Filas de Prioridade

Heap de Fibonacci

Definição

Heap de Fibonacci

- É uma coleção de árvores enraizadas (com raizes independentes), em que todas elas devem manter a propriedade de minHeap (ou todas maxHeap)
- As árvores que o formam podem ter qualquer forma
  - um nó pode ter mais do que dois filhos ou nenhum
  - as árvores podem ter todas apenas um único nó
- As árvores que o formam estão ligadas entre si pelas respetivas raizes, através de uma lista circular com início (cabeça) no nó com o elemento com maior prioridade (menor ou maior valor) do heap (que é raiz de uma árvore do heap)
- Difere dos heaps Binário e Binomial, pois
  - no heap Binário a única árvore que existe é binária
  - no heap Binomial cada árvore é uma árvore Binomial
- Em termos de complexidade é melhor do que o Binário e o Binomial

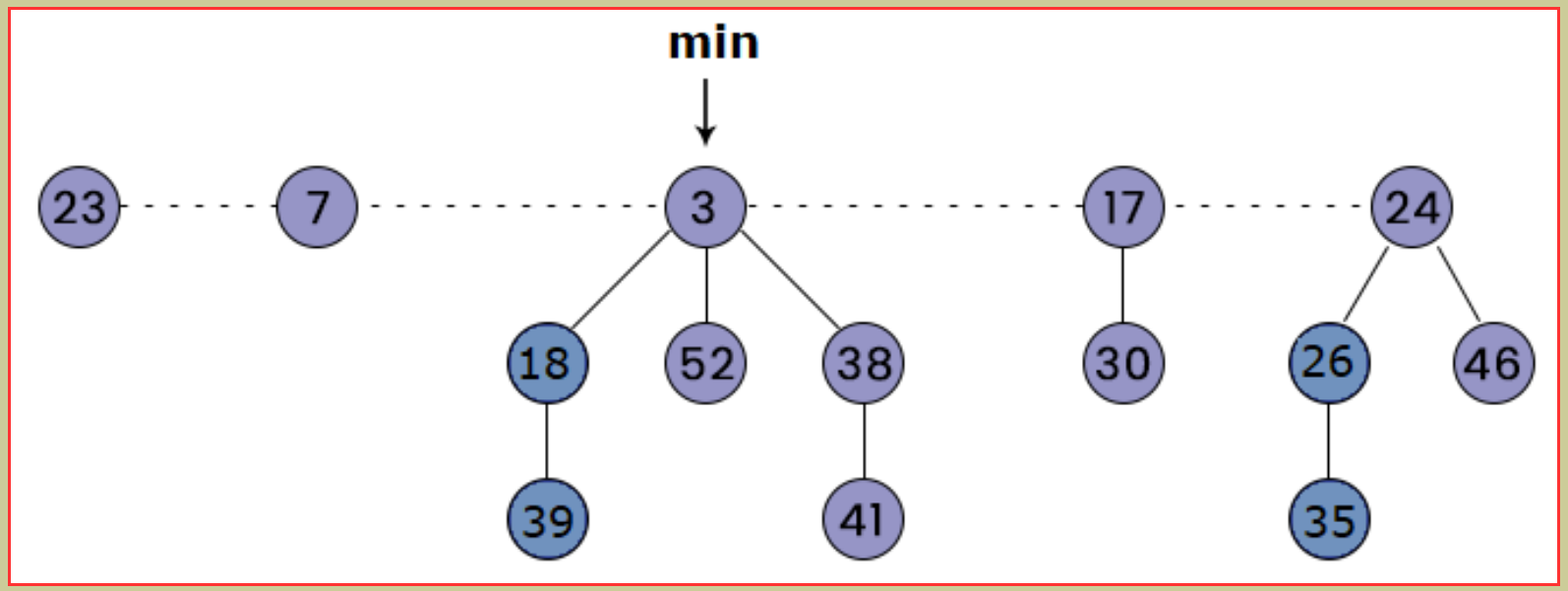
Heap de Fibonacci

- O nome de Fibonacci está relacionado com o facto das árvores serem construídas de tal forma que, uma árvore de grau **n** tem pelo menos F(n+2) nós
  - F(n+2) é o (n+2)-ésimo número de Fibonacci
- Sequência de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
  - F(0) = 0
  - F(1) = 1

    e
  - F(n) = F(n-1) + F(n-2), n ≥ 2
- Exemplos:
  - uma árvore de grau 0 (árvore só com um nó) tem pelo menos F(2) = 1 nó
  - uma árvore de grau 1 tem pelo menos F(3) = 2 nós
  - uma árvore de grau 2 tem pelo menos F(4) = 3 nós
  - uma árvore de grau 3 tem pelo menos F(5) = 5 nós
  - uma árvore de grau 4 tem pelo menos F(6) = 8 nós

Exemplo

- Representação gráfica de um heap de Fibonacci com propriedades de minHeap



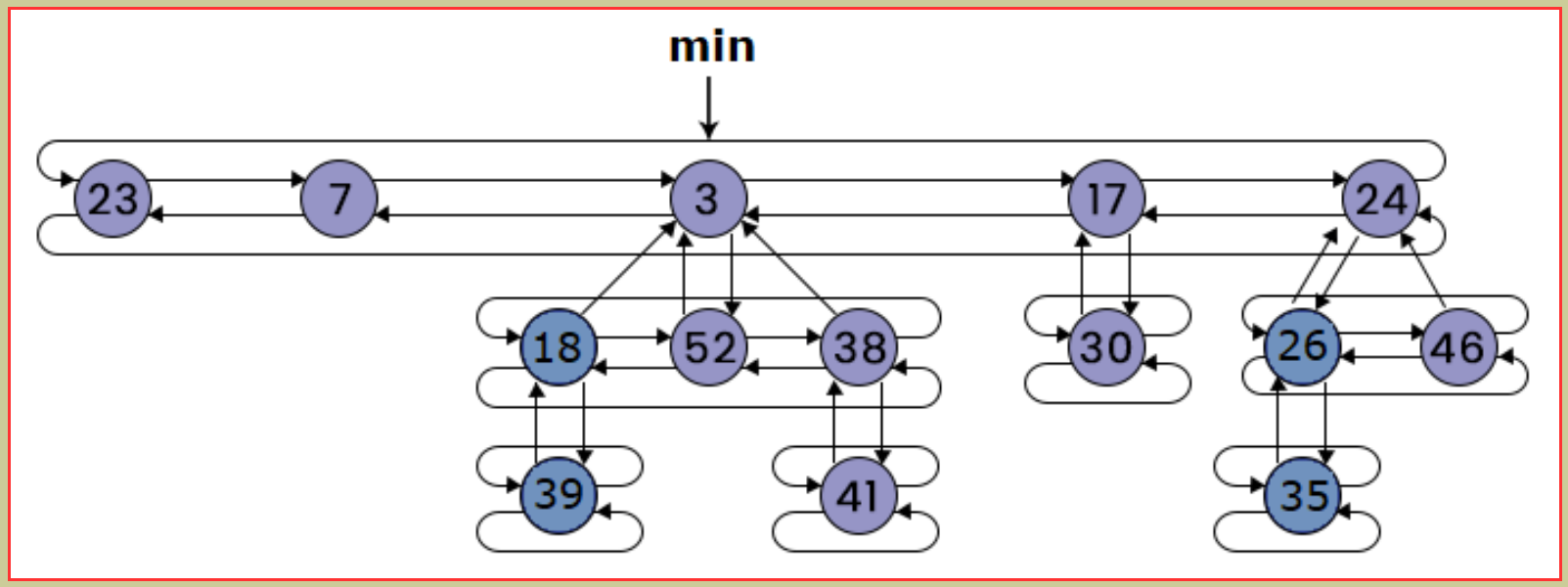
- heap de Fibonacci com 14 nós distribuídos por 5 árvores (com raizes independentes), todas com propriedades de minHeap
- as raizes das árvores estão ligadas através de uma lista circular (a tracejado)
- o nó com o elemento com maior prioridade é o nó com o valor 3 (menor valor), que é apontado pelo ponteiro **min** (ponteiro para o início da lista de raizes do heap)

