UNIVERSIDADE DA BEIRA INTERIOR

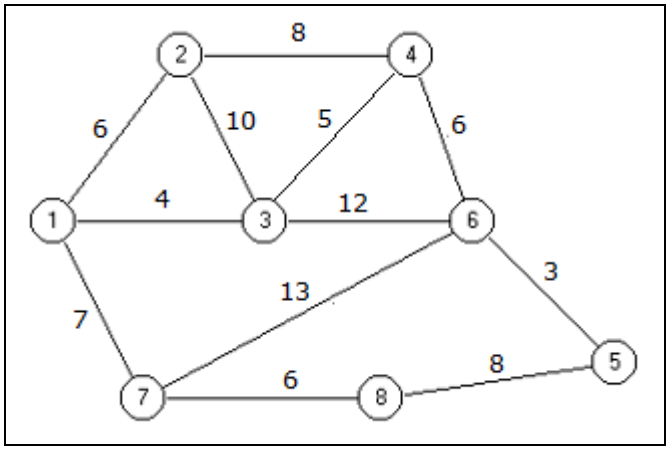
Algoritmos e Estruturas de Dados — **2º Semestre**

Frequência 2 — **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. Grafos

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



1. Determine a **Árvore dos Caminhos Mais Curtos (ACMC)** do **nó 1 (S = 1) para todos os outros nós**, simulando o algoritmo de **DIJKSTRA**. Apresente os dados da simulação: evolução das estruturas de dados (rótulo **R**, **Pai**, **Permanentes** e **Temporários**) e de todos os valores de **k** e de **j** (usar esquema de resposta em baixo).

S = 1

Passo1:

$[Pai_1, R_1] \leftarrow [1, 0]$

$[Pai_2, R_2] \leftarrow [1, 6]$ $[Pai_3, R_3] \leftarrow [1, 4]$ $[Pai_7, R_7] \leftarrow [1, 7]$

$[Pai_4, R_4] \leftarrow [1, \infty]$ $[Pai_5, R_5] \leftarrow [1, \infty]$ $[Pai_6, R_6] \leftarrow [1, \infty]$ $[Pai_8, R_8] \leftarrow [1, \infty]$

Permanentes ← { 1 }

Temporários ← { 2, 3, 4, 5, 6, 7, 8 }

	1	2	3	4	5	6	7	8
R	0 -	6 -	4 -	∞ 9 -	∞ 21 18 -	∞ 16 15 -	7 -	∞ 13 -
Pai	1 -	1 -	1 -	1 3 -	1 8 6 -	1 3 4 -	1 -	1 7 -

Passo 2:

Temporários ≠ ∅ ⇒ continuar

k: R_k = min{ R_2, R_3, R_4, R_5, R_6, R_7, R_8 } = min{ 6, 4, ∞, ∞, ∞, 7, ∞ } = 4 ⇒ **k ← 3**

Permanentes ← { 1, 3 }

Temporarios ← { 2, 4, 5, 6, 7, 8 }

Passo 3:

k ← 3; j ← 2, 4, 6 (1 é permanente, logo não é considerado)

j ← 2 ⇒ $R_3 + C_{32} < R_2$ (?) ⇔ 4 + 10 < 6 (NÃO); logo R_2 sem alteração

j ← 4 ⇒ $R_3 + C_{34} < R_4$ (?) ⇔ 4 + 5 < ∞ (SIM); logo alterar R_4 : **[Pai$_4$, R$_4$] ← [3, 9]**

j ← 6 ⇒ $R_3 + C_{36} < R_6$ (?) ⇔ 4 + 12 < ∞ (SIM); logo alterar R_6 : **[Pai$_6$, R$_6$] ← [3, 16]**

Passo 2:

Temporarios ≠ ∅ ⇒ continuar

k: R_k = min{ R_2, R_4, R_5, R_6, R_7, R_8 } = min{ 6, 9, ∞, 16, 7, ∞ } = 6 ⇒ **k ← 2**

Permanentes ← { 1, 3, 2 }

Temporarios ← { 4, 5, 6, 7, 8 }

Passo 3:

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

j ← 4 ⇒ $R_2 + C_{24} < R_4$ (?) ⇔ 6 + 8 < 9 (NÃO); logo R_4 sem alteração

Passo 2:

Temporarios ≠ ∅ ⇒ continuar

k: R_k = min{ R_4, R_5, R_6, R_7, R_8 } = min{ 9, ∞, 16, 7, ∞ } = 7 ⇒ **k ← 7**

Permanentes ← { 1, 3, 2, 7 }

Temporarios ← { 4, 5, 6, 8 }

Passo 3:

k ← 7; j ← 6, 8 (1 é permanente, logo não é considerado)

j ← 6 ⇒ $R_7 + C_{76} < R_6$ (?) ⇔ 7 + 13 < 16 (NÃO); logo R_6 sem alteração

j ← 8 ⇒ $R_7 + C_{78} < R_8$ (?) ⇔ 7 + 6 < ∞ (SIM); logo alterar R_8 : **[Pai$_8$, R$_8$] ← [7, 13]**

Passo 2:

Temporarios ≠ ∅ ⇒ continuar

k: R_k = min{ R_4, R_5, R_6, R_8 } = min{ 9, ∞, 16, 13 } = 9 ⇒ **k ← 4**

Permanentes ← { 1, 3, 2, 7, 4 }

Temporarios ← { 5, 6, 8 }

Passo 3:

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

j ← 6 ⇒ $R_4 + C_{46} < R_6$ (?) ⇔ 9 + 6 < 15 (SIM); logo alterar R_6 : **[Pai$_6$, R$_6$] ← [4, 15]**

Passo 2:

Temporarios ≠ ∅ ⇒ continuar

k: R_k = min{ R_5, R_6, R_8 } = min{ ∞, 15, 13 } = 13 ⇒ **k ← 8**

Permanentes ← { 1, 3, 2, 7, 4, 8 }

Temporarios ← { 5, 6 }

Passo 3:

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

j ← 5 ⇒ $R_8 + C_{85} < R_5$ (?) ⇔ 13 + 8 < ∞ (SIM); logo alterar R_5 : **[Pai$_5$, R$_5$] ← [8, 21]**

Passo 2:

Temporarios ≠ ∅ ⇒ continuar

k: $R_k = \min\{ R_5, R_6 \} = \min\{ 21, 15 \} = 15$ ⇒ **k ← 6**

Permanentes ← { 1, 3, 2, 7, 4, 8, 6 }

Temporarios ← { 5 }

Passo 3:

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

j ← 5 ⇒ $R_6 + C_{65} < R_5$ (?) ⇔ 15 + 3 < 21 (SIM); logo alterar R_5 : **[Pai$_5$, R$_5$] ← [6, 18]**

Passo 2:

Temporarios ≠ ∅ ⇒ continuar

k: $R_k = \min\{ R_5 \} = \min\{ 19 \} = 19$ ⇒ **k ← 5**

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

Temporarios ← { }

Passo 3:

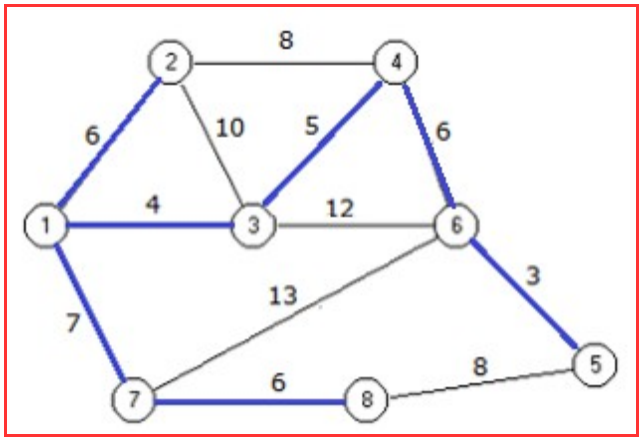
k ← 5; j ←

(6 e 8 são permanentes, logo não são considerados)

Passo 2:

Temporarios = ∅ ⇒ PARAR

Árvore dos Caminhos Mais Curtos do nó 1 para todos os outros:



2. Indique **o caminho mais curto** entre o nó **1** e o nó **5**, assim como o seu **comprimento**. Justifique usando os dados obtidos na simulação antes.

