

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

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 tempos¹ 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 [μ s] 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 (Texto 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

```
1 from re import split
2 from datetime import datetime
3 from random import randint
4
5 def procuraLinhaDaPalavra(texto, palavraProcurada):
6     linhas=""
7     for linha in range(len(texto)):
8         for palavra in texto[linha]:
9             if palavraProcurada==palavra and str(linha) not in linhas: linhas+=str(linha)+" "
10    if (linhas==""): return "-1\n"
11    else:
12        linhas = linhas[:-1]
13        return linhas+"\n"
14
15 def procuraPalavraNaLinha(texto, palavraProcurada, Linha):
16     for palavra in texto[Linha]:
17         if palavraProcurada==palavra: return "ENCONTRADA.\n"
18     return "NAO ENCONTRADA.\n"
19
20 def recebe_comandos():
21     resultados=""
22     texto=[]
23     while True:
24         comando=input().strip("\n")
25
26         if comando=="TEXT0":
27             start_time = datetime.now()
28             aux=input().strip("\n").upper()
29             while (aux!="FIM."):
30                 texto.append(split(r'[(\ ),.;]\s+',aux))
31                 aux=input().strip("\n").upper()
32             resultados+="GUARDADO.\n"
33             now1=(datetime.now()-start_time).seconds
34             break
35             You, 4 days ago • First Commit
36         else: continue
37
38 start_time = datetime.now()
39 # LINHAS
40 for i in range(50):
41     linha=texto[randint(0,999)]
42     palavra=linha[randint(0,len(linha)-1)]
43     procuraLinhaDaPalavra(texto, palavra)
44     now2=(datetime.now()-start_time).microseconds
45
46 start_time = datetime.now()
47 # ASSOC
48 for i in range(50):
49     num_linha=randint(0, 1000-1)
50     linha=texto[num_linha]
51     palavra=linha[randint(0,len(linha)-1)]
52     procuraPalavraNaLinha(texto, palavra, num_linha)
53     now3=(datetime.now()-start_time).microseconds
54
55 # LINHAS
56 lista_palavras=[]
57 for i in range(10):
58     linha=texto[randint(0, 1000-1)]
59     lista_palavras.append(linha[randint(0,len(linha)-1)])
60 start_time = datetime.now()
61 for i in range(500): procuraLinhaDaPalavra(texto, lista_palavras[randint(0, len(lista_palavras)-1)])
62 now4=(datetime.now()-start_time).microseconds
63
64 return now1, now2, now3, now4
65
66 if __name__ == "__main__":
67     tempos=[0,0,0,0]
68     for i in range(20):
69         aux=recebe_comandos()
70         tempos[0]+=aux[0]
71         tempos[1]+=aux[1]
72         tempos[2]+=aux[2]
73         tempos[3]+=aux[3]
74     print("TEMPOS MEDIOS:\nCARREGAMENTO: {} \n50 LINHAS {} \n50 ASSOC: {} \n500 LINHAS: {}".format(tempos[0]/20, tempos[1]/20, tempos[2]/20,
75     tempos[3]/20))
```

Algoritmo A1

```

1 from re import split
2 from datetime import datetime
3 from random import randint
4
5 You, a few seconds ago | 1 author (You)
6
7 class Node():
8     tot_palavras=0
9     palavras_distintas=0
10    numero_rotacoes=0
11
12    def __init__(self, palavra, linha):
13        self.palavra=palavra
14        self.linha=linha
15        self.fator_equi=0
16        self.altura=1
17        self.menor=None
18        self.maior=None
19        Node.tot_palavras+=1
20        Node.palavras_distintas+=1
21
22    def equilibrarArvore(self):
23        auxcima=self
24        if self.fator_equi<-1:
25            self=self.maior
26            auxcima.maior=None
27            if self.fator_equi==1:
28                if self.menor!=None:
29                    auxcima.maior=self.menor
30                    self.menor=None
31            elif self.fator_equi==1:
32                auxmeio=self
33                self=self.menor
34                if self.menor!=None:
35                    auxcima.maior=self.menor
36                if self.maior!=None:
37                    auxmeio.menor=self.maior
38            else: auxmeio.menor=None
39            self.maior=auxmeio
40            self.menor=auxcima
41        elif self.fator_equi>1:
42            self=self.menor
43            auxcima.menor=None
44            if self.fator_equi==1:
45                if self.maior!=None:
46                    auxcima.menor=self.maior
47                self.maior=None
48            elif self.fator_equi==1:
49                auxmeio=self
50                self=self.maior
51                if self.maior!=None:
52                    auxcima.menor=self.maior
53                if self.menor!=None:
54                    auxmeio.maior=self.menor
55            else: auxmeio.maior=None
