UNIVERSIDADE DA BEIRA INTERIOR

Algoritmos e Estruturas de Dados **2º Semestre**

Frequência 2 **Resolução** **2024/2025**

A. Árvores Binárias de Pesquisa (ABP)

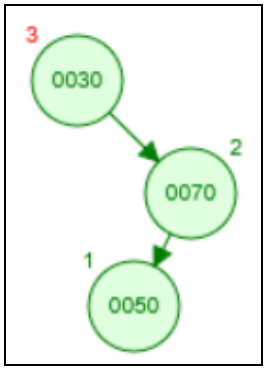
```
// Pergunta 1
int existeElemento (int X, PNodoABP T)
{
   if (T == NULL)
      return  0;
   if (X == T->Elemento)
      return  1;
   if (X < T->Elemento)
      return  existeElemento(X, T->Esquerda);
   else
      return  existeElemento(X, T->Direita);
}

//  Pergunta 2 - versão 1
PNodoABP paiNodoMenorMaiorX (PNodoABP T, PNodoABP pai, int X)
{
   PNodoABP  PEsq;
   if (T == NULL)
      return  NULL;
   if (T->Elemento > X)
      if (T->Esquerda == NULL)
         return  pai;
      else {
         PEsq = paiNodoMenorMaiorX(T->Esquerda, T, X);
         if (PEsq != NULL)
            return  PEsq;
         else
            return  pai;
      }
   else
      return  paiNodoMenorMaiorX(T->Direita, T, X);
}
```

NOTA: na primeira chamada desta função, o argumento pai recebe NULL.

```
// Pergunta 2 - versão 2
PNodoABP paiNodoMenorMaiorX_2 (PNodoABP T, int X)
{
   PNodoABP P;

   if (T == NULL)
      return NULL; // só acontece pata T original vazia

   if (T->Elemento <= X) {
      if (T->Direita == NULL)
         return NULL; // todos os elementos são <= X
      if (T->Direita->Elemento > X)
         if (T->Direita->Esquerda == NULL)  // T->Direita->Elemento é o menor dos maiores que X
            return T; // T é o pai no menor dos maiores que X
         else {
            P = paiNodoMenorMaiorX_2(T->Direita, X);
            if (P == NULL) // o menor dos maiores que X é a raiz da T->Direita
               return T;   // e o seu pai é T
            else
               return P;
         }
      else
         return paiNodoMenorMaiorX_2(T->Direita, X);
   }

   if (T->Elemento > X) {
      if (T->Esquerda == NULL)
         return NULL; // a raiz principal é o menor dos maiores que X
      if (T->Esquerda->Elemento > X)
         if (T->Esquerda->Esquerda == NULL)  // T->Esquerda->Elemento é o menor dos maiores que X
            return T; // T é o pai no menor dos maiores que X
         else {
            P = paiNodoMenorMaiorX_2(T->Esquerda, X);
            if (P == NULL) // o menor dos maiores é a raiz da T->Esquerda
               return T;   // e o seu pai é T
            else
               return P;
         }
      else
         return paiNodoMenorMaiorX_2(T->Esquerda, X);
   }

}
```

B. ABP Balanceadas/Equilibradas (AVL)

Inserir **30, 70** e **50**, por esta ordem:



A árvore fica **equilibrada** com a inserção dos nodos **30** e **70**

A árvore fica **desequilíbrada** com a inserção do nodo **50**

O **desequilibrio** acontece na árvore com raiz no nodo **30**, pois

- o seu filho direito (nodo **70**) tem altura 2
- o seu filho esquerdo (NULL) tem altura 0
- altura(filho direito) – altura(filho esquerdo) = 2 - 0 = 2

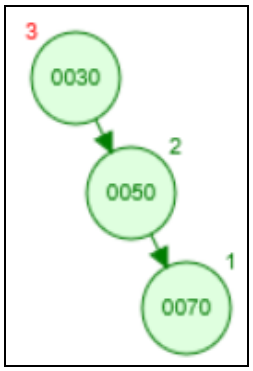
Todos as subárvores com raiz nos nodos descendentes do nodo **30** estão equilibradas

Como

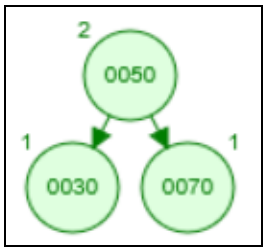
- o filho **direito** do nodo **30** (nodo **70**) tem **mais altura** que o filho esquerdo do nodo **30** (**NULL**), e
- o filho **esquerdo** do nodo **70** (nodo **50**) tem **mais altura** que o filho direito do **70** (**NULL**),

então **reequilibrar** a árvore aplicando uma **rotação dupla à esquerda**, usando o nodo **50**