Implementação computacional

Representação computacional

- As raízes de todas as árvores estão ligadas entre si para um acesso rápido
- Os nós filhos de um nó pai estão ligados entre si através de uma lista circular duplamente ligada
- Duas vantagens principais de se usar uma lista circular duplamente ligada:
  - remover um nó de uma árvore é uma operação com complexidade O(1)
  - a concatenação de duas destas listas é uma operação com complexidade O(1)

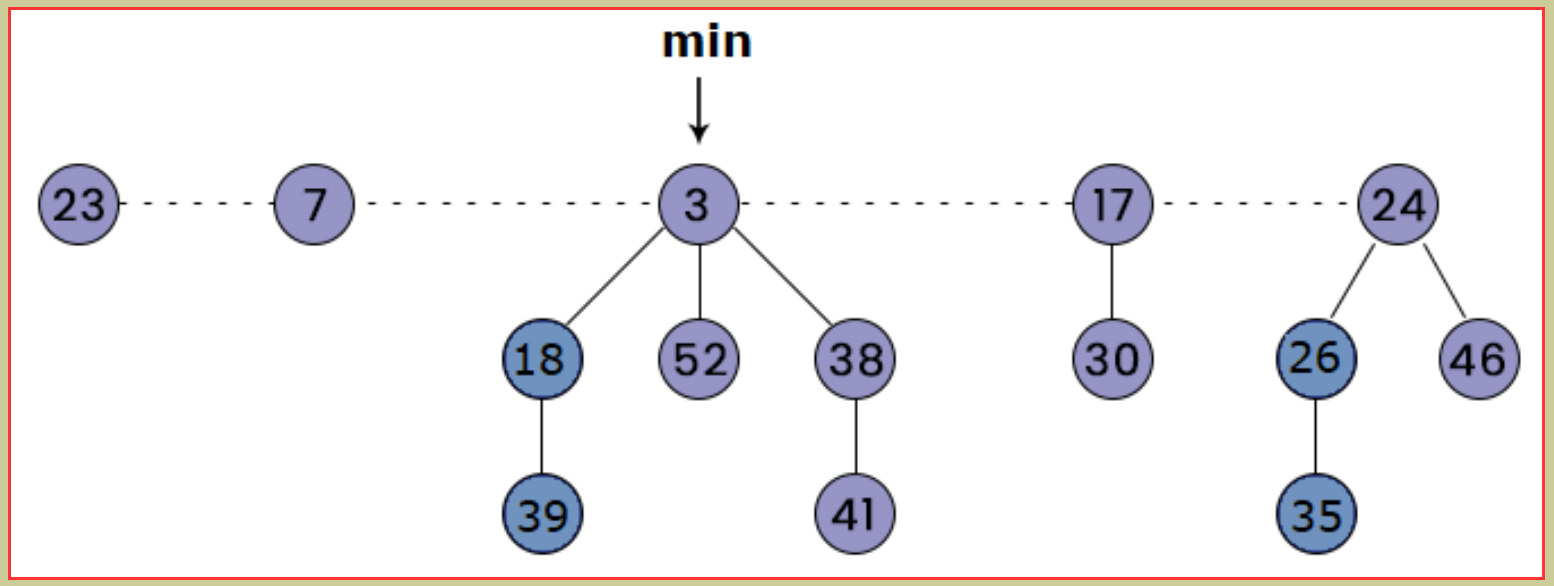
Exemplo



- Cada nó contém quatro ponteiros
  - um aponta para o pai (para cima)
  - um aponta para o filho (para baixo)
  - um aponta para o irmão do lado esquerdo (para a esquerda)
  - um aponta para o irmão do lado direito (para a direita)
- Uma lista circular duplamente ligada para as raizes das árvores (lista de raizes)
- Uma lista circular duplamente ligada entre o nós irmãos de cada nó das árvores

Operação principal consultar(H) - minHeap

- Consultar o elemento com maior prioridade de um heap H
  - devolve o elemento que se encontra na cabeça da lista de raizes, que corresponde à raiz com o elemento com maior prioridade (se minHeap, é o menor valor)
  - exemplo:



  - devolve o valor 3

Operação principal inserir(x,H) - minHeap

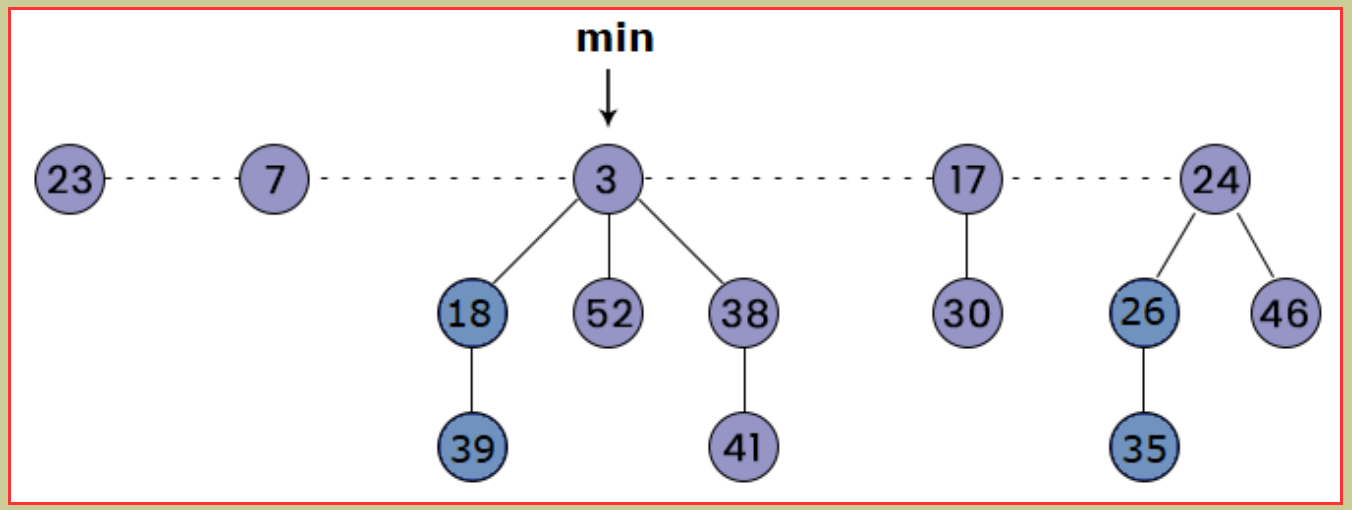
- Inserir um elemento x num heap H
  - consiste em inserir no heap uma árvore apenas com um nó (com o elemento x)

