1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA DEPARTAMENTO DE ENGENHARIA INFORMÁTICA	**Trabalho no 3 v3.1 – Lista de Referências** **Algoritmos e Estruturas de Dados** **2019/2020 – 2º Semestre** Upload: (link a disponibilizar no infoestudante) Data Limite: 17/Abril/2020, 18h00 Data Limite (PL1 e PL7): 24/Abril/2020, 18h00

O RELATÓRIO E LISTAGEM DO CÓDIGO DESENVOLVIDO DEVEM SER SUBMETIDOS NUM ÚNICO DOCUMENTO PDF

Nome: Rodrigo Fernando Henriques Sobral nº: 2018298209 PL: 2

Nº de horas de trabalho: Aulas Práticas de Laboratório: 10H Fora de Sala de Aula: 20H

CLASSIFICAÇÃO: (A Preencher pelo Docente)

Análise Empírica de Complexidade

Tarefa preparatória para o desenvolvimento desta ficha:

- Fazer o download dos 4 textos disponibilizados.
- Caracterizar cada texto em termos de número de palavras distintas, evidência de alguma ordem pré-estabelecida para as palavras, extensão do texto. Considere essa caracterização quando relevante na análise qualitativa que lhe é pedida mais adiante.
- Calcular na tabela abaixo os tempos[1] para as três/quatro versões do trabalho relativos às operações indicadas (a tarefa B é opcional).
- Analisar o número de rotações que vão ocorrer no carregamento do texto A (segunda tabela).

[1] Usar o tempo médio de 20 execuções do respetivo comando

TEXTO A

Núm. palavras distintas: 2920

Algum ordenamento? Caracterize? Não.

Núm. total de palavras: 10000

TEXTO B

Núm. palavras distintas: 2922

Algum ordenamento? Caracterize? Alfabeticamente crescente.

Núm. total de palavras: 10000

TEXTO C

Núm. palavras distintas: 2922

Algum ordenamento? Caracterize? Alfabeticamente decrescente.

Núm. total de palavras: 10038

TEXTO D

Núm. palavras distintas: 84

Algum ordenamento? Caracterize? Repetição igualmente intervalada de palavras.

Núm. total de palavras: 10000

#	Tarefa Tempos em [µs] Operação	A0	A1	A2	B
1	Carregamento **(Texto A)**	2,8 * 10^6	17,5 * 10^6	16,4 * 10^6	
2	Carregamento **(Texto B)**	2,95 * 10^6	16,65 * 10^6	15,35 * 10^6	
3	Carregamento **(Texto C)**	2,95 * 10^6	16.2 * 10^6	13,35 * 10^6	
4	50 chamadas do comando “LINHAS” com diferentes palavras do texto (escolha aleatória das palavras) **(Texto A)**	23 657,3	1 055,3	756,55	
5	50 chamadas do comando “ASSOC” com diferentes palavras do texto (escolha aleatória das palavras) e na linha definida aleatoriamente, dentro dos limites do texto (**Texto A)**	101,4	356,6	462,2	
6	500 chamadas do comando “LINHAS”	258 458,4	34 052,75	36 931,25	

	usando somente 10 palavras (T**exto D)** (escolha aleatória das palavras)				
7	Estrutura de dados auxiliar usada em cada uma das abordagens?	Array	Array	Array	

#	Tarefa Número Total de Rotações Simples Operação	A1	A2	B
8	Carregamento do texto D	31	29	

Reflexão sucinta sobre os resultados obtidos

(Formato de referência: Helvetica 10pt; texto para além do número de linhas não é considerado e desvaloriza o relatório)

1. Comente os resultados obtidos na tarefa A1 para os textos A, B e C.

Os resultados são os esperados. Usar uma árvore como estrutura de dados aumenta o acesso à informação comparativamente a um *array* por exemplo. Por outro lado, o tempo de carregamento do texto, que é, sem exceção, muito superior à tarefa A0, dado as várias rotações que têm de ser feitas.

2. Comente os resultados obtidos nas tarefas A0 a A2 e a B (opcional) para o texto A

São os esperados. O A0, limita-se a tornar as linhas recebidas num *array* e consecutivamente armazená-lo noutro array. Porém, o acesso a uma dada palavra é demorado, já que a pesquisa é exaustiva. O A2, apesar de também tornar as linhas num *array*, tem a função de manter constantemente, o equilíbrio, o que implica maior tempo de carregamento, mas um acesso mais rápido.