Caminho mais curto do nó 1 para o nó 5:

Valor: $R_5 = 18$

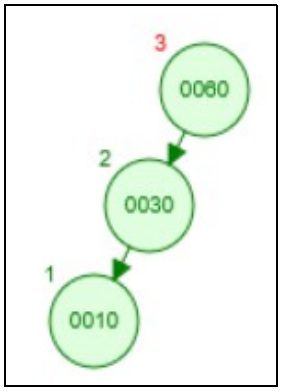
Caminho: [1, 3, 4, 6, 5]

Calcula-se andando de 5 para 1 usando os rótulos obtidos pelo algoritmo:

$5 \leftarrow Q_5=6 \leftarrow Q_6=4 \leftarrow Q_4=3 \leftarrow Q_3=1$

B. ABP Balanceadas/Equilibradas (AVL)

Inserir **60, 30** e **10**, por esta ordem:



A árvore fica **equilibrada** com a inserção dos nodos **60** e **30**

A árvore fica **desequilibrada** com a inserção do nodo **10**

O **desequilíbrio** acontece na árvore com raiz no nodo **60**, pois

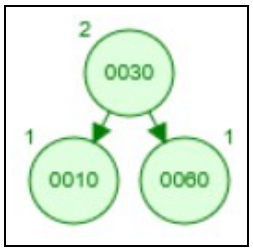
- o seu filho direito (**NULL**) tem altura 0
- o seu filho esquerdo (**30**) tem altura 2
- altura(filho esquerdo) – altura(filho direito) = 2 - 0 = 2

Todos as subárvores com raiz nos nodos descendentes do nodo **60** estão equilibradas

Como

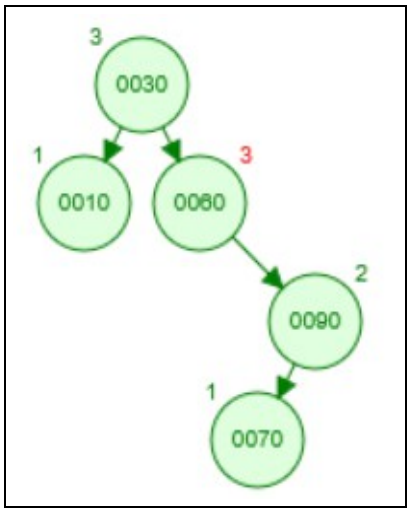
- o filho **esquerdo** do nodo **60** (**30**) tem **mais altura** que o filho **direito** do nodo **60** (**NULL**), e
- o filho **esquerdo** do nodo **30** (**10**) tem **mais altura** que o filho **direito** do nodo **30** (**NULL**),

então **reequilibrar** a árvore aplicando uma **rotação simples à direita** do nodo **60** usando o nodo **30**



A árvore está equilibrada.

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



A árvore fica equilibrada com a inserção dos nodos **90**

A árvore fica **desequilibrada** com a inserção do nodo **70**

O desequilíbrio acontece na árvore com raiz no nodo **60**, pois

- o seu filho esquerdo (**NULL**) tem altura o
- o seu filho direito (**90**) tem altura 2
- altura(filho direito) – altura(filho esquerdo) = 2 - 0 = 2

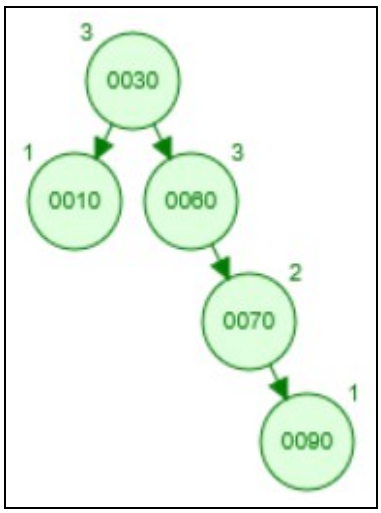
Todos as subárvores com raiz nos nodos descendentes do nodo **60** estão equilibradas

Como

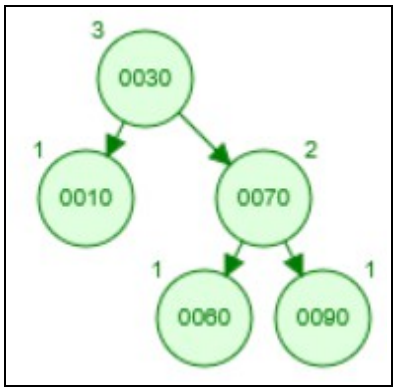
- o filho **direito** do nodo **60** (**90**) tem **mais altura** que o filho **esquerdo** do nodo **60** (**NULL**), e
- o filho **esquerdo** do nodo **90** (**70**) tem **mais altura** que o filho **direito** do nodo **90** (**NULL**),

então **reequilibrar** a árvore aplicando uma **rotação dupla à esquerda** do nodo **60**

