

## Algoritmos e Estruturas de Dados

Exame de Época Normal (08/02/2021)

Realizado à distância nos termos do DR20/2021

**08 de Fevereiro de 2021**

- Leia atentamente o enunciado antes de começar o exame.
- O exame está cotado para 20 valores.
- Preencha o seu nome e número em todas as folhas que entregar. Numere todas as folhas que entregar. Indique o total de folhas usadas na 1.ª folha.
- A entrega é feita tirando uma foto a cada uma das folhas, e enviando as fotos para [jovem.engenheiro@gmail.com](mailto:jovem.engenheiro@gmail.com) colocando como assunto "AED EN" e o número de aluno antes de sair da sessão Zoom. Não se consideram submissões que tenham sido feitas depois de sair de sessão.
- As submissões só serão consideradas válidas se o aluno estiver ligado na sessão ZOOM disponibilizada para o efeito.

---

1. (0.5) Indique a aproximação tilde mais correcta para a seguinte expressão:

$$\frac{\lg(2n)}{\lg(n)}$$

- a)  $\frac{1}{\lg(n)}$
- b) 2
- c)  $\lg(n)$
- d) 1

R:

$$\frac{\lg(2n)}{\lg(n)} = \frac{\lg(2) + \lg(n)}{\lg(n)} = \frac{\lg(2)}{\lg(n)} + 1 \sim 1$$

2. (0.5) Considere o seguinte algoritmo recursivo:

```
public void xpto(int[] a, int n)
{
    if(n < 1) return;

    for(int i = 0; i < n-1; i++)
    {
        func(a[i]);
    }
    xpto(a, n-1);
}
```

Admita que a função func(x) está definida e tem complexidade O(1). Indique a complexidade da função xpto():

- a) O(n lg n)
- b) O(n<sup>2</sup>)**
- c) O(n<sup>3</sup>)
- d) O(n)

R: Como xpto é uma função recursiva que vai subtraindo 1 de cada vez, para um dado n, a função func é invocada (n-1) + (n-2) + (n-3) + .. + 0 vezes. O que corresponde à soma de uma progressão aritmética, cuja soma dos n primeiros termos é dada por  $\frac{n(0+n-1)}{2} = \frac{n^2-1}{2}$ , logo se func(x) tem complexidade O(1) a complexidade temporal de xpto é dada por T(n) = O(n<sup>2</sup>)\*O(1) = O(n<sup>2</sup>).

Uma forma mais simples de chegar a este resultado seria reparar que a função xpto vai ser executada n vezes, e que em média, em cada invocação de xpto, o número de chamadas de func corresponde a  $\frac{0+n-1}{2}$

3. (0.5) Qual dos arrays seguintes não pode ser um max-heap?

- a) [null, 9, 3, 8, 1, 2, 8, 7]
- b) [null, 9, 8, 3, 7, 6, 1, 2]
- c) [null, 20, 15, 13, 12, 11, 10, 9]
- d) [null, 20, 14, 15, 13, 12, 16, 11]**

R: A melhor forma de responder a esta questão é desenhar um max-heap para cada array, e verificar quais o que verificam a propriedade dos max-heap: para todo nó n, n >= que os seus dois filhos.

É muito fácil representar um array num heap, pois sabemos que um heap é uma árvore binária, com 1, 2, 4, 8, ... nós em cada nível de profundidade. Por isso é só ir criando a árvore usando os elementos pela ordem que aparecem da esquerda para a direita (ignoramos a posição 0, pois assumimos que o heap começa na posição 1 do array). Quando acabamos de preencher um nível, passamos para o nível de baixo. Para tornar a resolução mais rápida, não é preciso colocarmos os traços e bolas, e no computador é mais fácil fazer isto com tabelas. A resposta seria a alínea d) que não respeita as propriedades de um heap (a amarelo).

a)			
9			
3		8	
1	2	8	7

b)			
9			
8		3	
7	6	1	2

c)			
20			
15		13	
12	11	10	9

d)			
20			
14		15	
13	12	16	11

4. (0.5) Considere o algoritmo de ordenação Shellsort usando a sequência de Knuth aplicado ao seguinte array [4, 2, 5, 0, 1, 7, 3, 8, 9, 6].

Indique qual das seguintes sequências corresponde a uma sequência válida e pela ordem correcta do estado do array após operações de h-sorting. Assuma que h inicial = 4.

a.

[0, 1, 5, 3, 2, 7, 4, 8, 9, 6]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

b.

