UNIVERSIDADE DA BEIRA INTERIOR

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

Exame Época Normal **Resolução** **2024/2025**

A. Análise de complexidade dos algoritmos

1.

k	k >= 1 (S/N)	j	j < k (S/N)	printf
(k=n) **n**	n >= 1 (S)	(j=0) **0** (j++) **1** . . . (j++) **n-1** (j++) **n**	0 < n (S) 1 < n (S) . . . n-1 < n (S) n < n (N)	OLA OLA . . . OLA
(k = k-1) **n-1**	n-1 >= 1 (S)	(j=0) **0** (j++) **1** . . . (j++) **n-2** (j++) **n-1**	0 < n-1 (S) 1 < n-1 (S) . . . n-2 < n-1 (S) n-1 < n-1 (N)	OLA OLA . . . OLA
...	...	...	...	...
(k=k-1) **2**	2 >= 1 (S)	(j=0) **0** (j++) **1** (j++) **2**	0 < 2 (S) 1 < 2 (S) 2 < 2 (N)	OLA OLA
(k=k-1) **1**	1 >= 1 (S)	(j=0) **0** (j++) **1**	0 < 1 (S) 1 < 1 (N)	OLA
(k=k-1) **0**	0 >= 1 (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+1)** [k=n] + **n** [k=n-1] + ... + **3** [k=2] + **2** [k=1] + **0** [k=0] =

(n+1) + **n** + ... + **3** + **2** [progressão aritmética com **n** termos] =

((**n+1 + 2**) / 2) x **n** = ((n + 3) / 2) x n = **(n^2 + 3n) / 2**

Logo, a ordem de complexidade é: **O(n^2)** (ou: **T(n) = O(n^2)**)

Podia-se também contabilizar todas as operações, somando o número de vezes que cada uma das operações é executada.

B. Listas ligadas

1.

```
PNodoLista ponteiroMaiorElemento (PNodoLista L) {
    PNodoLista  P;
    if (L == NULL)   // caso base/terminal
        return  NULL;
    // caso geral
    P = ponteiroMaiorElemento(L->Prox);
    if(P == NULL)
        return L;
    if (L->Elemento > P->Elemento)
        return  L;
    else
        return  P;
}
```

2.

```
PNodoLista  removerPositivos  (PNodoLista  L) {
    PNodoLista  PAnt, P, PRem;
    if (L == NULL)  return  L;
    while (L != NULL  &&  L->Elemento > 0) {     // remover os positivos na cabeça da lista L
        PRem = L;
        L = L->Prox;
        PRem = libertarNodoLista(PRem);
    }
    if ( L == NULL)  return L;
    // remover os restantes positivos da lista L, sendo que na cabeça está um elemento <= 0
    P = L->Prox;     // ponteiro para percorrer todos os restantes nodos da lista
    PAnt = L;        // ponteiro para o nodo anterior ao nodo apontado por P
    while (P != NULL) {
        if (P->Elemento > 0) {
            PRem = P;
            PAnt->Prox = P->Prox;      // PAnt não avança
            PRem = libertarNodoLista(PRem);
        }
        else
            PAnt = P;         // PAnt avança para P
        P = PAnt->Prox;    // P avança para o próximo de PAnt
    }
    return  L;
}
```

```
PNodoFila alterarFila (PNodoFila Q)
{
   int  fundo, cont;
   PNodoFila  filaPos, filaNeg;
   if (filaVazia(Q) == 0)
      return Q;
   // colocar os elementos >= 0 na pilha PPos e os < 0 na pilha PNeg
   filaPos = criarFila();
   filaNeg = criarFila();
   while (filaVazia(Q) == 0) {
      if(frente(Q) >= 0)
         filaPos = juntar(frente(Q), filaPos);
      else
         filaNeg = juntar(frente(Q), filaNeg);
      Q = remover(Q);
   }
   // colocar os elementos da filaPos (>= 0) na fila Q
   while (filaVazia(filaPos) == 0) {
      Q = juntar(frente(filaPos), Q);
      filaPos = remover(filaPos);
   }
   // acrescentar os elementos da filaNeg (< 0) na fila Q
   while (filaVazia(filaNeg) == 0) {
      Q = juntar(frente(filaNeg), Q);
      filaNeg = remover(filaNeg);
   }
   return Q;
}
```

D. Árvores Binárias de Pesquisa (ABP)

1.

