```java
public static List<Expedition> expeditionListOfACertainDay(int day, List<Localization> hubs) {
    //da um set de users de um determinado dia
    Set<Localization> users = mapDayClient.get(day);
    ArrayList<AVL.Node<ProductOrder>> list, biggerQuantityList = new ArrayList<>();
    Localization biggerProducer;
    ProductOrder productOrderDispatched;
    double totalQuantity, totalQuantityOrder, biggerQuantity;
    byte feedback;
    boolean found;
    int day3;
    String productName;
    HashMap<Localization, Set<ProductOrder>> mapProducerProducts;
    List<Expedition> expeditionList = new ArrayList<>();
    Set<ProductOrder> productOrderList;
    AVL<ProductOrder, ComparatorProductDay> avlProductOrder;

    day3 = SharedMethods.todayAndLast2Days(day);

    //vai percorrer todos os users existentes no set dos users do dia passado por parametro
    for (Localization user : users) {

        //vai ver se o user nao ta na lista dos hubs pois se tiver nao conta como cliente
        if (!hubs.contains(user)) {

            mapProducerProducts = new HashMap<>();
            //vai buscar a order de um user
            productOrderList = user.getUser().getOrder().get(day);

            //agora vai percorrer essa order toda para encontrar produtores para os produtos
            for (ProductOrder productOrder : productOrderList) {

                found = false;
                biggerProducer = producersList.get(0);
                biggerQuantity = 0;
                productName = productOrder.getProductName();

                //vai percorrer todos os producers existentes
                for (Localization producer : producersList) {

                    //vai buscar todos os produtos que o producer tem disponiveis para vender (ja contando com os do
                    avlProductOrder = producer.getUser().getProducerStock().get(day3);

                    //verifica se ele tem pelo menos um produto
                    if (avlProductOrder != null) {

                        list = new ArrayList<>();
                        //mete na "list" os produtos que sao do tipo do produto que o cliente quer
                        comparatorProductDay.find(avlProductOrder.root, productName, list);
                        //ve a quantidade total que o producer tem do produto que estamos a procura agora
                        totalQuantity = totalQuantity(list);
                        //ve a quantidade de produto que o cliente quer
                        totalQuantityOrder = productOrder.getProductQuantity();

                        //se a quantidade que o producer tem for maior que a que o cliente quer
                        if (totalQuantity >= totalQuantityOrder) {
                            //é tudo dado ao cliente e da-se ja break do loop por motivos de eficiencia
                            biggerQuantity = totalQuantityOrder;
                            biggerQuantityList = list;
                            biggerProducer = producer;
                            found = true;
                            break;

                            //este else é usado sempre para registar a maior quantidade existente
                            //de um determinado produto pois se nao se satisfizer a condiçao de cima
                            //é entregue ao cliente a maior quantidade de produto possivel
                        } else if (totalQuantity > biggerQuantity) {
```

```
                    } else if (totalQuantity > biggerQuantity) {
                        biggerQuantity = totalQuantity;
                        biggerQuantityList = list;
                        biggerProducer = producer;
                    }
                }
            }

            //se foi encontrada alguma quantidade de um determinado produto
            if (biggerProducer != null) {

                //se esta quantidade nao for a total pedida pelo cliente
                if (!found) {
                    //se for zero
                    if (biggerQuantity == 0) {
                        //da-mos um feedback de 0 ou seja o cliente nao recebeu absolutamente nada
                        feedback = 0;
                        //se for maior que zero mas nao for a total que o cliente queria
                    } else {
                        //com este metodo alteramos a quantidade de produto que o producer tem
                        //pois vai ser dado ao cliente
                        SharedMethods.alterProducerStock(biggerQuantityList, biggerQuantity);
                        //da-mos um feedback de 1 ou seja o cliente recebeu alguma coisa mas nao foi a total que
                        feedback = 1;
                    }
                    //se o cliente recebeu a quantidade total de produto que ele queria
                } else {
                    //com este metodo alteramos a quantidade de produto que o producer tem
                    //pois vai ser dado ao cliente
                    SharedMethods.alterProducerStock(biggerQuantityList, biggerQuantity);
                    //da-mos um feedback de 1 ou seja o cliente recebeu o produto na totalidade
                    feedback = 2;
                }

                //da-mos o feedback de um produto na order do cliente
                productOrder.setFeedBack(feedback);
                //cria-mos um objecto do tipo product order para registar o produto que vai ser dado ao cliente
                //e retirado ao produtor
                productOrderDispatched = new ProductOrder(productName, biggerQuantity, productOrder.getDay());

                //este mapa é utilizado para ver o que um producer deu de um produto a um certo cliente
                if (mapProducerProducts.containsKey(biggerProducer)) {
                    mapProducerProducts.get(biggerProducer).add(productOrderDispatched);
                } else {
                    Set<ProductOrder> productOrderSet = new HashSet<>();
                    productOrderSet.add(productOrderDispatched);
                    mapProducerProducts.put(biggerProducer, productOrderSet);
                }

            }
        }

        //por fim adiciono a lista de expediçao o clinete
        //a lista dos produtos que ele pediu com um determinado feedback (para ver se recebeu na totalidade o pr
        //e um mapa que tem como key produtores e depois tem um set que tem o que eles deram ao cliente
        expeditionList.add(new Expedition(user, productOrderList, mapProducerProducts));

    }
}

return expeditionList;

}
```

*Este método possui uma complexidade no pior caso de $O(n^3 \log n)$.*

```
    private static double totalQuantity(ArrayList<AVL.Node<ProductOrder>> list) {

        double total = 0;

        if (list.size() > 0) {
            for (AVL.Node<ProductOrder> productOrderNode : list) {
                total += productOrderNode.getElement().getProductQuantity();
            }
        }

        return total;

    }
}
```

*Este método possui uma complexidade no pior caso de O(n).*

# US309

```java
public static List<Expedition> expeditionListOfACertainDay(int nClosest, int day, List<Localization> hubs) {

    Set<Localization> users = mapDayClient.get(day);
    Map<Localization, ArrayList<Localization>> closestProducersPerClient = nClosestProducers(nClosest, users);
    ArrayList<AVL.Node<ProductOrder>> list, biggerQuantityList = new ArrayList<>();
    Localization biggerProducer;
    ProductOrder productOrderDispatched;
    double totalQuantity, totalQuantityOrder, biggerQuantity;
    byte feedback;
    boolean found;
    int day3;
    String productName;
    HashMap<Localization, Set<ProductOrder>> mapProducerProducts;
    List<Expedition> expeditionList = new ArrayList<>();
    Set<ProductOrder> productOrderList;
    AVL<ProductOrder, ComparatorProductDay> avlProductOrder;

    day3 = SharedMethods.todayAndLast2Days(day);

    for (Localization user : users) {

        if (!hubs.contains(user)) {

            mapProducerProducts = new HashMap<>();
            productOrderList = user.getUser().getOrder().get(day);

            for (ProductOrder productOrder : productOrderList) {

                found = false;
                biggerProducer = null;
                biggerQuantity = 0;
                productName = productOrder.getProductName();

                for (Localization producer : closestProducersPerClient.get(user)) {

                    avlProductOrder = producer.getUser().getProducerStock().get(day3);

                    if (avlProductOrder != null) {

                        list = new ArrayList<>();
                        comparatorProductDay.find(avlProductOrder.root, productName, list);
                        totalQuantity = SharedMethods.totalQuantity(list);
                        totalQuantityOrder = productOrder.getProductQuantity();

                        if (totalQuantity >= totalQuantityOrder) {
                            biggerQuantity = totalQuantityOrder;
                            biggerQuantityList = list;
                            biggerProducer = producer;
                            found = true;
                            break;

                        } else if (totalQuantity > biggerQuantity) {
                            biggerQuantity = totalQuantity;
                            biggerQuantityList = list;
                            biggerProducer = producer;
                        }
                    }
                }

                if (biggerProducer != null) {

                    if (!found) {
                        if (biggerQuantity == 0) {
                            feedback = 0;
                        } else {
```

```
                        SharedMethods.alterProducerStock(biggerQuantityList, biggerQuantity);
                        feedback = 1;
                    }
                } else {
                    SharedMethods.alterProducerStock(biggerQuantityList, biggerQuantity);
                    feedback = 2;
                }

                productOrder.setFeedBack(feedback);
                productOrderDispatched = new ProductOrder(productName, biggerQuantity, productOrder.getDay());

                if (mapProducerProducts.containsKey(biggerProducer)) {
                    mapProducerProducts.get(biggerProducer).add(productOrderDispatched);
                } else {
                    Set<ProductOrder> productOrderSet = new HashSet<>();
                    productOrderSet.add(productOrderDispatched);
                    mapProducerProducts.put(biggerProducer, productOrderSet);
                }

            }
        }

        expeditionList.add(new Expedition(user, productOrderList, mapProducerProducts));

    }
}

return expeditionList;

}
```