1º) **rotação simples à direita** do nodo **70** usando o nodo **50**:

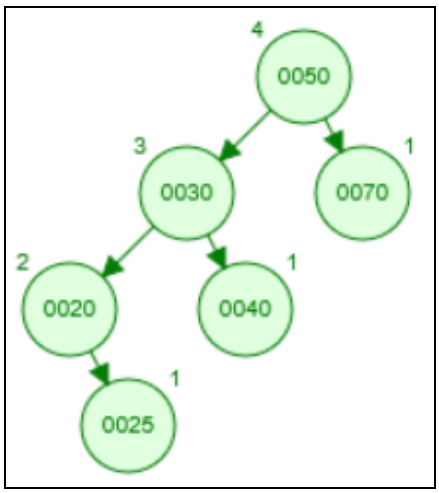


2º) **rotação simples à esquerda** do nodo **30** usando o nodo **50**:



A árvore está equilibrada.

Inserir **20, 40** e **25**, por esta ordem:



A árvore fica equilibrada com a inserção dos nodos **20** e **40**

A árvore fica **desequilíbrada** com a inserção do nodo **25**

O desequilibrio acontece na árvore com raiz no nodo **50**, pois

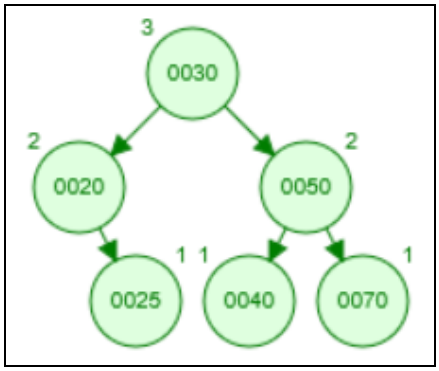
- o seu filho esquerdo (nodo **30**) tem altura 3
- o seu filho direito (nodo **70**) tem altura 1
- altura(filho esquerdo) – altura(filho direito) = 3 - 1 = 2

Todos as subárvores com raiz nos nodos descendentes do nodo **50** estão equilibradas

Como

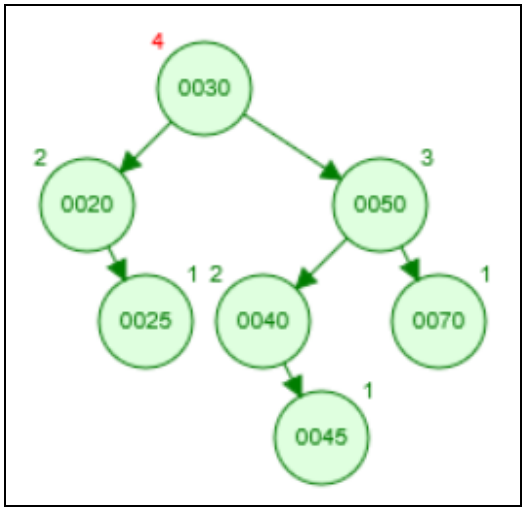
- o filho **esquerdo** do nodo **50** (nodo **30**) tem **mais altura** que o filho direito do nodo **50** (nodo **70**), e
- o filho **esquerdo** do nodo **30** (nodo **20**) tem **mais altura** que o filho direito do nodo **30** (nodo **40**),

então **reequilibrar** a árvore aplicando uma **rotação simples à direita** do nodo **50** usando o nodo **30**

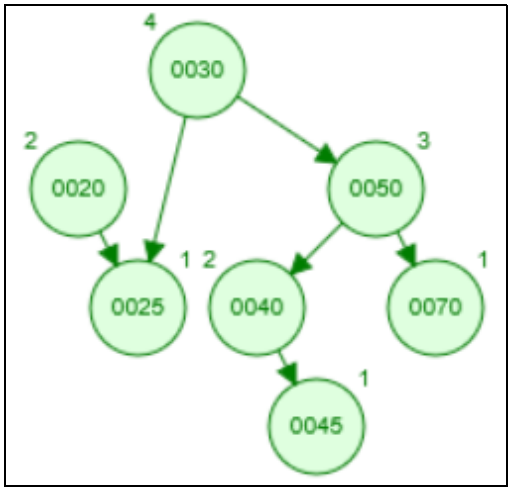


A árvore está equilibrada.

