Algoritmos e Tipos Abstratos de Dados

2019/2020

Licenciatura em Engª. Informática

Projeto final: Importação, Normalização e Classificação de Dados

Projeto elaborado por André Dias – 190221068 e Tomás Barroso – 190221029
Turma EI-01

Professora Patricia Macedo – Aula Prática de quinta-feira às 8:30

Índice

1. Introdução

É no âmbito da cadeira de Algoritmos e Tipos Abstratos de Dados da licenciatura em engenharia informática no 2º semestre 1º ano que desenvolvemos este projeto da melhor maneira possível como foi pedido. Sendo o tema deste projeto o mais apropriado para os dias de hoje e a situação que enfrentamos atualmente, podemos claramente dizer que foi uma ideia criativa de fazer os alunos verem esta a situação de uma maneira diferente, neste caso de uma maneira acadêmica, pelo contrário em vez de proporcionar receio, mas sim conhecimento na área da tecnologia e na programação em linguagem C.

Tem como objetivo principal desenvolver um programa em C para extrair informação útil de um ficheiro com dados sobre a incidência do Covid-19 num determinado país e consiste num interpretador de comandos que o utilizador usa para obter diversos tipos de informação, principalmente informação estatística.

Sendo composto por três estruturas de dados que representam os três tipos de dados existentes: Date, Patient e Region.

2. Descrição dos ADTs utilizados

Para poder armazenar a informação relativa aos pacientes foi implementado, como pedido, o 'ADT List', que é caracterizado por uma política de acesso por rank/índice/posição para que o acesso a cada instante possa ser feito a qualquer elemento da coleção a partir do seu rank. Já para poder armazenar a informação relativa às regiões foi implementado, como pedido o 'ADT Map' que implementa uma memória associativa (armazena tuplos chave <-> valor), cada tuplo é efetivamente um mapeamento de uma chave para um valor, notando que não pode haver chaves repetidas, o que o faz reger-se por uma politica de acesso baseada em chave(removemos ou inserimos um valor a partir da chave).

Para o 'ADT List' a implementação escolhida foi 'Array List', pois achávamos que o desenvolvimento do projeto se tornaria mais simples apenas com o objetivo de mapear diretamente os ranks do ADT List aos índices do array elements, e que neste caso iria resultar numa menor complexidade algorítmica do que com a linkedList. Um dos motivos adicionais foi que a 'Linked List' em comparação à 'Array List', para pesquisa, possui complexidade algorítmica maior (O(n) vs O(1)).

Já para o 'ADT Map' a implementação escolhida foi também 'Array List', pois era com essa que estávamos mais familiarizados.

3. Complexidade Algorítmica dos comandos implementados

Comando	Complexidade Algorítmica
LOADP	O(n)
LOADR	O(n)
AVERAGE	O(n)
FOLLOW	O(n)
SEX	O(n)
SHOW	O(1)
TOP5	O(n^2)
OLDEST	O(n)
GROWTH	O(n)
MATRIX	O(n)
REGIONS	O(n^2)
REPORT	O(n^2)

4. Implementação de funcionalidades em pseudocódigo

Funcionalidade B

- Comando Average

```
ALGORITHM commandAverage
Input : list - PtList
Begin
DECLARE patientsListSize <- 0,countDeceasedAges <- 0,countDeceasedTotal <- 0
        countReleasedAges <-0,countReleasedTotal <- 0,countIsolatedAges <- 0,countIsolatedTotal <- 0,
        age <- 0 AS INTEGER
DECLARE errorcode <- listSize(list,&patientsListSize) AS INTEGER
        if(errorcode = LIST_NULL) then
                PRINT "Patients list wasn't loaded"
                return
        End if

        if(patientsListSize = 0) then
                Begin
                PRINT "Patients list is empty"
                return
        End if
DECLARE patient AS Patient
        FOR int i <- 1 TO patientsListSize DO
        listGet(list,i,&patient)
        age <- getAge(patient)
        if(age != -1) then
                Swicth ON getStatus(patient)
                        Case r: countReleasedTotal <- countReleasedTotal + 1, countReleasedAges<-
                        countReleasedAges  + getAge(patient)
                        Case d: countDeceasedTotal<- countDeceasedTotal + 1, countDeceasedAges<-
                        countDeceasedAges+ getAge(patient)
                        Case i: countIsolatedTotal<- countIsolatedTotal+ 1, countIsolatedAges<-
                        countIsolatedAges+ getAge(patient)
                        Default:
                End switch
        End if
        End For
End
```

