

Gabarito da Segunda Avaliação à Distância - Segundo Semestre de 2011 - Estrutura de Dados e Algoritmos

1) (2,0) Determinar uma árvore binária de busca ótima supondo as seguintes frequências de acesso: $f_1 = 1, f_2 = 3, f_3 = 2, f_4 = 1, f_5 = 2, f'_0 = 2, f'_1 = 2, f'_2 = 1, f'_3 = 0, f'_4 = 1, f'_5 = 2$. Determinar as matrizes F, c, k associadas ao algoritmo e desenhar a árvore ótima obtida.

Resposta: As matrizes do algoritmo de cálculo da árvore ótima são:

Matriz dos valores $F[i, j]$:

2	5	9	11	13	17
-	2	6	8	10	14
-	-	1	3	5	9
-	-	-	0	2	6
-	-	-	-	1	5
-	-	-	-	-	2

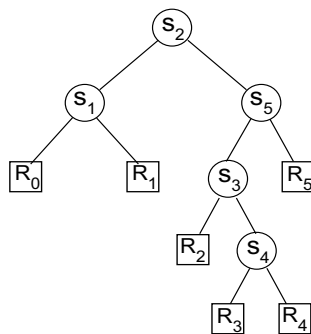
Matriz dos custos $c[i, j]$:

0	5	14	19	25	38
-	0	6	11	17	28
-	-	0	3	7	16
-	-	-	0	2	8
-	-	-	-	0	5
-	-	-	-	-	0

Matriz dos valores minimizantes k :

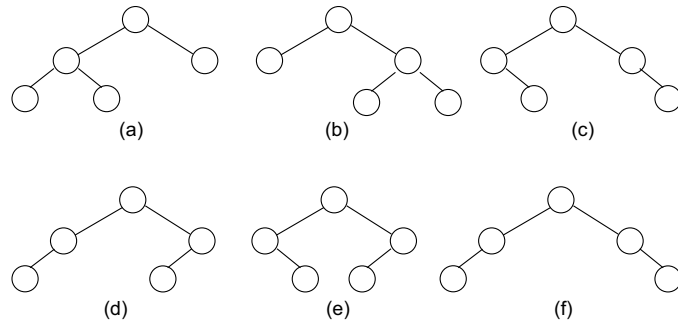
-	1	2	2	2	2
-	-	2	2	2	3
-	-	-	3	3	5
-	-	-	-	4	5
-	-	-	-	-	5
-	-	-	-	-	-

Da última matriz acima, temos a seguinte árvore ótima, de custo 38:

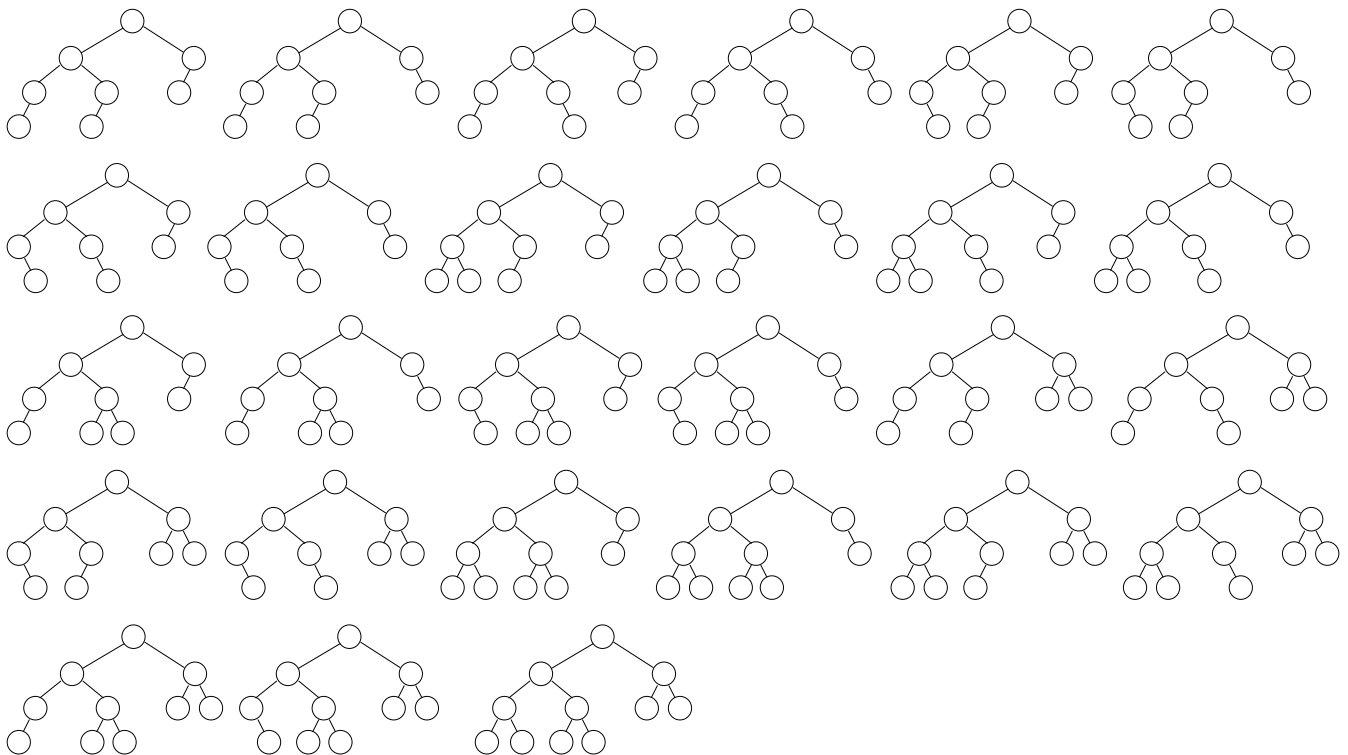


2) (1,5) Desenhe todos os formatos possíveis de uma árvore AVL com cinco nós internos.

Resposta: Se removermos todas as folhas de uma árvore AVL, temos que a árvore resultante também é uma árvore AVL. Assim, temos as seguintes configurações possíveis para os nós internos:

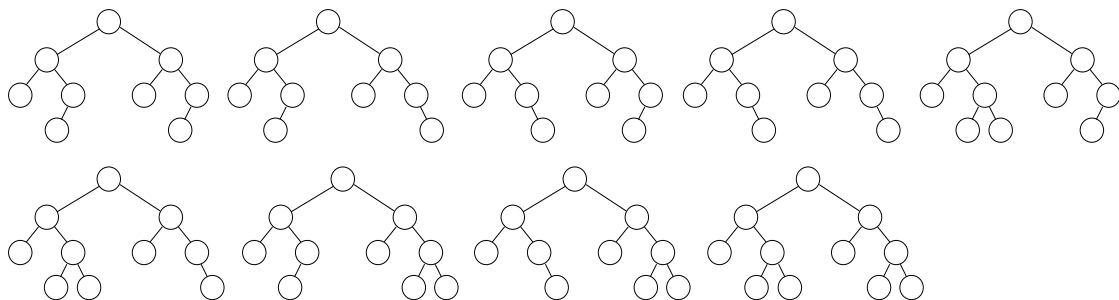


A partir da configuração (a), obtemos 27 árvores AVL, ilustradas a seguir:

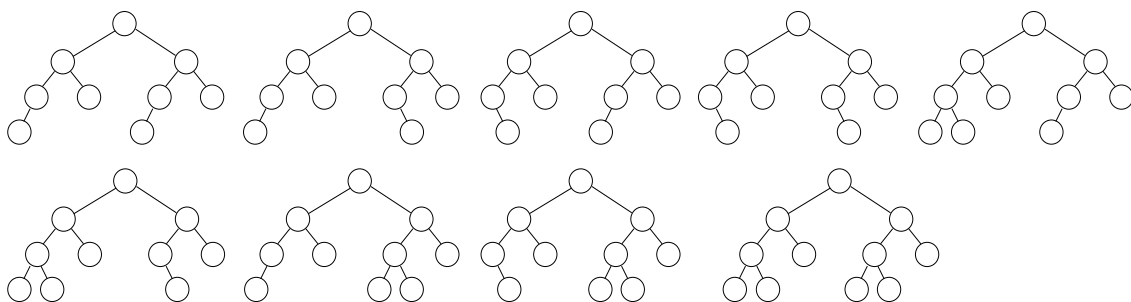


As árvores geradas a partir da configuração (b) são similares às ilustradas para (a), bastando trocar os filhos direito e esquerdo da raiz de posição.

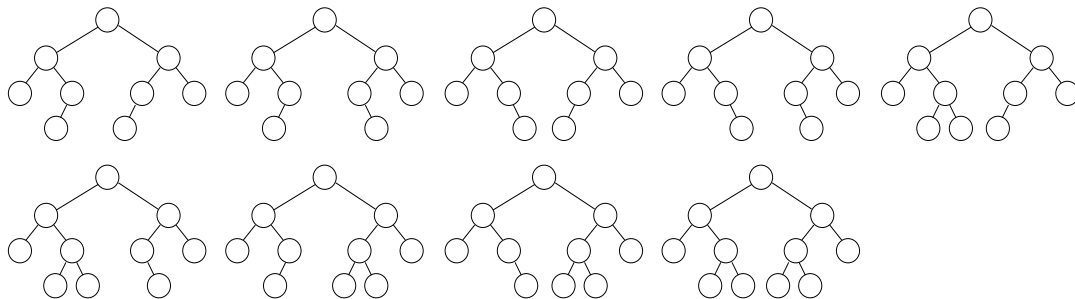
A partir da configuração (c), obtemos 9 árvores AVL, ilustradas a seguir:



A partir da configuração (d), obtemos 9 árvores AVL, ilustradas a seguir:



A partir da configuração (e), obtemos 9 árvores AVL, ilustradas a seguir:



As árvores geradas a partir da configuração (f) são similares às ilustradas para (e), bastando trocar os filhos direito e esquerdo da raiz de posição.

