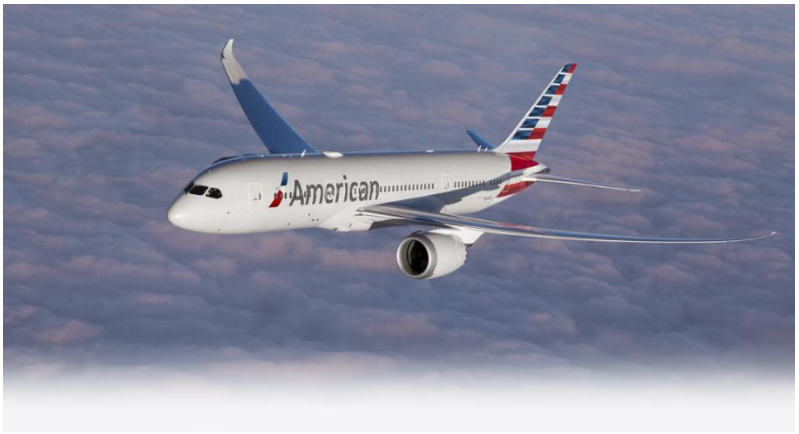
Projeto de recurso

ATAD 2021/22



André Meseiro - 202100225
Pedro Lopes – 202100319
Turma – 7
Docente – Miguel Bugalho
INSTITUTO POLITÉCNICO DE SETÚBAL

Índice

Introdução

Este projeto tem como objetivo a implementação de um programa que extrai/apresenta informação útil de ficheiros com dados sobre voos domésticos no território americano, nos primeiros dias de janeiro de 2015.

O programa deve ser desenvolvido de forma que o utilizador consiga interpretar de forma simples para obter diversos tipos de informação, principalmente informação estatística.

ADTs Utilizados

Os tipos abstratos de dados utilizados para a realização deste projeto são o ADT List e o ADT Map.

ADT List

O ADT List é uma coleção de elementos que possuem uma relação linear entre si. Uma relação linear significa que cada elemento da lista tem um único sucessor, além disso é também uma interface, ou seja, as outras classes fornecem a implementação real do tipo de dados.

A implementação escolhida para este ADT foi Array List porque tem as operações **get**, **size** e **destroy** com complexidade constante O(1) e a operação **add** com complexidade O(n) diferente da linked list que a complexidade de todas as operações é O(n), ou seja, a implementação Array List é mais eficiente.

ADT Map

O ADT Map é uma coleção constituída por conjuntos de pares chaves (exclusivas) e valores, onde cada chave está associada a um único valor.

A implementação escolhida para este ADT foi Array List pois apesar de ambas as implementações terem a mesma complexidade acreditamos que para este projeto seja a escolha mais indicada.

Algoritmos e complexidades

As 5 funcionalidades do tipo B e C são: SHOWF, SHOWALL, LISTAR, TOPDISTN, LISTAP.

SHOWF

Pseudo-código:

```
Algorithm SHOWF
        input: l – list, flights, *airlineCode
BEGIN
IF flights != NULL &&  originCode != NULL && destinationCode != NULL THEN
RETURN showFlightsWithRoute (flights, originCode, destinationCode)
END IF
END
```

A complexidade da funcionalidade SHOWF é **linear O(n)** porque a função **showFlightsWithRoute** têm complexidade **O(n)**, pois têm um ciclo for que depende do size dos flights introduzidos por parâmetro.

SHOWALL

Pseudo-código:

```
Algorithm SHOWALL
Input: l – list, flights
READ selectedOption = 0
BEGIN
DO
selectedOption = showAllMenu()
IF selectedOption == 1 THEN
READ size
RETURN listSize
READ numPages = size/ 30 + 1
READ numFinalPage size –  (numPages *30)
ListElem flight
READ currPage = 1
READ quit = false
WHILE quit == false THEN
READ max = currPage * 30
READ min = max -30
PRINT “Day Day of week Airline Flight Number Origin Destination scheduled Departure
Departure Time Real Travel Time Distance Scheduled Arrival Arrival Time Total Delay\n”
FOR i = min i < max i++ DO
listGet(flights, i, &flight)
flightPrint(&flight)
```

```
PRINT “\n”
END FOR
PRINT “\n”
PRINT “Menu Pages\n”
PRINT “1- Next Page\n”
PRINT “2- Return\n”
READ menuPgesOption;
RETURN readInteger (&menuPagesOption)
IF menuPagesOption = 1 THEN
RETURN currPage++
ELSE IF menuPagesOption = 2 THEN
 quit = true
END IF
END IF
END IF
END WHILE
ELSE IF selectedOption == 2 THEN
PRINT “----80 Random Flights Information---”
ListElem flight
READ size
listSize(flights, &size)
srand(time(0))
READ min = 0
READ max = 0
READ count = 0
READ random = (rand() % (min – max + 1)) + min
FOR i = random i < random + 79 i++
listGet(flights i &flight)
flightPrint(&flight)
printf”\n”
RETURN count++
END FOR
END IF
WHILE selectedOption != 3
END WHILE
END
```

A complexidade da funcionalidade SHOWALL é **quadrática O(n^2)** porque as funções **listGet** e **listSize** têm complexidade **O(n)**, pois ambas possuem uma condição que depende do size da list introduzidos por parâmetro.

LISTAR

Pseudo-código:

```
Algorithm LISTAR
Input: l – list, flights, airlines
BEGIN
PRINT ”---Airlines With Flights---”
FOR i = 0 i < ARRAY_CAPACITY i++ DO
IF (hashFlightsUnique(flights, airlines[i].iatacode) == true)
airlinePrint”&airlines[i]
END FOR
END IF
END
```

A complexidade da funcionalide LISTAR é **constante O(1)** porque a função **hasFlightUnique** têm complexidade **O(1)**, pois o ciclo for não depende de nenhum parâmetro.

TOPDISTN

Pseudo-código:

```
Algorithm TOPDISTN
Input: l – list, flights, n
BEGIN
READ size
Flight aux, f1, f2
listSize(flights, &size)
FOR i = 0 i < size i++ DO
listGet(flights, i, &aux)
listADD(sorted, i, aux)
END FOR
FOR i = 0 i < size i++ DO
FOR j = 0 j < size-1-I j++ DO
listGet(sorted, j, &f1)
listGet(sorted, j+1, f1, &f2)
IF f1.distance > f2.distance THEN
listSet(sorted, j, f2, &f1)
listSet(sorted, j+1, f1, &f2)
END IF
ELSE IF f1.distance = f2.distance THEN
IF f1.flightNumber < f2.flightNumber THEN
listSet(sorted, j, f2, &f1)
listSet(sorted, j+1, f1, &f2)
END IF
```

```
END IF
END FOR
END FOR
PRINT “----TOPDISTN----”
PRINT” Day Day of week Airline Flight Number Origin Destination scheduled Departure
Departure Time Real Travel Time Distance Scheduled Arrival Arrival Time Total Delay\n”
FOR i = 0 i < n i++
listGet(sorted, I, &aux)
flightPrint(&aux)
PRINT “\n”
END FOR
END
```

A complexidade da funcionalidade TOPDISTN é **quadrática O(n^2)** porque a função listGet têm complexidade O(n), pois têm um ciclo for que depende do size da list introduzidos por parâmetro e é usada duas vezes neste ciclo for.

LISTAP

Pseudo-código:

```
Algorithm LISTAP
Input: m – map, airports, flights
READ size1 , size 2
READ count
mapSize(airports, &size1)
listSize(flights, &size2)
Airport airport
FOR i = 0 i < size1 i++ DO
FOR j = 0 j < size2 j++ DO
listGet(flights, j, &flight)
StringCode originCode = stringCodeCreate(flight.originAirport)
StringCode destinationCode = stringCodeCreate(flight.destinationAirport)
StringCode airportCode = airportKeys[i]
IF equalsStringIgnoreCase(airportCode.code, originCode.code) AND
equalsStringIgnoreCase(airportCode.code, destinationCode.code) THEN
IF !mapContains(airportsWithFlights, airportCode) THEN
mapGet(airports, airportCode, &airport)
mapPut(airportsWithFlights, airportCode, airport)
END IF
END IF
END FOR
END FOR
mapSize(airportsWithFlights, &count)
PRINT “---- Airports List ----\n"
MapKey *awfKeys = mapKeys(airportsWithFlights)
MapValue *awfValues = mapValues(airportsWithFlights)
FOR i = 0 i < count i++ DO
```

```
PRINT “%-3: %-50s %-20s %-4s %-2d\n”, awfKeys[i].code, awfValues[i].airport,
awfValues[i].city, awfValues[i].city, awfValues[i].state, awfValues[i].timeZone
END FOR
PRINT “------------------\n”
PRINT “%d airports were found on the records\n”, count
mapValues(airportsWithFlights)
mapDestroy(&airportsWithFlights)
END
```

A complexidade da funcionalidade LISTAP é **linear O(n)** porque a função **listGet** tem complexidade **O(n)**, pois têm um ciclo for que depende do size da list introduzidos por parâmetro.

Limitações

Os comandos que não foram implementados são:

- **Indicadores simples:**
  - AVERAGEDELAY;
  - SHOWAP.
- **Indicadores complexos:**
  - AIRPORT_S;
  - AIRLINES.

Fora os comandos apresentados acima, todos os outros funcionam sem qualquer tipo de problemas.

Conclusões

Neste projeto pudemos demonstrar todos os conhecimentos adquiridos ao longo deste semestre na cadeira ATAD. Além disso, pudemos aprender como desenvolver um programa para um utilizador comum que irá ser muito útil no futuro como Engenheiros Informáticos. Em relação ao tema do projeto achamos que foi bem escolhido porque na atualidade existem muitas empresas que necessitam de programas que apresentam informação de ficheiros e com este projeto pudemos aprender a desenvolver um.