ESINF

RELATÓRIO TRABALHO 2

João Teixeira (1210957)
Jonas Antunes (1181478)
José Rente (1211155)
Marco Maia (1210951)
Ruben Ferreira (1210954)
2DD – G31

Índice

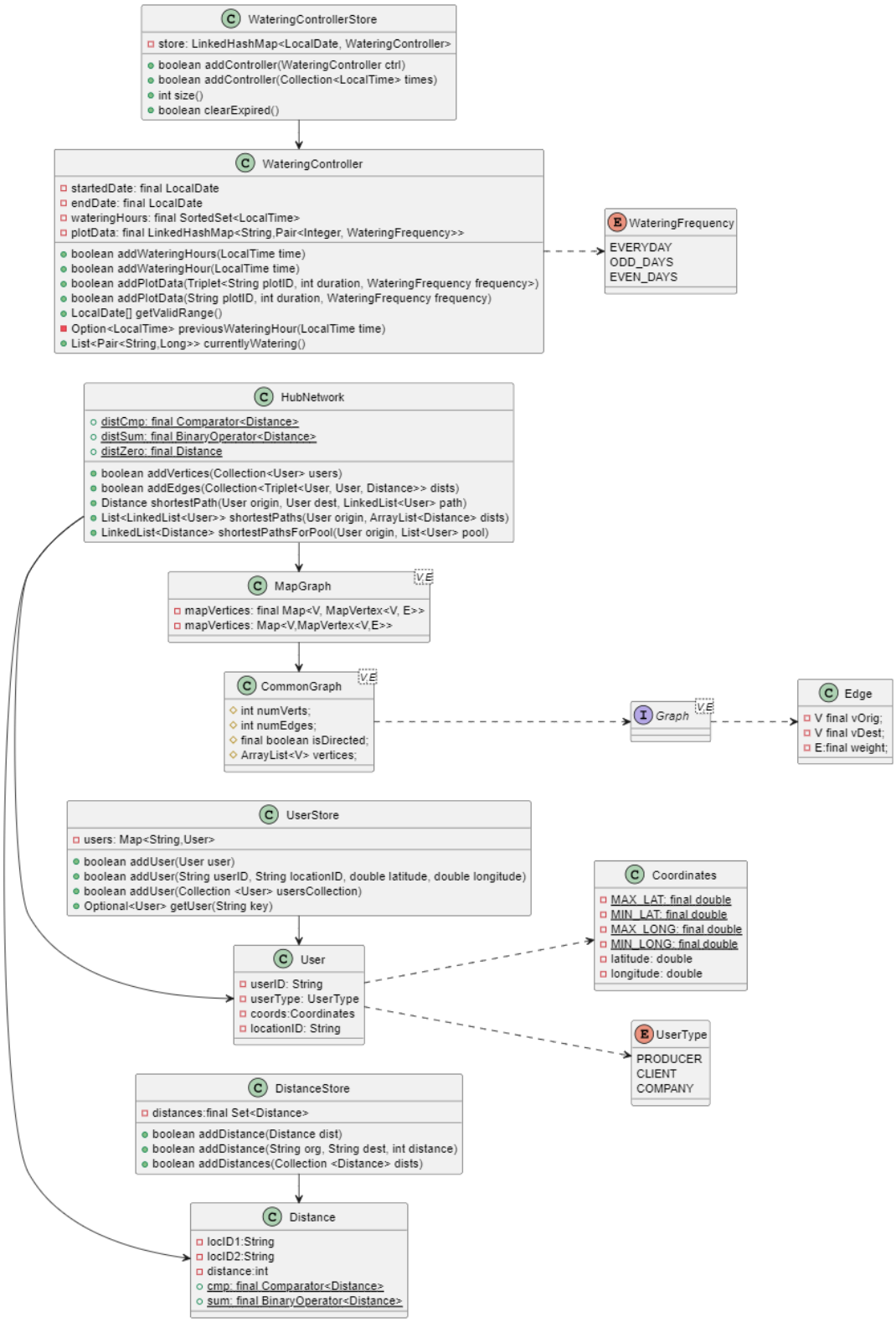

Diagrama de Classes 1

1 US301

```java
public void loadResources(boolean loadBig)
{
    for (var fileEnum : CSVFiles.values()) {                    // O(1) (enum values)
        String fpath = fileEnum.path(loadBig);
        List<String[]> data;

        try {
            data = CSVReader.readFromResources(fpath,           // O(l*c);
                    fileEnum == CSVFiles.BUNDLES && !loadBig    // l => lines of the file(s)
                    ? CSVHeader.BUNDLES_SMALL                   // c => columns of the file(s)
                    : fileEnum.header);
        } catch (IOException e) {
            e.printStackTrace();
            return;
        }

        if (fileEnum != CSVFiles.BUNDLES)
            this.parsers.get(fileEnum.header).parse(data);      // get: O(1); parse: O(l)

        /* NOTE: bundles are disabled because they're not yet needed */
        // if (fileEnum == CSVFiles.BUNDLES && !loadBig)
        //     ;// FIXME: parse differently
        // else
        //     parsers.get(fileEnum.header).parse(data);
    }

    // Net complexity: O(l*c)
}
```

*Método 1: loadResources() [O(l*c)] em CSVLoaderHandler*

```java
public void loadInteractive(Map<CSVHeader, File> files)
{
    files.forEach((header, file) -> {                           // O(1) (keys => enum values)
        List<String[]> data;
        try {
            data = CSVReader.readCSV(file, header);             // O(l*c)
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            return;
        }

        this.parsers.get(header).parse(data);                   // O(l)
    });

    // Net complexity: O(l*c)
}
```

*Método 2: loadInteractive() [O(l*c)] em CSVLoaderHandler*

```java
public boolean populateNetwork() {
    var users : UserStore = this.app.userStore();
    var distances : DistanceStore = this.app.distanceStore();

    var network : HubNetwork = this.app.hubNetwork();

    int edges = network.numEdges();                                  // O(1)