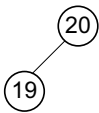
Totalizando as árvores geradas pelas 6 configurações possíveis de nós internos, temos 90 árvores AVL com 5 nós internos.

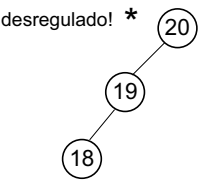
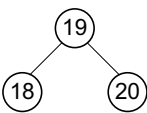
3) (1,5) Desenhe a árvore AVL obtida pela sequência de inserções das chaves 20, 19, 18, 16, 15, 17, 2, 6, nesta ordem. Mostre com desenhos as operações de rotação necessárias em cada inclusão.

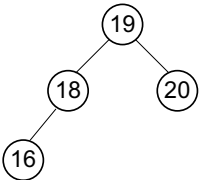
Resposta:

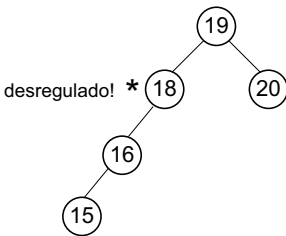
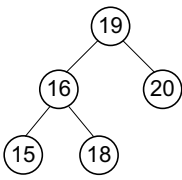
Início: árvore vazia

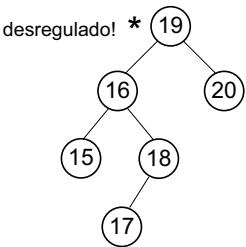
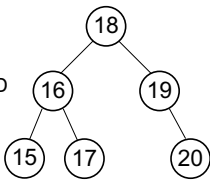
Inserir 20: (20)

Inserir 19: 

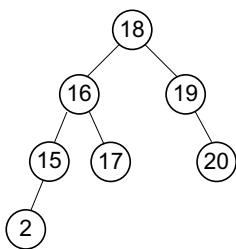
Inserir 18:  Efetuar RD 

Inserir 16: 

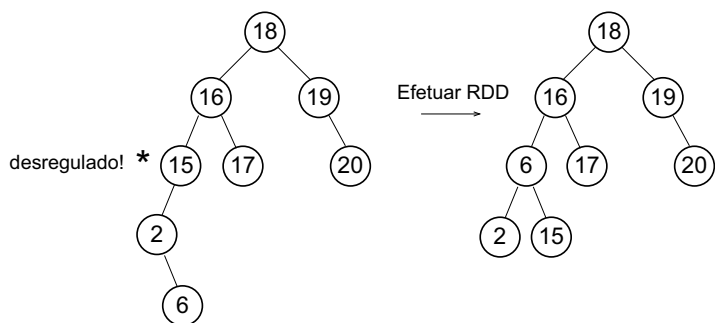
Inserir 15:  Efetuar RD 

Inserir 17:  Efetuar RDD 

Inserir 2:

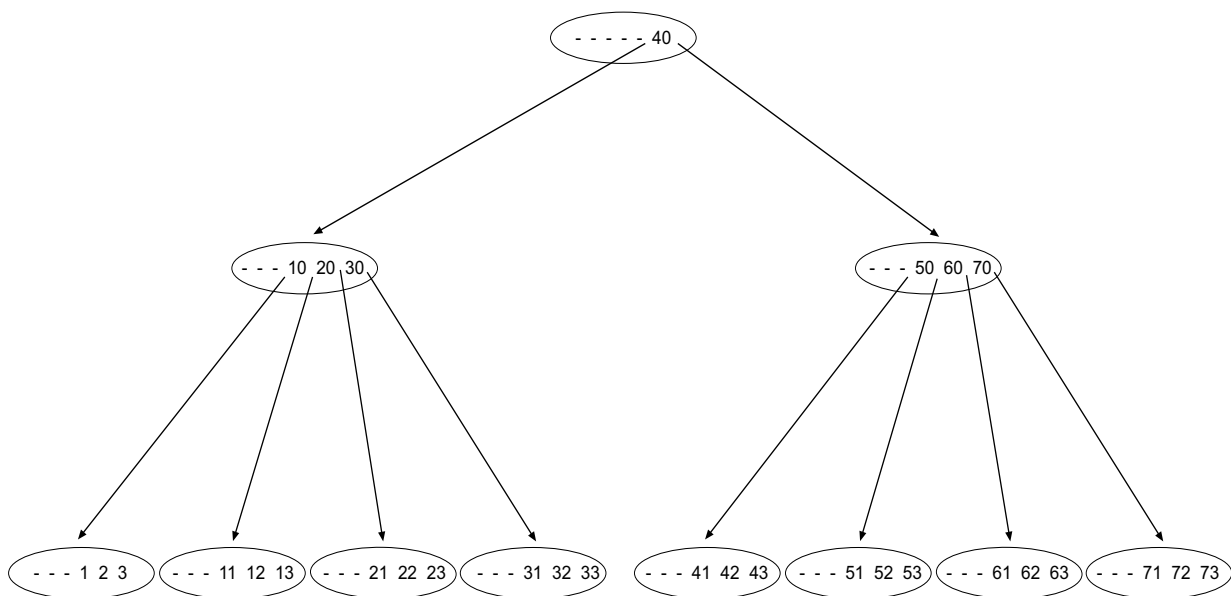


Inserir 6:

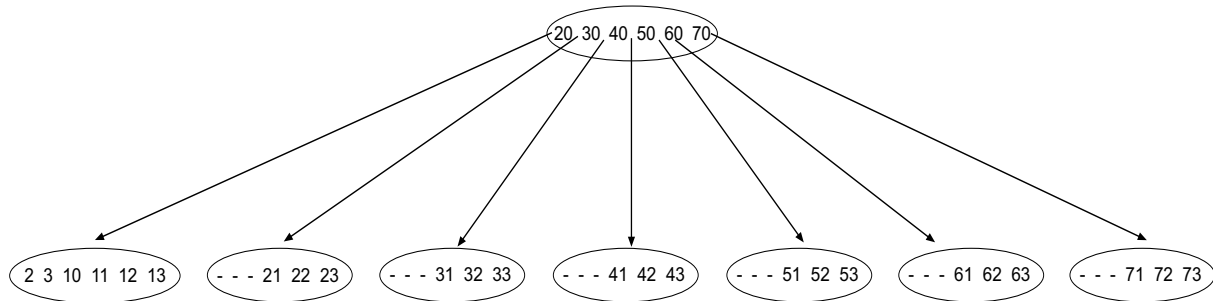


4) (1,5) Desenhar uma árvore B de ordem 3 e altura 3 contendo um número mínimo de chaves. A seguir, escolha uma chave para ser removida, e mostre a árvore resultante desta remoção.

Resposta:



Removendo a chave 1:



5) (2,0) Escreva versões *não recursivas* dos algoritmos de *subida* e *descida* de uma chave em uma lista de prioridades (*heap*) onde a raiz é a chave cuja prioridade tem valor máximo. Compare-os com as versões recursivas.

Resposta: Seja *fim* uma variável booleana que indica se a chave do nó *i* é menor que a de seu pai (procedimento subir) ou maior que de seus filhos (procedimento descer). Ambos os algoritmos não recursivos apresentados possuem complexidade $O(\log n)$ pois, assim como os recursivos, dependem apenas da altura da árvore. No entanto, os algoritmos não recursivos tendem a ser executados mais rapidamente, por não terem o custo extra das chamadas recursivas.

procedimento *subir*(*i*)

```

  j := ⌊i/2⌋
  fim := F
  enquanto j ≥ 1 e fim = F faça
    se T[i].chave > T[j].chave então
      T[i] ↔ T[j]
      i := j
      j := ⌊i/2⌋
    senão
      fim := V

```

procedimento *descer*(*i*, *n*)

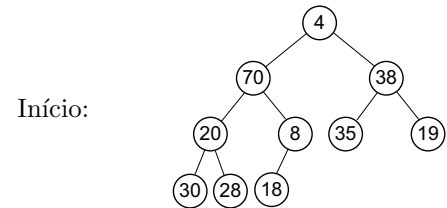
```

  j := 2i
  fim := F
  enquanto j ≤ n e fim = F faça
    se j < n então
      se T[j + 1].chave > T[j].chave então
        j := j + 1
    se T[i].chave < T[j].chave então
      T[i] ↔ T[j]
      i := j
      j := 2i
    senão
      fim := V

```

6) (1,5) Mostre os passos da construção de um *heap* contendo as seguintes chaves, colocadas nesta ordem em um vetor: 4, 70, 38, 20, 8, 35, 19, 30, 28, 18. A raiz é a chave cuja prioridade tem valor máximo. Utilize a Solução 3 vista em aula para a construção do heap.

Resposta:



Construção do heap:

