Universidade Federal do Ceará – UFC
Centro de Ciências – CC
Mestrado e Doutorado em Ciências da Computação - MDCC

Estruturas de Dados Avançadas

Exercício: Listas, Complexidade de Algoritmos, Tabelas de Dispersão, Hashing Dinâmico, Heap, Conjuntos e Partições
Objetivos: Exercitar os conceitos de Listas, Complexidade de Algoritmos, Tabelas de Dispersão, Hashing Dinâmico, Heap, Conjuntos e Partições

NOME: Pedro Roger Magalhães Vasconcelos

1)

```
class Hash {


  def doHash1(str: String): Int = {
    var tmp: Int = 0
    str.foreach(x => tmp += x.toInt)


    return tmp % 513
  }


}



object Main {
  def main(args: Array[String]): Unit = {
    var hash = new Hash
    println(hash.doHash1("Brasil"))
  }


}
```

2)

Dada uma função hash:

h(x) = x mod k, com k = 11

Temos:

h(13) = 2

h(2) = 2 (colisão, armazenamos, h(2) na próxima posição livre: 3)

h(3) = 3 (colisão, pois a posição 3 já está ocupada com h(2))

Assim, ocorre uma colisão com h(2) != h(3).

3)

Não podemos simplesmente apagar a entrada da tabela porque a referida chave pode conter mais de um elemento, sendo que o segundo elemento pode estar na próxima posição vazia encontrada. Ao apagar a entrada da tabela podemos estar excluindo um valor errado, i.e um elemento diferente daquele que buscamos excluir. Como alternativa podemos usar valores especias de chave, como "posição vazia" ou "elemento removido", assim uma inserção posterior pode reaproveitar posições marcadas como "elemento removido".

4)

a) h(x) = (13 * x) % 7

h(x)	0	1	2	3	4	5	6
x	14, 7			17, 10	16	15	

b)

h(x)	0	1	2	3	4	5	6
x	7		17	16	10	14	15

5)

a) x = 4, 12, 20, 28, 36, 44, 52, 60

h(x)	0	1	2	3	4	5	6	7

x 40 60 28 52 4 12 36 20

b) h(x,k) = (x+k) mod 8

c) Linear já que tenta agrupar tentativas consecutivas.

d) A função retorna um valor menor que zero quando estiver cheia, assim k não será menor que MAX, retornando -1.

e) Para valores menores ou igual a MAX (tamanho da tabela).

6)

a) n mod 5

-> (n - 5)

b) n mod 7

-> (n - 7)

c) n mod 35

-> (n - 35)

d) n mod (m/5)

-> (n - (m/5))

e) n mod m

-> (n - m)

7)

a)

h(x)	0	1	2	3	4	5	6	7	8	9	10	11
x	24	25	36	3	4		13	14		100	10	23

b)

h(x,i) = (x + 2i) mod m, onde i = 0, 1, 2, 3, 4…

8) Não, alguns elementos não poderão ser inseridos

9)

```
 def donoPais(novoDono: => String, tabelaPais: Hash, nomePais:
String ): Int = {

   var pais: Pais = null
   pais = hsh  busca(tabelaPais, nomePais)

   if(pais == null)
     return 0

   novoDono = nomePais

   return 1
 }
```

10)

```
 def hashFunc(chave: String): Int = {
```

```
    var i, k = 0
    while(chave.charAt(i) != '\0'){
      k += chave.charAt(i).toInt
      i += 1
    }

    return k % 307
  }
```

11)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
47	31		19		38						27			64	81	100

12)

a)

Elementos do diretório[0] : | 16 | 64 |
Elementos do diretório[1] : | 1 | 5 | 21 |
Elementos do diretório[2] : | 10 |
Elementos do diretório[3] : | 15 | 7 | 15
Elementos do diretório[4] : | 68 | 4 | 20
Elementos do diretório[5] : Referência para o diretório[1]
Elementos do diretório[6] : Referência para o diretório[2]
Elementos do diretório[7] : Referência para o diretório[3]
Elementos do diretório[8] : Referência para o diretório[0]
Elementos do diretório[9] : Referência para o diretório[1]
Elementos do diretório[10] : Referência para o diretório[2]
Elementos do diretório[11] : Referência para o diretório[3]
Elementos do diretório[12] : | 12 |
Elementos do diretório[13] : Referência para o diretório[1]
Elementos do diretório[14] : Referência para o diretório[2]
Elementos do diretório[15] : Referência para o diretório[3]

b)

Elementos do bucket no diretório[0] : | 16 | 64 |
Elementos do bucket no diretório[1] : | 17 | 1 | 5 | 21 |
Elementos do bucket no diretório[2] : | 10 |
Elementos do bucket no diretório[3] : | 15 | 7 | 15 |
Elementos do bucket no diretório[4] : | 36 | 20 | 4 | 12 |
Elementos do bucket no diretório[5] : | 69 | 21 | 5 |
Elementos do bucket no diretório[6] : Referência para o diretório[2]
Elementos do bucket no diretório[7] : Referência para o diretório[3]

c)

Elementos do bucket no diretório[0] : | 16 | 64 |
Elementos do bucket no diretório[1] : | 1 | 5 |
Elementos do bucket no diretório[2] : | 10 |
Elementos do bucket no diretório[3] : | 15 | 7 | 15 |
Elementos do bucket no diretório[4] : | 36 | 20 | 4 | 12 |
Elementos do bucket no diretório[5] : Referência para o diretório[1]
Elementos do bucket no diretório[6] : Referência para o diretório[2]
Elementos do bucket no diretório[7] : Referência para o diretório[3]

14)

1. Oito elementos: 8/2=4 estão em ordem pois não tem filhos.

| | | | | 28 | 66 | 70 | 33 |

2.Insere o 39 e troca com o 33.

| | | | 39 | 28 | 66 | 70 | 33 |

| | | | 33 | 28 | 66 | 70 | 39 |

3. Insere o 78 e troca com o 66 (menor filho)

| | | 78 | 33 | 28 | 66 | 70 | 39 |

| | | 66 | 33 | 28 | 78 | 70 | 39 |

4.Insere o 60 e troca com o 28 (menor filho)

| | 60 | 66 | 33 | 28 | 66 | 78 | 39 |

| | 28 | 66 | 33 | 60 | 78 | 70 | 39 |

5.Insere o 95 e troca com o 28.