    int verts = network.numVertices();                               // O(1)


    users.forEach(network::addVertex);                               // O(V) ~ O(l)

    distances.forEach(distance -> {                                  // O(E)
        var orig : Optional<User> = users.getUser(distance.getLocID1());          // O(1)
        var dest : Optional<User> = users.getUser(distance.getLocID2());          // O(1)

        if (orig.isPresent() && dest.isPresent())
            network.addEdge(orig.get(), dest.get(), distance);  // O(1)
    });

    // Net complexity: O(E), since E ~ V^2
    return edges < network.numEdges() || verts < network.numVertices();
}
```

Método 3: populateNetwork() [O(E)] em CSVLoaderHandler

```java
@Override
public void parse(List<String[]> data) {
    // O(l); l => lines of the file
    data.forEach(line -> {
        String loc1, loc2;
        int length;

        loc1 = line[DistanceColumns.LOC_ID_1.col];
        loc2 = line[DistanceColumns.LOC_ID_2.col];

        try {
            length = Integer.parseUnsignedInt(line[DistanceColumns.LENGTH.col]);
        } catch (NumberFormatException e) {
            throw new InvalidCSVFileException("CSV File contained an invalid length!!");
        }

        // O(1)
        app.distanceStore().addDistance(loc1, loc2, length);
    });

    // Net Complexity: O(l)
}
```

Método 4: parse() [O(l)] em DistanceParser

```java
@Override
public void parse(List<String[]> data) {
    // O(l); l => lines of the file
    data.forEach(line -> {
        String locID, userID;
        double latitude, longitude;

        locID = line[UserColumns.LOC_ID.col];

        try {
            latitude = Double.parseDouble(line[UserColumns.LATITUDE.col]);
            longitude = Double.parseDouble(line[UserColumns.LONGITUDE.col]);
        } catch (NumberFormatException e) {
            throw new InvalidCSVFileException("CSV File contained invalid coordinates!!");
        }

        userID = line[UserColumns.USER_ID.col];

        // O(1)
        this.app.userStore().addUser(userID, locID, latitude, longitude);
    });

    // Net Complexity: O(l)
}
```