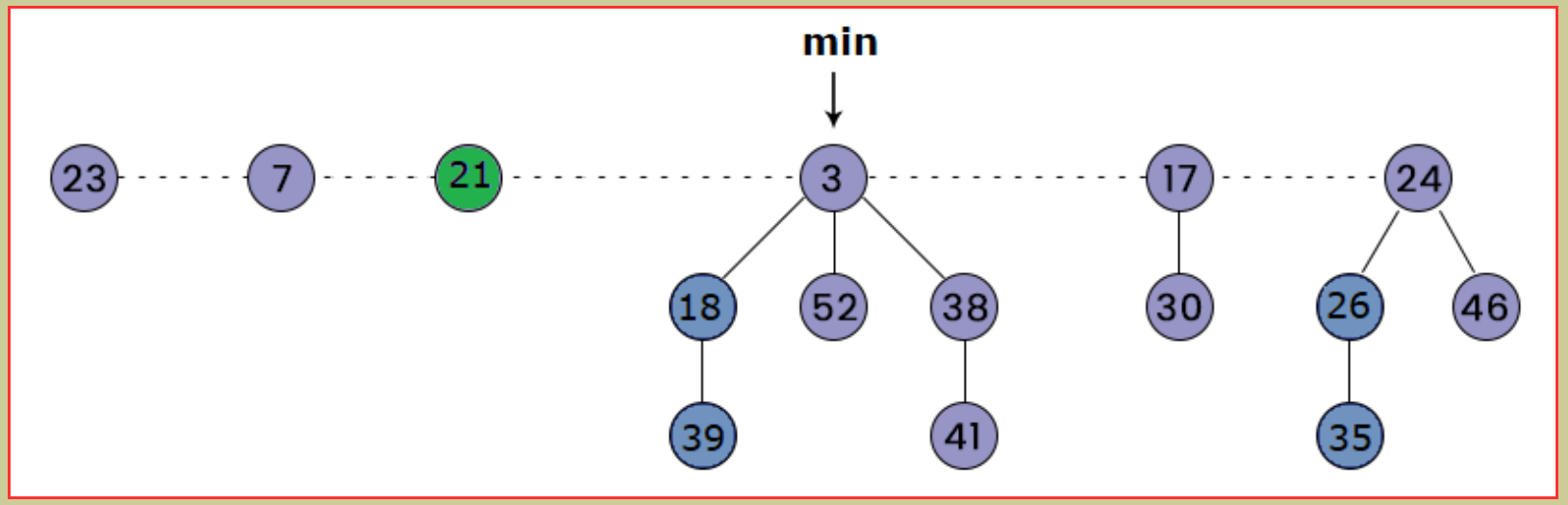
```
algoritmo inserir (x, H)
   criar um novo nodo com o elemento x (árvore só com um nodo)
   se (o heap H é vazio) então
      criar a lista de raizes, sendo o novo nodo a cabeça da lista (marcar com min)
   senão
      inserir o novo nodo na lista de raizes
      atualizar a cabeça da lista (min)
   fim_se
fim_algoritmo
```

  - o local de inserção do nó deve ser aquele que realize menos operações
    - sendo a lista de raizes duplamente ligadas e circular, o mais adequado é ser logo a seguir à cabeça ou antes dela
  - atualização do valor mínimo: nodo da cabeça ou novo nó (estão seguidos na lista)

Operação principal inserir(x, H) - minHeap

- Inserir um elemento **x** num heap H (exemplo: inserir o 21)





Operação principal remover(H) - minHeap

- Esta operação consiste em realizar as seguintes acções:
  - extrair o elemento com menor valor da chave (maior prioridade)
  - remover o nodo com o elemento com menor valor da chave (maior prioridade)
  - reajustar o heap, aplicando a operação **união** entre as árvores com o mesmo grau

Operação principal remover(H) - minHeap

- Algoritmo

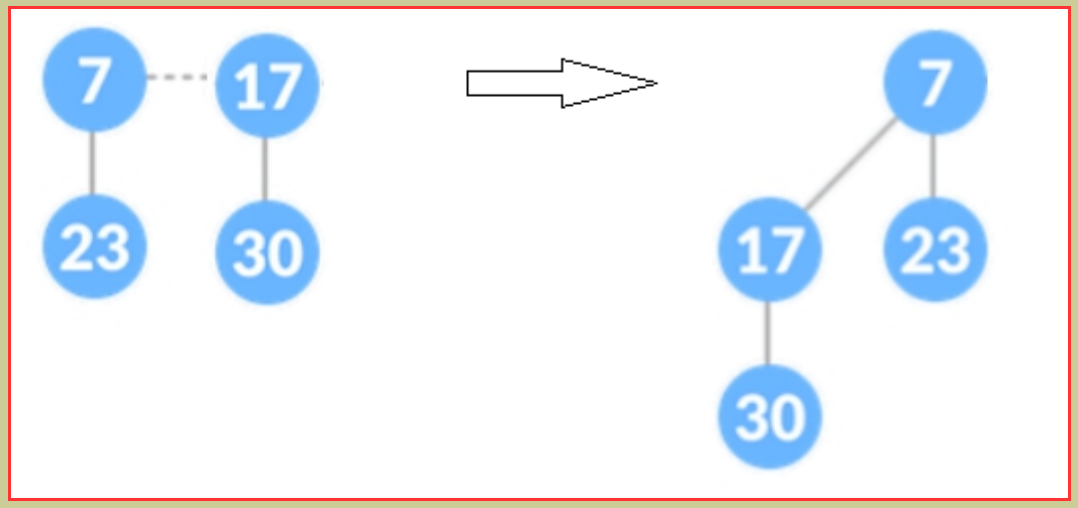
algoritmo remover(H)
1. extrair o elemento da lista com o menor valor da chave (raiz apontada por **min**)
2. remover a raiz com o menor valor da chave (apontada por **min**)
3. colocar o ponteiro **min** a apontar para a próxima raiz da lista
4. acrescentar os filhos da raiz removida à lista de raizes
5. criar um array de tamanho igual ao máximo de graus das árvores antes da remoção
fazer
6. mapear o grau da raiz atual (ponteiro **min**) no array de graus
7. mapear os graus das raizes seguintes da lista de raizes no array de graus
enquanto houver raizes com mapeamento no array (menor para maior grau) **fazer**
se há árvores com o mesmo grau **então**
8. aplicar a operação **união** a essas árvores, mantendo a propriedade minHeap
fim_se
fim_enquanto
enquanto houver várias árvores (raízes) com o mesmo grau
fim_algoritmo

Operação principal remover(H) - minHeap

- Operação auxiliar **união** de duas árvores de uma minHeap

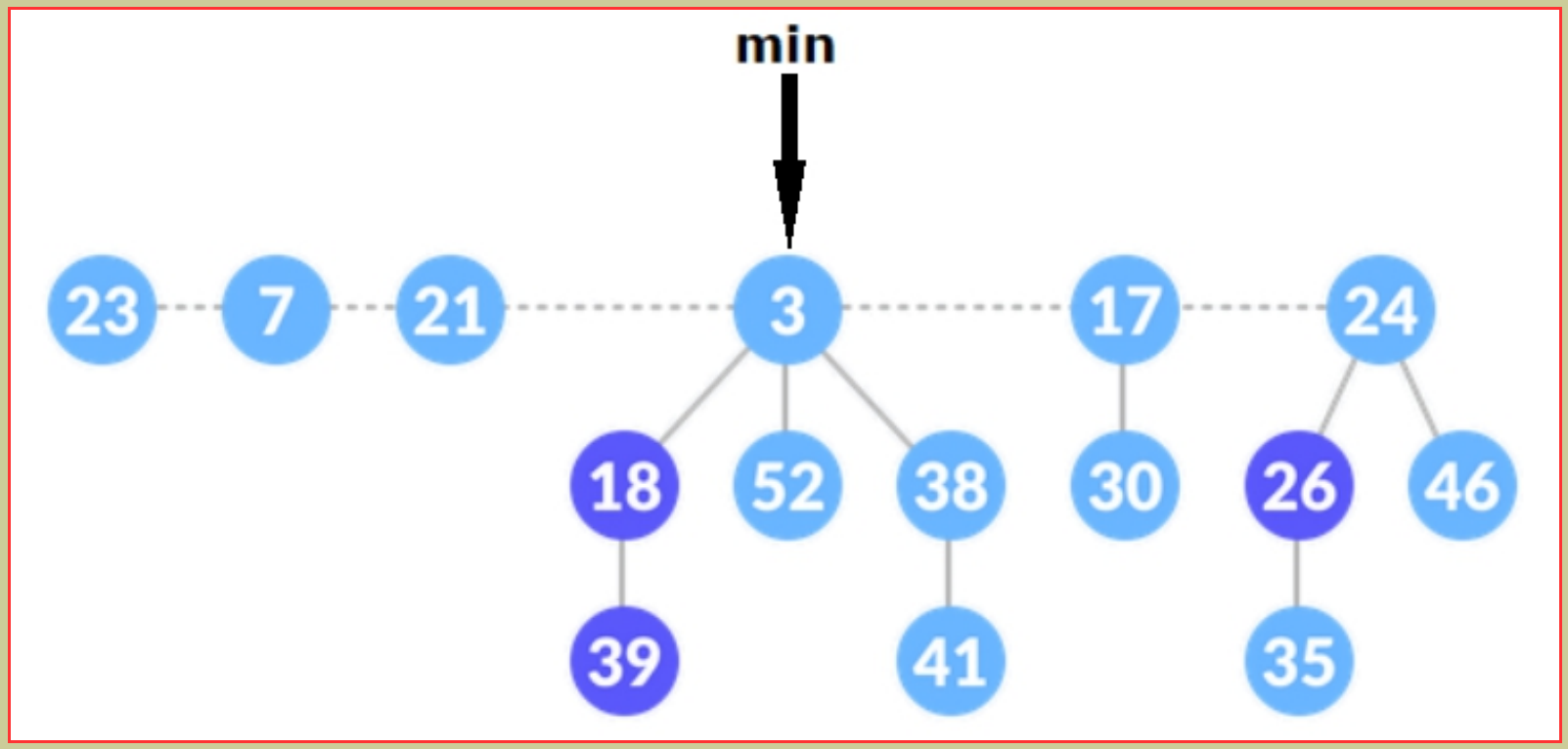
```
algoritmo união (T1, T2)
   se raiz(T1) < raiz(T2) então
      T2 passa a ser filho esquerdo de T1
   senão
      T1 passa a ser filho esquerdo de T2
   fim_se
fim_algoritmo
```

- exemplo: **união** de duas árvores de grau 1 de uma minHeap, formando uma de grau 2


7
17
23
30
7
17
23
30

Operação principal remover(H) - minHeap

- Exemplo:

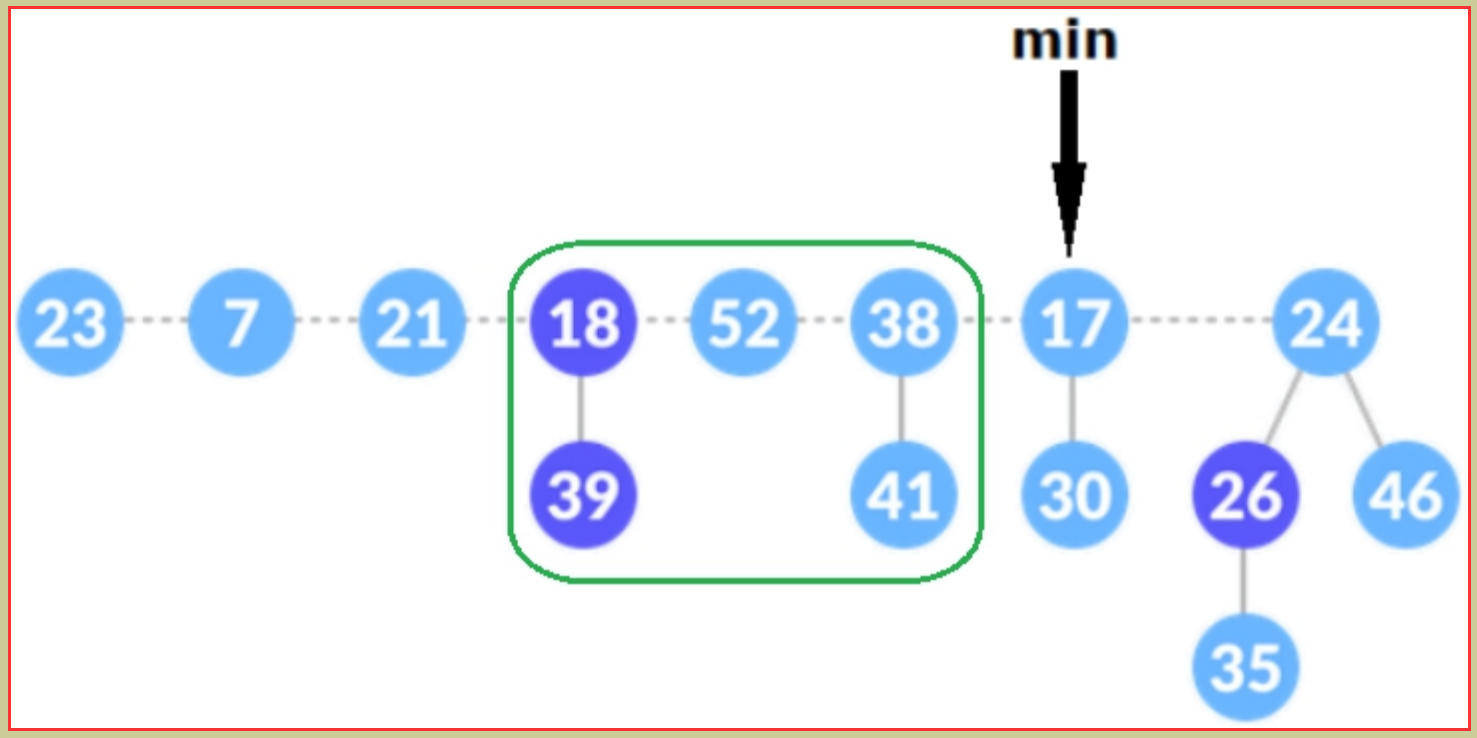


1. extrair o elemento com a chave 3 (contido no nodo apontado por **min**)

Operação principal remover(H) - minHeap

- Exemplo:

2. remover a raiz cujo elemento tem a chave 3 (menor valor) apontado por **min**
3. colocar o ponteiro **min** a apontar para a próxima raiz da lista de raizes (chave 17)
4. acrescentar os filhos da raiz removida à lista das raizes



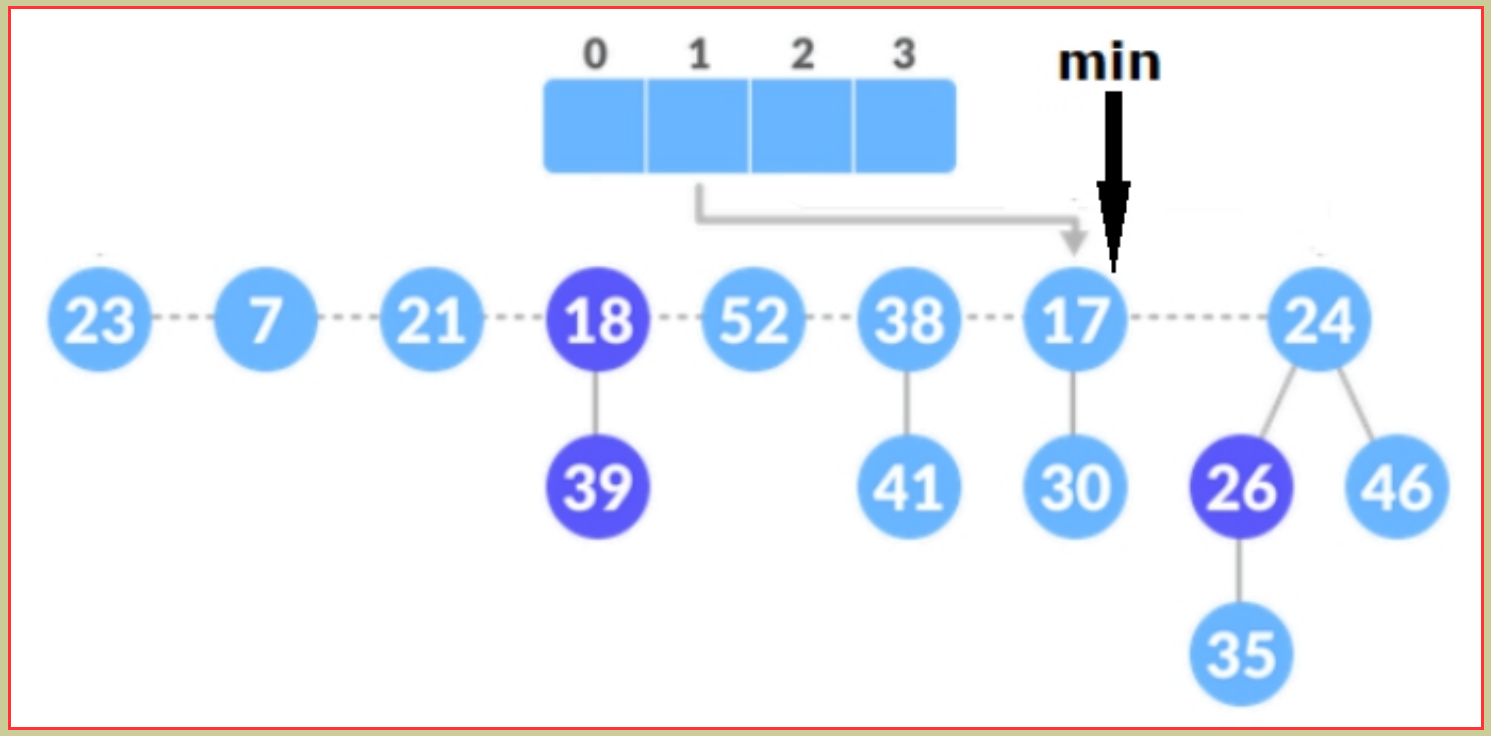
Operação principal remover(H) - minHeap

- Exemplo:

5. criar um array de graus de tamanho 4 (0 a 3), pois o grau máximo da árvore é 3