*Este método possui uma complexidade no pior caso de $O(n^3 \log n)$.*

```
private static double totalQuantity(ArrayList<AVL.Node<ProductOrder>> list) {

    double total = 0;

    if (list.size() > 0) {
        for (AVL.Node<ProductOrder> productOrderNode : list) {
            total += productOrderNode.getElement().getProductQuantity();
        }
    }

    return total;

}
```

*Este método possui uma complexidade no pior caso de $O(n)$.*

```java
private static Map<Localization, ArrayList<Localization>> nClosestProducers(int n, Set<Localization> clients) {
    ArrayList<LinkedList<Localization>> paths;
    ArrayList<Double> dists;
    ArrayList<CompanyMediaPair> cmpList;
    ArrayList<Localization> cmpSet;

    int arraySize;
    int auxN;

    Map<Localization, ArrayList<Localization>> mapClosestProducers = new HashMap<>();

    for(Localization client : clients) {
        paths = new ArrayList<>();
        dists = new ArrayList<>();
        cmpList = new ArrayList<>();
        cmpSet = new ArrayList<>();
        Algorithms.shortestPathsWeighted(mapGraph, client.getUser().getHub(), paths, dists);
        arraySize = paths.size();
        for(int x = 0 ; x < arraySize ; x++){
            Localization producer = paths.get(x).getLast();
            if(producer.getUser().getUserCode() == 'P') {
                cmpList.add(new CompanyMediaPair(producer,dists.get(x)));
            }
        }
        Collections.sort(cmpList);
        auxN = Math.min(n, cmpList.size());
        for(int x = 0 ; x < auxN ; x++) {
            cmpSet.add(cmpList.get(x).getLocalization());
        }
        mapClosestProducers.put(client,cmpSet);
    }
    return mapClosestProducers;
}
```

*Este método possui uma complexidade no pior caso de $O(n^3)$.*

## US310

```java
public static Distribution shortestPath(List<Expedition> expeditionList) {
    ArrayList<LinkedList<Localization>> paths = new ArrayList<>();
    ArrayList<Double> dists = new ArrayList<>();
    ArrayList<Localization> hubsWhereProducerHasToGo;

    HashMap<Localization, Set<Localization>> producerHubs = new HashMap<>();
    HashMap<Localization, Set<Expedition>> basketsPerHub = new HashMap<>();
    HashMap<Localization, ArrayList<Edge<Localization, Double>>> distanceBetweenEveryCrossingPoint = new HashMap<>()

    Set<Localization> producersOfEachExpedition;

    double totalDistance = 0;

    for (Expedition expedition : expeditionList) {
        producersOfEachExpedition = expedition.getListOfProductsDispatched().keySet();
        Localization hub = expedition.getClient().getUser().getHub();
        for (Localization producer : producersOfEachExpedition) {
            if (producerHubs.get(producer) != null) {
                producerHubs.get(producer).add(hub);
            } else {
                Set<Localization> setLocalization = new TreeSet<>(); //para o contains tem complexidade logn
                setLocalization.add(hub);
                producerHubs.put(producer, setLocalization);
            }
        }
        if(basketsPerHub.get(hub) != null) {
            basketsPerHub.get(hub).add(expedition);
        } else {
            Set<Expedition> setExpedition = new HashSet<>();
            setExpedition.add(expedition);
            basketsPerHub.put(hub,setExpedition);
        }
    }

    for (Localization producer : producerHubs.keySet()) {
        Algorithms.shortestPathsWeighted(mapGraph, producer, paths, dists);
        ArrayList<Edge<Localization, Double>> edges = new ArrayList<>();
        LinkedList<Localization> path;
        Localization possibleHub, closestHub = null;
        double compare = 100000000, distanceBetween2Localization;

        for(int x = 0 ; x < paths.size() ; x++) {
            path = paths.get(x);
            distanceBetween2Localization = dists.get(x);
            possibleHub = path.getLast();
            if(producerHubs.get(producer).contains(possibleHub) && distanceBetween2Localization < compare){
                compare = distanceBetween2Localization;
                closestHub = possibleHub;
            }
        }
        hubsWhereProducerHasToGo = new ArrayList<>(producerHubs.get(producer));
        hubsWhereProducerHasToGo.remove(closestHub);
        edges.add(new Edge<>(producer,closestHub,compare));

        totalDistance += findEdges(hubsWhereProducerHasToGo,closestHub,edges,paths,dists) + compare;
        distanceBetweenEveryCrossingPoint.put(producer,edges);
    }

    return new Distribution(basketsPerHub,distanceBetweenEveryCrossingPoint,totalDistance);
}
```