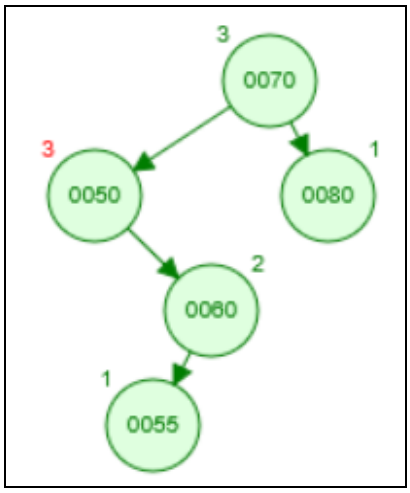
```
int  numeroFolhasMenoresX (PNodoABP T, int X)
{
   // casos base
   if (T == NULL)
      return 0;
   if (T->Esquerda == NULL && T->Direita == NULL && T->Elemento < X)
      return 1;   // T é uma folha e tem valor < X
   // caso geral
   if (T->Elemento < X)
      return numeroFolhasMenoresX(T->Esquerda, X) + numeroFolhasMenoresX(T->Direita, X);
   else
      return numeroFolhasMenoresX(T->Esquerda, X);
}
```

2.

```
PNodoABP ponteiroFilhoNodoXMaiorValor (PNodoABP T, int X)
{
   if (T == NULL)
      return  NULL;
   while (T != NULL)
   {
      if (T->Elemento == X) {    //  encontrou o nodo com o valor X
         if (T->Direita == NULL)
            return  T->Esquerda;   // mesmo que seja NULL
         else
            // como T tem filho direito, então este é o maior
            return  T->Direita;
      }
      if (T->Elemento > X)
         T = T->Esquerda;
      else
         T = T->Direita;
   }
   return NULL;   // não existe nodo com o valor X
}
```

E. ABP Balanceadas/Equilibradas (AVL)

Inserir **70, 50, 80, 60** e **55**, por esta ordem:



A árvore fica **equilibrada** com a inserção dos nodos **70, 50, 80** e **60**

A árvore fica **desequilíbrada** com a inserção do nodo **55**

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

- o seu filho direito (nodo **60**) tem altura 2
- o seu filho esquerdo (NULL) tem altura 0
- altura(filho direito) – altura(filho esquerdo) = 2 - 0 = 2

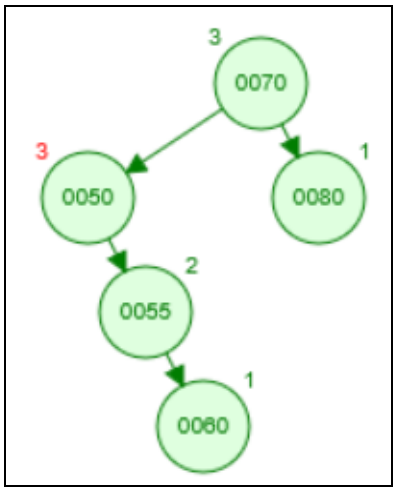
Todos as subárvores com raiz nos nodos descendentes do nodo **50** estão equilibradas

Como

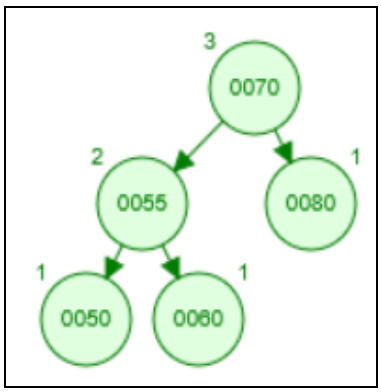
- o filho **direito** do nodo **50** (nodo **60**) tem **mais altura** que o filho **esquerdo** do nodo **50** (**NULL**), e
- o filho **esquerdo** do nodo **60** (nodo **55**) tem **mais altura** que o filho **direito** do **60** (**NULL**),

então **reequilibrar** a árvore aplicando uma **rotação dupla à esquerda** do nodo **50**, que consiste em realizar duas rotações simples seguidas:

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

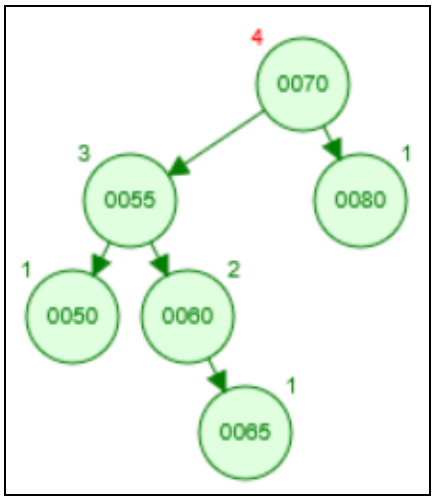


2º) rotação simples à esquerda do nodo 50 usando o nodo 55



A árvore está equilibrada.