1º) **rotação simples à direita** do nodo **90** usando o nodo **70**:

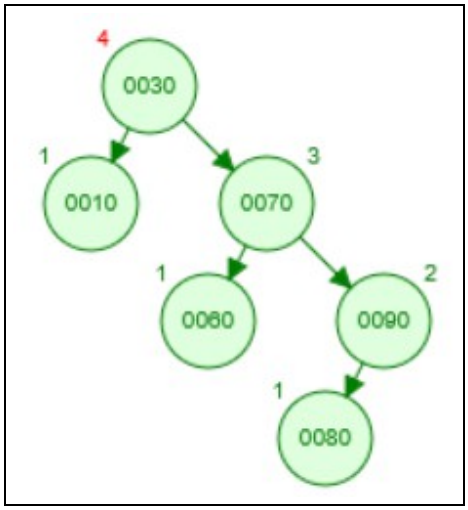


2º) **rotação simples à esquerda** do nodo **60** usando o nodo **70**:



A árvore está equilibrada.

Inserir **80**:



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

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

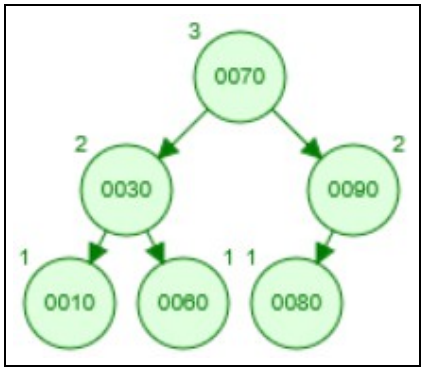
- o seu filho direito (**70**) tem altura 3
- o seu filho esquerdo (**10**) tem altura 1
- altura(filho direito) – altura(filho esquerdo) = 3 - 1 = 2

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

Como

- o filho **direito** do nodo **30** (**70**) tem **mais altura** que o filho **esquerdo** do nodo **30** (**10**), e
- o filho **direito** do nodo **70** (**90**) tem **mais altura** que o filho **esquerdo** do nodo **70** (**NULL**),

então **reequilibrar** a árvore aplicando uma **rotação simples à esquerda** do nodo **30** usando o nodo **70**



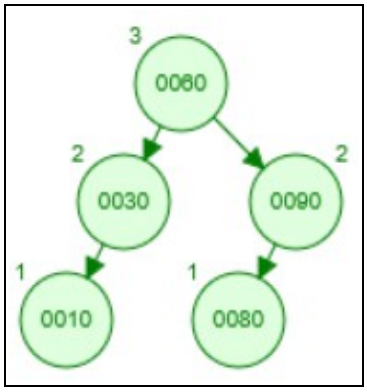
A árvore está equilibrada.

Remover o nodo 70 (raiz da ABP obtida)

Como o nodo **70** tem dois filhos, a remoção deste nodo pode ser feito de uma de duas forma:

1º caso:

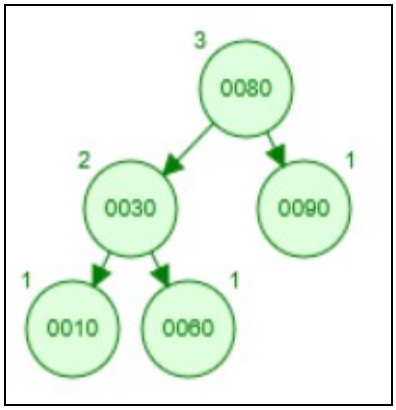
- copiar o valor do nodo com maior valor menor que 70 (60) para o nodo com 70 (raiz passa a 60),
- remover o nodo com 60



A árvore continua **equilibrada**

2º caso:

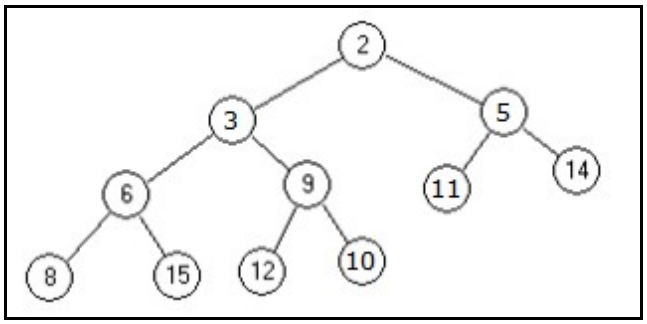
- copiar o valor do nodo com menor valor maior que 70 (80) para o nodo com 70 (raiz passa a 80),
- remover o nodo com 80



A árvore continua **equilibrada**

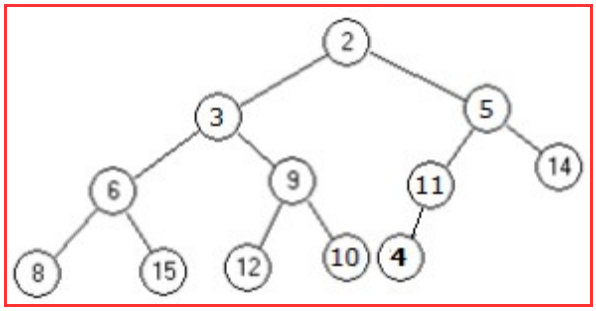
C. Heaps (Filas de Prioridade)

Considere o seguinte **minHeap** binário:



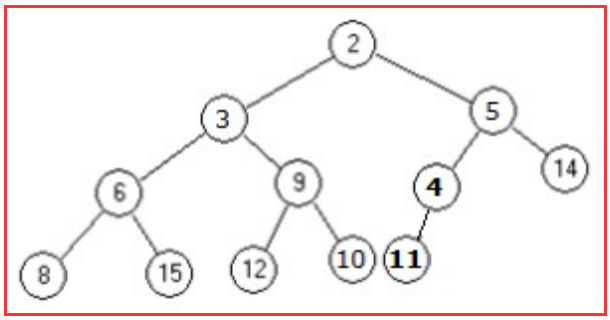
1. **Insira** um nodo com o valor **4** no minHeap dado e desenhe o minHeap obtido. Justifique, apresentando a evolução da minHeap desde a estrutura inicial até à final.

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

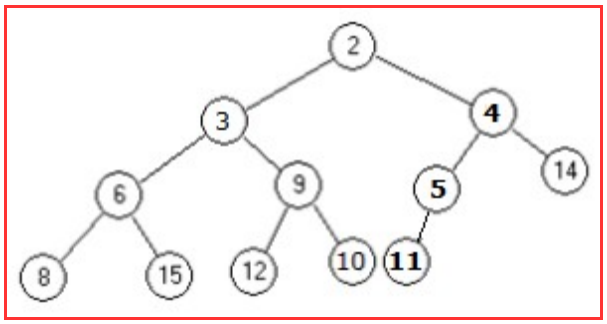


   2º) Comparar o valor inserido com o valor do seu nó pai e, se for melhor (menor), então trocá-los; realizar este processo até o valor inserido ficar no local correto

   a) como 4 < 11, então trocar o 4 com o 11



   b) como 4 < 5, então trocar o 4 com o 5

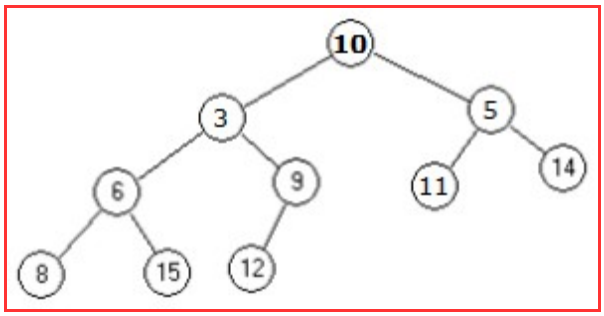


   c) como 4 > 2, então terminar processo; a árvore obtida em b) é a resultante da inserção de 4.

2. Aplique a operação **remover** ao minHeap dado (original) e redesenhe o minHeap obtido. Justifique, apresentando a evolução da minHeap desde a estrutura inicial até à final.

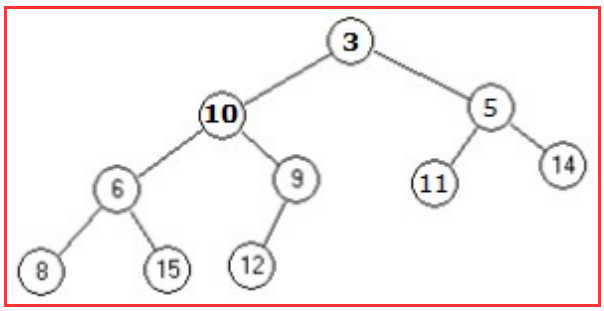
1º) Tomar o elemento que se encontra na raiz da minHeap (árvore): **2**

2º) Substituir o elemento que se encontra na raiz da heap (2), pelo elemento que se encontra no nó mais à direita do último nível da árvore (10) e de seguida remover este nó (com 10) da heap

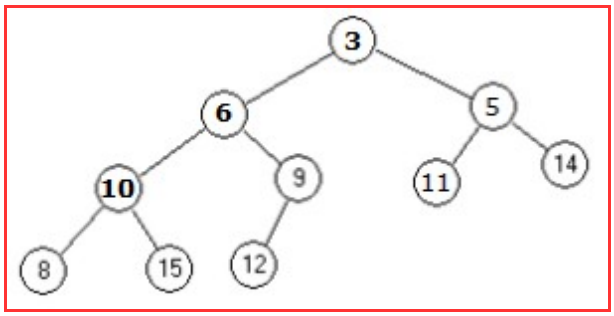


3º) Comparar o valor que se encontra na raiz (10) com o melhor (menor) dos seus filhos e, se for pior (maior), então trocá-los; realizar este processo até o valor da raiz (10) ficar no local correto

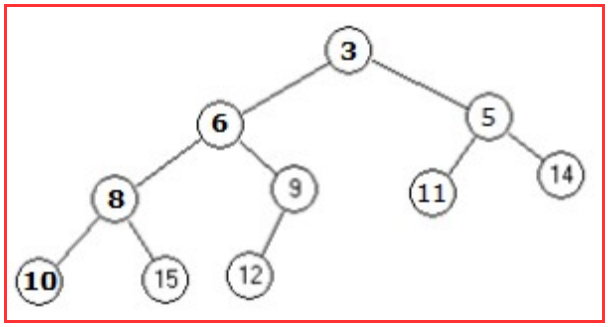
a) como 10 > 3 [= min(3, 5)], então trocar o 10 com o 3



b) como 10 > 6 [= min(6, 9)], então trocar o 10 com o 6



c) como 10 > 8 [= min(8, 15)], então trocar o 10 com o 8



Como o nó com 10 não tem filhos, o processo termina

D. Árvores Binárias de Pesquisa (ABP)

// Pergunta 1

```
int quantElementos (PNodoABP T, int X)
{
   if (T == NULL)
      return  0;
   if (T->Elemento > X)
      return  1 + quantElementos(T->Esquerda, X) + quantElementos(T->Direita, X);
   else
      return  quantElementos(T->Direita, X);
}
```

// Pergunta 2

```
int ligacoesEntreXY (PNodoABP T, int X, int Y)
{
   int cont = 0, k;
   PNodoABP P;

   if (T == NULL)
      return -1;

   if (T->Elemento < X)
      return ligacoesEntreXY (T->Direita, X, Y);
   if (T->Elemento > Y)
      return ligacoesEntreXY (T->Esquerda, X, Y);

   if (T->Elemento > X) {       // != X
      P = T->Esquerda;
      k = 1;
      while (P != NULL && P->Elemento != X){
         k = k + 1;
         if (P->Elemento < X)
            P = P->Direita;
         else
            P = P->Esquerda;
      }
      if (P == NULL)
         return -1;
      else
         cont = cont + k;
   }

   if (T->Elemento < Y) {       // != Y
      P = T->Direita;
      k = 1;
      while (P != NULL && P->Elemento != Y){
         k = k + 1;
         if (P->Elemento < Y)
            P = P->Direita;
         else
            P = P->Esquerda;
      }
      if (P == NULL)
         return -1;
      else
         cont = cont + k;
   }

   return cont;
}
```