[1, 2, 3, 0, 4, 6, 5, 8, 9, 7]

[1, 0, 3, 2, 4, 6, 5, 7, 9, 8]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

c.

[0, 1, 5, 3, 2, 7, 4, 8, 9, 6]

[0, 1, 2, 3, 4, 6, 5, 7, 9, 8]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

d.











[1, 2, 3, 0, 4, 6, 5, 8, 9, 7]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

R: A melhor forma de responder a este tipo de questões (mesmo sendo de escolha múltipla) é executar os passos do algoritmo. Usando a sequência de Knuth o h a usar num nível seguinte é dado pela divisão inteira do h anterior por 3. Como é dito que o h inicial é 4, só vamos ter 2 h's: h=4, e h=4/3 = 1. O Algoritmo termina **depois** de ter sido 1-sorted.

Começemos com 4-sorted. Vamos fazer sort do array 4 a 4 elementos, vamos usar cores para melhor ilustrar:

Array inicial:

[, , , , , , , , , ]. Ou seja, temos de ordenar as casas verdes entre si, as casas azuis entre si, etc.

Array depois de 4-sorted terminado:

[, , , , , , , , , ].

A seguir a 4-sorted, temos de fazer 1-sorted. 1-sorted corresponde a ordenarmos todos os elementos entre si, ou seja um insertion sort normal.

Array depois de 1-sorted terminado:

[0,1,2,3,4,5,6,7,8,9]

Portanto a resposta correcta é a d). A alínea b) começa da mesma forma, mas a seguir a fazer 4-sorted foi feito um 2-sorted, e finalmente 1-sorted. Está errada pois a sequência de Knuth não vai fazer 2-sorted.

**5. (2.0)** Escreva um método estático chamado xpto2 em linguagem Java, que dados dois argumentos, um array de inteiros, e um inteiro n, retorna a soma dos elementos do array que sejam divisores de n (ou seja, o resto da divisão inteira por n seja 0).

Exemplos:

xpto2([2, 4, 6, 3, 5, 9],3) = 6 + 3 + 9 = 18

**R:** Este exercício é trivial para quem tenha feito algum dos projectos da cadeira ao longo do semestre.

```
public static int xpto2(int[] a, int n) {
    int result=0;
    for(int i = 0; i < a.length ; i++) {
        if(a[i] % n == 0) result += a[i];
    }
    return result;
}
```

**6. (2.0)** Considere uma pilha (stack) e uma fila (queue) inicialmente vazias, sobre as quais é executada a seguinte sequência de instruções:

```
StackList<Integer> s = new StackList<Integer>();
QueueList<Integer> q = new QueueList<Integer>();

for(int i = 1; i < 7; i++)
{
    s.push(i);
    q.enqueue(i);
}

while(!q.isEmpty()) s.push(q.dequeue());

while(!s.isEmpty())
{
    System.out.print(s.pop() + ",");
}
```

Indique a sequência impressa no ecrã quando o código é executado.

**R:** Quase todos acertaram nesta pergunta. Aqui a melhor forma de responder é desenhar a fila e a pilha, e ir vendo como é que são preenchidas e vazadas. No computador, vou usar 2 tabelas.

6
5
4
3
2
1

1	2	3	4	5	6		
---	---	---	---	---	---	--	--

6
5
4
3
2
1
6
5
4
3
2
1

No final do 1.º for, este é o resultado. O 1.º while vai tirar os elementos da fila e pô-los na pilha, resultando na pilha da direita.

O último while vai retirar os elementos da pilha direita (a partir do topo), imprimindo-os: 6,5,4,3,2,1,6,5,4,3,2,1