Inserir **65**:



A árvore fica **desequilíbrada** com a inserção do nodo **65**

O desequilibrio acontece na árvore com raiz no nodo **70**, pois

- o seu filho esquerdo (nodo **55**) tem altura 3
- o seu filho direito (nodo **80**) tem altura 1
- altura(filho esquerdo) – altura(filho direito) = 3 - 1 = 2

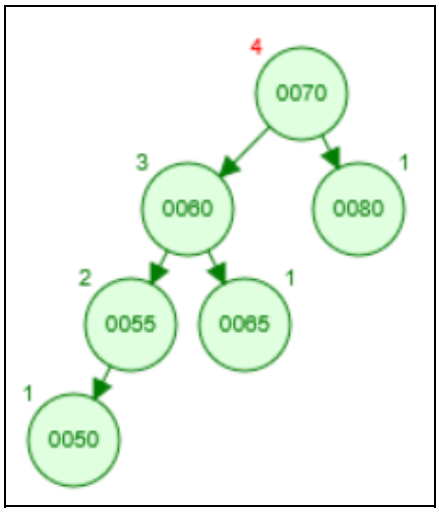
Todos as subárvores com raiz nos nodos descendentes do nodo **70** estão equilibradas

Como

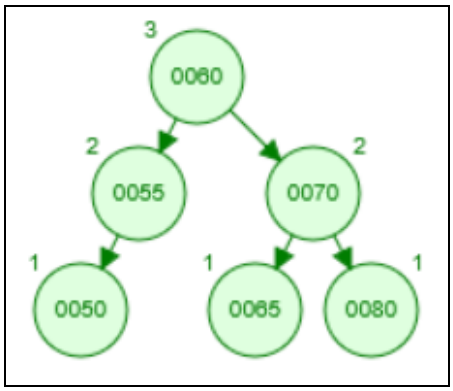
- o filho **esquerdo** do nodo **70** (nodo **55**) tem **mais altura** que o seu filho **direito** (nodo **80**), e
- o filho **direito** do nodo **55** (nodo **60**) tem **mais altura** que o seu filho **esquerdo** (nodo **50**),

então **reequilibrar** a árvore aplicando uma **rotação dupla à direita** do nodo **70,** que consiste em realizar duas rotações simples seguidas:

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

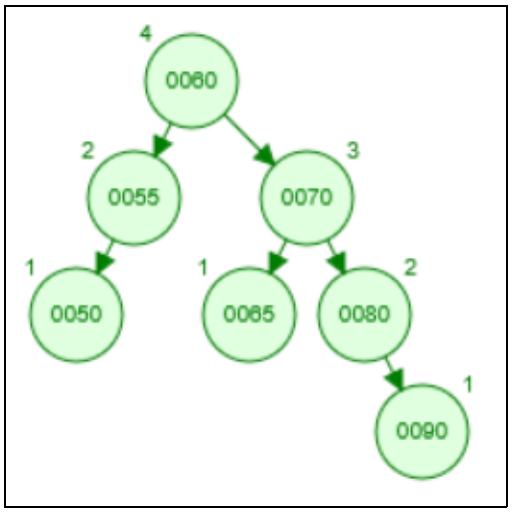


2º) **rotação simples à direita** do nodo **80** usando o nodo **75**:



A árvore está equilibrada.

Inserir **90**:



A árvore está equilibrada.

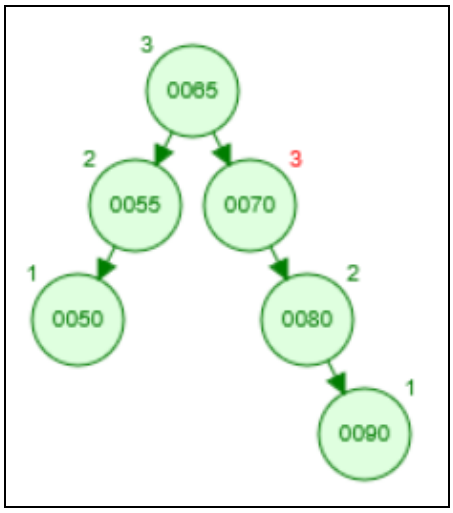
Remover a raiz da ABP (nodo 60):

Remover a raiz da árvore (nodo **60**) implica substituir o elemento **60** por um dos seguintes elementos:

- **65** (menor elemento da subárvore direita – subárvore com os maiores que 60) [Caso 1]
- **55** (maior elemento da subárvore esquerda – subárvore com os menores que 60) [Caso 2]

e de seguida remover o nodo substituto.