Método 5: parse() [O(l)] em UserParser

2 US302

```java
public Optional<Integer> minReachability() {
    if (connected) {
        var matrix : MatrixGraph<User, Distance> = Algorithms.minDistGraph(network, Distance.cmp, Distance.sum);  // O(V^3) (Floyd-Warshall)
        return Optional.of(matrix.numEdges());  // O(1)
    }

    // Net Complexity: O(V^3)
    return Optional.empty();
}
```

Método 6: minReachability() [O(V^3)] em IsConnectedHandler

```java
public static <V,E> boolean isConnected(Graph<V, E> g) {
    Objects.requireNonNull(g);

    g = getUndirectedGraph(g);  // O(V*E)

    // O(V*E)
    return BreadthFirstSearch(g, g.vertex( key: 0)).size() == g.numVertices();
}
```

*Método 7: isConnected() [O(V*E)] em Algorithms*

```java
public static <V,E> Graph<V,E> getUndirectedGraph(Graph<V,E> g) {
    Objects.requireNonNull(g);

    if (!g.isDirected())
        return g;

    Graph<V,E> newGraph = g.clone();     // O(V*E)

    for (var e : newGraph.edges())       // O(E)
        newGraph.addEdge(e.getVDest(), e.getVOrig(), e.getWeight());

    // Net complexity: O(V*E)
    return newGraph;
}
```

*Método 8: getUndirectedGraph() [O(V*E)] em Algorithms*

3 US303

```java
public ArrayList<Pair<User,Double>> findCompaniesAverageWeight (){

    ArrayList<Distance> dists = new ArrayList<>();

    for (User user : mapGraph.vertices()) {     // O(V)

        if(user.getUserType()==UserType.COMPANY){

            //paths & dists are being cleared at Algorithms.shortestPaths()
            mapGraph.shortestPaths(user, dists);    // O(V^2)

            //for each path
            Double average = getAverageAllPaths( dists); // O(E)

            companiesByOrder.add(new Pair<>(user,average)); // O(1)

        }
    }
    orderCompanies();   // O(V*logV)
    return getList();   // O(V)
    // Net complexity:  O(V^3)
}
```

Método 9: findCompaniesAverageWeight() [O(V^3)] em TopNCompaniesHandler

```java
private void orderCompanies(){
    if (companiesByOrder.isEmpty())
        throw new ArrayIndexOutOfBoundsException("List with companies and correspondent weights is empty");
    else
        companiesByOrder.sort(cmpDist);     // O(V*logV)
}
```

*Método 10: orderCompanies() [O(V*log(v))]em TopNCompaniesHandler*

```java
protected ArrayList<Pair<User,Double>> getList(){
    // O(V)
    return new ArrayList<>(companiesByOrder);
}
```

Método 11: getList() [O(V)] em TopNCompaniesHandler

```java
private Double getAverageAllPaths(ArrayList<Distance> dists)
{
    int soma = 0;

    for (Distance pathsDist: dists) {        // O(E)
        soma += pathsDist.getDistance();
    }

    // Net complexity: O(E)
    return (double) soma / dists.size();
}
```

Método 12: getAverageAllPaths() [O(E)] em TopNCompaniesHandler

```java
protected ArrayList<Pair<User,Double>> getTopNCompanies(int n){
    ArrayList<Pair<User,Double>> topN = new ArrayList<>();

    if(companiesByOrder.size() < n || n <= 0){
        return null;
    }

    int counter=0;

    for (Pair<User,Double> pair : companiesByOrder) { // O(V)

        topN.add(pair);
        counter++;
        if(counter==n)
            break;
    }

    // Net complexity: O(V)
    return topN;
}
```

Método 13: getTopNCompanies() [O(V)] em TopNCompaniesHandler

4 US304

```java
public LinkedList<Triplet<User, User, Distance>> findNearestHubs() {
    LinkedList<Triplet<User, User, Distance>> list = new LinkedList<>();

    var companies = new ArrayList<User>();
    var clients = new ArrayList<User>();

    // Get all existing companies && clients to speed up the process
    for (User u : network.vertices()) {     // O(V)
        switch (u.getUserType()) {
            case COMPANY:
                companies.add(u);
            case CLIENT: /* FALLTHROUGH */
                clients.add(u);
                break;
            default: // Do nothing
        }
    }

    for (User client : clients) {           // O(V)
        var companyDist : Optional<Distance> = nearestHub(client, companies);    // O(V^3)

        if (companyDist.isPresent()) {
            Distance d = companyDist.get();
            list.add(new Triplet<>(client, userStore.getUser(d.getLocID2()).get(), d));
        }
    }

    // O(V^4)
    return list;
}
```

Método 14: findNearestHubs() [O(V^4)] em NearestHubToClientsHandler

```java
private Optional<Distance> nearestHub(User client, List<User> companies) {
    // O(V^3)
    var dists : LinkedList<Distance> = this.network.shortestPathsForPool(client, companies);

    // Closest Hubs first
    dists.sort(Distance.cmp);

    for (Distance d : dists) {  // O(E)
        if (userStore.getUser(d.getLocID2()).isPresent())
            return Optional.of(d); // NOTE: User is guaranteed to be a company because of findNearestHubs()
    }

    // Nothing found :(
    return Optional.empty();    // Net complexity: O(V^3)
}
```

Método 15: nearestHub() [O(V^3)] em NearestHubToClientsHandler

```java
public static <V, E> LinkedList<E>
shortestPathsForPool(Graph<V,E> g, V vOrig, List<V> pool,
                     Comparator<E> ce, BinaryOperator<E> sum, E zero)
{
    var dists = new LinkedList<E>();

    // Required by shortestPath()
    var tmp = new LinkedList<V>();

    for (V dest : pool) {   // O(V)
        // O(V^2)
        dists.push(shortestPath(g, vOrig, dest, ce, sum, zero, tmp));
    }

    // Net complexity: O(V^3)
    return dists;
}
```

Método 16: shortestPathsForPool() [O(V^3)] em Algorithms

5 US305

```java
public Graph<User, Distance> getMinimalUserNetwork(){
    return Algorithms.kruskalMST(hubNetwork, HubNetwork.distCmp);   //O(E*log E)
}
```

*Método 17: getMinimalUserNetwork() [O(E*log(E))] em MinimumDistanceHandler*

```java
public int getMinimumCost(Graph<User, Distance> mst){
    int minimumCost=0;
    for (Edge<User,Distance> edge: mst.edges()) {                   //O(E)
        minimumCost += edge.getWeight().getDistance();
    }
    return minimumCost;
}
```

Método 18: getMinimumCost() [O(E)] em MinimumDistanceHandler

```java
public static <V,E> Graph<V,E> kruskalMST(Graph<V,E> g, Comparator<E> ce) {   //O(E*log E)
    PriorityQueue<Edge<V,E>> lstEdges = new PriorityQueue<>(Comparator.comparing(Edge::getWeight, ce));

    Graph<V, E> mst = new MapGraph<>(g.isDirected());

    for (V vertex : g.vertices()) {             // O(V)
        mst.addVertex(vertex);
    }

    lstEdges.addAll(g.edges());                 // O(E)

    // Kruskal stops when V-1 edges are 'selected'
    int n = g.numVertices() - 1;

    while (n > 0 && !lstEdges.isEmpty()) {      // O(V)
        Edge<V, E> e1 = lstEdges.poll();
        if (!hasConnection(mst, e1.getVOrig(), e1.getVDest())) { // O(V*E)
            mst.addEdge(e1.getVOrig(), e1.getVDest(), e1.getWeight());
            n--;
        }
    }

    // O(V^2*E)
    return mst;
}
```

*Método 19: kruskalMST() [O(V^2*E)] em Algorithms*