56            self.maior=auxmeio
57            self.menor=auxcima
58        Node.numero_rotacoes+=1
59        return self
60
61    def atualizaAlturasFatores(self):
62        if self.menor==None and self.maior==None:
63            self.altura=1
64            self.fator_equi=0
65            return self
66        altura_menor, altura_maior = 0, 0
67        if self.menor!=None:
68            self.menor=self.menor.atualizaAlturasFatores()
69            altura_menor=self.menor.altura
70        if self.maior!=None:
71            self.maior=self.maior.atualizaAlturasFatores()
72            altura_maior=self.maior.altura
73        self.fator_equi= altura_menor-altura_maior
74
75        if self.fator_equi>1 or self.fator_equi<-1:
76            self=self.equilibrarArvore()
77            self.atualizaAlturasFatores()
78            if self.maior.altura>self.menor.altura: self.altura= 1+self.maior.altura
79            else: self.altura= 1+self.menor.altura
80            return self
81
82        if altura_maior>altura_menor: self.altura= 1+altura_maior
83        else: self.altura= 1+altura_menor
84        return self
85
86    def adicionarElementosAVL(self, palavra, linha):
87        while True:
88            if self.palavra==palavra:
89                if (self.menor!=None):
90                    self.menor=Node(palavra, linha)
91                    break
92                else: self=self.menor
93            elif self.palavra<palavra:
94                if (self.maior!=None):
95                    self.maior=Node(palavra, linha)
96                    break
97                else: self=self.maior

```

```

98    else:
99        Node.tot_palavras+=1
100        if linha not in self.linha: self.linha.append(linha)
101        break
102
103    def procuraPalavraNaArvore(self, palavra, linha):
104        ocorrencias=""
105        while True:
106            if self.palavra==palavra:
107                if linha==None:
108                    for i in range(len(self.linha)):
109                        if i!=len(self.linha)-1: ocorrencias+=str(self.linha[i])+" "
110                        else: ocorrencias+=str(self.linha[i])+"\n"
111                    return ocorrencias
112                elif linha in self.linha: return "ENCONTRADA.\n"
113                else: return "NAO ENCONTRADA.\n"
114            elif self.palavra>palavra:
115                if (self.menor==None): return "-1\n"
116                else: self=self.menor
117            elif self.palavra<palavra:
118                if (self.maior==None): return "-1\n"
119                else: self=self.maior
120
121    def recebe_comandos():
122        arvore=None
123        resultados=""
124        while True:
125            comando=input().strip("\n")
126
127            if comando=="TEXTO":
128                start_time = datetime.now()
129                num_linha=0
130                texto=[]
131                aux=input().strip("\n").upper()
132                while (aux!="FIM."):
133                    conteudo=split(r'([ ,.;])\s*',aux)
134                    texto.append(conteudo)
135                    arvore= adicionaPalavras(arvore, conteudo, num_linha)
136                    aux=input().strip("\n").upper()
137                    num_linha+=1
138                resultados+= "GUARDADO.\n"
139                now1=(datetime.now()-start_time).seconds
140                break
141
142            # 50 LINHAS
143            start_time = datetime.now()
144            for i in range(50):
145                linha=texto[randint(0, 999)]
146                arvore.procuraPalavraNaArvore(linha[randint(0, len(linha)-1)], None)
147                now2=(datetime.now()-start_time).microseconds
148
149            # ASSOC
150            start_time = datetime.now()
151            for i in range(50):
152                linha=texto[randint(0, 999)]
153                arvore.procuraPalavraNaArvore(linha[randint(0, len(linha)-1)], randint(0,999))
154                now3=(datetime.now()-start_time).microseconds
155            # 500 LINHAS
156            palavras=[]
157            for i in range(10):
158                linha=texto[randint(0, 999)]
159                palavras.append(linha[randint(0, len(linha)-1)])
160
161            start_time = datetime.now()
162            for i in range(500): arvore.procuraPalavraNaArvore(palavras[randint(0, len(palavras)-1)], None)
163            now4=(datetime.now()-start_time).microseconds
164
165            print("\nDADOS:\nPalavras: {} \nDistintas: {}".format(arvore.tot_palavras, arvore.palavras_distintas))
166            print("Numero de rotacoes simples:", Node.numero_rotacoes)
167            return now1, now2, now3, now4
168
169    def adicionaPalavras(arvore: Node, linha, n_linha):
170        if n_linha==0:
171            arvore=Node(linha[0], n_linha)
172            for palavra in range(1, len(linha)):
173                if linha[palavra]!=" ":
174                    arvore.adicionarElementosAVL(linha[palavra], n_linha)
175                    arvore=arvore.atualizaAlturasFatores()
176            else:
177                for palavra in range(len(linha)):
178                    if linha[palavra]!=" ":
179                        arvore.adicionarElementosAVL(linha[palavra], n_linha)
180                        arvore=arvore.atualizaAlturasFatores()
181            return arvore
182
183    if __name__ == "__main__":
184        tempos=[0,0,0,0]
185        for i in range(20):
186            aux=recebe_comandos()
187            tempos[0]+=aux[0]
188            tempos[1]+=aux[1]
189            tempos[2]+=aux[2]
190            tempos[3]+=aux[3]
191            print("TEMPOS MEDIOS:\nCARREGAMENTO: {} \n50 LINHAS: {} \n500 ASSOC: {} \n500 LINHAS: {}".format(tempos[0]/20, tempos[1]/20, tempos[2]/20,
192            tempos[3]/20))

```

Algoritmo A2

```
1 from re import split
2 from datetime import datetime
3 from random import randint
4
5 You, 4 days ago | 1 author (You)
6
7 class Node():
8     numero_rotacoes=0
9     def __init__(self, palavra, linha):
10         self.palavra=palavra
11         self.linha=[linha]
12         self.isRed=True
13         self.menor=None
14         self.maior=None
15
16     def equilibrarArvore(self, no_pai):
17         if no_pai.maior==self:
18             if no_pai.menor!=None and no_pai.menor.isRed==True:
19                 no_pai.isRed, no_pai.menor.isRed, no_pai.maior.isRed= True, False, False
20                 return 1, self
21             else:
22                 if self.menor!=None and self.menor.isRed==True:
23                     aux_meio=self
24                     self=self.menor
25                     no_pai.maior, aux_meio.menor=self.menor, self.maior
26                     self.maior, self.menor=aux_meio, no_pai
27                     self.isRed, self.maior.isRed, self.menor.isRed= False, True, True
28                 elif self.maior!=None and self.maior.isRed==True:
29                     no_pai.maior=self.maior
30                     self.menor=no_pai
31                     self.isRed, self.maior.isRed, self.menor.isRed= False, True, True
32                     Node.numero_rotacoes+=1
33                     return 2, self
34             if no_pai.maior!=None and no_pai.maior.isRed==True:
35                 no_pai.isRed, no_pai.menor.isRed, no_pai.maior.isRed= True, False, False
36                 return 1, self
37             else:
38                 if self.maior!=None and self.maior.isRed==True:
39                     aux_meio=self
40                     self=self.maior
41                     no_pai.menor, aux_meio.maior=self.maior, self.menor
42                     self.menor, self.maior=aux_meio, no_pai
43                     self.isRed, self.menor.isRed, self.maior.isRed= False, True, True
44                 elif self.menor!=None and self.menor.isRed==True:
45                     no_pai.menor=self.maior
46                     self.maior=no_pai
47                     self.isRed, self.maior.isRed, self.menor.isRed= False, True, True
48                     Node.numero_rotacoes+=1
49                     return 2, self
50
51     def atualizaCores(self, no_pai):
52         if self.menor==None and self.maior==None: return -1, self
53         if self.menor!=None:
54             resultado= self.menor.atualizaCores(self)
55             if resultado[0]==-1: pass
56             elif resultado[0]==0 or resultado[0]==1: self.menor=resultado[1]
57             elif resultado[0]==2: self=resultado[1]
58         if self.maior!=None:
59             resultado= self.maior.atualizaCores(self)
60             if resultado[0]==-1: pass
61             elif resultado[0]==0 or resultado[0]==1: self.maior=resultado[1]
62             elif resultado[0]==2: self=resultado[1]
63         if self.maior!=None and self.isRed==True and self.maior.isRed==True:
64             return self.equilibrarArvore(no_pai)
65         if self.menor!=None and self.isRed==True and self.menor.isRed==True:
66             return self.equilibrarArvore(no_pai)
67         return 0, self
68
69     def adicionarElementosAVL(self, palavra, linha):
70         while True:
71             if self.palavra==palavra:
72                 if (self.menor==None):
73                     self.menor=Node(palavra, linha)
