UNIVERSIDADE DA BEIRA INTERIOR

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

Frequência 2 **Resolução** **2023/2024**

A. Árvores Binárias de Pesquisa (ABP)

Considere a seguinte declaração de EAD Árvore Binária:

```
struct NodoABP {
   int    Elemento;
   struct  NodoABP  *Esquerda;
   struct  NodoABP  *Direita;
};
typedef  struct  NodoABP *PNodoABP;
```

Implemente uma função em C que

- **receba** uma ABP **T**,
- **devolva** o menor elemento positivo (inteiro > 0) entre todos os elementos da ABP T (se T está vazia ou se todos os elementos de T são não positivos (≤ 0), então a função deve devolver **-1**).

Nota: otimize o algoritmo criado (código escrito) tendo em conta a **definição de ABP**.

```
int  menorPositivo (PnodoABP T)
{
   int  menorPosEsq;
   // Caso base/terminal: T vazia
   if (T == NULL)
      return -1;
   // caso geral: raiz ≤ 0 e raiz > 0
   if (T->Elemento <= 0)
      return (T->Direita);
   // T->Elemento > 0
   menorPosEsq = menorPositivo(T->Esquerda);
   if (menorPosEsq == -1)
      return T->Elemento;
   if (T->Elemento < menorPosEsq)
      return T->Elemento;
   else
      return menorPosEsq;
}
```

B. ABP Balanceadas/Equilibradas (AVL)

Equilibrar uma **ABP do tipo AVL**, consiste em aplicar uma das 4 operações de **rotação**: simples à esquerda, simples à direita, dupla à esquerda e dupla à direita.

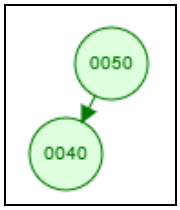
a) Construa uma **ABP do tipo AVL inserindo** os nodos com os seguintes valores (um a um e pela ordem apresentada): **50, 40, 30, 60, 55 e 65**. (**NOTA**: não é necessário colocar as alturas).

Sempre que **inserir um nodo**, deve **redesenhar** a **ABP** obtida (acrescentar o nodo à árvore) e **verificar se continua balanceada/equilibrada**. Caso **não fique equilibrada**, deve

1º) indicar o nodo onde se deteta o desequilíbrio e o tipo de rotação a aplicar para reequilibrá-la,

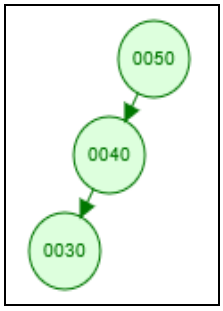
2º) reequilibrar a árvore e redesenhá-la (apresentando todos os passos efetuados)

Inserir **50** e **40**, por esta ordem:



A árvore está equilibrada em ambos os casos.

Inserir **30**:



A árvore fica desequilíbrada no nodo **50**, pois

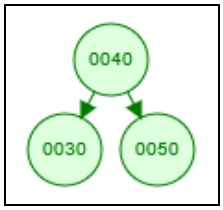
- o seu filho esquerdo (nodo 40) tem altura 1
- o seu filho direito (NULL) tem altura -1
- altura(filho esquerdo) – altura(filho direito) = 1 - (-1) = 2

Como

- o filho esquerdo de 50 (40) tem mais altura que o filho direito de 50 (NULL), e
- o filho esquerdo de 40 (30) tem mais altura que o filho direito de 40 (NULL),

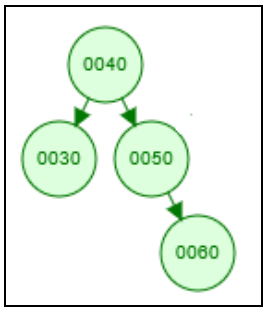
então reequilibrar a árvore aplicando uma **rotação simples à direita** do nodo 50 usando o nodo 40

Árvore após rotação simples à direita do nodo 50:



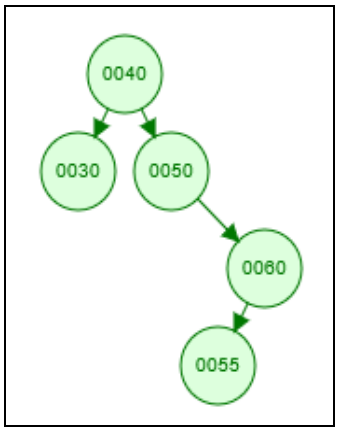
A árvore está equilibrada.