- Comando Sex

```
ALGORITHM commandSex
        Input : list - PtList
Begin
DECLARE size <- 0, maleCount <- 0, femaleCount <- 0, unknownCount <- 0 AS INTEGER
DECLARE errorcode <- listSize(list,&size) AS INTEGER
DECLARE patient AS Patient
        if(errorcode = LIST_NULL) then
                PRINT "Patients list wasn't loaded"
                return
        End if
        if(size = 0) then
                PRINT "Patients list is empty"
                return
        End if
        FOR int i <-1 TO size do
                listGet(list,i,&patient)
                if(strcmp(patient.sex,"male") = 0) then
                        maleCount <- maleCount+ 1
                else if(strcmp(patient.sex," female") = 0) then
                        femaleCount = femaleCount+ 1
                else
                        unknownCount = unknownCount+ 1
                End if
                End if
                End if
        End for
        PRINT "Average Age for deceased patients: " round(getAverage(countDeceasedAges,
        countDeceasedTotal))
        PRINT "Average Age for released patients: "
        round(getAverage(countReleasedAges,countReleasedTotal))
        PRINT "Average Age for isolated patients: " round(getAverage(countIsolatedAges, countIsolatedTotal))
End
```

Funcionalidade C

- Comando Regions

```
ALGORITHM commandRegions
        Input : list – PtList, map – PtMap
Begin
DECLARE lSize <- 0 AS INTEGER
DECLARE mSize <-0 AS INTEGER
DECLARE arrSize <-0 AS INTEGER
DECLARE lerrorCode <- listSize(list, &lSize) AS INTEGER
DECLARE merrorCode <- mapSize(map, &mSize) AS INTEGER
        if(lerrorCode = LIST_NULL) then
                PRINT "Patients list wasn't loaded"
                Return
        End if
        if(lSize = 0) then
                Begin
                PRINT "Patients list is empty"
                return
        End if
        if(merrorCode = MAP_NULL) then
                PRINT "Regions list wasn't loaded"
                return
        End if
        if(mSize = 0) then
                PRINT "Regions list is empty"
                return
        End if
DECLARE **str <- (**)malloc(20*sizeof(*)) AS CHARACTER
        FOR int i <-1 TO 20 Do
                str(i) <- (char*)malloc(50*sizeof(char))
        End for
DECLARE patient AS Patient
DECLARE mapValue AS MapValue
DECLARE key AS KeyString
        FOR int i <-1 TO lSize Do
                listGet(list,i,&patient)
                if(getStatus(patient) = 'i') then
                        key <- createKey(patient.region)
                        mapGet(map, key, &mapValue)
                        if(!stringIsRepeated(str, arrSize, key.content)) then
                                strcpy(str(arrSize),key.content)
                                arrSize <- arrSize + 1
                        End if
                End if
        End for
        orderStringArrAlphaBBsort(str, arrSize)
        PRINT "Active regions (alphabetically): "
```

```
        FOR int i <- 1 TO arrSize Do
                key <- createKey(str(i))
                mapKeyPrint(key)
        End for
        FOR int i <- 1 TO 20 Do
                free(str(i))
        End for
        free(str)
End
```

5. Limitações

Todos os comandos e aspetos pedidos foram implementados e estão a funcionar da melhor maneira possível exceto o comando TOP5 que não desempata as idades e o comando REPORT não parece estar completamente de acordo com o pedido.

6. Conclusões e Análise Crítica

A partir de todo o trabalho feito e matéria lecionada em aula, achamos que este tema foi mais que indicado não só para pormos em prática tudo o que aprendemos, mas também por ter um projeto com um tema que poderá ser utilizado numa situação real no mercado de trabalho.
Foi de deveras interessante não só a implementação deste projeto, mas também o seu conteúdo e certamente que adquirimos diversos conhecimentos a partir do mesmo e da cadeira em global.

É claro que surgiram obstáculos, sendo o mais problemático a gestão do tempo, que felizmente foi ultrapassada e que permitiu evoluir as nossas capacidades.