

Gabarito Segunda Avaliação a Distância – Estrutura de Dados – 2º Semestre de 2018
Todas as questões valem 1.0, exceto as questões 1 e 6 (que vale 2.0 cada)

1. Responda os seguintes itens:

(a) Determinar os valores dos números mínimo e máximo de páginas que uma árvore B de ordem d e altura h pode armazenar.

R = Os números, máximo (P_{max}) e mínimo (P_{min}), de páginas são dados pelas seguintes expressões:

$$P_{max} = \frac{(2d+1)^h - 1}{2d} \qquad P_{min} = 1 + \frac{2}{d} [(d+1)^{h-1} - 1]$$

(b) Determinar os valores dos números mínimo e máximo de chaves que uma árvore B de ordem d e altura h pode armazenar.

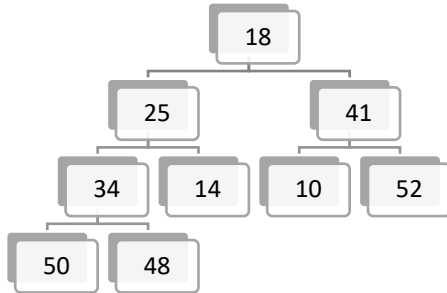
R = Os números, máximo (C_{max}) e mínimo (C_{min}), de chaves são dados pelas seguintes expressões:

$$C_{max} = (2d+1)^h - 1 \qquad C_{min} = 2(d+1)^{h-1} - 1$$

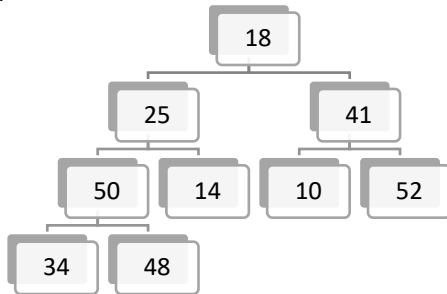
2. Determine o heap obtido pela aplicação do algoritmo de construção (de tempo linear) às seguintes prioridades: 18,25,41,34,14,10,52,50,48.

R = Acompanhe a construção do heap em tempo linear (isto é, complexidade de $O(n)$), baseando-se na lista de prioridades apresentada no enunciado da questão.

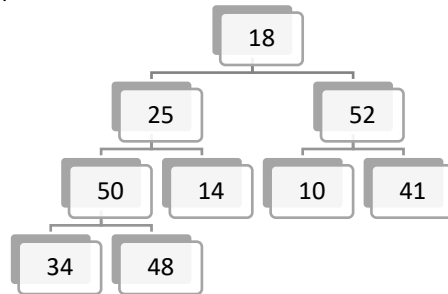
Condição Inicial:



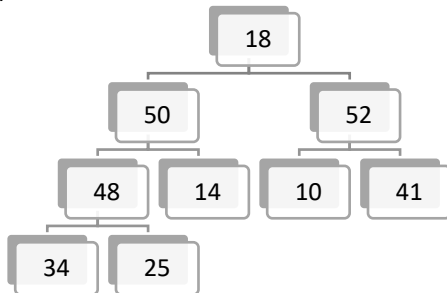
1) Descer 34:



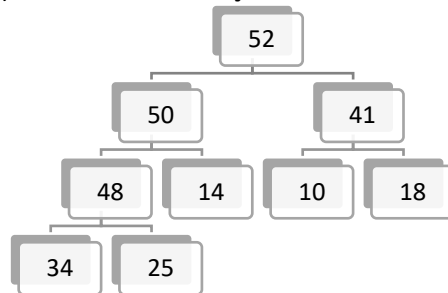
2) Descer 41:



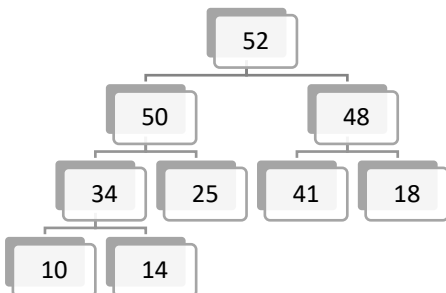
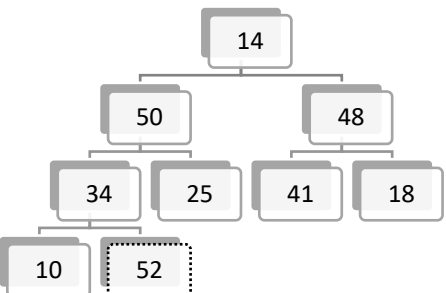
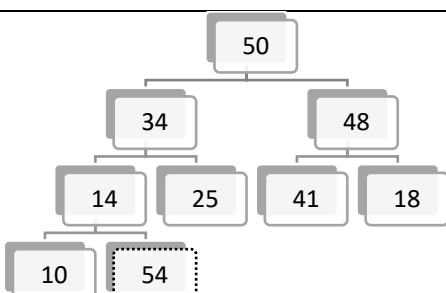
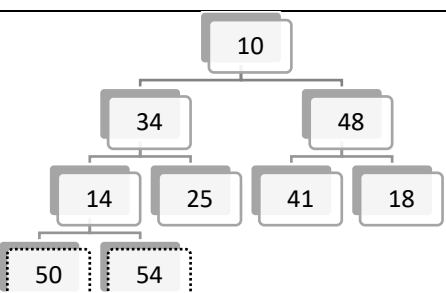
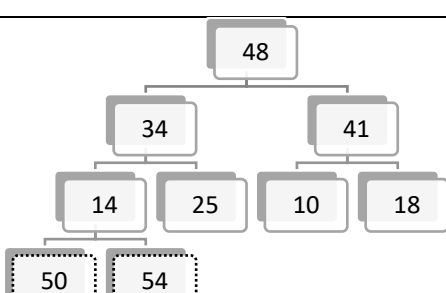
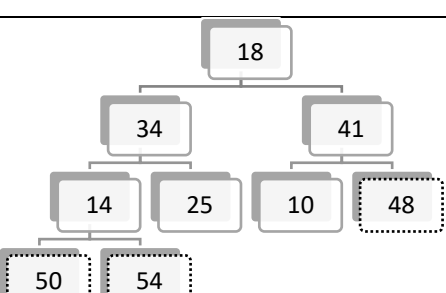
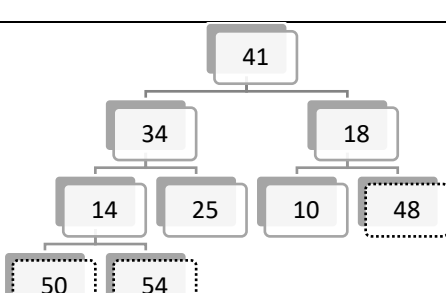
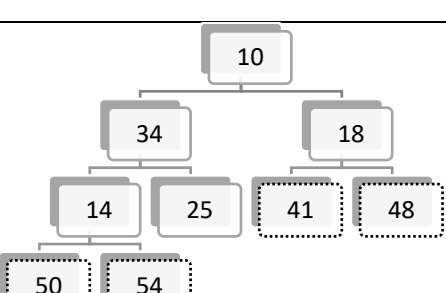
3) Descer 25:

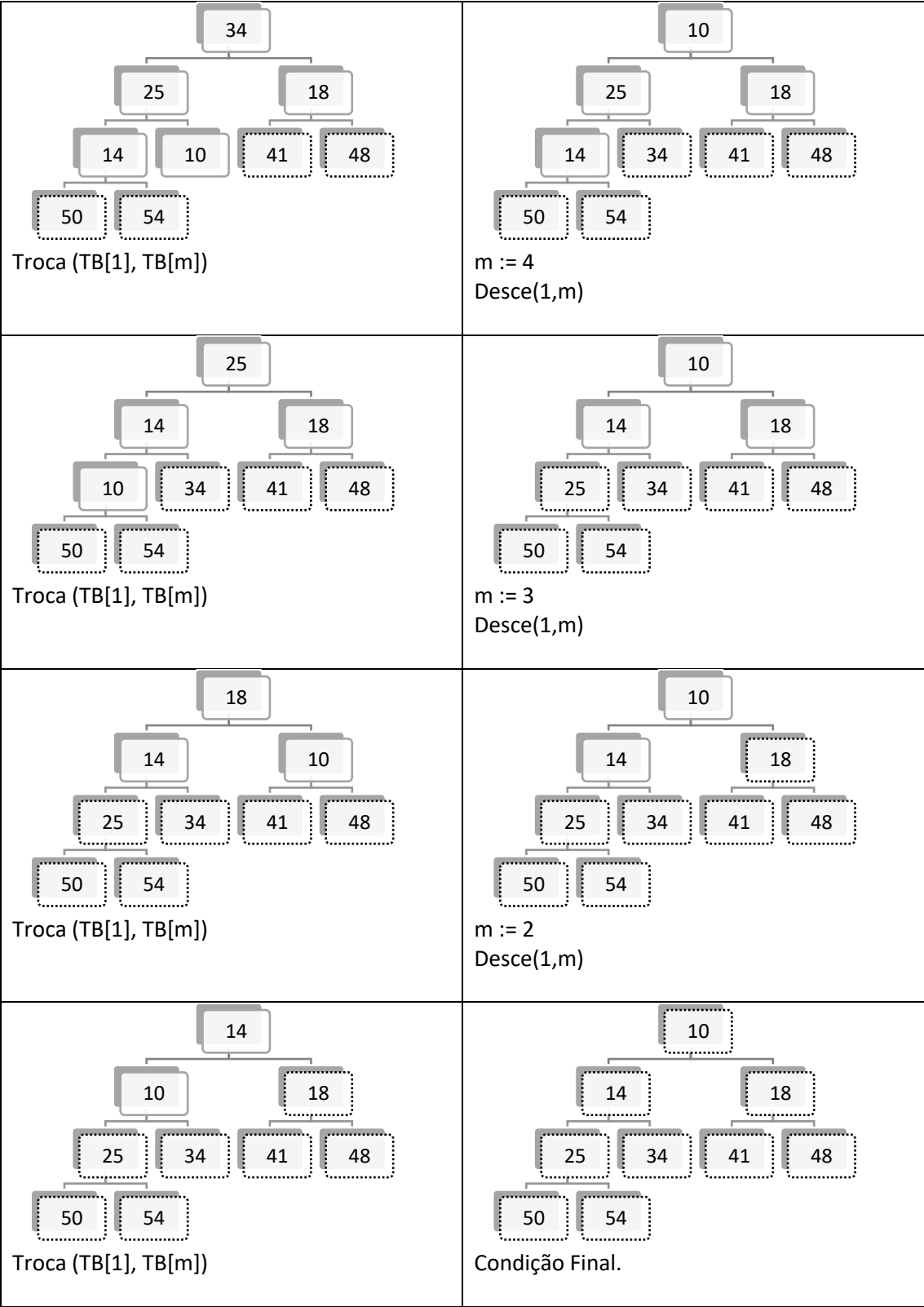


4) Descer 18 – Condição Final:



3. Execute o método de ordenação por heap ("heapsort"), aplicando-o ao seguinte heap: 52, 50, 48, 34, 25, 41, 18, 10, 14. (Desenhe as configurações sucessivas da árvore durante o processo de ordenação.)

<p>Condição Inicial:</p>  <p>$m := 9$ Troca (TB[1], TB[m])</p>	 <p>$m := 8$ Desce(1,m)</p>
 <p>Troca (TB[1], TB[m])</p>	 <p>$m := 7$ Desce(1,m)</p>
 <p>Troca (TB[1], TB[m])</p>	 <p>$m := 6$ Desce(1,m)</p>
 <p>Troca (TB[1], TB[m])</p>	 <p>$m := 5$ Desce(1,m)</p>



4. Descrever um algoritmo de inserção em uma tabela de dispersão por encadeamento aberto, supondo a não-existência de remoções. Deve-se verificar se o elemento a ser inserido já se encontra armazenado; neste caso, a inserção não deve ser feita.

R = Suponha que x é a chave a ser incluída na tabela $T[0 \dots m-1]$, e tamanho m .

```
end := h(x) mod m
pont := T[end]    % ponteiros para percorrer a lista
achou := falso
enquanto pont ≠ λ faça
    ant := pont
    se pont↑.chave = x então
        achou := verdadeiro
        pont := λ    % x já está na lista
    senão
        pont := pont↑.prox
se achou = falso então
    ocupar(pt)
    pt↑.chave := x
    pt↑.prox := λ
    se T[end] = λ então
        T[end] := pt
    senão
        ant↑.prox := pt
```

5. Descreva um algoritmo que percorra os nós de uma árvore binária na ordem de seus níveis. Sugestão: use uma fila.

R = Considere F como sendo a fila utilizada e f como a variável que indica o final da fila.

```
se  $ptrai z \neq \lambda$  então
    insere( $F$ ,  $ptrai z$ )
enquanto  $f \neq 0$  faça
     $pt := \text{remove}(F)$     % remove o 1º elemento de  $F$ 
    visita( $pt$ )
    se  $(pt \uparrow .esq \neq \lambda)$  então
        insere( $F$ ,  $pt \uparrow .esq$ )
    se  $(pt \uparrow .dir \neq \lambda)$  então
        insere( $F$ ,  $pt \uparrow .dir$ )
```

6. Descreva um algoritmo que imprima os nós de uma árvore binária na ordem de suas alturas. Isto é, primeiro são impressos os nós de altura 1, depois os de altura 2, e assim por diante. (A ordem de impressão dos nós de mesma altura pode ser arbitrária.)

R = Sendo a altura denotada pela variável h , o algoritmo abaixo irá percorrer a árvores $h-1$ vezes, imprimindo e removendo suas folhas. A condição de parada é dada quando a raiz for a folha da árvore (ou seja, a árvore só possuir o nó raiz). O procedimento *busca-folhas*, realiza a procura de uma folha considerando a variável *lado*. Essa variável pode assumir os valores 0 ou 1, indicando, respectivamente, o filho à esquerda ou à direita do seu pai.

enquanto $(ptrai\uparrow.esq \neq \lambda)$ ou $(ptrai\uparrow.dir \neq \lambda)$ faça

se $(ptrai\uparrow.esq \neq \lambda)$ então

busca-folhas($ptrai\uparrow.esq, ptrai$, 0)

se $(ptrai\uparrow.dir \neq \lambda)$ então

busca-folhas($ptrai\uparrow.dir, ptrai$, 1)

imprimir($ptrai\uparrow.info$)

procedimento busca-folhas($no, pai, lado$)

se $(no\uparrow.esq = \lambda)$ e $(no\uparrow.dir = \lambda)$ então

se $(lado = 0)$ então $pai\uparrow.esq = \lambda$

senão $pai\uparrow.dir = \lambda$

imprimir(no)

desocupar(no)

senão

se $(no\uparrow.esq \neq \lambda)$ então busca-folhas($no\uparrow.esq, no$, 0)

se $(no\uparrow.dir \neq \lambda)$ então busca-folhas($no\uparrow.dir, no$, 1)

7. V ou F? Toda árvore binária de busca completa é uma árvore AVL. (Justifique).

R = Uma árvore completa é uma árvore cujas folhas estão dispostas exclusivamente nos dois últimos níveis. Partindo deste princípio podemos acusar a afirmação desta questão como sendo uma afirmação **verdadeira**, uma vez que uma árvore AVL também pode possuir as suas folhas dispostas nestes dois últimos níveis. Isso acontece, pois, este tipo de árvore busca minimizar o número de comparações necessárias na procura de uma determinada chave. Em resumo, uma árvore binária completa naturalmente estará balanceada, ou seja, a diferença entre as sub-árvores da esquerda e direita de um nó sempre será menor ou igual a 1.

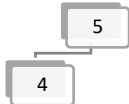
8. Mostre o passo-a-passo da construção da árvore AVL a partir da inserção das seguintes chaves (nesta ordem): 5, 4, 3, 10, 8 e 15.

R = Acompanhe a construção de árvore AVL abaixo.

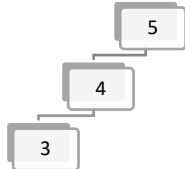
Inserção chave 5:



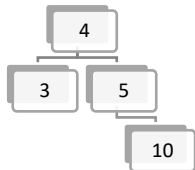
Inserção chave 4:



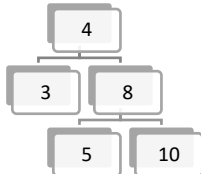
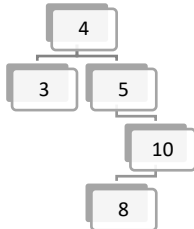
Inserção chave 3:



Inserção chave 10:



Inserção chave 8:



Inserção chave 15:

