ANÁLISE DE COMPLEXIDADE

Projeto Integrador - ESINF

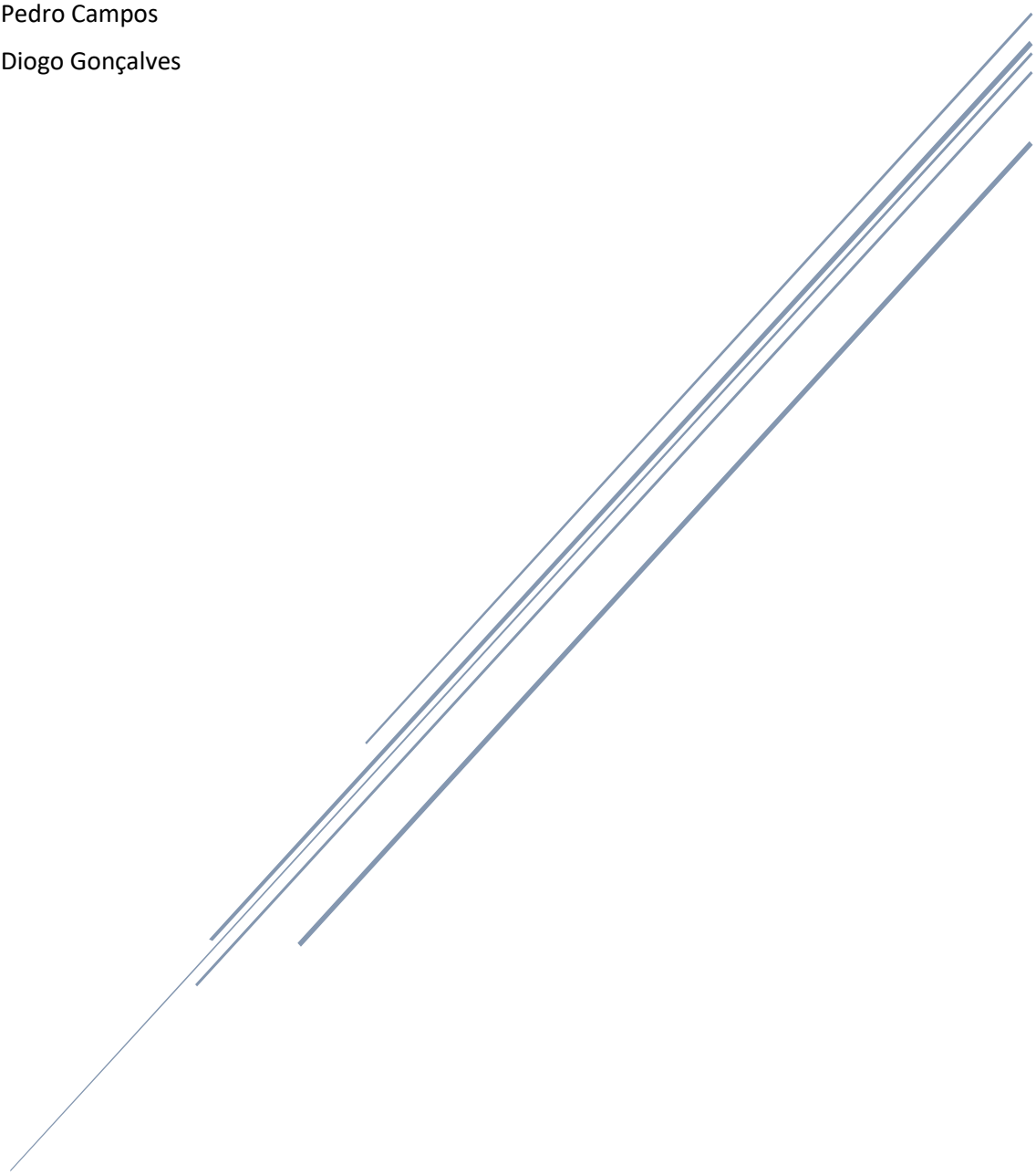
1211546 – João Costa

1211504 – Pedro Lopes

1211514 – Diogo Costa

1211511 – Pedro Campos

1211509 – Diogo Gonçalves



ISEP

Engenharia Informática

Índice

US301 – Construir a rede de distribuição de cabazes a partir da informação fornecida nos ficheiros.