Inserir **60**:



A árvore está equilibrada.

Inserir **55**:



A árvore fica desequilíbrada no nodo **50**, pois

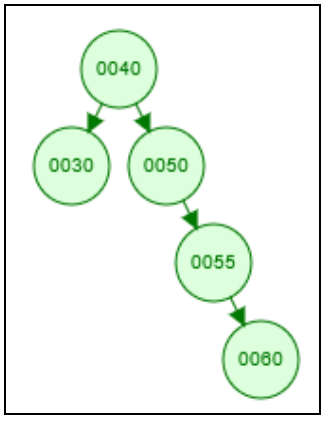
- o seu filho esquerdo (NULL) tem altura -1
- o seu filho direito (60) tem altura 1
- altura(filho direito) – altura(filho esquerdo) = 1 - (-1) = 2

Como

- o filho direito do nodo 50 (nodo 60) tem mais altura que o filho esquerdo de 50 (NULL), e
- o filho esquerdo de 60 (55) tem mais altura que o filho direito de 60 (NULL),

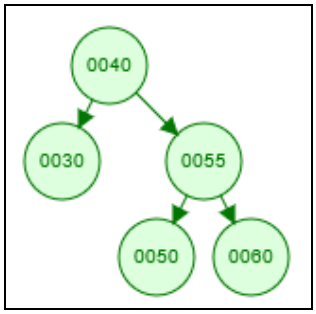
então reequilibrar a árvore aplicando uma **rotação dupla à esquerda** (rotação simples à direita seguida de rotação simples à esquerda), usando o nodo 55

1º) **rotação simples à direita** do nodo 60 usando o nodo **55**:



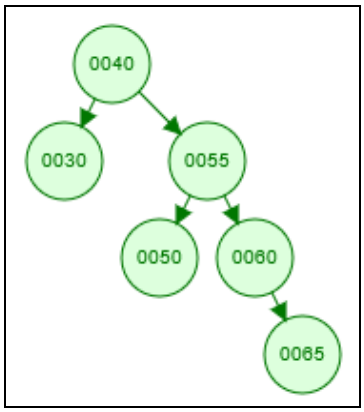
A árvore continua desequilibrada no nodo 50

2º) **rotação simples à esquerda** do nodo 50 usando o nodo **55**:



A árvore está equilibrada.

Inserir **65**:



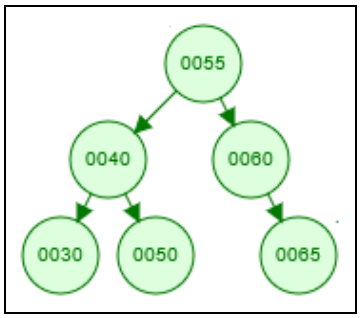
A árvore fica desequilíbrada no nodo **40**, pois

- o seu filho esquerdo (nodo 30) tem altura 0
- o seu filho direito (nodo 55) tem altura 2
- altura(filho direito) – altura(filho esquerdo) = 2 - 0 = 2

Como

- o filho direito do nodo 40 (55) tem mais altura que o filho esquerdo do nodo 40 (30), e
- o filho direito do nodo 55 (60) tem mais altura que o filho esquerdo do nodo 55 (50),

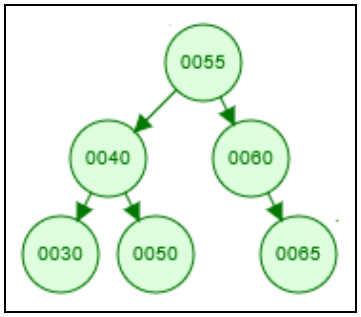
então reequilibrar a árvore aplicando uma **rotação simples à esquerda** do nodo 40 usando o nodo 55



A árvore está equilibrada.

---

b) Remova o nodo **raiz** da ABP **obtida em a)**, **redesenhe** a árvore obtida, **verifique** se continua balanceada/equilibrada e **reequilibre-a**, caso seja necessário. Apresente a árvore final.