Caso 1 (substituir pelo elemento **65**):



A árvore fica **desequilíbrada** com a remoção do nodo **60** e a substituição pelo nodo **65**

O desequilibrio acontece na árvore com raiz no nodo **70**, pois

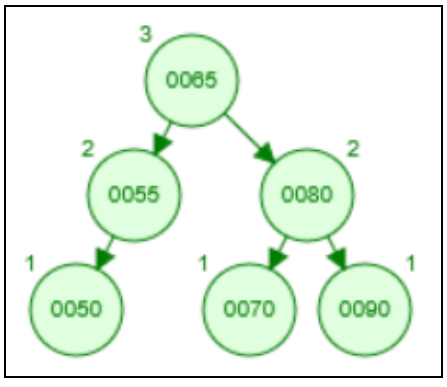
- o seu filho direito (nodo **80**) tem altura 2
- o seu filho esquerdo (**NULL**) tem altura 0
- altura(filho direito) – altura(filho esquerdo) = 2 - 0 = 2

Todos as subárvores com raiz nos nodos descendentes do nodo **70** estão equilibradas

Como

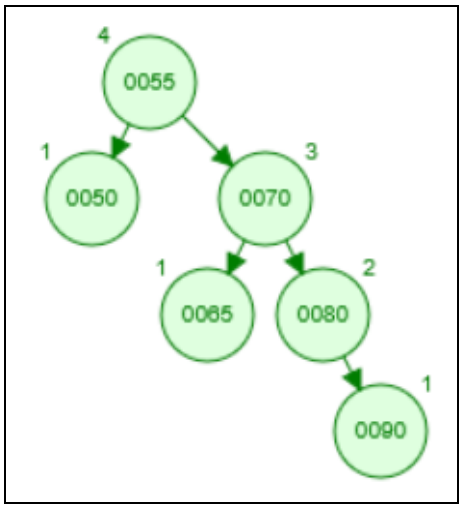
- o filho **direito** do nodo **70** (nodo **80**) tem **mais altura** que o seu filho **esquerdo** (**NULL**), e
- o filho **direito** do nodo **80** (nodo **90**) tem **mais altura** que o seu filho **esquerdo** (**NULL**),

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



A árvore está equilibrada.

Caso 2 (substituir pelo elemento **55**):



A árvore fica **desequilíbrada** com a remoção do nodo **60** e a substituição pelo nodo **55**

O desequilibrio acontece na árvore com raiz no nodo **55**, pois

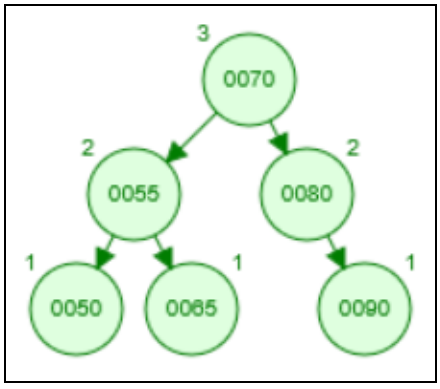
- o seu filho direito (nodo **70**) tem altura 3
- o seu filho esquerdo (nodo **50**) tem altura 1
- altura(filho direito) – altura(filho esquerdo) = 3 - 1 = 2

Todos as subárvores com raiz nos nodos descendentes do nodo **55** estão equilibradas

Como

- o filho **direito** do nodo **55** (nodo **70**) tem **mais altura** que o seu filho **esquerdo** (nodo **50**), e
- o filho **direito** do nodo **70** (nodo **80**) tem **mais altura** que o seu filho **esquerdo** (nodo **65**),

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



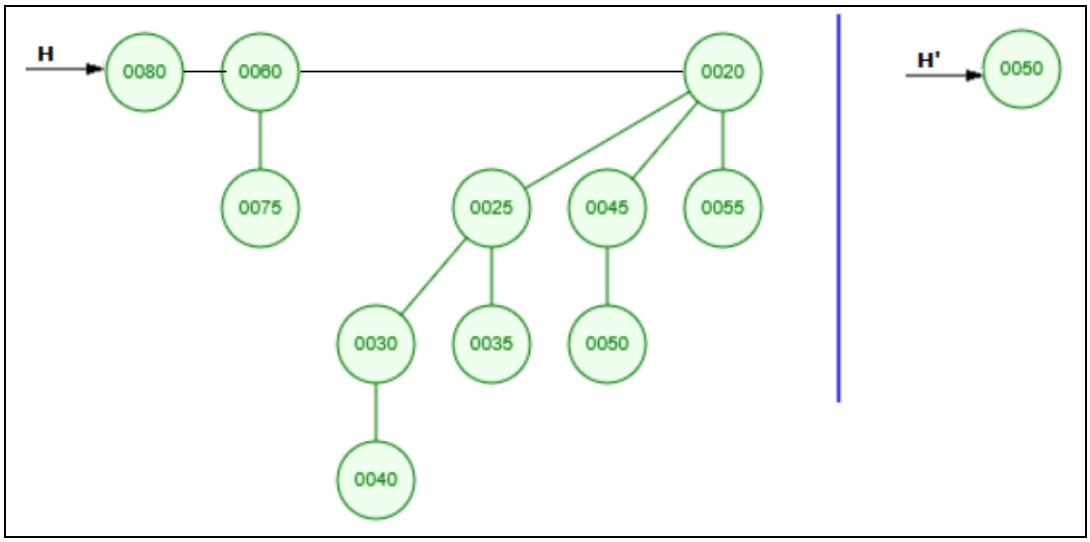
A árvore está equilibrada.