3. Compare os resultados obtidos nas operações 5 e 6 e relacione com as opções tomadas em termos de estrutura auxiliar de dados. Se achar que não há relação justifique. Comente os resultados obtidos para a tarefa B com estas duas operações.

Pondo de parte a diferença presente na quantidade de chamadas que são feitas, relativamente à operação 5, verificamos tempos reduzidos. Uma das razões é o facto de já ser conhecida a linha de ocorrência da palavra *a priori,* o que faz com que não precisemos de percorrer a estrutura auxiliar para imprimir as linha de ocorrência, e, em particular, no caso A0, não precisemos de percorrer todas as linhas do texto até encontrar a palavra, por isso é que resultou num tempo de execução ainda mais baixo que os outros algoritmos. Por outro lado, a operação 6 obriga a iteração da estrutura auxiliar, e sendo ela um *array*, implica um aumento do tempo de execução. Este tempo poderia ser possivelmente reduzido usando como estrutura auxiliar uma outra árvore. Além das apresentadas, outra causa do aumento abrupto de tempo, nomeadamente do algoritmo A0, é o facto de se estar constantemente a percorrer a estrutura de dados em busca da palavra requisitada.

4. Analise e comente os resultados da operação 8.

São esperados. O Texto D, apesar de possuir uma grande quantidade de palavras, possui pouca variedade. Isto implica que a quantidade de inserções de novos nós seja reduzida, logo a probabilidade de efetuar algum tipo de rotação ao ler uma nova palavra vai ser também menor

Algoritmo A0

```python
from re import split
from datetime import datetime
from random import randint

def procuraLinhaDaPalavra(texto, palavraProcurada):
    linhas=""
    for linha in range(len(texto)):
        for palavra in texto[linha]:
            if palavraProcurada==palavra and str(linha) not in linhas: linhas+=str(linha)+" "
    if (linhas==""): return "-1\n"
    else:
        linhas = linhas[:-1]
        return linhas+"\n"

def procuraPalavraNaLinha(texto, palavraProcurada, linha):
    for palavra in texto[linha]:
        if palavraProcurada==palavra: return "ENCONTRADA.\n"
    return "NAO ENCONTRADA.\n"

def recebe_comandos():
    resultados=""
    texto=[]
    while True:
        comando=input().strip("\n")

        if comando=="TEXTO":
            start_time = datetime.now()
            aux=input().strip("\n").upper()
            while (aux!="FIM."):
                texto.append(split(r'[( ),.;]\s*',aux))
                aux=input().strip("\n").upper()
            resultados+="GUARDADO.\n"
            now1=(datetime.now()-start_time).seconds
            break       You, 4 days ago • First Commit
        else: continue

    start_time = datetime.now()
    # LINHAS
    for i in range(50):
        linha=texto[randint(0,999)]
        palavra=linha[randint(0,len(linha)-1)]
        procuraLinhaDaPalavra(texto, palavra)
    now2=(datetime.now()-start_time).microseconds

    start_time = datetime.now()
    # ASSOC
    for i in range(50):
        num_linha=randint(0, 1000-1)
        linha=texto[num_linha]
        palavra=linha[randint(0,len(linha)-1)]
        procuraPalavraNaLinha(texto, palavra, num_linha)
    now3=(datetime.now()-start_time).microseconds

    # LINHAS
    lista_palavras=[]
    for i in range(10):
        linha=texto[randint(0, 1000-1)]
        lista_palavras.append(linha[randint(0,len(linha)-1)])
    start_time = datetime.now()
    for i in range(500): procuraLinhaDaPalavra(texto, lista_palavras[randint(0, len(lista_palavras)-1)])
    now4=(datetime.now()-start_time).microseconds

    return now1, now2, now3, now4
if __name__ == "__main__":
    tempos=[0,0,0,0]
    for i in range(20):
        aux=recebe_comandos()
        tempos[0]+=aux[0]
        tempos[1]+=aux[1]
        tempos[2]+=aux[2]
        tempos[3]+=aux[3]
    print("TEMPOS MEDIOS:\nCARREGAMENTO: {}\n50 LINHAS {}\n50 ASSOC: {}\n500 LINHAS: {}".format(tempos[0]/20, tempos[1]/20, tempos[2]/20,
    tempos[3]/20))