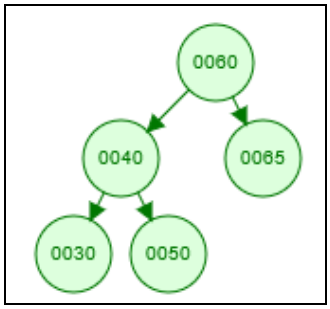


Remover a raiz da árvore (nodo 55) implica substituir o elemento 55 por um dos seguintes elementos:

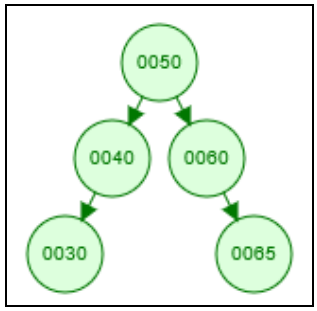
- 60 (menor elemento da árvore maior que 55)
- 50 (maior elemento da árvore menor que 55)

e de seguida remover o nodo substituto.

Caso 1: substituir pelo elemento 60



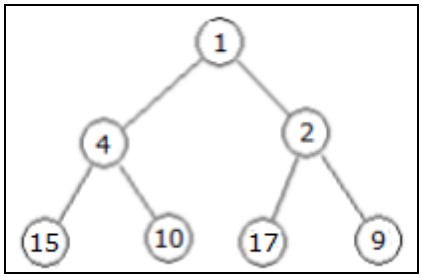
Caso 2: substituir pelo elemento 50



Em ambos os casos, a árvore está equilibrada.

C. Heaps

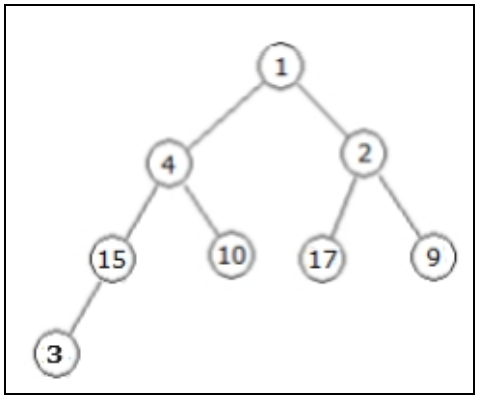
Considere a seguinte **minHeap**:



NOTA: Descreva os processos aplicados para obter as **minHeaps** finais nas duas questões.

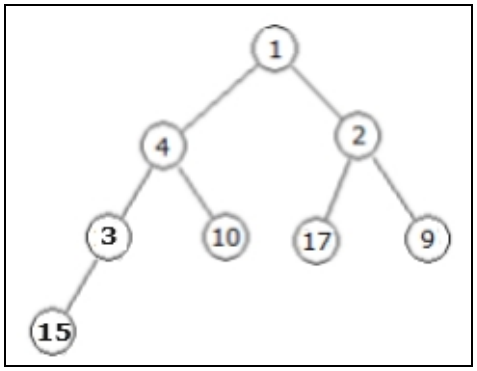
a) Insira um nodo com o valor **3** na minHeap dada (inicial) e desenhe a minHeap obtida. Justifique, apresentando a evolução da minHeap desde a estrutura inicial até à final.

1º) Inserir um nodo com o valor **3** como filho esquerdo do nodo com o valor 15

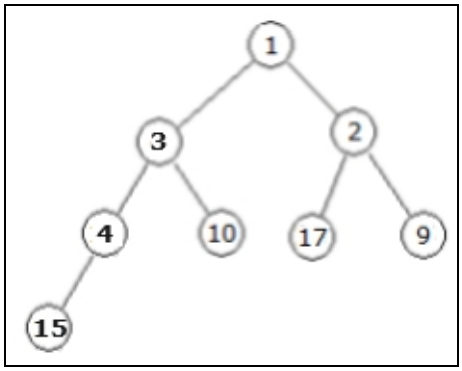


2º) Comparar o nodo com o valor 3 com o seu pai, e caso tenha menor (melhor) valor trocá-los. Fazer isto até um nodo com valor 3 ter valor mior (pior) do que o seu pai ou chegar à raiz da minHeap:

a) como 3 < 15 (melhor que o seu pai), então trocá-los



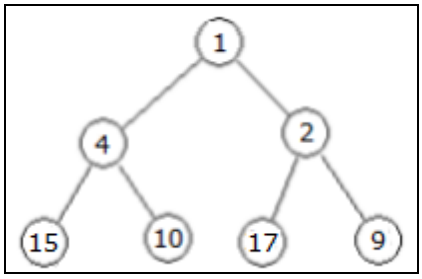
b) como 3 < 4 (melhor que o seu pai), então trocá-los



c) como 3 > 1 (pior que o seu pai), então parar o processo.

---

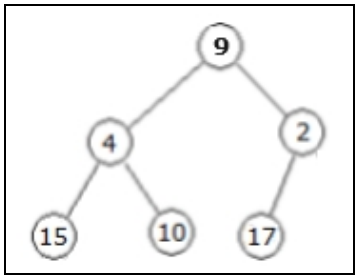
b) Aplique a operação **remover** à minHeap dada (inicial) e desenhe a minHeap obtida. Justifique, apresentando a evolução da minHeap desde a estrutura inicial até à estrutura final.



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

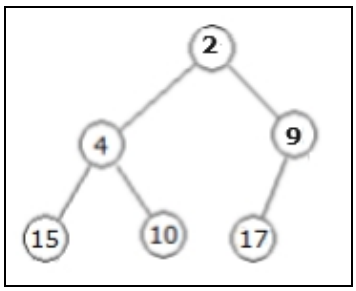
2º) Remover o nó que se encontra na raiz da árvore, da seguinte forma:

a) copiar para a raiz (com valor 1) 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:

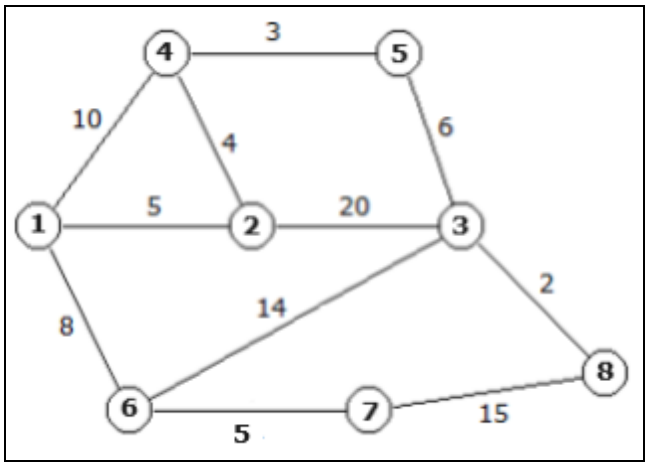
i) como 9 > 2 (= min{4,2}) (pior que o melhor dos seus filhos), então trocar o elemento com valor 9 pelo elemento com valor 2



ii) como 9 < 17 (= valor do seu único filho) (melhor que o melhor dos seus filhos), então terminar processo.

D. Grafos

Considere a seguinte rede G = (A, N, C), onde o valor de cada arco corresponde ao seu custo (C_{ij}):



1. Construa a **lista de Adjacência** da rede G.

---

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. Determine a Árvore dos Caminhos Mais Curtos **do nó 1 (S = 1) para todos os outros nós**, simulando o algoritmo de **Dijkstra**. Apresente os dados da simulação, em especial: evolução dos rótulos, conjuntos dos Permanentes e dos Temporários, evolução dos valores de **k** e de **j**.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
R(i)	0 /	5 /	∞ 25 22 18 /	10 9 /	∞ 12 /	8 /	∞ 13 /	∞ 28 20 /
Q(i)	1	1	1 2 6 5	1 2	1 4	1	1 6	1 7 3