F. Heaps

1.

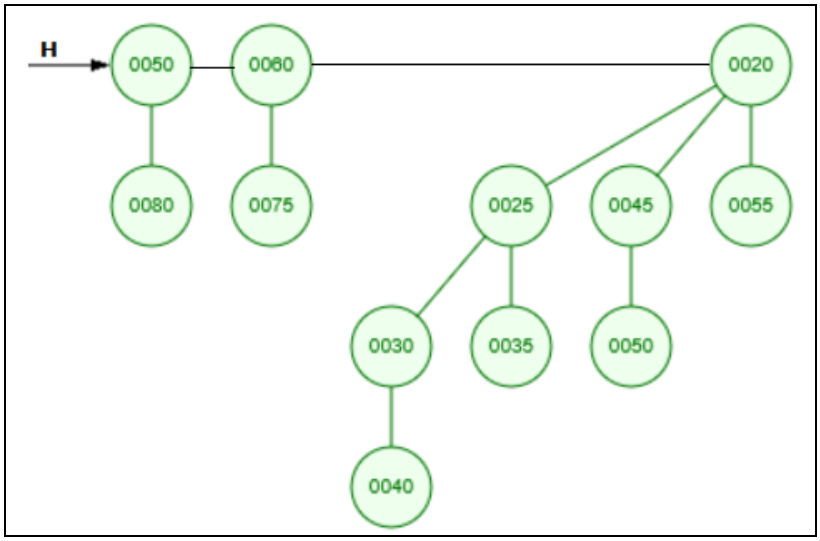
O processo é o seguinte:

1º) criar um heap H' apenas com um nodo com o valor **50**

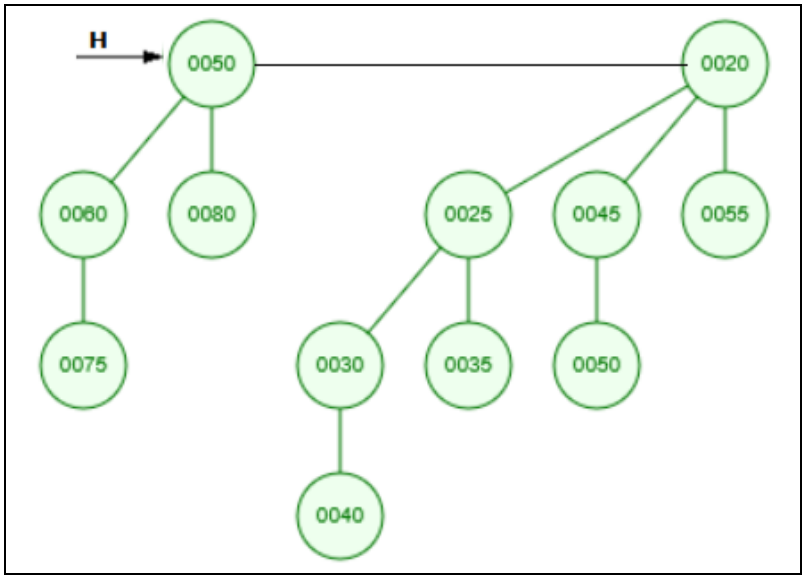


2º) aplicar a operação **união** entre os dois heaps, H e H':

- como existem 2 árvores binomiais de grau 0 (B0) cujas raizes têm os valores **50** e **80**, então aplicar a operação **união** entre árvores binomiais fomando uma árvore de grau 1 (B1) com raiz com o valor **50** (min{50, 80})



- como agora existem 2 árvores binomiais de grau 1 (B1) cujas raizes têm os valores **50** e **60**, então aplicar a operação **união** entre árvores binomiais fomando uma árvore de grau 2 (B2) com raiz com o valor **50** (min{50, 60})



- como agora existem 2 árvores binomiais de diferentes graus (uma de grau 2 (B2) com raiz com o valor **50** e a outra de grau 3 (B3) com raiz com o valor **20**), o processo termina

2.