Inserir **45**:



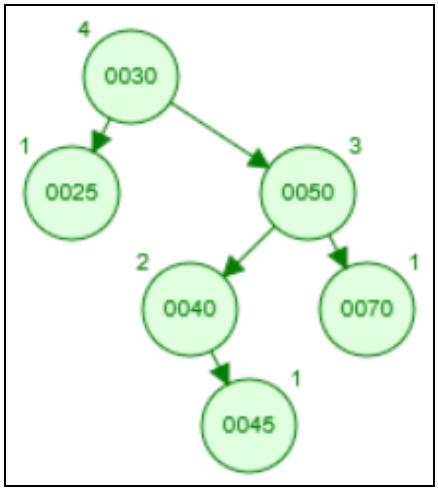
A árvore está equilibrada.



Remover o nodo **20**:

como o nodo **20** tem apenas um filho, a remoção deste nodo consiste nos dois passos seguintes:

- ligar o nodo **30** ao nodo filho de **20** (nodo **25**), através da sua ligação esquerda (filho esquerdo do nodo **30** passa a ser o nodo **25**)
- remover o nodo **20**



A árvore fica **desequilíbrada** com a remoção do nodo **20**

O **desequilibrio** acontece na árvore com raiz no nodo **30**, pois

- o seu filho direito (nodo **50**) tem altura 3
- o seu filho esquerdo (nodo **25**) tem altura 1
- altura(filho direito) – altura(filho esquerdo) = 3 - 1 = 2

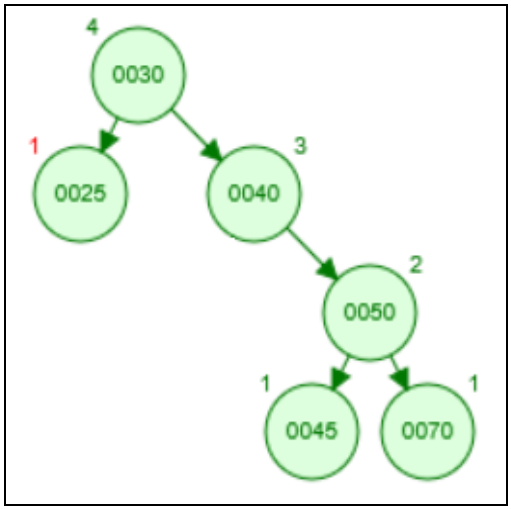
Todos as subárvores com raiz nos nodos descendentes do nodo **30** estão equilibradas

Como

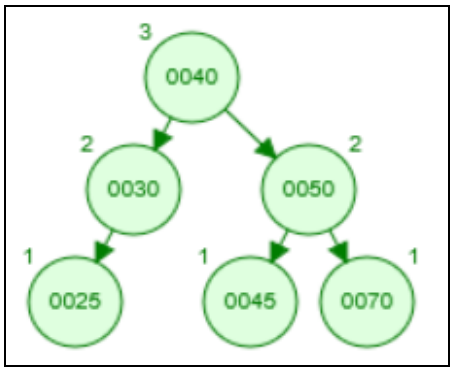
- o filho **direito** do nodo **30** (nodo **50**) tem **mais altura** que o filho esquerdo do nodo **30** (nodo **25**), e
- o filho **esquerdo** do nodo **50** (nodo **40**) tem **mais altura** que o filho direito do nodo **50** (nodo **70**),

então **reequilibrar** a árvore aplicando uma **rotação dupla à esquerda** do nodo **30**

1º) **rotação simples à direita** do nodo **50** usando o nodo **40**:



2º) **rotação simples à esquerda** do nodo **30** usando o nodo **40**:



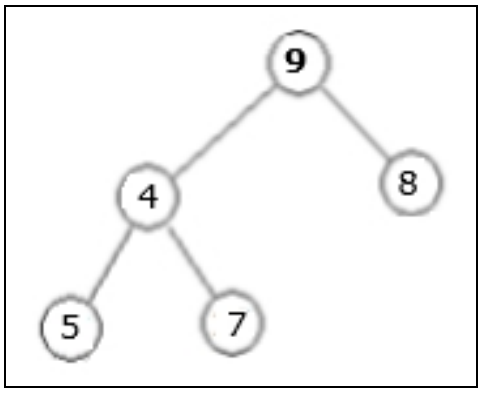
C. Heaps

1.