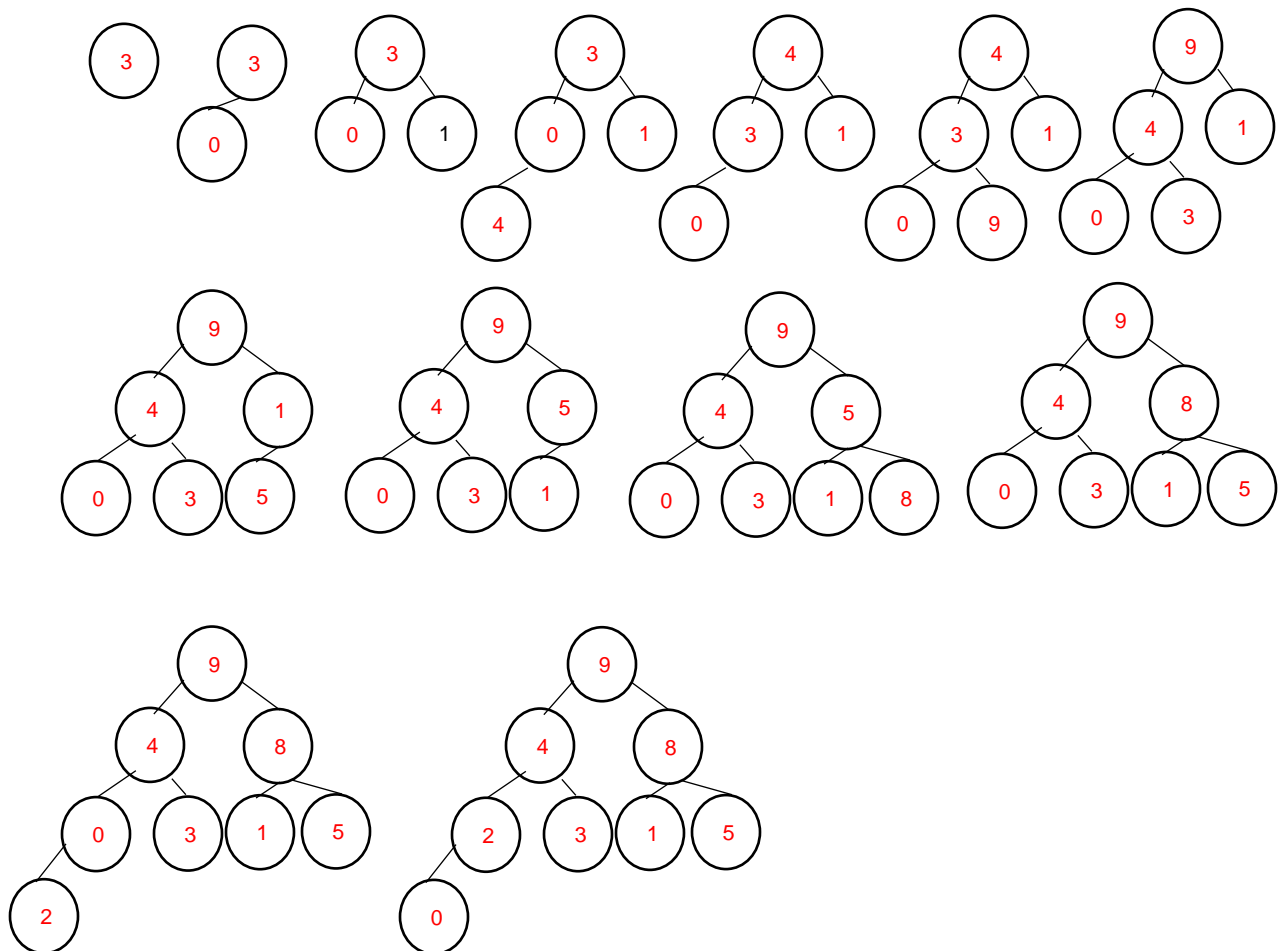
7. (2.0) Suponha que está a fazer uma aplicação onde existe um numero enorme de operações de inserção, mas apenas um número muito limitado de operações de remoção do máximo. Indique, justificando, que implementação de uma fila prioritária será mais apropriada para esta situação: heap, array não ordenado, ou array ordenado?

**R:** Como o número de operações de inserção é muito elevado face ao número de operações de remoção, a operação mais importante em termos de eficiência é a inserção. Das estruturas indicadas o array não ordenado é aquela que tem a operação de inserção mais eficiente com complexidade temporal amortizada  $O(1)$ .

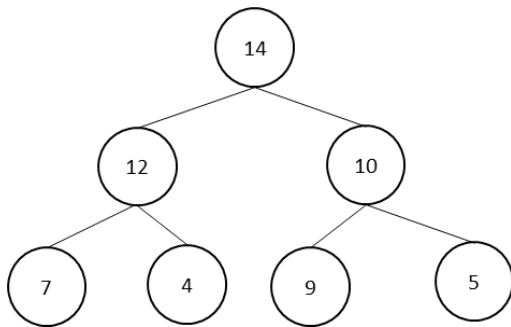
8. (3.0) Resolva os seguintes exercícios sobre montes (heaps).

a) Desenhe a representação em árvore do max-heap obtido quando se insere os seguintes elementos (da esquerda para a direita) num heap inicialmente vazio: [3, 0, 1, 4, 9, 5, 8, 2]

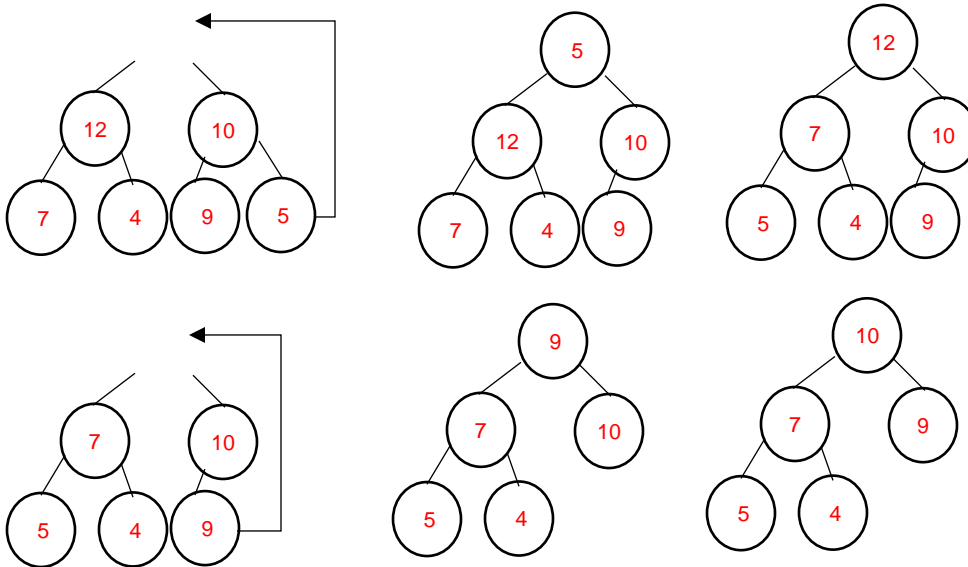
**R:** O importante aqui é ter em conta que é inserido um elemento de cada vez no heap, e que o novo elemento é sempre inserido o mais em baixo à direita possível. Depois de introduzido, faz-se heapify bottom-up.



b) Considere o seguinte max-heap. Desenhe a representação em árvore do max-heap obtido depois de se executar as seguintes operações sobre o heap: removeMax(), removeMax() (ou seja, remover duas vezes o maior elemento).



R: Duas operações de remove. Em cada uma delas, removemos o topo do heap, colocamos o nó mais em baixo à direita como sendo o novo topo, e fazemos heapify top-down.



9. (3.0) Considere o método partition usado no algoritmo de ordenação quicksort, com a assinatura `int partition(Comparable[] a, int low, int high)`. Suponha que o método partition é invocado com os seguintes argumentos:

`a=[5, 7, 2, 10, 8, 4, 9, 3]`, `low = 0`, `high=7`

Indique qual o conteúdo do array após a execução do método partition, assumindo que o pivô é escolhido como sendo o elemento da posição low. Indique também qual a posição do pivô, retornada pelo método.

R:

`low=0`, `high=7`,

pivô = 5, `i=1`, `j=7`

`a=[5, 2, 10, 8, 4, 9, 3]` – para melhor visualizarmos, i a verde, j a azul.

[é mais fácil se indicarmos para cada iteração onde é que o i e o j param (i e j são índices)]

1ª iteração: `i=1`, `j=7` [pois  $7 \geq 5$ , e  $3 \leq 5$ ]

`exchange(1,7)`

`a=[5,3,2,10,8,4,9,7]`

2ª iteração: `i=3`, `j=5` [pois  $10 \geq 5$ , e  $4 \leq 5$ ]

`exchange(3,5)`

`a=[5,3,2,4,10,9,7]`

3ª iteração: `i=4`, `j=3` [pois  $8 \geq 5$ ,  $4 \leq 5$ ]

i e j cruzaram-se, não há trocas na 3.ª iteração, acabaram-se as iterações

No final das iterações, faz-se exchange da posição do pivô pela posição j

`exchange(0,3)`

`a=[4,3,2,5,8,10,9,7]`

`j=3` (posição final do pivô retornada);

10. (3.0) Considere uma árvore rubro-negra (red-black) inicialmente vazia, onde ligações vermelhas são representadas como linhas a tracejado e ligações negras como linhas contínuas. Desenhe a representação da árvore, depois de inseridos os elementos no array indicado abaixo, da esquerda para a direita.

[1, 10, 3, 9, 8, 2, 5]

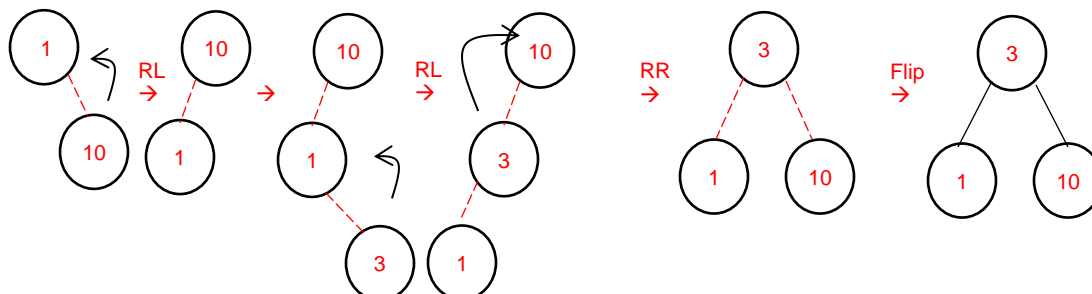
R: A maior parte dos alunos não fez correctamente esta pergunta, porque por exemplo esqueceram-se da operação de flip colors. Ao longo do exercício vamos fazer 3 tipos de operação, e vamos apontar cada vez que fizermos cada uma delas:

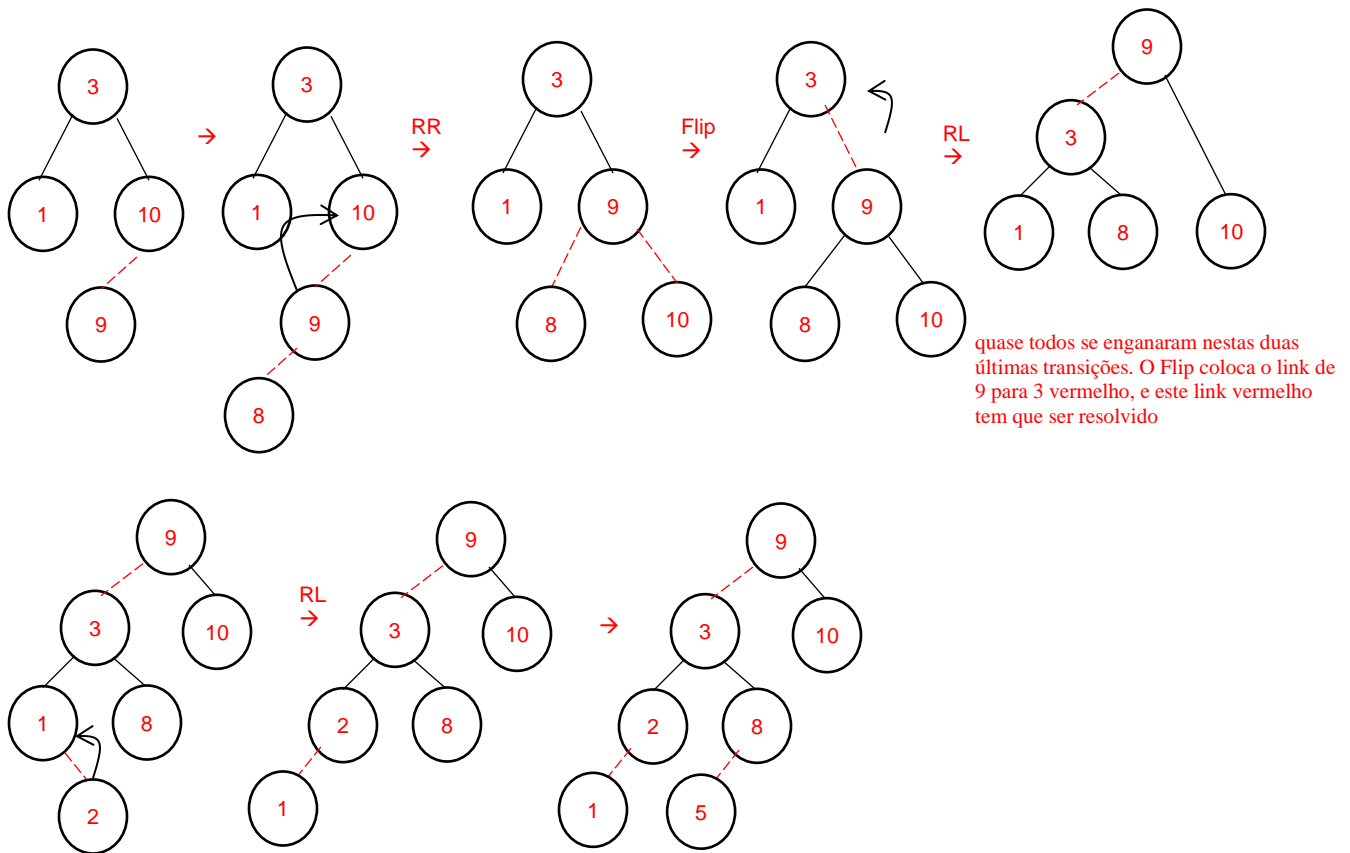
Única ligação vermelha à direita – RL - rotate left

Duas ligações vermelhas esquerdas seguidas –RR - rotate right

Ligação vermelha à esquerda e direita – Flip - flip colors

Quando um novo nó é colocado numa árvore red-black a cor para o seu pai é sempre vermelha (excepto a raiz que tem sempre cor negra).





**11. (3.0)** Considere uma tabela de dispersão de dimensão  $M = 9$ , com resolução de colisões por dupla dispersão, e funções de dispersão  $h_1(k) = k \bmod M$  e  $h_2(k) = 7 - k \bmod 7$ .

a) Transcreva a seguinte tabela para a sua folha de exame. Sabendo que inicialmente a tabela se encontra vazia, indique quais os elementos presentes em cada posição do array de chaves e de valores, após a inserção da sequência de pares <chave, valor> abaixo. Indique uma posição vazia com —.

<3, “a”>, <12, “b”>, <21, “c”>, <1, “d”>, <4, “e”>, <13, “f”>, <5, “g”>

i	0	1	2	3	4	5	6	7	8
chaves	5	21		3	4	12	13	1	
valores	g	c		a	e	b	f	d	

R: Neste exercício é boa ideia escrevermos explicitamente os valores de  $h_1$  e  $h_2$  para cada chave

$h_1(3) = 3 \bmod 9 = 3$ , a posição 3 está livre, colocar

$h_1(12) = 12 \bmod 9 = 3$ ,  $h_2(12) = 7 - 12 \bmod 7 = 7 - 5 = 2$

a posição 3 está ocupada, vamos experimentar posição  $3 + 2 (h_1(k) + h_2(k))$ , está livre

$h_1(21) = 21 \bmod 9 = 3$ ,  $h_2(21) = 7 - 21 \bmod 7 = 7 - 0 = 7$

a posição 3 está ocupada, experimentamos  $(3 + 7) \bmod 9$  (o índice dá a volta) = 1, está livre

$h_1(1) = 1 \bmod 9 = 1$ ,  $h_2(1) = 7 - 1 \bmod 7 = 7 - 1 = 6$

a posição 1 está ocupada, experimentamos  $1 + 6$ , está livre

$h_1(4) = 4 \bmod 9 = 4$ , está livre

$h_1(13) = 13 \bmod 9 = 4$ ,  $h_2(13) = 7 - 13 \bmod 7 = 7 - 6 = 1$

posição 4 ocupada, experimentamos  $4 + 1$  que está ocupada, experimentamos  $5 + 1$  que está livre



$$h1(5) = 5\%9 = 5, h2(5) = 7 - 5\%7 = 7 - 5 = 2$$

posição 5 ocupada, tentar  $5+2$  que está ocupada, tentar  $(7 + 2)\%9 = 0$  que está livre

b) (1.0) Considere agora que o conteúdo da tabela de dispersão é a seguinte:

i	0	1	2	3	4	5	6	7	8
chaves	—	28	29	2	3	14	—	—	—
valores	—	“a”	“b”	“c”	“d”	—	—	—	—

Indique o que muda nos *arrays* de chaves e valores, após a remoção das chaves 2, 3 e 14. Apenas precisa de assinalar o que é diferente. Indique uma posição vazia ou nula com —.

i	0	1	2	3	4	5	6	7	8
chaves				2	3				
valores				—	—				

R: quando se usa dupla dispersão não se pode usar remoção imediata, tem que se usar remoção preguiçosa. Ou seja as chaves removidas são apenas assinaladas com valor nulo. A chave 14 já se encontrava marcada como removida, por isso não é preciso mudar nada na chave 14.