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

Temporários ← ~~2~~ – ~~3~~ – ~~4~~ – ~~5~~ – ~~6~~ – ~~7~~ - ~~8~~

Temporários ≠ ∅ ⇒ Continuar

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

j ← 1, 3, 4

1: não pode ser rotulado, pois já é Permanente

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

5 + 20 < ∞ SIM ⇒ **R(3) = 25 e Q(3) = k = 2**

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

5 + 4 < 10 SIM ⇒ **R(4) = 9 e Q(4) = k = 2**

Temporários ≠ ∅ ⇒ Continuar

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

j ← 1, 3, 7

1: não pode ser rotulado, pois já é Permanente

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

8 + 14 < 25 SIM ⇒ **R(3) = 22 e Q(3) = k = 6**

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

8 + 5 < ∞ SIM ⇒ **R(7) = 13 e Q(7) = k = 6**

Temporários ≠ ∅ ⇒ Continuar

k ← 4 ---> R(4) = 9 = minimo entre os rótulos dos nós temporários (o nó 4 passa a Permanente)

j ← 1, 2, 5

1: não pode ser rotulado, pois já é Permanente

2: não pode ser rotulado, pois já é Permanente

5: R(4) + C(4,5) < R(5) ?

9 + 3 < ∞ SIM ⇒ **R(5) = 12 e Q(5) = k = 4**

Temporários ≠ ∅ ⇒ Continuar

k ← 5 ---> R(5) = 12 = minimo entre os rótulos dos nós temporários (o nó 5 passa a Permanente)

j ← 3, 4

3: R(5) + C(5,3) < R(3) ?

12 + 6 < 22 SIM ⇒ **R(3) = 18 e Q(3) = k = 5**

4: não pode ser rotulado, pois já é Permanente

Temporários ≠ ∅ ⇒ Continuar

k ← 7 ---> R(7) = 13 = minimo entre os rótulos dos nós temporários (o nó 7 passa a Permanente)

j ← 6, 8

6: não pode ser rotulado, pois já é Permanente

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

13 + 15 < ∞ SIM ⇒ **R(8) = 28 e Q(8) = k = 7**

Temporários ≠ ∅ ⇒ Continuar

k ← 3 ---> R(3) = 18 = minimo entre os rótulos dos nós temporários (o nó 3 passa a Permanente)

j ← 2, 5, 6, 8

2: não pode ser rotulado, pois já é Permanente

5: não pode ser rotulado, pois já é Permanente

6: não pode ser rotulado, pois já é Permanente

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

18 + 2 < 28 SIM ⇒ **R(8) = 20 e Q(8) = k = 3**

Temporários ≠ ∅ ⇒ Continuar

k ← 8 ---> R(8) = 20 = minimo entre os rótulos dos nós temporários (o nó 8 passa a Permanente)

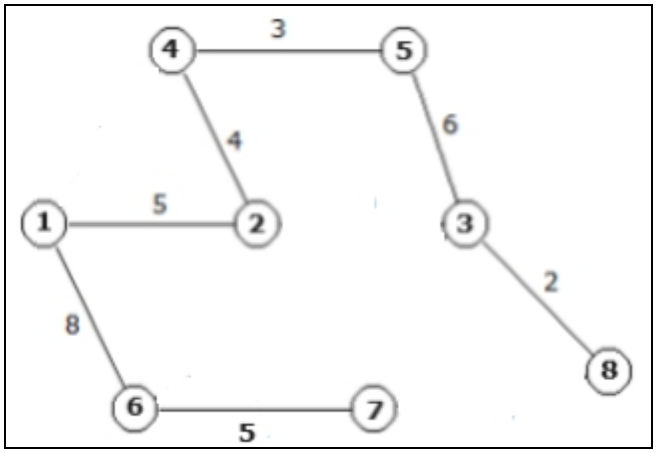
j ← 3, 7

3: não pode ser rotulado, pois já é Permanente

7: não pode ser rotulado, pois já é Permanente

Temporários = ∅ ⇒ PARAR

Árvore dos Caminhos Mais Curtos entre o nó 1 e todos os outros nós, é a seguinte:



---

3. Qual o C.M.C. entre o nó 1 e o nó 8 ? E qual o custo deste caminho ? Justifique.

C.M.C. (1, 8):

Percorrer a rede em sentido inverso, do nó 8 até ao nó 1 e usando os valores de Q(i).

8 ⇐ Q(8)=3 ⇐ Q(3)=5 ⇐ Q(5)=4 ⇐ Q(4)=2 ⇐ Q(2)=**1**

Portanto, o **C.M.C. (1, 8) = [1, 2, 4, 5, 3, 8]**

Custo do C.M.C. (1, 8) = R(8) = **20**