1º Extrair/tomar o elemento que se encontra na raiz do minHeap: **2**

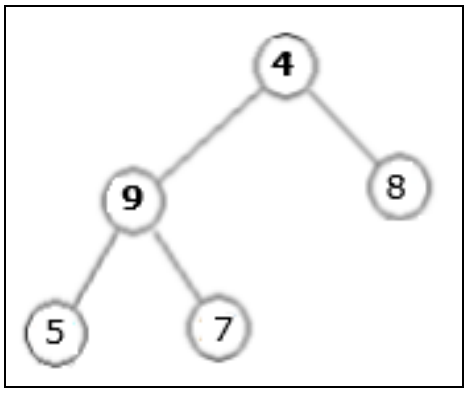
2º Remover o nó que se encontra na raiz da árvore, da seguinte forma:

a) copiar para a raiz (com valor 2) o elemento do nodo mais à direita do último nível do minHeap (com valor 9) e remover este último nodo

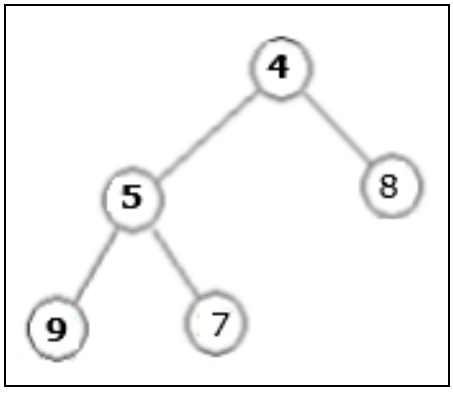


b) repor as propriedade de minHeap: comparar o nodo com valor 9 (agora raiz) com o menor (melhor) dos seus filhos e, caso seja maior (pior) do que esse filho, então trocá-los; fazer isto até o nodo com valor 9 ser menor (melhor) que os seus filhos ou atingir o último nível do minHeap:

i) como 9 > 4 (4 = min{4, 8}) (pior que o melhor dos seus filhos), então trocar o elemento com valor 9 pelo elemento com valor 4



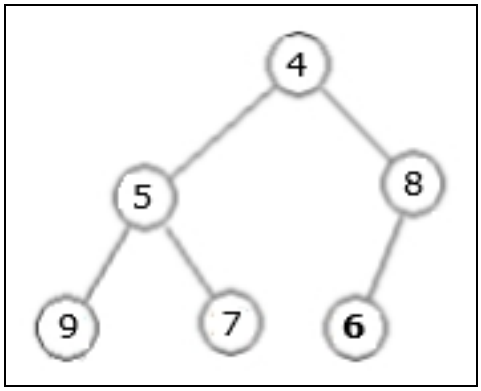
ii) como 9 > 5 (5 = min{5, 7}) (pior que o melhor dos seus filhos), então trocar o elemento com valor 9 pelo elemento com valor 5



iii) como o nodo 9 não tem filhos, então terminar processo.

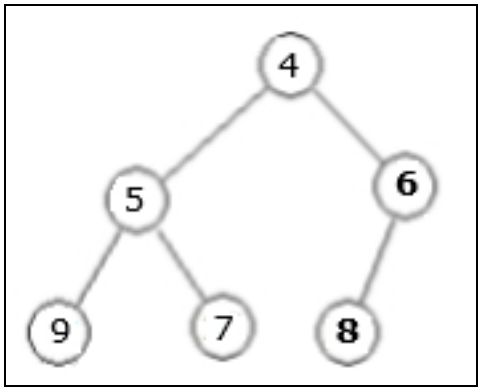
2.

a) inserir um nodo com o valor **6** como filho esquerdo do nodo com o valor **8**



b) comparar o nodo com o valor **6** com o seu pai, e caso tenha menor (melhor) valor trocá-los. Fazer isto até um nodo com valor **6** ter valor maior (pior) do que o seu pai ou chegar à raiz da minHeap:

i) como 6 < 8 (melhor que o seu pai), então trocá-los



ii) como 6 > 4 (pior que o seu pai), então parar o processo.

D. Grafos

1.

			destino/custo			
1	3	--->	2/5	4/10	6/8	
2	3	--->	1/5	3/20	4/4	
3	4	--->	2/20	5/6	6/14	8/2
4	3	--->	1/10	2/4	5/3	
5	2	--->	3/6	4/3		
6	3	--->	1/8	3/14	7/5	
7	2	--->	6/5	8/15		
8	2	--->	3/2	7/15		

2.

Passo 1:

S = 5

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
R	∞	∞	6	3	0	∞	∞	∞
Pai	-	-	5	5	5	-	-	-

Permanentes ← 5

Temporários ← 1 - 2 – 6 – 4 – 7 – 3 - 8

Passos 2 e 3:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
R	∞ 10 5	∞ 4	6	3	0	∞ 8	∞ 15 5	∞ 2
Pai	- 4 2	- 4	5	5	5	- 1	- 8 6	- 3

Permanentes ← 5 - 4 - 2 - 1 - 3 - 8 - 6 - 7

Temporários ← 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8

k ← 4 ---> R(4) = 3 = minimo entre os rótulos dos nós temporários (o nó 4 passa a Permanente)

Temporários ≠ ∅ ⇒ Continuar

j ← 1, 2, 5 (adjacentes ao nó 4)

1: C(4,1) < R(1) ?