*Este método possui uma complexidade no pior caso de $O(n^3)$.*

```java
    private static double findEdges(ArrayList<Localization> hubsWhereProducerHasToGo, Localization origin, ArrayList<Edg
        if(hubsWhereProducerHasToGo.size() == 0){
            return 0;
        }

        Algorithms.shortestPathsWeighted(mapGraph, origin, paths, dists);
        LinkedList<Localization> path;
        Localization possibleHub, closestHub = null;
        double compare = 100000000, distanceBetween2Localization;

        for(int x = 0 ; x < paths.size() ; x++) {
            path = paths.get(x);
            distanceBetween2Localization = dists.get(x);
            possibleHub = path.getLast();
            if(hubsWhereProducerHasToGo.contains(possibleHub) && distanceBetween2Localization < compare){
                compare = distanceBetween2Localization;
                closestHub = possibleHub;
            }
        }

        hubsWhereProducerHasToGo.remove(closestHub);
        edges.add(new Edge<>(origin,closestHub,compare));


        return findEdges(hubsWhereProducerHasToGo,closestHub,edges,paths,dists) + compare;
    }
}
```

*Este método possui uma complexidade no pior caso de $O(n^2)$.*

## US311

```java
public static List<BasketStatistics> getExpeditionStatisticsPerBasket(List<Expedition> expeditions){

    byte feedBack;
    int totalProductsCompleted, totalProductsNotComplete, totalProductsEmpty, totalOfProducers;
    double successPercentage;
    List<BasketStatistics> basketStatisticsList = new ArrayList<>();
    Set<ProductOrder> productOrderList;

    for (Expedition expedition : expeditions){

        totalProductsCompleted = 0;
        totalProductsNotComplete = 0;
        totalProductsEmpty = 0;
        productOrderList = expedition.getListOfProductsOrder();

        for (ProductOrder productOrder : productOrderList){

            feedBack = productOrder.getFeedBack();
            if (feedBack == 0){
                totalProductsEmpty ++;
            }else if (feedBack == 1){
                totalProductsNotComplete ++;
            }else if (feedBack == 2){
                totalProductsCompleted ++;
            }

        }

        totalOfProducers = expedition.getListOfProductsDispatched().keySet().size();
        successPercentage = totalProductsCompleted / (double) (productOrderList.size()) * 100;
        basketStatisticsList.add(new BasketStatistics(expedition.getClient(), totalProductsCompleted, totalProductsN
                totalProductsEmpty, successPercentage, totalOfProducers));
    }

    return basketStatisticsList;

}
```

*Este método possui uma complexidade no pior caso de $O(n^2)$.*

```java
public static List<HubStatistics> getExpeditionStatisticsPerHub(List<Localization> hubs, List<Expedition> expedition

    if (hubs == null || hubs.isEmpty()){
        return null;
    }

    int totalClients, totalProducers;
    Localization hubKey, client;
    List<HubStatistics> statistics = new ArrayList<>();
    Map<Localization, Set<Localization>> mapHubPerUser = new HashMap<>();

    for (Localization hub : hubs) {

        mapHubPerUser.put(hub, new HashSet<>());

    }

    for (Expedition expedition : expeditions){

        client = expedition.getClient();
        hubKey = client.getUser().getHub();
        mapHubPerUser.get(hubKey).add(client);
        mapHubPerUser.get(hubKey).addAll(expedition.getListOfProductsDispatched().keySet());

    }

    for (Localization hub : mapHubPerUser.keySet()){

        totalClients = 0;
        totalProducers = 0;

        for (Localization user : mapHubPerUser.get(hub)){
        totalClients = 0;
        totalProducers = 0;

        for (Localization user : mapHubPerUser.get(hub)){

            switch (user.getUser().getUserCode()) {
                case 'P', 'p' -> totalProducers++;
                case 'C', 'c', 'E', 'e' -> totalClients++;
                default -> System.out.println("Wrong Code!!!");
            }

        }

        statistics.add(new HubStatistics(hub ,totalClients, totalProducers));

    }

    return statistics;

}
}
```

*Este método possui uma complexidade no pior caso de $O(n^2)$.*

```java
public static List<ProducerStatistics> getExpeditionStatisticsPerProducer(List<Localization> producers, List<Expedit

    byte feedBack;
    int totalBasketsCompleted, totalBasketsNotCompleted, totalClientsCompleted, totalProductsOutOfStock, totalHubsCo
    List<ProducerStatistics> producerStatisticsList = new ArrayList<>();
    Set<ProductOrder> productOrderSet;
    List<Localization> distinctClients = new ArrayList<>();

    for (Localization producer : producers){

        totalProductsOutOfStock = 0;
        totalBasketsNotCompleted = 0;
        totalBasketsCompleted = 0;
        totalHubsCompleted = 0;
        totalProductsNotCompleted = 0;

        for (Expedition expedition : expeditions){

            productOrderSet = expedition.getListOfProductsDispatched().get(producer);

            if (!distinctClients.contains(expedition.getClient())) distinctClients.add(expedition.getClient());

            for (ProductOrder productOrder : productOrderSet) {
                if (productOrder.getProductQuantity() == 0) totalProductsOutOfStock++;

                feedBack = productOrder.getFeedBack();
                if (feedBack == 1 || feedBack == 0) totalProductsNotCompleted ++;
            }

            if (totalProductsNotCompleted == 0) totalBasketsCompleted++;
            else                                totalBasketsNotCompleted++;
        }
        totalClientsCompleted = distinctClients.size();

        producerStatisticsList.add(new ProducerStatistics(producer, totalBasketsCompleted, totalBasketsNotCompleted,

    }
    return producerStatisticsList;
}
```

*Este método possui uma complexidade no pior caso de $O(n^3)$.*

```java
public static List<ClientStatistics> getExpeditionStatisticsPerClient(List<Expedition> expeditions){

    byte feedBack;
    int totalBasketsCompleted, totalBasketsNotCompleted, totalDistinctSuppliersCompleted;
    List<ClientStatistics> clientStatisticsList = new ArrayList<>();
    Set<ProductOrder> productOrderSet;
    List<Localization> distinctSuppliers = new ArrayList<>();

    for (Expedition expedition : expeditions){

        totalBasketsNotCompleted = 0;
        totalBasketsCompleted = 0;
        totalDistinctSuppliersCompleted = 0;

        productOrderSet = expedition.getListOfProductsOrder();

        for (ProductOrder productOrder : productOrderSet){

            feedBack = productOrder.getFeedBack();
            if (feedBack == 1 || feedBack == 0) totalBasketsNotCompleted ++;
            else                                totalBasketsCompleted++;

        }
        totalDistinctSuppliersCompleted = expedition.getListOfProductsDispatched().keySet().size();;

        clientStatisticsList.add(new ClientStatistics(expedition.getClient(), totalBasketsCompleted, totalBasketsNot

    }
    return clientStatisticsList;
}
}
```

*Este método possui uma complexidade no pior caso de $O(n^2)$.*