```

Algoritmo A1

```python
from re import split
from datetime import datetime
from random import randint

class Node():
    tot_palavras=0
    palavras_distintas=0
    numero_rotacoes=0
    def __init__(self, palavra, linha):
        self.palavra=palavra
        self.linha=[linha]
        self.fator_equi=0
        self.altura=1
        self.menor=None
        self.maior=None
        Node.tot_palavras+=1
        Node.palavras_distintas+=1

    def equilibrarArvore(self):
        auxcima=self
        if self.fator_equi<-1:
            self=self.maior
            auxcima.maior=None
            if self.fator_equi==-1:
                if self.menor!=None:
                    auxcima.maior=self.menor
                    self.menor=None
            elif self.fator_equi==1:
                auxmeio=self
                self=self.menor
                if self.menor!=None:
                    auxcima.maior=self.menor
                if self.maior!=None:
                    auxmeio.menor=self.maior
                else: auxmeio.menor=None
                self.maior=auxmeio
            self.menor=auxcima
        elif self.fator_equi>1:
            self=self.menor
            auxcima.menor=None
            if self.fator_equi==1:
                if self.maior!=None:
                    auxcima.menor=self.maior
                    self.maior=None
            elif self.fator_equi==-1:
                auxmeio=self
                self=self.maior
                if self.maior!=None:
                    auxcima.menor=self.maior
                if self.menor!=None:
                    auxmeio.maior=self.menor
                else: auxmeio.maior=None
                self.menor=auxmeio
            self.maior=auxcima
        Node.numero_rotacoes+=1
        return self

    def atualizaAlturasFatores(self):
        if self.menor==None and self.maior==None:
            self.altura=1
            self.fator_equi=0
            return self
        altura_menor, altura_maior= 0, 0
        if self.menor!=None:
            self.menor=self.menor.atualizaAlturasFatores()
            altura_menor=self.menor.altura
        if self.maior!=None:
            self.maior= self.maior.atualizaAlturasFatores()
            altura_maior=self.maior.altura

        self.fator_equi= altura_menor-altura_maior

        if self.fator_equi>1 or self.fator_equi<-1:
            self=self.equilibrarArvore()
            self.atualizaAlturasFatores()
            if self.maior.altura>self.menor.altura: self.altura= 1+self.maior.altura
            else: self.altura= 1+self.menor.altura
            return self

        if altura_maior>altura_menor: self.altura= 1+altura_maior
        else: self.altura= 1+altura_menor
        return self

    def adicionarElementosAVL(self, palavra, linha):
        while True:
            if self.palavra>palavra:
                if (self.menor==None):
                    self.menor=Node(palavra, linha)
                    break
                else: self=self.menor
            elif self.palavra<palavra:
                if (self.maior==None):
                    self.maior=Node(palavra, linha)
                    break
                else: self=self.maior
            else:
                Node.tot_palavras+=1
                if linha not in self.linha: self.linha.append(linha)
                break

    def procuraPalavraNaArvore(self, palavra, linha):
        ocorencias=""
        while True:
            if self.palavra==palavra:
                if linha==None:
                    for i in range(len(self.linha)):
                        if i!=len(self.linha)-1: ocorencias+=str(self.linha[i])+" "
                        else: ocorencias+=str(self.linha[i])+"\n"
                    return ocorencias
                elif linha in self.linha: return "ENCONTRADA.\n"
                else: return "NAO ENCONTRADA.\n"
            elif self.palavra>palavra:
                if (self.menor==None): return "-1\n"
                else: self=self.menor
            elif self.palavra<palavra:
                if (self.maior==None): return "-1\n"
                else: self=self.maior

def recebe_comandos():
    arvore=None
    resultados=""
    while True:
        comando=input().strip("\n")

        if comando=="TEXTO":
            start_time = datetime.now()
            num_linha=0
            texto=[]
            aux=input().strip("\n").upper()
            while (aux!="FIM."):
                conteudo=split(r'[( ),.;]\s*',aux)
                texto.append(conteudo)
                arvore= adicionaPalavras(arvore, conteudo, num_linha)
                aux=input().strip("\n").upper()
                num_linha+=1
            resultados+= "GUARDADO.\n"
            now1=(datetime.now()-start_time).seconds
            break

    # 50 LINHAS
    start_time = datetime.now()
    for i in range(50):
        linha=texto[randint(0, 999)]
        arvore.procuraPalavraNaArvore(linha[randint(0, len(linha)-1)], None)
    now2=(datetime.now()-start_time).microseconds

    # ASSOC
    start_time = datetime.now()
    for i in range(50):
        linha=texto[randint(0, 999)]
        arvore.procuraPalavraNaArvore(linha[randint(0, len(linha)-1)], randint(0,999))
    now3=(datetime.now()-start_time).microseconds
    # 500 LINHAS
    palavras=[]
    for i in range(10):
        linha=texto[randint(0, 999)]
        palavras.append(linha[randint(0, len(linha)-1)])

    start_time = datetime.now()
    for i in range(500): arvore.procuraPalavraNaArvore(palavras[randint(0, len(palavras)-1)], None)
    now4=(datetime.now()-start_time).microseconds

    print("\nDADOS:\nPalavras: {}\nDistintas: {}".format(arvore.tot_palavras, arvore.palavras_distintas))
    print("Numero de rotacoes simples:", Node.numero_rotacoes)
    return now1, now2, now3, now4

def adicionaPalavras(arvore: Node, linha, n_linha):
    if n_linha==0:
        arvore=Node(linha[0], n_linha)
        for palavra in range(1, len(linha)):
            if linha[palavra]!='':
                arvore.adicionarElementosAVL(linha[palavra], n_linha)
                arvore=arvore.atualizaAlturasFatores()
    else:
        for palavra in range(len(linha)):
            if linha[palavra]!='':
                arvore.adicionarElementosAVL(linha[palavra], n_linha)
                arvore=arvore.atualizaAlturasFatores()
    return arvore
if __name__ == "__main__":
    tempos=[0,0,0,0]
    for i in range(20):
        aux=recebe_comandos()
        tempos[0]+=aux[0]
        tempos[1]+=aux[1]
        tempos[2]+=aux[2]
        tempos[3]+=aux[3]
    print("TEMPOS MEDIOS:\nCARREGAMENTO: {}\n50 LINHAS {}\n50 ASSOC: {}\n500 LINHAS: {}".format(tempos[0]/20, tempos[1]/20, tempos[2]/20,
    tempos[3]/20))
```

Algoritmo A2

```python
from re import split
from datetime import datetime
from random import randint

You, 4 days ago | 1 author (You)
class Node():
    numero_rotacoes=0
    def __init__(self, palavra, linha):
        self.palavra=palavra
        self.linha=[linha]
        self.isRed=True
        self.menor=None
        self.maior=None

    def equilibrarArvore(self, no_pai):
        if no_pai.maior==self:
            if no_pai.menor!=None and no_pai.menor.isRed==True:
                no_pai.isRed, no_pai.menor.isRed, no_pai.maior.isRed= True, False, False
                return 1, self
            else:
                if self.menor!=None and self.menor.isRed==True:
                    aux_meio=self
                    self=self.menor
                    no_pai.maior, aux_meio.menor=self.menor, self.maior
                    self.maior, self.menor=aux_meio, no_pai
                    self.isRed, self.maior.isRed, self.menor.isRed= False, True, True
                elif self.maior!=None and self.maior.isRed==True:
                    no_pai.maior=self.menor
                    self.menor=no_pai
                    self.isRed, self.maior.isRed, self.menor.isRed= False, True, True
                Node.numero_rotacoes+=1
                return 2, self
        if no_pai.maior!=None and no_pai.maior.isRed==True:
            no_pai.isRed, no_pai.menor.isRed, no_pai.maior.isRed= True, False, False
            return 1, self
        else:
            if self.maior!=None and self.maior.isRed==True:
                aux_meio=self
                self=self.maior
                no_pai.menor, aux_meio.maior=self.maior, self.menor
                self.menor, self.maior= aux_meio, no_pai
                self.isRed, self.menor.isRed, self.maior.isRed= False, True, True
            elif self.menor!=None and self.menor.isRed==True:
                no_pai.menor=self.maior
                self.maior=no_pai
                self.isRed, self.maior.isRed, self.menor.isRed= False, True, True
        Node.numero_rotacoes+=1
        return 2, self

    def atualizaCores(self, no_pai):
        if self.menor==None and self.maior==None: return -1, self
        if self.menor!=None:
            resultado= self.menor.atualizaCores(self)
            if resultado[0]==-1: pass
            elif resultado[0]==0 or resultado[0]==1: self.menor=resultado[1]
            elif resultado[0]==2: self=resultado[1]
        if self.maior!=None:
            resultado= self.maior.atualizaCores(self)
            if resultado[0]==-1: pass
            elif resultado[0]==0 or resultado[0]==1: self.maior=resultado[1]
            elif resultado[0]==2: self=resultado[1]
        if self.maior!=None and self.isRed==True and self.maior.isRed==True:
            return self.equilibrarArvore(no_pai)
        if self.menor!=None and self.isRed==True and self.menor.isRed==True:
            return self.equilibrarArvore(no_pai)
        return 0, self

    def adicionarElementosAVL(self, palavra, linha):
        while True:
            if self.palavra>palavra:
                if (self.menor==None):
                    self.menor=Node(palavra, linha)
                    break
                else: self=self.menor
            elif self.palavra<palavra:
                if (self.maior==None):
                    self.maior=Node(palavra, linha)
                    break
                else: self=self.maior
            else:
                if linha not in self.linha: self.linha.append(linha)
                break