10 < ∞ SIM ⇒ **R(1) = 10 e Pai(1) = k = 4**

2: C(4,2) < R(2) ?

4 < ∞ SIM ⇒ **R(2) = 4 e Pai(2) = k = 4**

5: não pode ser rotulado, pois já é Permanente

k ← 2 ---> R(2) = 5 = minimo entre os rótulos dos nós temporários (o nó 2 passa a Permanente)

Temporários ≠ ∅ ⇒ Continuar

j ← 1, 3, 4 (adjacentes ao nó 2)

1: C(2,1) < R(1) ?

5 < 10 SIM ⇒ **R(1) = 5 e Pai(1) = k = 2**

3: C(2,3) < R(3) ?

20 < 6 NÃO

4: não pode ser rotulado, pois já é Permanente

k ← 1 ---> R(1) = 5 = minimo entre os rótulos dos nós temporários (o nó 1 passa a Permanente)

Temporários ≠ ∅ ⇒ Continuar

j ← 2, 4, 6 (adjacentes ao nó 1)

2: não pode ser rotulado, pois já é Permanente

4: não pode ser rotulado, pois já é Permanente

6: C(1,6) < R(6) ?

8 < ∞ SIM ⇒ **R(6) = 8 e Pai(6) = k = 1**

k ← 3 ---> R(3) = 5 = minimo entre os rótulos dos nós temporários (o nó 3 passa a Permanente)

Temporários ≠ ∅ ⇒ Continuar

j ← 2, 5, 6, 8 (adjacentes ao nó 3)

2: não pode ser rotulado, pois já é Permanente

5: não pode ser rotulado, pois já é Permanente

6: C(3,6) < R(6) ?

14 < 8 NÃO

8: C(3,8) < R(8) ?

2 < ∞ SIM ⇒ **R(8) = 2 e Pai(8) = k = 3**

k ← 8 ---> R(8) = 2 = minimo entre os rótulos dos nós temporários (o nó 8 passa a Permanente)

Temporários ≠ ∅ ⇒ Continuar

j ← 3, 7 (adjacentes ao nó 8)

3: não pode ser rotulado, pois já é Permanente

7: C(8,7) < R(7) ?

15 < ∞ SIM ⇒ **R(7) = 15 e Pai(7) = k = 8**

k ← 6 ---> R(6) = 8 = minimo entre os rótulos dos nós temporários (o nó 6 passa a Permanente)

Temporários ≠ ∅ ⇒ Continuar

j ← 1, 3, 7 (adjacentes ao nó 6)

1: não pode ser rotulado, pois já é Permanente

3: não pode ser rotulado, pois já é Permanente

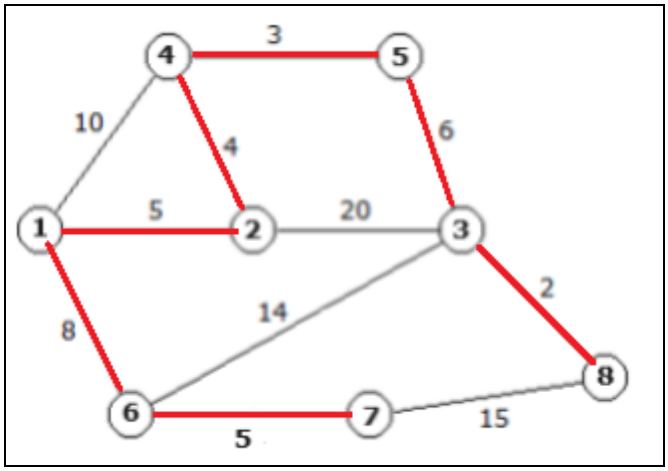
7: C(6,7) < R(7) ?

5 < 15 SIM ⇒ **R(7) = 5 e Pai(7) = k = 6**

k ← 7 ---> R(7) = 5 = minimo entre os rótulos dos nós temporários (o nó 7 passa a Permanente)

Temporários = ∅ ⇒ PARAR

Árvore Abrangente Mínima determinada tomando o nó 5 com inicial (S = 0)=dos Caminhos Mais Curtos entre o nó 1 e todos os outros nós, é a seguinte:



3.

Custo totla da AAM = C(1,2) + C(1,6) + C(2,4) + C(4,5) + C(5,3) + C(6,7) + C(3,8) =

5 + 8 + 4 + 3 + 6 + 5 + 2 = **33**