74                     break
75                 else: self=self.menor
76             elif self.palavra<palavra:
77                 if (self.maior==None):
78                     self.maior=Node(palavra, linha)
79                     break
80                 else: self=self.maior
81             else:
82                 if linha not in self.linha: self.linha.append(linha)
83                 break
84
85     def procuraPalavraNaArvore(self, palavra, linha):
86         ocorencias=""
87         while True:
88             if self.palavra==palavra:
89                 if linha==None:
90                     for i in range(len(self.linha)):
91                         if i!=len(self.linha)-1: ocorencias+=str(self.linha[i])+ "
92                         else: ocorencias+=str(self.linha[i])+"\n"
93                     return ocorencias
94                 elif linha in self.linha: return "ENCONTRADA.\n"
95                 else: return "NAO ENCONTRADA.\n"
```

```
96         elif self.palavra>palavra:
97             if (self.menor==None): return "-1\n"
98             else: self=self.menor
99         elif self.palavra<palavra:
100             if (self.maior==None): return "-1\n"
101             else: self=self.maior
102
103     def recebe_comandos():
104         arvore=None
105         resultados=""
106         while True:
107             comando=input().strip("\n")
108
109             if comando=="TEXT0":
110                 start_time = datetime.now()
111                 linha=0
112                 texto=[]
113                 aux=input().strip("\n").upper()
114                 while (aux!="FIM."):
115                     conteudo=split('([ (,.;])\s+',aux)
116                     texto.append(conteudo)
117                     arvore= adicionaPalavras(arvore, conteudo, linha)
118                     aux=input().strip("\n").upper()
119                     linha+=1
120                 resultados+= "GUARDADO.\n"
121                 now1=(datetime.now()-start_time).seconds
122                 break
123
124             # LINHAS
125             start_time = datetime.now()
126             for i in range(50):
127                 linha=texto[randint(0, 999)]
128                 arvore.procuraPalavraNaArvore(linha[randint(0, len(linha)-1)], None)
129                 now2=(datetime.now()-start_time).microseconds
130
131             # ASSOC
132             start_time = datetime.now()
133             for i in range(50):
134                 linha=texto[randint(0, 999)]
135                 arvore.procuraPalavraNaArvore(linha[randint(0, len(linha)-1)], randint(0,999))
136                 now3=(datetime.now()-start_time).microseconds
137
138             palavras=[]
139             for i in range(10):
140                 linha=texto[randint(0, 999)]
141                 palavras.append(linha[randint(0, len(linha)-1)])
142
143             start_time = datetime.now()
144             for i in range(500): arvore.procuraPalavraNaArvore(palavras[randint(0, len(palavras)-1)], None)
145             now4=(datetime.now()-start_time).microseconds
146
147             print("\nDADOS:\nPalavras: {} \nDistintas: {}".format(arvore.tot_palavras, arvore.palavras_distintas))
148             print("Numero de rotacoes simples:", Node.numero_rotacoes)
149             You, a few seconds ago • Uncommitted changes
150             return now1, now2, now3, now4
151
152     def adicionaPalavras(arvore: Node, linha, n_linha):
153         if n_linha==0:
154             arvore=Node(linha[0], n_linha)
155             arvore.isRed=False
156             for palavra in range(1, len(linha)):
157                 if linha[palavra]!="":
158                     arvore.adicionarElementosAVL(linha[palavra], n_linha)
159                     arvore= arvore.atualizaCores(arvore)[1]
160                     arvore.isRed=False
161             return arvore
162         else:
163             for palavra in range(len(linha)):
164                 if linha[palavra]!="":
165                     arvore.adicionarElementosAVL(linha[palavra], n_linha)
166                     arvore= arvore.atualizaCores(arvore)[1]
167                     arvore.isRed=False
168             return arvore
169
170     if __name__ == "__main__":
171         tempos=[0,0,0,0]
172         for i in range(20):
173             aux=recebe_comandos()
174             tempos[0]+=aux[0]
175             tempos[1]+=aux[1]
176             tempos[2]+=aux[2]
177             tempos[3]+=aux[3]
178         print("TEMPOS MEDIOS:\nCARGAMENTO: {} \n50 LINHAS {} \n50 ASSOC: {} \n50 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