    def procuraPalavraNaArvore(self, palavra, linha):
        ocorencias=""
        while True:
            if self.palavra==palavra:
                if linha==None:
                    for i in range(len(self.linha)):
                        if i!=len(self.linha)-1: ocorencias+=str(self.linha[i])+" "
                        else: ocorencias+=str(self.linha[i])+"\n"
                    return ocorencias
                elif linha in self.linha: return "ENCONTRADA.\n"
                else: return "NAO ENCONTRADA.\n"
            elif self.palavra>palavra:
                if (self.menor==None): return "-1\n"
                else: self=self.menor
            elif self.palavra<palavra:
                if (self.maior==None): return "-1\n"
                else: self=self.maior

def recebe_comandos():
    arvore=None
    resultados=""
    while True:
        comando=input().strip("\n")

        if comando=="TEXTO":
            start_time = datetime.now()
            linha=0
            texto=[]
            aux=input().strip("\n").upper()
            while (aux!="FIM."):
                conteudo=split(r'[( ),.;]\s*',aux)
                texto.append(conteudo)
                arvore= adicionaPalavras(arvore, conteudo, linha)
                aux=input().strip("\n").upper()
                linha+=1
            resultados+= "GUARDADO.\n"
            now1=(datetime.now()-start_time).seconds
            break

    # LINHAS
    start_time = datetime.now()
    for i in range(50):
        linha=texto[randint(0, 999)]
        arvore.procuraPalavraNaArvore(linha[randint(0, len(linha)-1)], None)
    now2=(datetime.now()-start_time).microseconds

    # ASSOC
    start_time = datetime.now()
    for i in range(50):
        linha=texto[randint(0, 999)]
        arvore.procuraPalavraNaArvore(linha[randint(0, len(linha)-1)], randint(0,999))
    now3=(datetime.now()-start_time).microseconds

    palavras=[]
    for i in range(10):
        linha=texto[randint(0, 999)]
        palavras.append(linha[randint(0, len(linha)-1)])

    start_time = datetime.now()
    for i in range(500): arvore.procuraPalavraNaArvore(palavras[randint(0, len(palavras)-1)], None)
    now4=(datetime.now()-start_time).microseconds

    print("\nDADOS:\nPalavras: {}\nDistintas: {}".format(arvore.tot_palavras, arvore.palavras_distintas))
    print("Numero de rotacoes simples:", Node.numero_rotacoes)
        You, a few seconds ago • Uncommitted changes
    return now1, now2, now3, now4

def adicionaPalavras(arvore: Node, linha, n_linha):
    if n_linha==0:
        arvore=Node(linha[0], n_linha)
        arvore.isRed=False
        for palavra in range(1, len(linha)):
            if linha[palavra]!='':
                arvore.adicionarElementosAVL(linha[palavra], n_linha)
                arvore= arvore.atualizaCores(arvore)[1]
                arvore.isRed=False
    else:
        for palavra in range(len(linha)):
            if linha[palavra]!='':
                arvore.adicionarElementosAVL(linha[palavra], n_linha)
                arvore= arvore.atualizaCores(arvore)[1]
                arvore.isRed=False
    return arvore

if __name__ == "__main__":
    tempos=[0,0,0,0]
    for i in range(20):
        aux=recebe_comandos()
        tempos[0]+=aux[0]
        tempos[1]+=aux[1]
        tempos[2]+=aux[2]
        tempos[3]+=aux[3]
    print("TEMPOS MEDIOS:\nCARREGAMENTO: {}\n50 LINHAS {}\n50 ASSOC: {}\n500 LINHAS: {}".format(tempos[0]/20, tempos[1]/20, tempos[2]/20,
    tempos[3]/20))
```

Bom trabalho, os Docentes da Disciplina,

Carlos L Bento e Catarina Silva