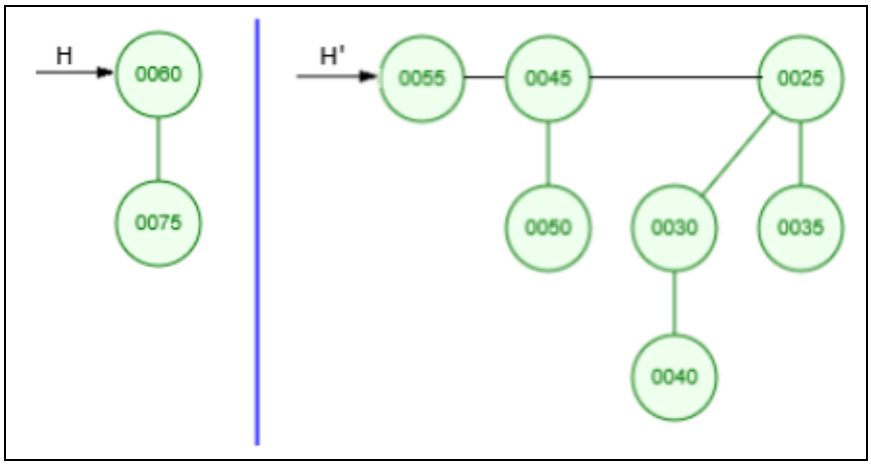
O processo é o seguinte:

1º) extrair o menor valor entre as raizes do heap H: min{ 80, 20 } = **20**

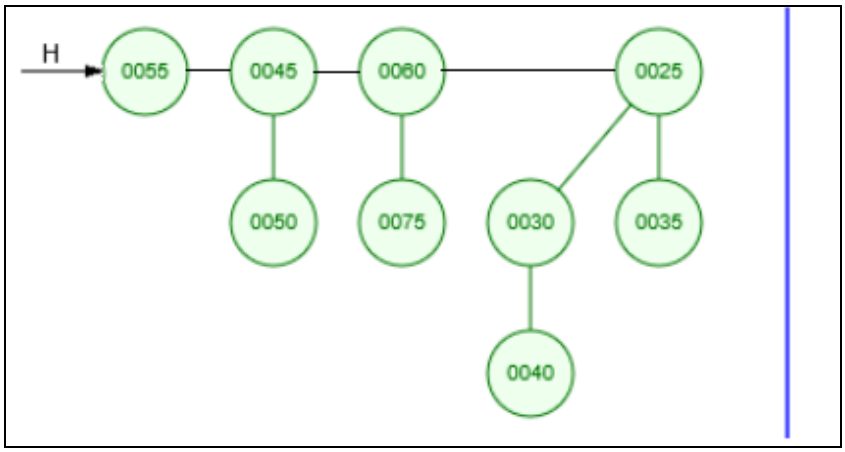
2º) remover o nodo raiz com o valor **20** (menor raiz),

- remover a árvore binomial com raiz com valor **20** do heap H, ficando H apenas com a árvore binomial de grau 1 (B1) com raiz com o valor **60**

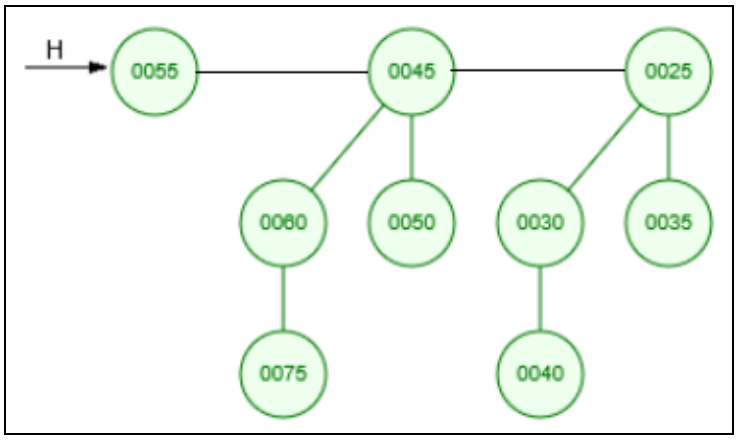
- criar um novo heap H' com os filhos da árvore com raiz com o valor **20**, o qual é formado por 3 árvores binomiais de graus 0, 1 e 2 (B0 com raiz com o valor **55**, B1 com raiz com o valor **45** e B2 com raiz com o valor 2**5**)



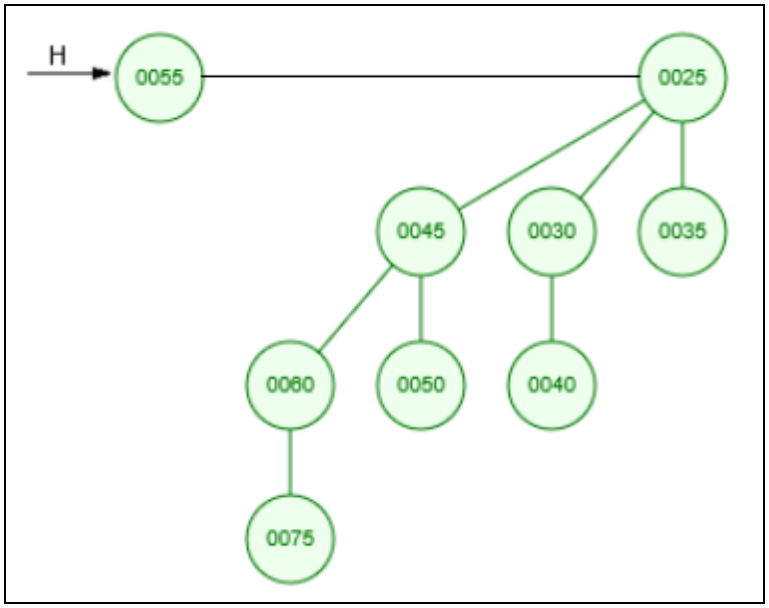
3º) aplicar a operação **união** entre os dois heaps, H e H':



- como existem 2 árvores binomiais de grau 1 (B1) cujas raizes têm os valores 45 e 60, então aplicar a operação **união** entre estas duas árvores binomiais para fomar uma árvore de grau 2 (B2) com raiz com o valor 45 (min{45, 60})



– como existem 2 árvores binomiais de grau 2 (B2) cujas raizes têm os valores 45 e 25, então aplicar a operação **união** entre estas duas árvores binomiais para fomar uma árvore de grau 3 (B3) com raiz com o valor 25 (min{45, 25})



- a árvore resultante da operação "remover" é formada por duas árvores binomiais de graus 0 e 3 (B0 com raiz com o valor 55 e B2 com raiz com o valor 25).

D. Grafos

1.

Fluxo atual = fluxo que sai do nó S = fluxo que chega ao nó T

Como S = 1, fluxo que sai do nó 1 = $x_{12} + x_{13} + x_{14} = 5 + 4 + 4 = 13$

2.

	1	2	3	4	5	6
R(k)	[1+, ∞]	[3-, 1]	[1+, 2]	[3-, 2]	[2+, 1]	[5+, 1]

Nós **rotulados** e **não varridos** ← 1 – 3 – 2 – 4 – 5 - 6

Nós **rotulados** e **varridos** ← **1 - 3 – 2 – 4 - 5**

Passo1:

R1 ← [1+, ∞]

Passo 2:

j ← 1 [1 rotulado e não varrido]

k ← 2, 3, 4 (arcos de 1 para k)

2 --> x12 < b12 <=> 5 < 5 NÃO, logo não é possível rotular o nó 2

3 --> x13 < b13 <=> 4 < 6 SIM, logo R(3) = [1+, min{ ∞, 6-4 }] = [1+, 2]

4 --> x14 < b14 <=> 4 < 4 NÃO, logo não é possível rotular o nó 4

k ← ? (arcos de k para 1 não existem)

j ← 3 [3 rotulado e não varrido]

k ← 5, 6 (arcos de 3 para k)

5 --> x35 < b35 <=> 1 < 1 NÃO, logo não é possível rotular o nó 5

6 --> x36 < b36 <=> 6 < 6 NÃO, logo não é possível rotular o nó 6

k ← 1, 2, 4 (arcos de k para 3)

1 --> nó 1 já rotulado

2 --> x23 > 0 <=> 1 > 0 SIM, logo R(2) = [3-, min{ 2, 1 }] = [3-, 1]

4 --> x43 > 0 <=> 2 > 0 SIM, logo R(4) = [3-, min{ 2, 2 }] = [3-, 2]

j ← 2 [2 rotulado e não varrido] (podia ser o nó 4)

k ← 3, 5, 6 (arcos de 2 para k)

3 --> nó 3 já rotulado

5 --> x25 < b25 <=> 2 < 4 SIM, logo R(5) = [2+, min{ 1, 4-2 }] = [2+, 1]

6 --> x26 < b26 <=> 2 < 2 NÃO, logo não é possível rotular o nó 6

k ← 1 (arcos de k para 2)

1 --> nó 1 já rotulado

j ← 4 [4 rotulado e não varrido] (podia ser o nó 5)

k ← 3, 6 (arcos de 4 para k)

3 --> nó 3 já rotulado

6 --> x46 < b46 <=> 2 < 2 NÃO, logo não é possível rotular o nó 6

k ← 1 (arcos de k para 4)

1 --> nó 1 já rotulado

j ← 5 [5 rotulado e não varrido]

k ← 6 (arcos de 5 para k)

6 --> x56 < b56 <=> 3 < 5 SIM, logo R(6) = [5+, min{ 1, 5-2 }] = [5+, 1]

k ← 2, 3 (arcos de k para 5)

2 --> nó 2 já rotulado

3 --> nó 3 já rotulado

Como o nó T = 6 já está rotulado, foi encontrado um c.a.f.; logo, passar ao Passo 3.

Passo 3:

R(T=6) = [5+, 1] => k = 5 e Q(6) = 1

x56 = x56 + Q(6) = 3 + 1 = 4

k=5 ≠ S=1 SIM

R(5) = [2+, 1] => x25 = x25 + Q(6) = 2 + 1 = 3

k = 2

k=2 ≠ S=1 SIM

R(2) = [3-, 2] => x23 = x23 - Q(6) = 1 - 1 = 0

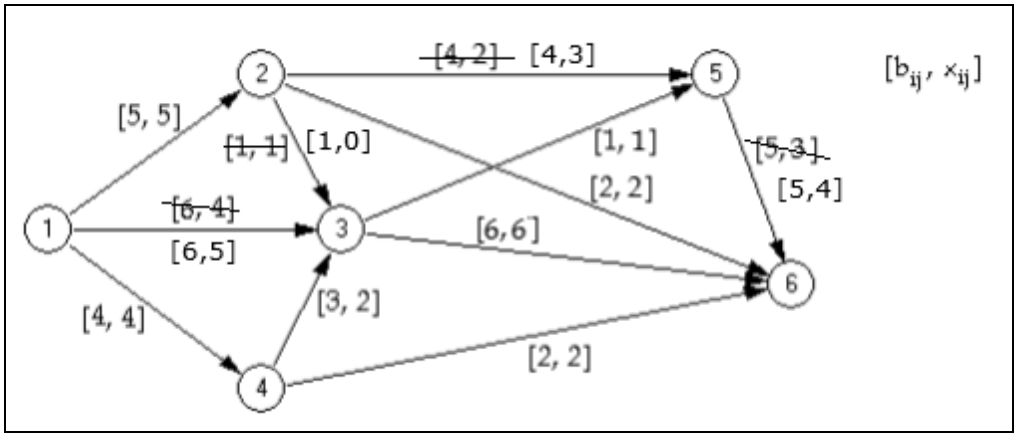
k = 3

k=3 ≠ S=1 SIM

R(3) = [1+, 2] => x13 = x13 + Q(6) = 4 + 1 = 5

k = 1

k=1 ≠ S=1 NÃO => Atualização da rede terminado



Nova iteração do algoritmo de Ford-Fulkerson, aplicado à rede atualizada.

	1	2	3	4	5	6
R(k)	[1+, ∞]		[1+, 1]	[3-, 1]		

Nós **rotulados** e não varridos ← 1 – 3 - 4

Nós **rotulados** e **varridos** ← **1 – 3 - 4**

Passo1:

R1 ← [1+, ∞]

Passo 2:

j ← 1 [1 rotulado e não varrido]

k ← 2, 3, 4 (arcos de 1 para k)

2 --> x12 < b12 <=> 5 < 5 NÃO, logo não é possível rotular o nó 2

3 --> x13 < b13 <=> 5 < 6 SIM, logo R(3) = [1+, min{ ∞, 6-5 }] = [1+, 1]

4 --> x14 < b14 <=> 4 < 4 NÃO, logo não é possível rotular o nó 4

j ← 3 [3 rotulado e não varrido]

k ← 5, 6 (arcos de 3 para k)

5 --> x35 < b35 <=> 3 < 3 NÃO, logo não é possível rotular o nó 5

6 --> x36 < b36 <=> 6 < 6 NÃO, logo não é possível rotular o nó 6

k ← 1, 2, 4 (arcos de k para 3)

1 --> nó 1 já rotulado

2 --> x23 > 0 <=> 1 > 0 NÃO, logo não é possível rotular o nó 2

4 --> x43 > 0 <=> 2 > 0 SIM, logo R(4) = [3-, min{ 1, 2 }] = [3-, 1]

j ← 4 [4 rotulado e não varrido]

k ← 3, 6 (arcos de 4 para k)

3 --> nó 3 já rotulado

6 --> x46 < b46 <=> 2 < 2 NÃO, logo não é possível rotular o nó 6

k ← 1 (arcos de k para 4)

1 --> nó 1 já rotulado

Como o nó T = 6 **não está rotulado e não é possível rotular** (todos os nós rotulados estão varridos), então **não existe c.a.f.**; logo, o fluxo atual é máximo ==> PARAR

Fluxo máximo = fluxo que sai do nó S = fluxo que chega ao nó T

Como S = 1, fluxo que sai do nó 1 = x12 + x13 + x14 = 5 + 5 + 4 = 14