importEntityInfo

- Melhor caso
  O(n^2)
- Pior caso
  O(n^2)

```java
/**
 * Imports the entity information from a file to entityStore
 * @param file the file to import from
 * @throws FileNotFoundException if the file is not found
 */
public void importEntityInfo(String file) throws FileNotFoundException {
    List<Entity> entityList = new ArrayList<>(); // list containing all the entities
    Scanner sc = new Scanner(new File(file));
    int lineNumber = 1;
    sc.nextLine(); // skip header
    while (sc.hasNextLine()) {
        try {
            String line = sc.nextLine();
            String[] elements = line.split( regex: ",");
            Location entityLocation = new Location(elements[0], Double.parseDouble(elements[1]), Double.parseDouble(elements[2]));
            char entityType = elements[3].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
                    entityList.add(new Client(entityLocation, elements[3]));
                    break;
                case 'E': // if the entity is a company
                    entityList.add(new Company(entityLocation, elements[3]));
                    break;
                case 'P': // if the entity is a productor
                    entityList.add(new Productor(entityLocation, elements[3]));
                    break;
            }
        } catch (Exception e) {
            System.out.printf("Error in line: %d %s ", lineNumber, e.getCause());
        }
        lineNumber++;
    }

    MapGraph<Entity, Integer> entitiesGraph = entityStore.getEntitiesGraph();
    for (Entity e : entityList) {
        entitiesGraph.addVertex(e);
    }

}
```

findEntityByLocationId

- Melhor caso
  O(1)
- Pior caso
  O(V)

```java
/**
 * Finds the correct entity object, given its location id
 * @param entityList list of entities
 * @param locationId location id of the entity
 * @return the entity object
 */
public Entity findEntityByLocationId(List<Entity> entityList, String locationId){
    for (Entity e : entityList) {
        if (e.getLocation().getLocationId().equals(locationId)) {
            return e;
        }
    }
    return null;
}
```

importEntityConnections

- Melhor caso
  $O(nV^2)$
- Pior caso
  $O(nV^2)$

```java
/**
 * Imports the entity connections from a file and adds them to the entityStore
 * @param file the file to import from
 * @throws FileNotFoundException if the file is not found
 */
public void importEntityConnections(String file) throws FileNotFoundException {
    Scanner sc = new Scanner(new File(file));
    int lineNumber = 1;
    sc.nextLine(); // skip header
    while (sc.hasNextLine()) {
        try {
            String line = sc.nextLine();
            String[] elements = line.split( regex: ",");
            String firstLocationId = elements[0]; // location id of the first entity
            String secondLocationId = elements[1]; // location id of the second entity
            int distance = Integer.parseInt(elements[2]); // distance between the two locations
            MapGraph<Entity, Integer> entitiesGraph = entityStore.getEntitiesGraph();
            List<Entity> entityList = entitiesGraph.vertices(); // get all the entities from the graph
            Entity firstEntity = findEntityByLocationId(entityList, firstLocationId); // find the first entity by location id
            Entity secondEntity = findEntityByLocationId(entityList, secondLocationId); // find the second entity by location id
            entitiesGraph.addEdge(firstEntity, secondEntity, distance); // add the edge to the graph
        } catch (Exception e) {
            System.out.printf("Error in line: %d %s ", lineNumber, e.getCause());
        }
        lineNumber++;
    }
}
```

US302 – Verificar se o grafo carregado é conexo e devolver o número mínimo de ligações necessário para nesta rede qualquer cliente/produtor conseguir contactar um qualquer outro.

isConnected

- Melhor caso
  O(1)
- Pior caso
  O(E)

```
public boolean isConnected(){
    LinkedList<Entity> queue = Algorithms.DepthFirstSearch(mapGraph,mapGraph.vertex( key: 0));
    if(queue == null){
        return false;
    }
    if (mapGraph.numVertices() == queue.size()){
        return true;
    }else {
        return false;
    }
}
```

ignoreWeights

- Melhor caso
  O(E)
- Pior caso
  O(E)

```
public MapGraph<Entity, Integer> IgnoreWeights(){
    MapGraph<Entity, Integer> noWeightsMapGraph = new MapGraph<>( directed: false);
    noWeightsMapGraph = mapGraph.clone();
    for (Edge<Entity,Integer> edge : noWeightsMapGraph.edges()){
            edge.setWeight(1);
    }
    return noWeightsMapGraph;
}
```

minimumNumberOfConnections

- Melhor caso
  O(V^3)
- Pior caso
  O(V^3)

```
public int minimumNumberOfConnections(){
    LinkedList<Entity> path = new LinkedList<>();
    int maxNumberOfConnections = 0;
    MapGraph<Entity, Integer> noWeightsMapGraph = IgnoreWeights();
    for(Entity entity : noWeightsMapGraph.vertices()) {
        LinkedList<Entity> queue = new LinkedList<>();
        queue = Algorithms.BreadthFirstSearch(noWeightsMapGraph, entity);
        Algorithms.shortestPath(noWeightsMapGraph,entity,queue.getLast(),Integer::compareTo,Integer::sum, zero: 0,path);
        if(maxNumberOfConnections < path.size()-1)
            maxNumberOfConnections = path.size()-1;
    }
    return maxNumberOfConnections;
}
```

US303 – Definir os hubs da rede de distribuição

getAverageProximityDistanceOfCompany

- Melhor caso
  O(V*E*log(V))
- Pior caso
  O(V*E*log(V))

```
/**
 * Method to get the average proximity distance to all other entities for a respective company
 * @param company the company to get the average proximity distance to all other entities
 * @return the average proximity distance to all other entities
 */
public int getAverageProximityDistanceOfCompany(Company company) {
    MapGraph<Entity, Integer> entityGraph = entityStore.getEntitiesGraph();
    int totalDistance = 0;

    ArrayList<LinkedList<Entity>> paths = new ArrayList<>();
    ArrayList<Integer> pathsDistances = new ArrayList<>();

    Algorithms.shortestPaths(entityGraph, company, Integer::compareTo, Integer::sum, zero: 0, paths, pathsDistances);
    for (int i = 0; i < paths.size(); i++) {
        if (!(paths.get(i).getLast().equals(company))) {
            totalDistance += pathsDistances.get(i);
        }

    }
    int averageDistance = totalDistance / (paths.size() - 1);

    return averageDistance;
}
```

getCompanyAverageDistanceList

- Melhor caso
  O(V)
- Pior caso
  O(V)

```java
/**
 * Method to get a list of all companies with their respective average proximity distance to all other entities
 * @return a list of pairs with the companies and their respective average proximity distance to all other entities
 */
public List<Pair<Company, Integer>> getCompanyAverageDistanceList() {
    List<Pair<Company, Integer>> companyAverageDistanceList = new ArrayList<>();

    for (Entity entity : entityStore.getEntitiesGraph().vertices()) {
        if (entity instanceof Company) {
            int averageDistance = getAverageProximityDistanceOfCompany((Company) entity);
            companyAverageDistanceList.add(new Pair<>((Company) entity, averageDistance));
        }

    }
    return companyAverageDistanceList;
}
```

defineHubs

- Melhor caso
  O(n*log(n))
- Pior caso
  O(n*log(n))

```java
/**
 * Method to define the n companies with the lowest average proximity distance to all other entities as Distribution Hubs
 */
public void defineHubs(int numberOfHubs) {

    List<Pair<Company, Integer>> companyAverageDistanceList = getCompanyAverageDistanceList();
    Collections.sort(companyAverageDistanceList, Comparator.comparing(p -> p.second()));
    if (numberOfHubs > companyAverageDistanceList.size()) {
        numberOfHubs = companyAverageDistanceList.size();
        System.out.println("Number of hubs is greater than the number of companies, setting number of hubs to " + numberOfHubs);
    }

    for (int i = 0; i < numberOfHubs; i++) {
        companyAverageDistanceList.get(i).first().setDistributionHub(true);
        System.out.println("Company " + companyAverageDistanceList.get(i).first().getName() + " is now a distribution hub with a aver
    }
}
```

US304 – Para cada cliente determinar o hub mais próximo.

findNearestHubController

- Melhor caso
  O(1)
- Pior caso
  O(V*E*log(V))

```java
/**
 * Method to get a list of reachable Distribution Hubs from a given entity with the respective path distance
 * @param entity the entity to find the Distribution Hubs from
 * @return a list of pairs with the reachable Distribution Hubs and the distance to each one
 */
public List<Pair<Company, Integer>> getEntityReachableDistributionHubs(Entity entity) {
    List<Pair<Company, Integer>> reachableDistributionHubs = new ArrayList<>();
    MapGraph<Entity, Integer> entityGraph = entityStore.getEntitiesGraph();
    ArrayList<LinkedList<Entity>> paths = new ArrayList<>();
    ArrayList<Integer> pathsDistances = new ArrayList<>();

    Algorithms.shortestPaths(entityGraph, entity, Integer::compareTo, Integer::sum, zero: 0, paths, pathsDistances);
    for (int i = 0; i < paths.size(); i++) {
        if (paths.get(i).getLast() instanceof Company && ((Company) paths.get(i).getLast()).isDistributionHub()) {
            reachableDistributionHubs.add(new Pair<>((Company) paths.get(i).getLast(), pathsDistances.get(i)));
        }
    }

    return reachableDistributionHubs;
}
```

getNeasrestDistributionHub

- Melhor caso
  O(n)
- Pior caso
  O(n)

```java
/**
 * Method to get the nearest Distribution Hub to a given entity
 * @param entity the entity to find the nearest Distribution Hub from
 * @return a pair with the nearest company and the distance to it
 */
public Pair<Company, Integer> getNearestDistributionHub(Entity entity) {
    List<Pair<Company, Integer>> reachableDistributionHubs = getEntityReachableDistributionHubs(entity);
    Pair<Company, Integer> nearestDistributionHub = null;
    for (Pair<Company, Integer> pair : reachableDistributionHubs) {
        if (nearestDistributionHub == null || pair.second() < nearestDistributionHub.second()) {
            nearestDistributionHub = pair;
        }
    }
    return nearestDistributionHub;
}
```

getNearestDistributionHubForEachClient

- Melhor caso
  O(1)
- Pior caso
  O(E)

```java
/**
 * Method to get the nearest Distribution Hub for each client
 * @return a list of pairs with the nearest Distribution Hub for each client and the distance to it
 */
public List<Pair<Entity, Pair<Company, Integer>>> getNearestDistributionHubForEachClient() {
    List<Pair<Entity, Pair<Company, Integer>>> nearestDistributionHubForAllClients = new ArrayList<>();
    for (Entity entity : entityStore.getEntitiesGraph().vertices()) {
        if (!(entity instanceof Productor)) {
            if(entity instanceof Company){
                if(((Company) entity).isDistributionHub()){
                    continue;
                }
            }
            nearestDistributionHubForAllClients.add(new Pair<>(entity, getNearestDistributionHub(entity)));
        }
    }
    return nearestDistributionHubForAllClients;
}
```

US305 – Rede que conecte todos os clientes e produtores agrícolas com uma distância total mínima

getMinimumDistanceNetworkDistance

- Melhor caso
  O(1)
- Pior caso
  O(E)

```java
/**
 * Method to get minimum total distance of the network to all entities
 * @return the minimum network distance
 */
public int getMinimumNetworkDistance(MapGraph<Entity,Integer> mapGraph){
    int totalDistance = 0;
    for (Edge<Entity, Integer> edge : mapGraph.edges()) {
        totalDistance = totalDistance + edge.getWeight();
    }
    return totalDistance;
}
```

getMinimumDistGraph

- Melhor caso
  O(1)
- Pior caso
  O(E log V)

```java
/**
 * Method to get the minimum network to connect all entities
 * @return the minimum network graph to connect all entities
 */
public MapGraph<Entity, Integer> getMinimumDistGraph() {
    MapGraph<Entity, Integer> minimumDistGraph = App.getInstance().getOrganization().getEntityStore().getEntitiesGraph().clone();
    minimumDistGraph = Algorithms.kruskallAlgorithm(minimumDistGraph);
    return minimumDistGraph;
}
```