| 95 | 28 | 66 | 33 | 60 | 78 | 70 | 39 |

| 28 | 95 | 66 | 33 | 60 | 78 | 70 | 39 |

6.Troca o 95 pelo 33

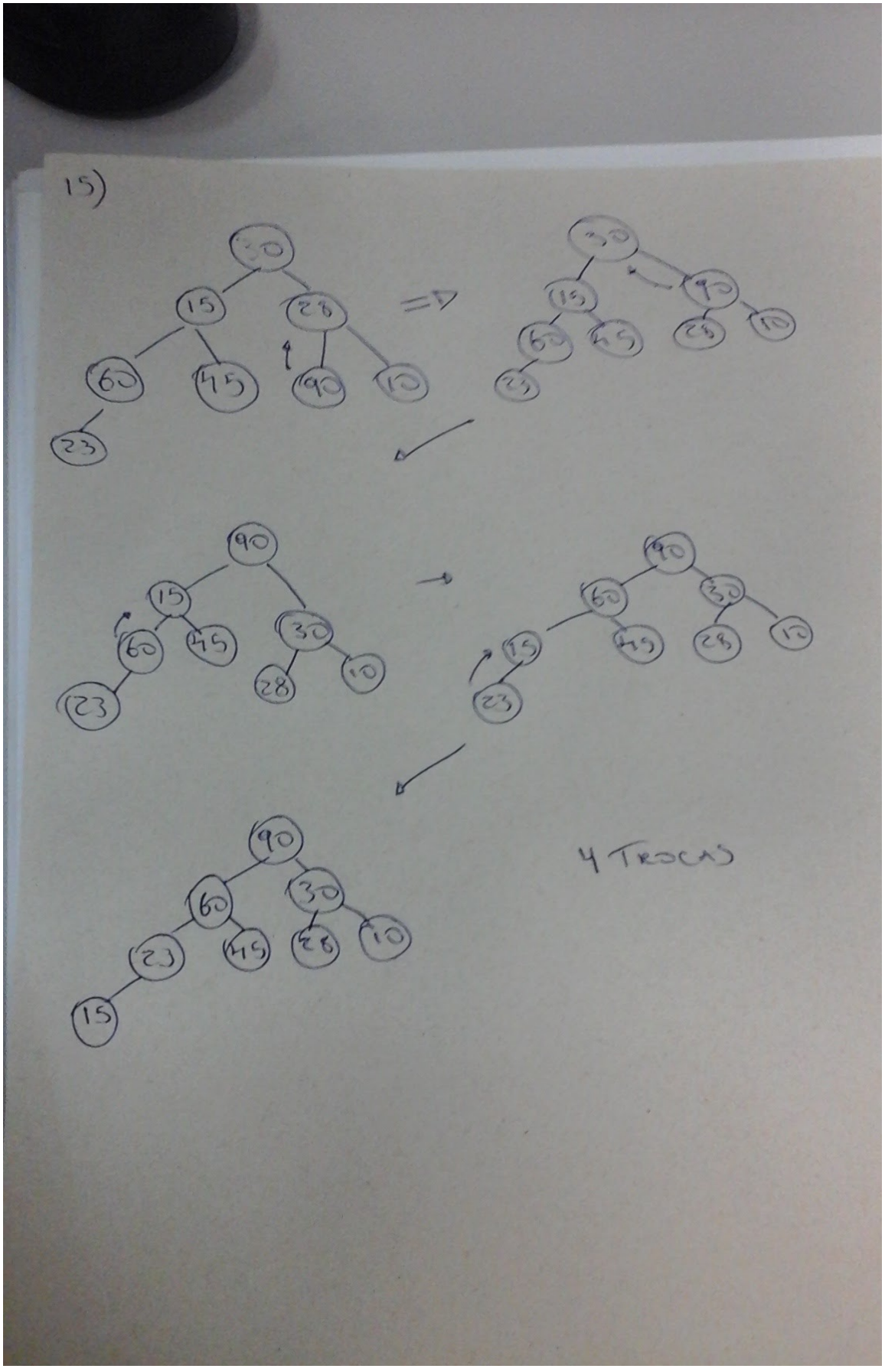
| 28 | 33 | 66 | 95 | 60 | 78 | 70 | 39 |

