UNIVERSIDADE DA BEIRA INTERIOR

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

Exame Época Normal **Resolução** **2025/2026**

NOTA: A forma como esta resolução se apresenta, para além de mostrar uma possível solução para cada questão, serve também para explicar os métodos e algoritmos usados. Assim sendo, não seria necessário, em especial nalgumas questões, detalhar tanto as respostas.

A. Análise de complexidade dos algoritmos

1.

k	k < 4 (S/N)	j	j < n (S/N)	printf
(k=1) 1	1 < 4 (S)	(j=k) 1 (j++) 2 ... (j++) n-1 (j++) n	1 < n (S) 2 < n (S) ... n-1 < n (S) n < n (N)	OLA OLA ... OLA
(k++) 2	2 < 4 (S)	(j=k) 2 (j++) 3 ... (j++) n-1 (j++) n	2 < n (S) 3 < n (S) ... n-1 < n (S) n < n (N)	OLA OLA ... OLA
(k++) 3	3 < 4 (S)	(j=k) 3 (j++) 4 ... (j++) n-1 (j++) n	3 < n (S) 4 < n (S) ... n-1 < n (S) n < n (N)	OLA OLA ... OLA
(k++) 4	4 < 4 (N)			

2.

Para determinar a ordem de complexidade do algoritmo, **basta** contabilizar o número de vezes que a operação dominante é executada. Analisando a tabela de simulação, conclui-se que a operação dominante é a comparação "**j < n**".

T(n) = **n** [k=1] + **(n-1)** [k=2] + **(n-2)** [k=3] = **3 n – 3 => T(n)** = **O(n)**

B. Algoritmos de ordenação e de pesquisa

```
int segundoMaiorValor (int A[], int N)
{
   int  k, max1, max2;
   k = N-2;
   max1 = A[N-1];  // maior valor
   while(k >= 0 && A[k] == max1)
      k = k - 1;
   if(k >= 0)
      max2 = A[k];  // 2º maior valor
   else
      return -1;
   k = k – 1;
   while(k >= 0 && A[k] == max2)
      k = k – 1;
   if(k >= 0)
      return A[k];   // 3º maior valor
   else
      return -1;
}
```

C. Listas ligadas

1.

```
int numeroElementosAeB (PNodoLista L, int A, int B) {
   int  cont;
   // caso base/terminal
   if (L == NULL)
      return  0;
   // caso geral
   cont = numeroElementosAeB(L->Prox, A, B);
   if (L->Elemento >= A  &&  L->Elemento <= B)
      return  1 + cont;
   else
      return  cont;
}
```

2.

```
PNodoLista inserirAntesMaior (PNodoLista L, int X) {
   PNodoLista  P, PAnt, PAntMaior, Novo;
   int  maior;
   Novo = criarNodoLista(X);
   if (Novo == NULL)
      return  L;
   if (L == NULL)
      return  Novo;
   // procurar o maior elemento (maior) e marcar o seu antecessor com PAntMaior
   PAntMaior = NULL;
   PAnt = L;
   P = L->Prox;
   maior = L->Elemento;
   while (P != NULL) {
      if (P->Elemento > maior){
         PAntMaior = PAnt;
         maior = P->Elemento;
      }
      PAnt = P;
      P = P->Prox;
   }
   // inserir o novo elemento como antecessor do maior
   if (PAntMaior == NULL) {  // inserir o novo elemento na cabeça da lista
      Novo->Prox = L;
      L = Novo;  //  ou: return Novo;
   }
   else {
      Novo->Prox = PAntMaior->Prox;
      PAntMaior->Prox = Novo;
   }
   return  L;
}
```

D. Pilhas

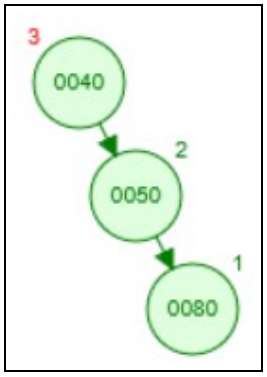
```
int numeroElementosMaioresFundo (PNodoPilha *S) {
   PNodoPilha  SAux;
   int  cont, fundo;
   if (pilhaVazia(*S) == 1)  // se a pilha está vazia, a soma é 0
      return  -1;
   SAux = criarPilha();
   // determinar o fundo da pilha e construir uma pilha auxiliar com os elementos de S
   while (pilhaVazia(*S) == 0) {
      SAux = push(topo(*S), SAux);
      *S = pop(*S);
   }
   fundo = topo(SAux);    // o fundo da pilha S é o topo da pilha SAux
   while (pilhaVazia(SAux) == 0) {  // reconstruir a pilha S e
      if(topo(SAux) > fundo)            // contar quantos elementos que são maiores que fundo
         cont = cont + 1;
      *S = push(topo(SAux), *S);
      SAux = pop(SAux);
   }
   return  cont;
}
```

E. Árvores Binárias de Pesquisa (ABP)

```
int maiorElementoParMaiorK (PNodoABP T, int K) {
   int  max;
   // caso base/terminal
   if (T == NULL)
      return -1;
   // caso geral
   if (T->Elemento > K) {
      maxPar = maiorElementoParMaiorK(T->Direita, K);
      if (maxPar != -1)
         return  maxPar;
      if (T->Elemento % 2 == 0)
         return  T->Elemento;
      return  maiorElementoParMaiorK(T->Esquerda, K);
   }
   else  // T->Elemento <= K
      return  maiorElementoParMaiorK(T->Direita, K);
}
```

F. ABP Balanceadas/Equilibradas (AVL)

Inserir **40, 50** e **80**, por esta ordem:



A árvore fica **equilibrada** com a inserção dos nodos **40** e **50**

A árvore fica **desequilibrada** com a inserção do nodo **80**

O **desequilíbrio** acontece na árvore com raiz no nodo **40**, pois

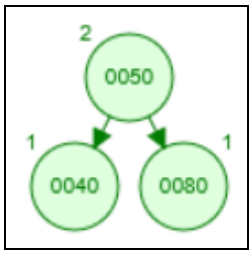
- o seu filho esquerdo (**NULL**) tem altura 0
- o seu filho direito (**50**) tem altura 2
- altura(filho direito) – altura(filho esquerdo) = 2 - 0 = 2

Todos as subárvores com raiz nos nodos descendentes do nodo **40** estão equilibradas

Como:

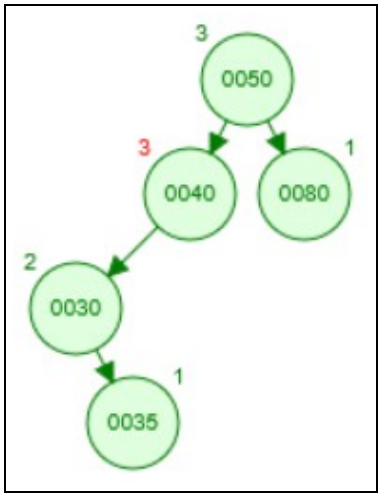
- o filho **direito** do nodo **40** (**50**) tem **mais altura** que o filho **esquerdo** do nodo **40** (**NULL**), e
- o filho **direito** do nodo **50** (**80**) tem **mais altura** que o filho **esquerdo** do nodo **50** (**NULL**),

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



A árvore está equilibrada.

Inserir **90** e **70**, por esta ordem:



A árvore fica equilibrada com a inserção do nodo **30**

A árvore fica **desequilibrada** com a inserção do nodo **35**

O desequilíbrio acontece na árvore com raiz no nodo **40**, pois

- o seu filho **direito** (**NULL**) tem altura o
- o seu filho **esquerdo** (**30**) tem altura 2
- altura(filho esquerdo) – altura(filho direito) = 2 - 0 = 2

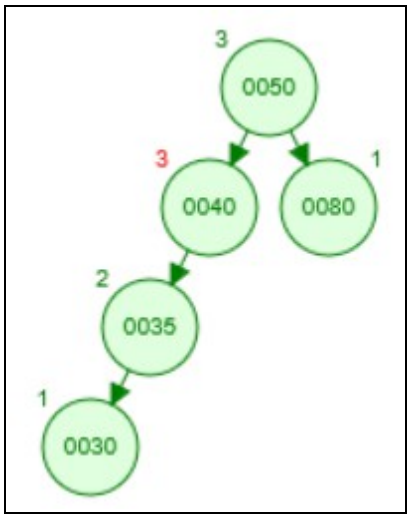
Todos as subárvores com raiz nos nodos descendentes do nodo **40** estão equilibradas

Como:

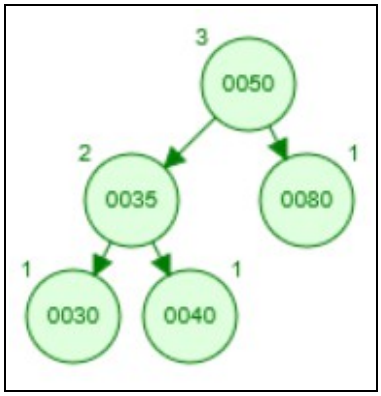
- o filho **esquerdo** do nodo **40** (**30**) tem **mais altura** que o filho **direito** do nodo **40** (**NULL**), e
- o filho **direito** do nodo **30** (**35**) tem **mais altura** que o filho **esquerdo** do nodo **30** (**NULL**),

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

1º) **rotação simples à esquerda** do nodo **30** usando o nodo **35**:

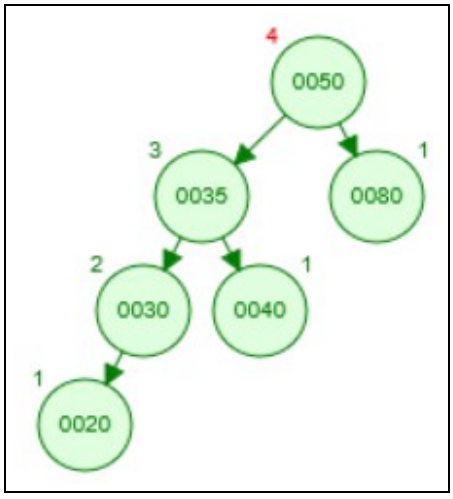


2º) **rotação simples à direita** do nodo **40** usando o nodo **35**:



A árvore está equilibrada.

Inserir **20**:



A árvore fica **desequilibrada** com a inserção do nodo **20**

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

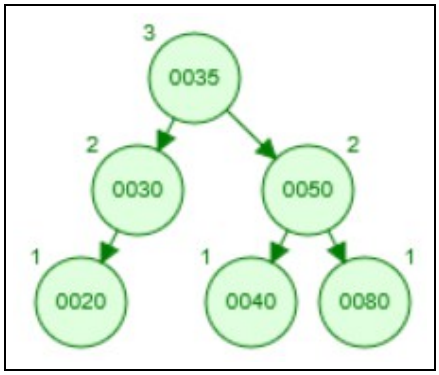
- o seu filho **esquerdo** (**35**) tem altura 3
- o seu filho **direito** (**80**) tem altura 1
- altura(filho esquerdo) – altura(filho direito) = 3 - 1 = 2

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

Como:

- o filho **esquerdo** do nodo **50** (**35**) tem **mais altura** que o filho **direito** do nodo **50** (**80**), e
- o filho **esquerdo** do nodo **35** (**30**) tem **mais altura** que o filho **direito** do nodo **40** (**NULL**),

então **reequilibrar** a árvore aplicando uma **rotação simples à direita** do nodo **50** usando o nodo **35**



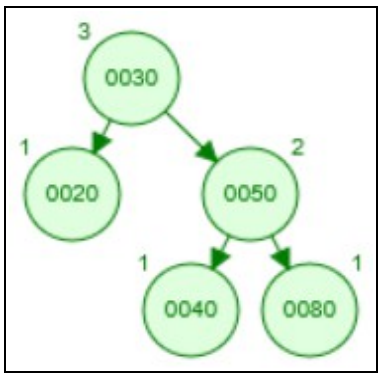
A árvore está equilibrada.

Remover o nodo 35 (raiz da ABP obtida)

Como o nodo **35** tem dois filhos, a remoção deste nodo pode ser feito de duas formas:

1º caso:

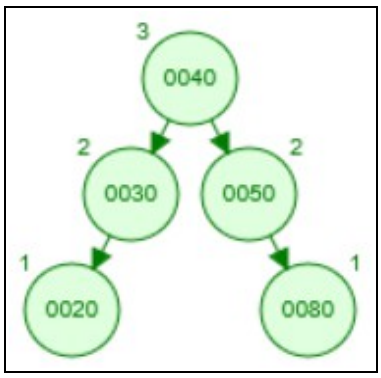
- copiar o valor do nodo com maior valor menor que 35 (**30**) para o nodo com **35** (raiz passa a **30**),
- remover o nodo com **30**



A árvore continua **equilibrada**

2º caso:

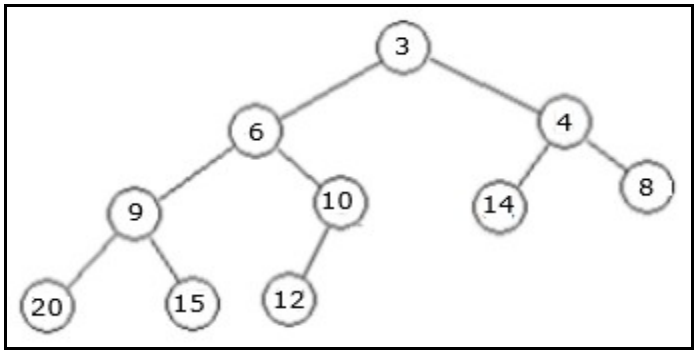
- copiar o valor do nodo com menor valor maior que 35 (**40**) para o nodo com **35** (raiz passa a **40**),
- remover o nodo com **40**



A árvore continua **equilibrada**

G. Heaps (Filas de Prioridade)

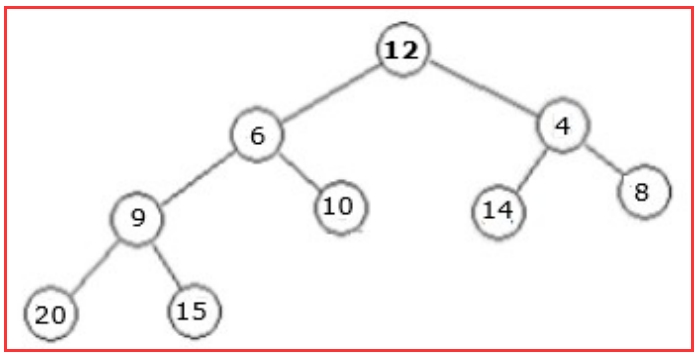
Considere o seguinte **minHeap** binário:



1. A operação **remover** implica realizar as seguintes ações:

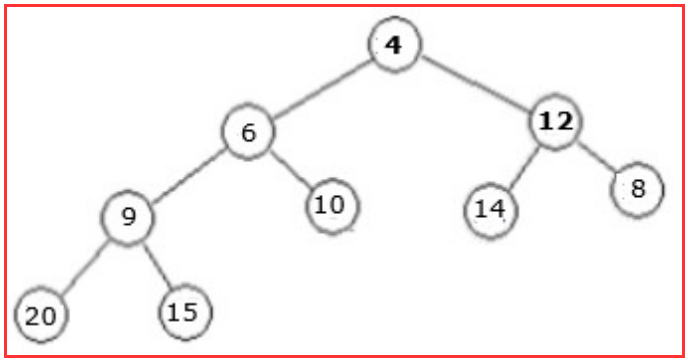
1º) Tomar o elemento que se encontra na raiz da minHeap (árvore): **3**

2º) Substituir o elemento que se encontra na raiz do minHeap (**3**), pelo elemento que se encontra no nó mais à direita do último nível da árvore (**12**) e de seguida remover este nó (com **12**) do minHeap

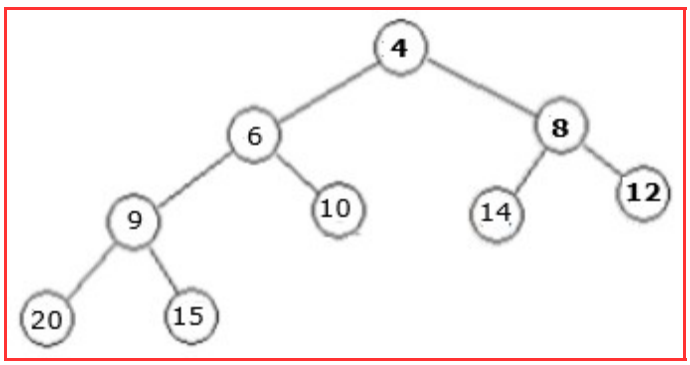


3º) Comparar o valor que se encontra agora na raiz (**12**) com o melhor (menor) dos seus filhos e, se for pior (maior), então trocá-los; realizar este processo até o valor da raiz (**12**) ficar no local correto

a) como **12 > 4** [= min(6, 4)], então **trocar** o **12** com o **4**



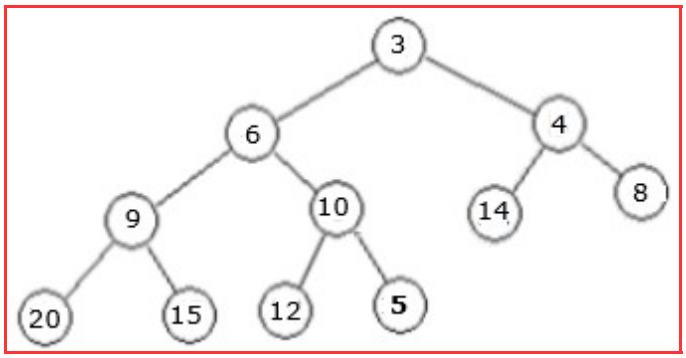
b) como **12 > 8** [= min(14, 8)], então **trocar** o **12** com o **8**



c) Como o nó com **12** não tem filhos, o processo termina

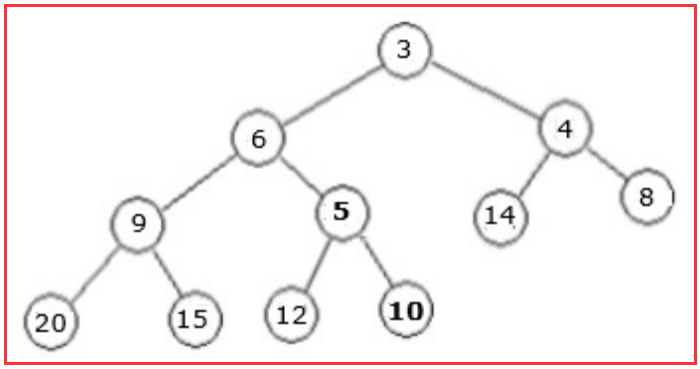
2. A operação **inserir** implica realizar as seguintes ações:

1º Inserir um nó com o valor 4 como filho esquerdo do nó com 11.

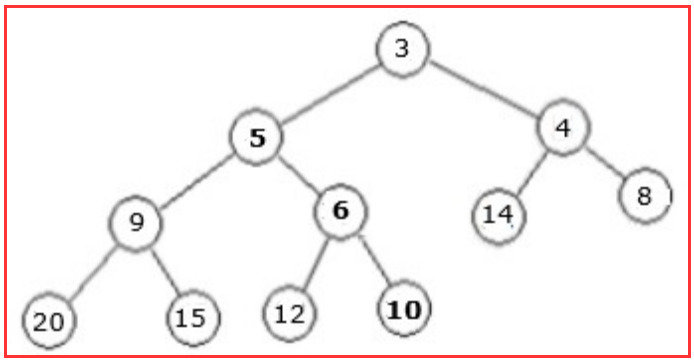


2º) Comparar o valor inserido com o valor do seu pai e, se for melhor (menor), então trocá-los; realizar este processo até o valor inserido ficar no local correto

a) como **5 < 10**, então **trocar** o **5** com o **10**



b) como **5 < 6**, então **trocar** o **5** com o **6**



c) como 5 > 3, então terminar processo; a árvore obtida em b) é a resultante da inserção de 5.

H. Grafos

1.

Tome-se **S = 7**

Passo1:

$R_7 \leftarrow 0$ e $Pai_7 \leftarrow 0$

$R_1 \leftarrow 7$ e $Pai_1 \leftarrow 7$ $R_6 \leftarrow 13$ e $Pai_6 \leftarrow 7$ $R_8 \leftarrow 6$ e $Pai_8 \leftarrow 7$

$R_2 \leftarrow \infty$ e $Pai_2 \leftarrow$ NULO $R_3 \leftarrow \infty$ e $Pai_3 \leftarrow$ NULO $R_4 \leftarrow \infty$ e $Pai_4 \leftarrow$ NULO $R_5 \leftarrow \infty$ e $Pai_5 \leftarrow$ NULO

Permanentes $\leftarrow$ { 7 }

Temporários $\leftarrow$ { 1, 2, 3, 4, 5, 6, 8 }

	1	2	3	4	5	6	7	8
R	9 4	∞ 8 7	∞ 12 5	∞ 6	∞ 8	13 3	0	6
Pai	7 3	- 4 1	- 6 4	- 6	- 8	7 5	7	7

Passo 2:

k: $R_k = \min\{ R_1, R_2, R_3, R_4, R_5, R_6, R_8 \} = \min\{ 9, \infty, \infty, \infty, \infty, 13, 6 \} = 6 \Rightarrow$ **k ← 8**

Permanentes $\leftarrow$ { 7, 8 } Temporários $\leftarrow$ { 1, 2, 3, 4, 5, 6 }

Temporários $\neq \varnothing \Rightarrow$ continuar

Passo 3:

k ← 8; j ← 5 (7 é permanente, logo não é considerado)

j ← 5 $\Rightarrow C_{85} < R_5$ (?) $\Leftrightarrow 8 < \infty$ (SIM); logo **$R_5 \leftarrow 8$ e $Pai_5 \leftarrow 8$**

Passo 2:

k: $R_k = \min\{ R_1, R_2, R_3, R_4, R_5, R_6 \} = \min\{ 9, \infty, \infty, \infty, 8, 13 \} = 8 \Rightarrow$ **k ← 5**

Permanentes $\leftarrow$ { 7, 8, 5 } Temporários $\leftarrow$ { 1, 2, 3, 4, 6 }

Temporários $\neq \varnothing \Rightarrow$ continuar

Passo 3:

k ← 5; j ← 6 (8 é permanente, logo não é considerado)

j ← 6 $\Rightarrow C_{56} < R_6$ (?) $\Leftrightarrow 3 < 13$ (SIM); logo **$R_6 \leftarrow 3$ e $Pai_6 \leftarrow 5$**

Passo 2:

k: $R_k = \min\{ R_1, R_2, R_3, R_4, R_6 \} = \min\{ 9, \infty, \infty, \infty, 3 \} = 3 \Rightarrow$ **k ← 6**

Permanentes $\leftarrow$ { 7, 8, 5, 6 } Temporários $\leftarrow$ { 1, 2, 3, 4 }

Temporários $\neq \varnothing \Rightarrow$ continuar

Passo 3:

k ← 6; j ← 3, 4 (5, 7 são permanentes, logo não são considerados)

j ← 3 $\Rightarrow C_{63} < R_3$ (?) $\Leftrightarrow 12 < \infty$ (SIM); logo **$R_3 \leftarrow 12$ e $Pai_3 \leftarrow 6$**

j ← 4 $\Rightarrow C_{64} < R_4$ (?) $\Leftrightarrow 6 < \infty$ (SIM); logo **$R_4 \leftarrow 6$ e $Pai_4 \leftarrow 6$**

Passo 2:

k: $R_k = \min\{ R_1, R_2, R_3, R_4 \} = \min\{ 9, \infty, 12, 6 \} = 6 \Rightarrow$ **k ← 4**

Permanentes ← { 7, 8, 5, 6, 4 } Temporários ← { 1, 2, 3 }

Temporários ≠ ∅ ⇒ continuar

Passo 3:

k ← 4; j ← 2, 3 (6 é permanente, logo não é considerado)

j ← 2 ⇒ $C_{42} < R_2$ (?) ⇔ 8 < ∞ (SIM); logo **R_2 ← 8 e Pai_2 ← 4**

j ← 3 ⇒ $C_{43} < R_3$ (?) ⇔ 5 < 12 (SIM); logo **R_3 ← 5 e Pai_3 ← 4**

Passo 2:

k: $R_k = \min\{ R_1, R_2, R_3 \} = \min\{ 9, 8, 5 \} = 5 \Rightarrow$ **k ← 3**

Permanentes ← { 7, 8, 5, 6, 4, 3 } Temporários ← { 1, 2 }

Temporários ≠ ∅ ⇒ continuar

Passo 3:

k ← 3; j ← 1, 2 (4, 6 são permanentes, logo não são considerados)

j ← 1 ⇒ $C_{31} < R_1$ (?) ⇔ 4 < 9 (SIM); logo **R_1 ← 4 e Pai_1 ← 3**

j ← 2 ⇒ $C_{32} < R_2$ (?) ⇔ 10 < 8 (NÃO)

Passo 2:

k: $R_k = \min\{ R_1, R_2 \} = \min\{ 4, 8 \} = 4 \Rightarrow$ **k ← 1**

Permanentes ← { 7, 8, 5, 6, 4, 3, 1 } Temporários ← { 2 }

Temporários ≠ ∅ ⇒ continuar

Passo 3:

k ← 1; j ← 2 (3, 7 são permanentes, logo não são considerados)

j ← 2 ⇒ $C_{12} < R_2$ (?) ⇔ 7 < 8 (SIM); logo **R_2 ← 7 e Pai_2 ← 1**

Passo 2:

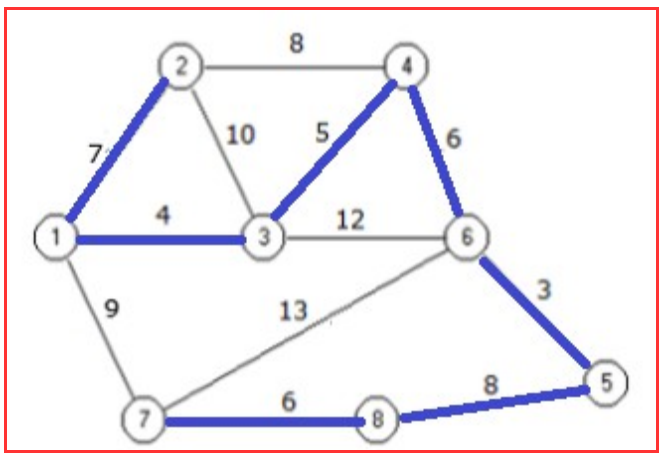
k: $R_k = \min\{ R_2 \} = \min\{ 7 \} = 7 \Rightarrow$ **k ← 2**

Permanentes ← { 7, 8, 5, 6, 4, 3, 1, 2 } Temporários ← { }

Temporarios = ∅ ⇒ PARAR

2.

Árvore Abrangente Mínima obtida tomando S = 7:



Custo total da AAM:

$C_{78} + C_{12} + C_{31} + C_{43} + C_{85} + C_{64} + C_{56} =$

6 + 7 + 4 + 5 + 6 + 8 + 3 = **39**