Licenciatura em Engenharia Informática

Relatório Trabalho Prático 1

Diogo Guedes (1170617)
João Teixeira (1180590)
Rui Gonçalves (1191831)

Novembro 2021

Para este primeiro trabalho, começamos por usar as classes “BST”, “BSTInterface” as quais foram implementadas nas aulas PL e TP.

Classe CSVReaderUtil

Nesta classe desenvolvemos métodos para a leitura dos ficheiros CSV e carregamento dos dados recebidos nos respetivos construtores.

```
public static ArrayList<Ship> readCSV(String path) throws Exception {

    DateTimeFormatter formatDate = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");

    ArrayList<Ship> shipArray = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(new FileReader(path))) {

        String line = br.readLine();

        while ((line = br.readLine()) != null) {

            String[] values = line.split( regex: ",");

            ShipData sd = new ShipData(LocalDateTime.parse(values[1], formatDate),
                    Double.parseDouble(values[2]),
                    Double.parseDouble(values[3]),
                    Double.parseDouble(values[4]),
                    Double.parseDouble(values[5]),
                    Double.parseDouble(values[6]),
                    values[15].charAt(0));

            int index = verifyShip(values[0], shipArray);
            if (index == -1) { // if there's ship
                int imo = newImo(values[8]);
                int cargo = newCargo(values[14]);

                Ship ship = new Ship(
                        Integer.parseInt(values[0]), // mmsi
                         dynamicShip: null, // dynamic ship data
                        values[7], // name
                        imo, // imo
                        values[9], // callsign
                        Integer.parseInt(values[10]), // vessel
                        Double.parseDouble(values[11]), // length
                        Double.parseDouble(values[12]), // width
                        Double.parseDouble(values[13]),// draft
                        cargo);// cargo
                ship.initializeDynamicData();
                ship.addDynamicShip(sd);
                shipArray.add(ship);

            } else {
                shipArray.get(index).addDynamicShip(sd);

            }
        }
```

Complexidade: O(n)

Para verificar se um já existe, utilizamos o método verifyShip. Caso exista devolvemos o indíce desse barco, caso contrário devolvemos –1.

```java
public static int verifyShip(String value, ArrayList<Ship> shipArray) {

    for (int i = 0; i < shipArray.size(); i++) {
        Ship ship = shipArray.get(i);
        if (ship.getMmsi() == Integer.parseInt(value)) {
            return i;
        }
    }

    return -1;
}
```

Complecidade: O(N)

Para ordenar os registos dos barcos criamos este método que ordena do registo mais antigo para o mais recente.

```java
public static ArrayList<Ship> sortByDate(ArrayList<Ship> shipArray) {

    for (int i = 0; i < shipArray.size(); i++) {
        ArrayList<ShipData> sortedArray = (ArrayList<ShipData>) shipArray.get(i).getDynamicShip().stream()
                .sorted(Comparator.comparing(ShipData::getDateTime).reversed())
                .collect(Collectors.toList());

        shipArray.get(i).setDynamicShip(sortedArray);
    }

    return shipArray;
}
```

Complexidade: O(e)

Exercício 2

No segundo exercício, é pedido para fazer um sumário que contém determinados parâmetros acerca de um navio.

Optamos por usar um Array List (sumary) onde iriam ser guardados todos os parâmetros desejados.

Quanto ao nível de complexidade deste exercício, nós usamos uma class "Sumary" que contém o método "createSumary()" que tem a complexidade **O(n)**, visto que, apenas têm um ciclo "for" que percorre o Array Dinâmico do navio. *(Figura 1 e 2)*

```
public static ArrayList createSumary(Ship ship, String code) {
    ArrayList<Object> sumary = new ArrayList<>();
    LocalDateTime inicialTime = null;
    LocalDateTime finalTime = null;
    int nMoves = 0;
    int totalTime;
    double maxSog = 0.0;
    double meanSog = 0.0;
    double maxCog = 0.0;
    double meanCog = 0.0;
    double departLat = 0.0;
    double departLong = 0.0;
    double arrLat = 0.0;
    double arrLong = 0.0;

    for (ShipData sd : ship.getDynamicShip()) {
        if (nMoves == 0) {
            inicialTime = sd.getDateTime();
            departLat = sd.getLatitude();
            departLong = sd.getLongitude();
        } else if (nMoves + 1 == ship.getDynamicShip().size()) {
            finalTime = sd.getDateTime();
            arrLat = sd.getLatitude();
            arrLong = sd.getLongitude();
        }
        if (maxSog < sd.getSog()) {
            maxSog = sd.getSog();
        }
        if (maxCog < sd.getCog()) {
            maxCog = sd.getCog();
        }
        meanSog += sd.getSog();
        meanCog += sd.getCog();
        nMoves++;
    }
```

Figura 1

```
Double travelDistance = Calculator.totalDistance(ship.getDynamicShip());
Double deltaDistance = Calculator.distanceBetween(departLat, departLong, arrLat, arrLong);


// Code
if (code.equalsIgnoreCase( anotherString: "MMSI")) {
    sumary.add(ship.getMmsi());
} else if (code.equalsIgnoreCase( anotherString: "IMO")) {
    sumary.add(ship.getImo());
} else if (code.equalsIgnoreCase( anotherString: "CallSign")) {
    sumary.add(ship.getCallSign());
}
sumary.add(ship.getName()); // Name
sumary.add(ship.getVessel()); // VasselType
sumary.add(inicialTime); // BDT Inicial
sumary.add(finalTime); // BDT Final
sumary.add(totalTime); // Tempo total dos movimentos
sumary.add(nMoves); // Numero total de movimentos
sumary.add(maxSog); // MaxSog
sumary.add(meanSog); // MeanSog
sumary.add(maxCog); // MaxCog
sumary.add(meanCog); // MeanCog
sumary.add(departLat); // DepartureLatitude
sumary.add(departLong); // DepartureLongitude
sumary.add(arrLat); // ArrivalLatitude
sumary.add(arrLong); // ArrivalLongitude
sumary.add(travelDistance); // TraveledDistance
sumary.add(deltaDistance); // DeltaDistance

return sumary;
```

Figura 2

De seguida, ainda neste método, chama-mos uma função "convertToDateViaInstant()" que, como não tem nenhum ciclo, a complexidade é **O(1).** *(Figura 3)*

```
public static Date convertToDateViaInstant(LocalDateTime dateToConvert) {
    return java.util.Date
            .from(dateToConvert.atZone(ZoneId.systemDefault())
                    .toInstant());
}
```

Figura 1

Finalmente, ainda dentro do método principal, temos os métodos "distanceBetween()" que tem a complexidade **O(1)** e o método "totalDistance()" que tem complexidade **O(n)** por ter um ciclo "for". *(Figura 4)*

```java
public static double distanceBetween(double lat1, double lon1, double lat2, double lon2){
    //if latitude/longitude aren't available
     if((lat1 < -90)  || (lat1 > 90)  || (lat2 < -90)  || (lat2 > 90) ||
       (lon1 < -180) || (lon1 > 180) || (lon2 < -180) || (lon2 > 180)  ){
        return 0;
    }

    double lat1Rad = degToRad(lat1);
    double lat2Rad = degToRad(lat2);
    double Δlat = degToRad(lat2-lat1);
    double Δlon = degToRad(lon2-lon1);
    double a = Math.sin(Δlat/2) * Math.sin(Δlat/2) + Math.cos(lat1Rad) * Math.cos(lat2Rad) * Math.sin(Δlon/2) * Math.sin(Δlon/2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

    return (R * c); // in metres
}


//handle shipData before
//also works for deltaDistance
public static double totalDistance(ArrayList<ShipData> shipData){
    if(shipData.isEmpty()) { return 0;}

    double totalDistance = -1;
    double lat1 = 91;
    double lat2 = 91;
    double lon1 = 181;
    double lon2 = 181;
    ShipData pos1 = null;
    ShipData pos2 = null;

    for(int i = 0; i < shipData.size() - 1; i++) {
        pos1= shipData.get(i);
        pos2= shipData.get(i+1);
        lat1= pos1.getLatitude();
        lat2= pos2.getLatitude();
        lon1= pos1.getLongitude();
        lon2= pos2.getLongitude();

        totalDistance+= distanceBetween(lat1, lon1, lat2, lon2);
    }
    return totalDistance;
}
```

Terminal Profiler Build

Figura 2

Assim sendo, para este exercício, podemos concluir que o grau de complexidade é **O(n)**.

O(n)+O(1)+O(n)+O(1) = 2 x O(1) + 2 x O(n) = O(n)

Exercício 4

Neste use case (US6) o utilizador escolhe o número de navios que pretende listar e o período de datas a filtrar partir da consola no main.

Se não for necessário apresentar uma data inicial e/ou final será considerado todos os valores sem limites temporais de tempo e as duas datas são recebidas como null.

O método getNTopShips() em TopShipsController recebe estes valores e cria duas ArrayLists ordenadas pela distância percorrida: uma com os navios e outra com os respetivos MeanSOG's.

Para filtrar as shipDatas por período do tempo é usado o método filterShipData() da classe Ship ; no caso de não haver limites o método simplesmente retorna shipData sem a alterar.

Com os Ships organizados o método menuTopShips() em Menu organiza os respetivos dados conforme é pedido numa tabela:

```
+------------------------------+
  CARGO APP 103 > Main Menu
+------------------------------+
  0 - Exit
  1 - Import Menu
  2 - Manage Ships
  3 - Get the top-N ships
+------------------------------+
 > Please make a selection: 3
How many ships?
Choose a value above 0.
8
Input initial date or press '0' for none.
Which starting year?
0
Input final date or press '0' for none.
Which final year?
0
Number of Ships: 8
Inital date: none
Final date: none
Review your inputs, and press '1' to continue or '0' to restart:
```

Figura 3 exemplo do US6 com 8 navios e sem período de tempo

```
-----------------The top-N ships with the most kilometres travelled-----------------
---Vessel Type : 70
       OREGON TRADER 4,80
   HYUNDAI SINGAPORE 8,30
        RHL AGILITAS 24,20
            ARABELLA 0,22
---Vessel Type : 90
        STENA ICEMAX 24,40
---Vessel Type : 60
           TUSTUMENA 1,22
---Vessel Type : 79
           MSC ILONA 0,24
---Vessel Type : 30
          ARCTIC SEA 0,01
```

Figura 4 a tabela com os navios agrupados, mostrando o nome do navio e o respetivo MeanSOG