7.Troca o 95 pelo 39

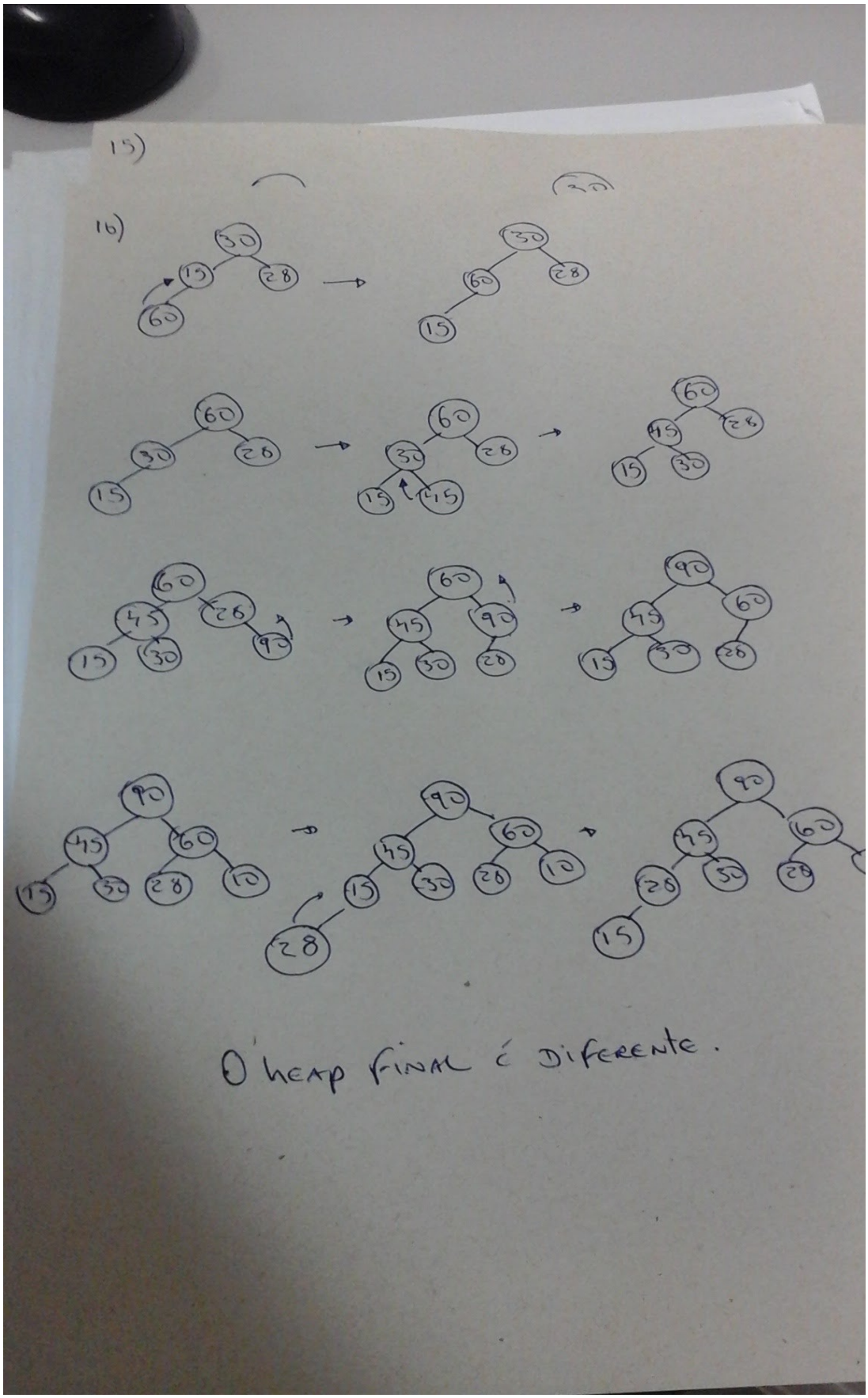
| 28 | 33 | 66 | 39 | 60 | 78 | 70 | 95 |

Resposta: 6 trocas

15)



16)



17)

```
#define PAI(x) floor((x-2) / 2)
int maxHeap(int n, Aluno** v) {
     int filho = n - 1;
     for(; filho > 0; filho--) {
          if(v[PAI(filho)]->nota < v[filho]->nota) {
               return 0;
          }
          return 1;
     }
     return 0;
}
```

18)

```
void countBiggerThan(Heap* heap, int x){
     int bit = 0;
     int count = 0;
     for(int i=0; i < heap->pos && bit==0; i++){
          if(heap->valor[i] > x)
               count++;
          else
               bit = 1;
     }
     return count;
}
```

20)

a)

índice:	1	2	3	4	5	6
valor:	2	-1	1	2	2	2

b)

índice:	1	2	3	4	5	6
valor:	-1	1	1	1	1	1

21)

índice:	1	2	3	4	5	6
valor:	2	-1	2	2	2	2