

|  |
| --- |
| Algoritmos e Tipos Abstratos de Dados  2020/2021  Licenciatura em Engª. Informática |
| Netflix Titles |

|  |  |  |
| --- | --- | --- |
| Turma: 1 | Data/Horário Lab.: 2ª Feira 13h30 | Docente: Aníbal Paulo Lopes da Ponte |

|  |  |  |
| --- | --- | --- |
| **Nº aluno** | **Nome** | **Endereço de Correio Eletrónico** |
| 200221024 | Gonçalo Mendes | 200221024@estudantes.ips.pt |
| 20021060 | Henrique Leote | 20021060@estudantes.ips.pt |

Índice

[1. Introdução 3](#_Toc77524855)

[2. ADTs Utilizados 3](#_Toc77524856)

[3. Complexidade Algorítmicas 4](#_Toc77524857)

[4. Algoritmos 5](#_Toc77524858)

[5. Limitações 10](#_Toc77524859)

[6. Conclusões 10](#_Toc77524860)

# Introdução

O seguinte projeto foi realizado no âmbito da unidade curricular de Algoritmos e Tipos Abstratos de Dados (ATAD). No referido, foram desenvolvidas diferentes funções com o intuito de manipular um ficheiro com diferentes títulos da Netflix. Na implementação dos algoritmos foram utilizados diferentes “abstract data types” (ADTs), que são o foco geral da UC.

# ADTs Utilizados

Para a realização do projeto foram utilizados os ADTs List e Map, com recurso a ArrayList como estrutura de dados. Fora o ADT List que era de critério obrigatório para a realização deste projeto, foi utilizado o ADT Map, de forma a filtrar valores duplicados para apresentação ao utilizador.

Em termos de complexidade, houve uma maior facilidade na utilização destes ADTs com a referida estrutura de dados ArrayList, porém, a eficiência destes não é elevada, resultando em varias funções com alta complexidade.

# Complexidade Algorítmicas

Uma imagem com mesa

Descrição gerada automaticamenteUma imagem com mesa

Descrição gerada automaticamente  
Em termos de complexidade algorítmica, ambas são definidas pela estrutura de dados ArrayList:

Figura 1- Complexidades do ADT List implementado com ArrayList

Figura 2- Complexidades do ADT Map implementado com ArrayList

As complexidades dos ADT influenciam diretamente as complexidades das funções posteriormente desenvolvidas. Das funções realizadas no âmbito do projeto, estas são as suas complexidades:

|  |  |  |
| --- | --- | --- |
| Função | Complexidade | Justificação |
| LOADF |  | Iterações a depender do tamanho do ficheiro e que executam a função ListAdd |
| LOADD |  | Iterações a depender do tamanho do ficheiro e que executam a função ListAdd |
| DEL |  | Iterações a depender do tamanho da lista e que executam a função convertToLower |
| GET |  | Iterações a depender do tamanho da lista e que executam a função convertToLower |
| LIST |  | Iterações a depender do tamanho da lista |
| STATS |  | Iterações a depender do tamanho da lista |
| MTIME |  | Depende da iteração da função sortDuration |
| SEARCHT |  | Iterações a depender do tamanho da lista e que executam a função convertToLower |
| SEARCHC |  | Iterações a depender do tamanho da lista e que executam a função convertToLower |
| RATINGS |  | Iterações a depender do tamanho da lista e que executam a função ListAdd |
| CATEGORIES |  | Iterações a depender do tamanho da lista que realiza outro set de iterações a depender do tamanho do elemento "token", que por sua vez executa a função trimWhiteSpaces |
| SEGMENT |  | Depende das iterações da função Directors |

Tabela 1 - Complexidades Algorítmicas das funções

# Algoritmos

Algorithm MTIME

input: mainList – List with the Netflix titles

input TYPE\_MOVIE - integer

BEGIN

PRINT “Min: “

READ min

PRINT “Max: “

READ max

IF min < max THEN

sortDuration mainList

LIST(mainList, min, max, TYPE\_MOVIE)

ELSE

PRINT “Min can't be bigger than max. “

END IF

END

Figura 3 – Pseudo-código da função MTIME

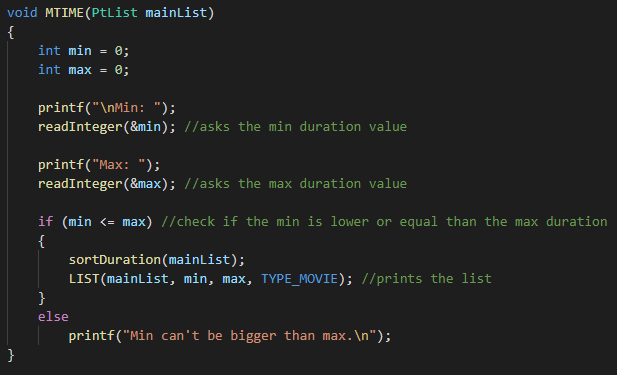


Figura 4 – Código em C da função MTIME

Figura 5 – Pseudo-código da função SEARCHC

Algorithm SEARCHC

Input: mainList – List with the Netflix titles

input TYPE - integer

Variables

castName[640], netflixcastName – char array

netflix – Netflix struct

cCount, mainSizeList, netflixCastNameCount – integer

PtList - titlesList

BEGIN

titlesList <- listCreate

sizeList <- listSize list

cCount <- 0

PRINT “Please insert the cast member(s) name: “

READ castName

FOR i <- 0 TO mainSizeList DO

netflix <- listGet list

strcpy(netflixCastName, netflix.cast)

convertToLower netflixCastName

IF strstr(netflixCastName, castName) != NULL THEN

listAdd titlesList, count, netflix

cCount +1

END IF

END FOR

sortDate titleList

PRINT “Query is $castname”

free castName

free netflixCastName

LIST(titlesList, -1, -1, TYPE)

END

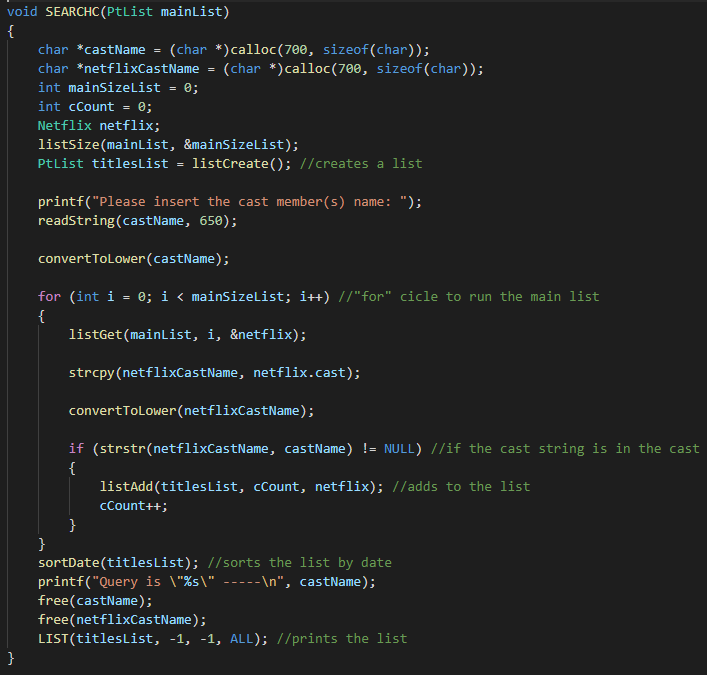


Figura 6 – Código em C da função SEARCHC

Figura 7 – Pseudo-código da função STATS

Algorithm STATS

Input: mainList – List with the Netflix titles

Variables

statsListSize, min, max, minS, maxS – integer

movieCount, totalMinutes, totalSeasons, tvCount, averageDuration, averageSeasons – double

netflix – Netflix struct

titlesList - PtList

BEGIN

titlesList <- listCreate

sizeList <- listSize list

FOR i <- 0 TO statsListSize DO

netflix <- listGet list

IF strcmp(netflix.type, “Movie”) == 0 THEN

movieCount++

totalMinutes = totalMinutes + netflix.duration

IF (netflix.duration < min) THEN

min = netflix.duration

END IF

IF (netflix.duration > max) THEN

max = netflix.duration

END IF

END IF

IF strcmp(netflix.type, “TV SHow”) == 0 THEN

tvCount++

totalSeasons = totalSeasons + netflix.duration

IF (netflix.duration < minS) THEN

minS = netflix.duration

END IF

IF (netflix.duration > maxS) THEN

maxS = netflix.duration

END IF

END IF

END FOR

averageDuration = totalMinutes / movieCount

averageSeasons = totalSeasons / tvCount

PRINT(“Movie Count: $movieCount”, “Min. duration: $min”)

PRINT(“Max. duration: $max”, “Avg. Duration $averageDuration”)

PRINT(“TV Show Count $tvCount”, “Min. Seasons: $minS”)

PRINT(“Max. Seasons: $maxS”m “Avg. Seasons: $averageSeasons”)

PRINT(“Total minutes of movie time: $totalMinutes”)

PRINT(“Total seasons of tv shows $totalSeasons”)

END

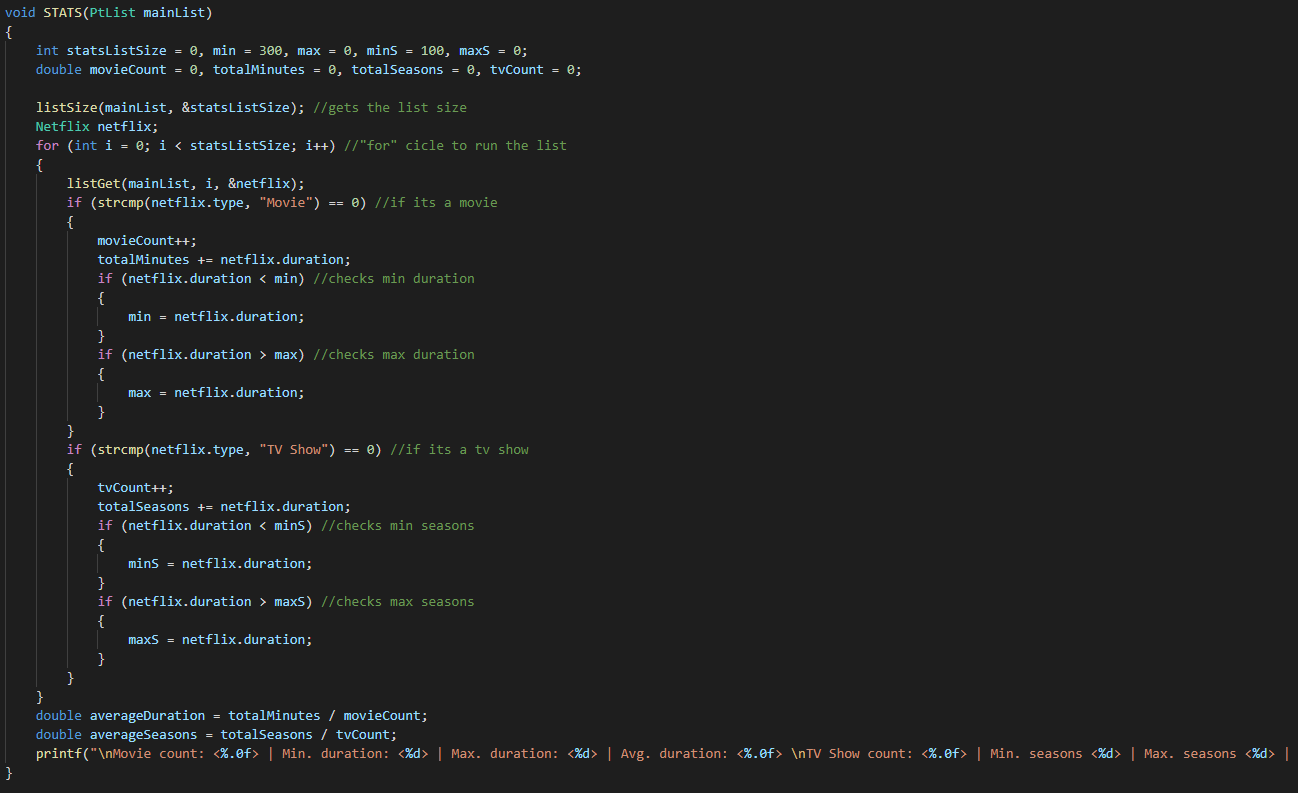


Figura 8 – Código em C da função STATS

# Limitações

Em termos de limitações, todos os comandos funcionam a 100% à exceção do SEARCHC. Apesar de estar bem implementado, sendo semelhante ao SEARCHT, mas mudando o campo, dá um erro de “segmentation fault” que mesmo com ajuda e bastante cuidado nos testes com debug, não foi possível resolver.

Ao testarmos o problema do SEARCHC, quando executávamos a função sem menu, seguida do LOADD para carregar dados, funciona, mas se executarmos uma segunda vez, já dá “segmentation fault”. Se carregarmos a função através do menu, dá diretamente “segmentation fault” na primeira vez.

Fora o SEARCHC, todos os comandos funcionam sem qualquer problema ou erro.

# Conclusões

Em geral este projeto foi bastante interessante de se desenvolver, divertido e trabalhoso no bom sentido. Em comparação aos projetos das outras unidades curriculares do curso, este foi o mais acessível em termos do enunciado estar bem estruturado, simplificado e claro. Prova disso é que sendo a cadeira com a teoria e prática mais “complicada”, foi possível, do nosso ponto de visto, desenvolver um bom projeto.

Em análise, o projeto foi feito a 100%, sem deixar qualquer função de fora. À exceção da função SEARCHC que não sabemos o porquê de não funcionar visto que é semelhante à SEARCHT, tirámos ótimo proveito e consolidamos com certeza os conceitos da linguagem C e das implementações de ADTs, e estamos bastante orgulhosos do nosso desempenho na realização deste projeto.