
# 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

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.

# 2. 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.

# 3. Complexidade Algorítmicas

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

| Operação | Complexidade | Motivo |
|---|---|---|
| `add` | $O(n)$ | "deslocar" elementos |
| `remove` | $O(n)$ | "deslocar" elementos |
| `get` | $O(1)$ | indexação `[]` |
| `set` | $O(1)$ | indexação `[]` |

*Figura 1- Complexidades do ADT List implementado com ArrayList*

| Operação | Complexidade | Motivo |
|---|---|---|
| `put` | $O(n)$ | pesquisa seq. de chave |
| `get` | $O(n)$ | pesquisa seq. de chave |
| `remove` | $O(n)$ | pesquisa seq. de chave |
| `contains` | $O(n)$ | pesquisa seq. de chave |

*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 | $O(n^2)$ | Iterações a depender do tamanho do ficheiro e que executam a função ListAdd |
| LOADD | $O(n^2)$ | Iterações a depender do tamanho do ficheiro e que executam a função ListAdd |
| DEL | $O(n^2)$ | Iterações a depender do tamanho da lista e que executam a função convertToLower |
| GET | $O(n^2)$ | Iterações a depender do tamanho da lista e que executam a função convertToLower |
| LIST | $O(n)$ | Iterações a depender do tamanho da lista |
| STATS | $O(n)$ | Iterações a depender do tamanho da lista |
| MTIME | $O(n^2)$ | Depende da iteração da função sortDuration |
| SEARCHT | $O(n^2)$ | Iterações a depender do tamanho da lista e que executam a função convertToLower |
| SEARCHC | $O(n^2)$ | Iterações a depender do tamanho da lista e que executam a função convertToLower |
| RATINGS | $O(n^2)$ | Iterações a depender do tamanho da lista e que executam a função ListAdd |
| CATEGORIES | $O(n^3)$ | 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 | $O(n^3)$ | Depende das iterações da função Directors |

*Tabela 1 - Complexidades Algorítmicas das funções*

# 4. 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*

```c
void MTIME(PtList mainList)
{
    int min = 0;
    int max = 0;

    printf("\nMin: ");
    readInteger(&min); //asks the min duration value

    printf("Max: ");
    readInteger(&max); //asks the max duration value

    if (min <= max) //check if the min is lower or equal than the max duration
    {
        sortDuration(mainList);
        LIST(mainList, min, max, TYPE_MOVIE); //prints the list
    }
    else
        printf("Min can't be bigger than max.\n");
}
```

*Figura 4 – Código em C da função MTIME*

```
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 5 – Pseudo-código da função SEARCHC*

```c
void SEARCHC(PtList mainList)
{
    char *castName = (char *)calloc(700, sizeof(char));
    char *netflixCastName = (char *)calloc(700, sizeof(char));
    int mainSizeList = 0;
    int cCount = 0;
    Netflix netflix;
    listSize(mainList, &mainSizeList);
    PtList titlesList = listCreate(); //creates a list

    printf("Please insert the cast member(s) name: ");
    readString(castName, 650);

    convertToLower(castName);

    for (int i = 0; i < mainSizeList; i++) //"for" cicle to run the main list
    {
        listGet(mainList, i, &netflix);

        strcpy(netflixCastName, netflix.cast);

        convertToLower(netflixCastName);

        if (strstr(netflixCastName, castName) != NULL) //if the cast string is in the cast
        {
            listAdd(titlesList, cCount, netflix); //adds to the list
            cCount++;
        }
    }
    sortDate(titlesList); //sorts the list by date
    printf("Query is \"%s\" -----\n", castName);
    free(castName);
    free(netflixCastName);
    LIST(titlesList, -1, -1, ALL); //prints the list
}
```

*Figura 6 – Código em C da função SEARCHC*

```
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 7 – Pseudo-código da função STATS*

```c
void STATS(PtList mainList)
{
    int statsListSize = 0, min = 300, max = 0, minS = 100, maxS = 0;
    double movieCount = 0, totalMinutes = 0, totalSeasons = 0, tvCount = 0;

    listSize(mainList, &statsListSize); //gets the list size
    Netflix netflix;
    for (int i = 0; i < statsListSize; i++) //"for" cicle to run the list
    {
        listGet(mainList, i, &netflix);
        if (strcmp(netflix.type, "Movie") == 0) //if its a movie
        {
            movieCount++;
            totalMinutes += netflix.duration;
            if (netflix.duration < min) //checks min duration
            {
                min = netflix.duration;
            }
            if (netflix.duration > max) //checks max duration
            {
                max = netflix.duration;
            }
        }
        if (strcmp(netflix.type, "TV Show") == 0) //if its a tv show
        {
            tvCount++;
            totalSeasons += netflix.duration;
            if (netflix.duration < minS) //checks min seasons
            {
                minS = netflix.duration;
            }
            if (netflix.duration > maxS) //checks max seasons
            {
                maxS = netflix.duration;
            }
        }
    }
    double averageDuration = totalMinutes / movieCount;
    double averageSeasons = totalSeasons / tvCount;
    printf("\nMovie count: <%.0f> | Min. duration: <%d> | Max. duration: <%d> | Avg. duration: <%.0f> \nTV Show count: <%.0f> | Min. seasons <%d> | Max. seasons <%d> |
}
```

*Figura 8 – Código em C da função STATS*

## 5. 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.

## 6. 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.