6. mapear o grau da árvore com raiz apontada por **min** no array de graus

- a árvore enraizada no nodo com a chave 17 (**min**) tem grau 1

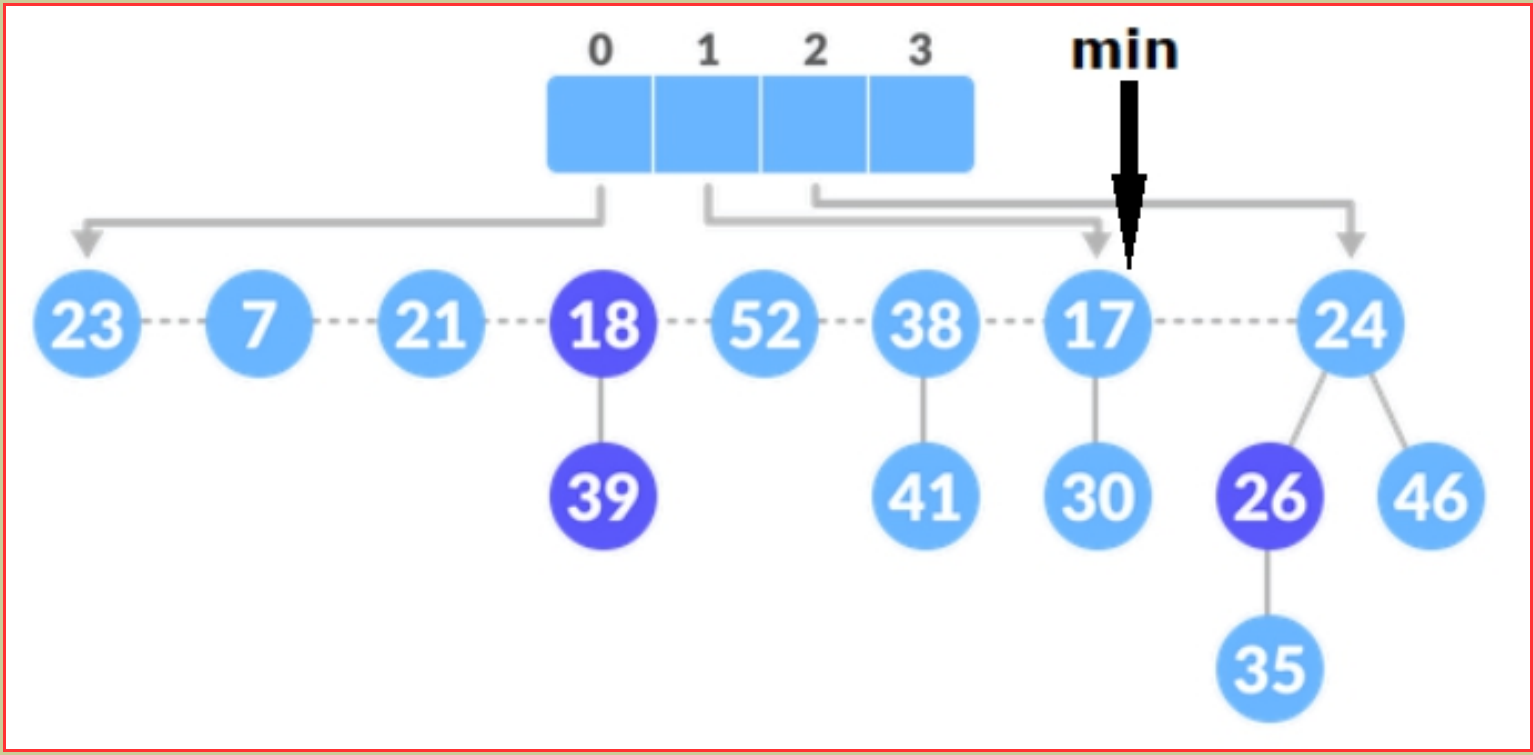


array de graus: [-; 17; -; -]

Operação principal remover(H) - minHeap

- Exemplo:

7. mapear o grau das árvores enraizadas nas próximas raizes da lista no array de graus
- a árvore enraizada no nodo com a chave 24 (**min→prox**) tem grau 2
- a árvore enraizada no nodo com a chave 23 (**min→prox→prox**) tem grau 0

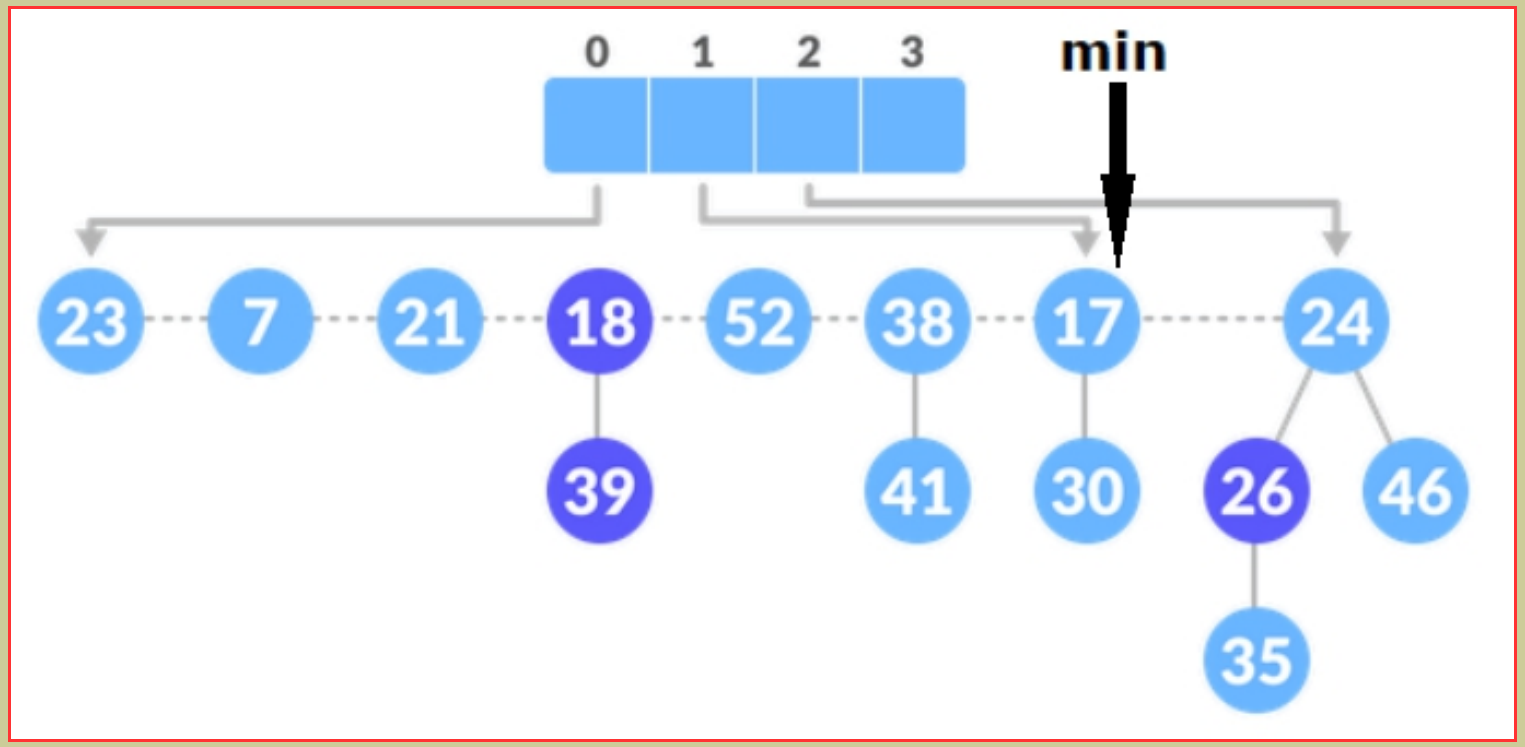


array de graus: [23; 17; 24; -]

Operação principal remover(H) - minHeap

- Exemplo:

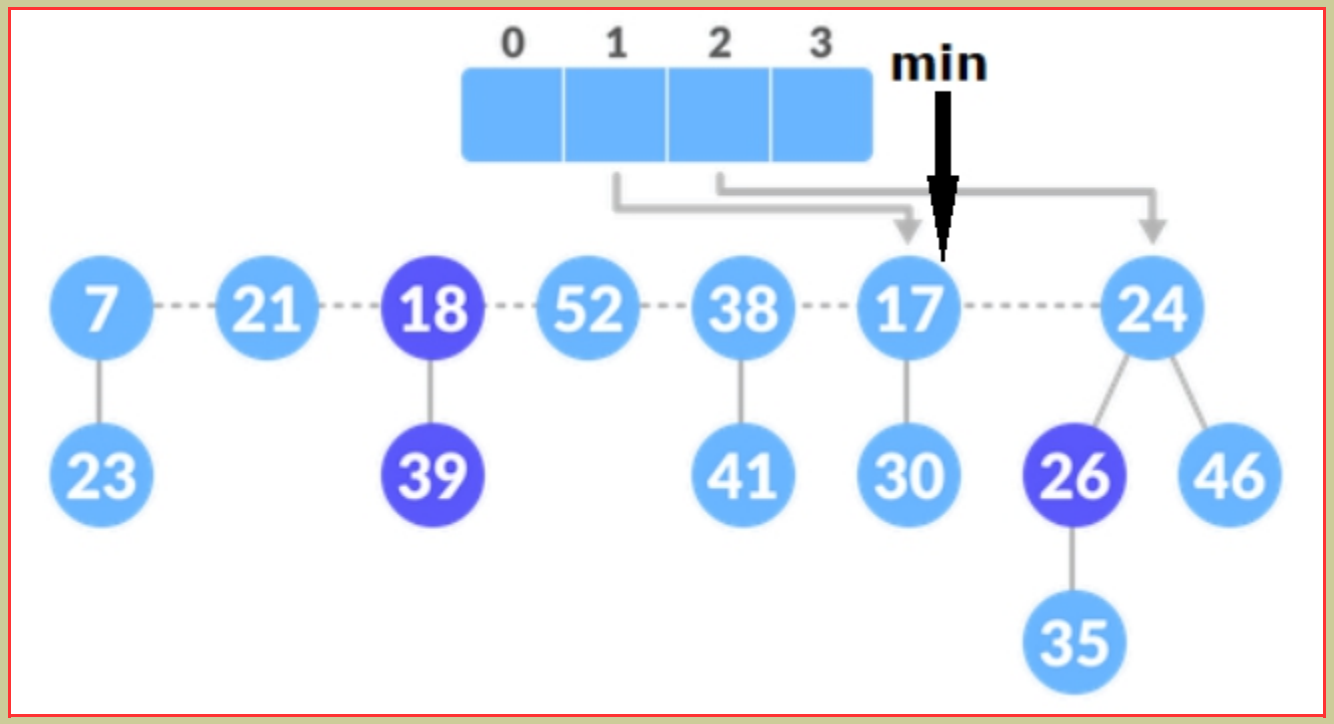
array de graus: [**23**; 17; 24; -]
como as árvores enraizadas nos nodos com **23** e **7** têm o mesmo grau (0) **então**
8. aplicar a operação **união** entre estas duas árvores



Operação principal remover(H) - minHeap

- Exemplo:

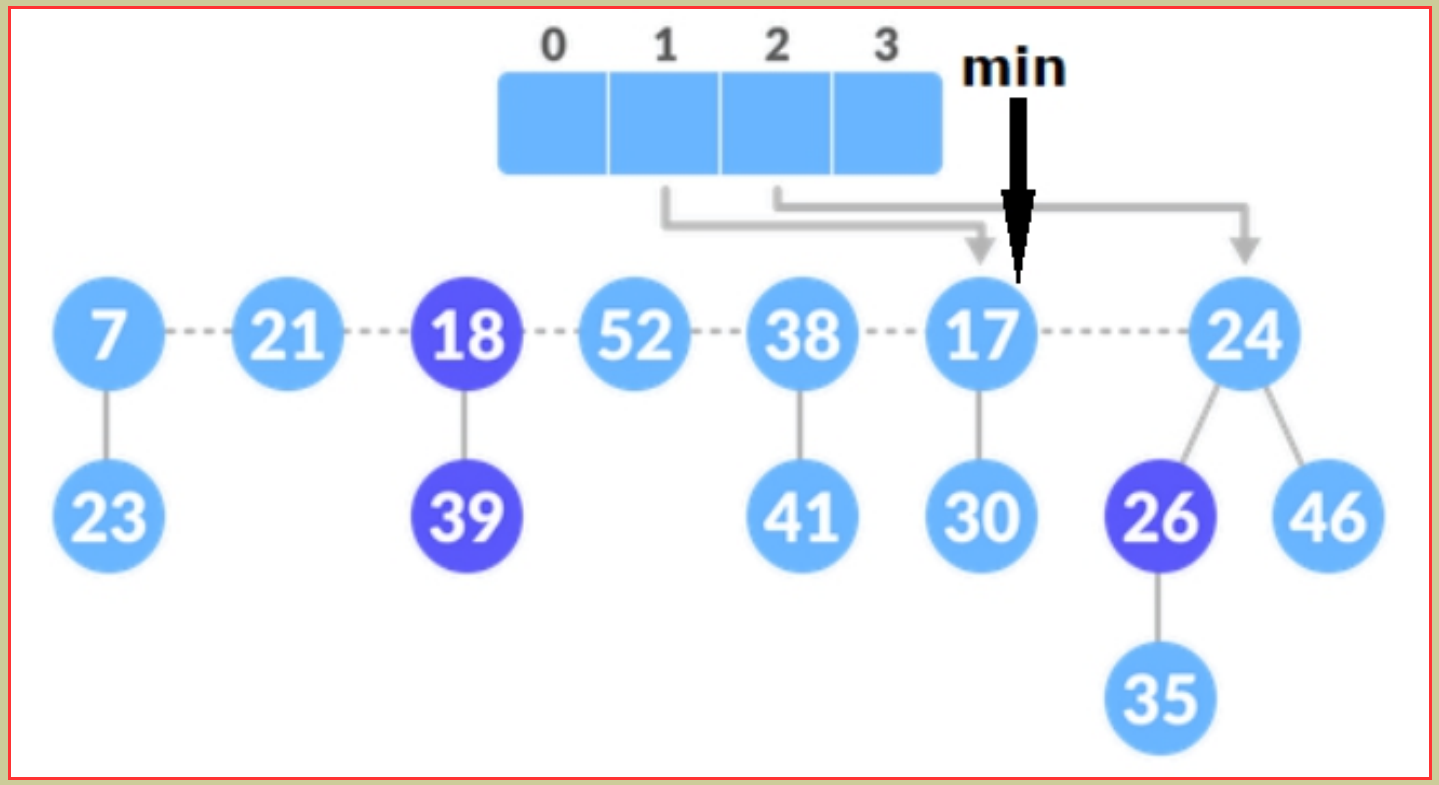
array de graus: [**23**; 17; 24; -]
como as árvores enraizadas nos nodos com **23** e **7** têm o mesmo grau (0) **então**
8. aplicar a operação **união** entre estas duas árvores
- a árvore resultante (minHeap) tem raiz com **7** e grau **1**



Operação principal remover(H) - minHeap

- Exemplo:

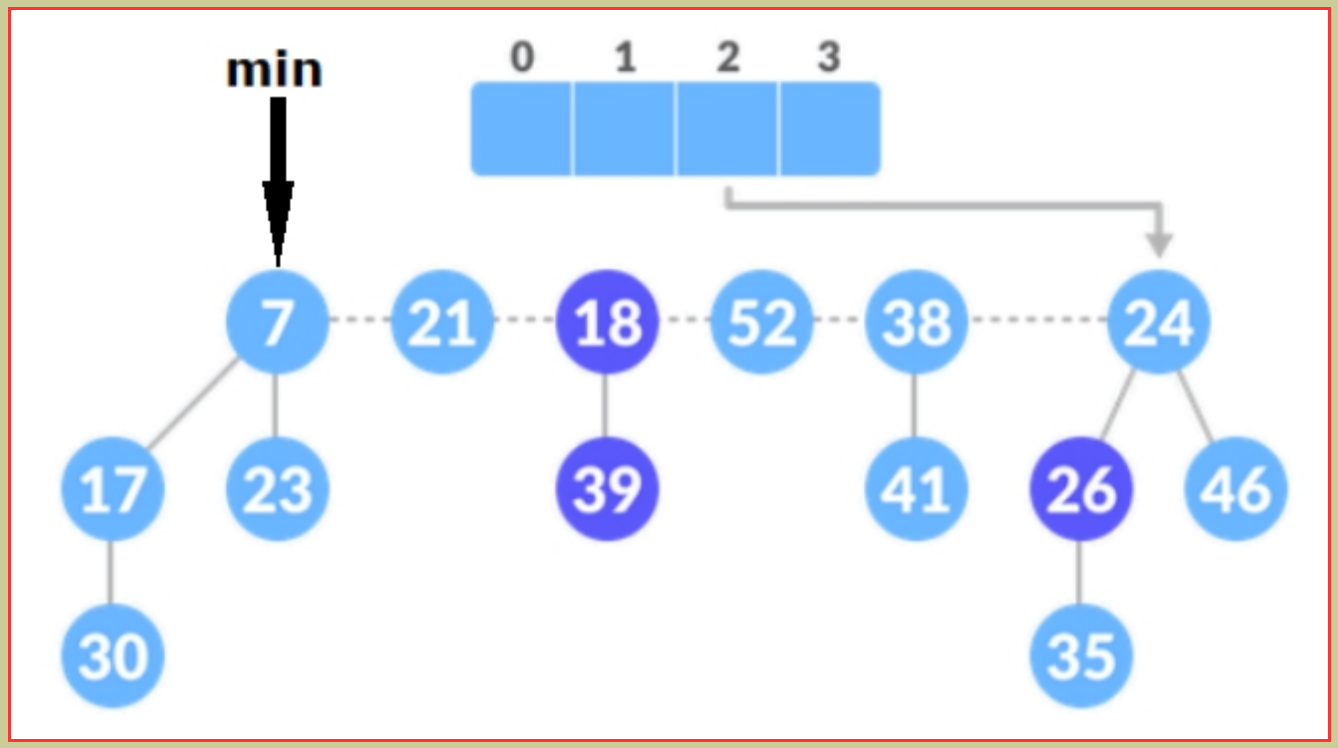
array de graus: [**23**; **17**; 24; -]
como as árvores enraizadas nos nodos com **7** e **17** têm **agora** o mesmo grau (1) **então**
8. aplicar a operação **união** entre estas duas árvores



Operação principal remover(H) - minHeap

- Exemplo:

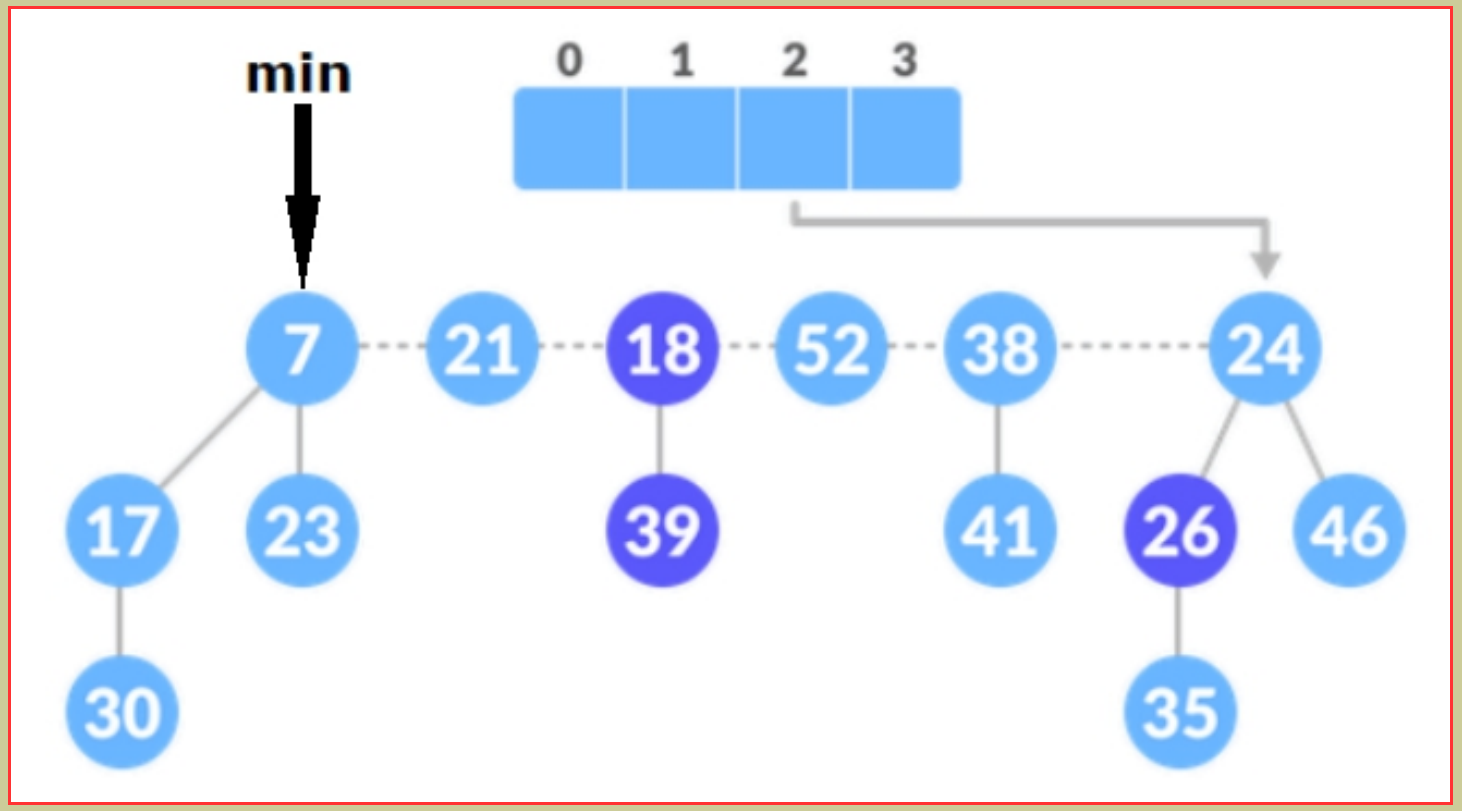
array de graus: [**23**; **17**; 24; -]
como as árvores enraizadas nos nodos com **7** e **17** têm **agora** o mesmo grau (1) **então**
8. aplicar a operação **união** entre estas duas árvores
- a árvore resultante (minHeap) tem raiz com **7** e grau **2**



Operação principal remover(H) - minHeap

- Exemplo:

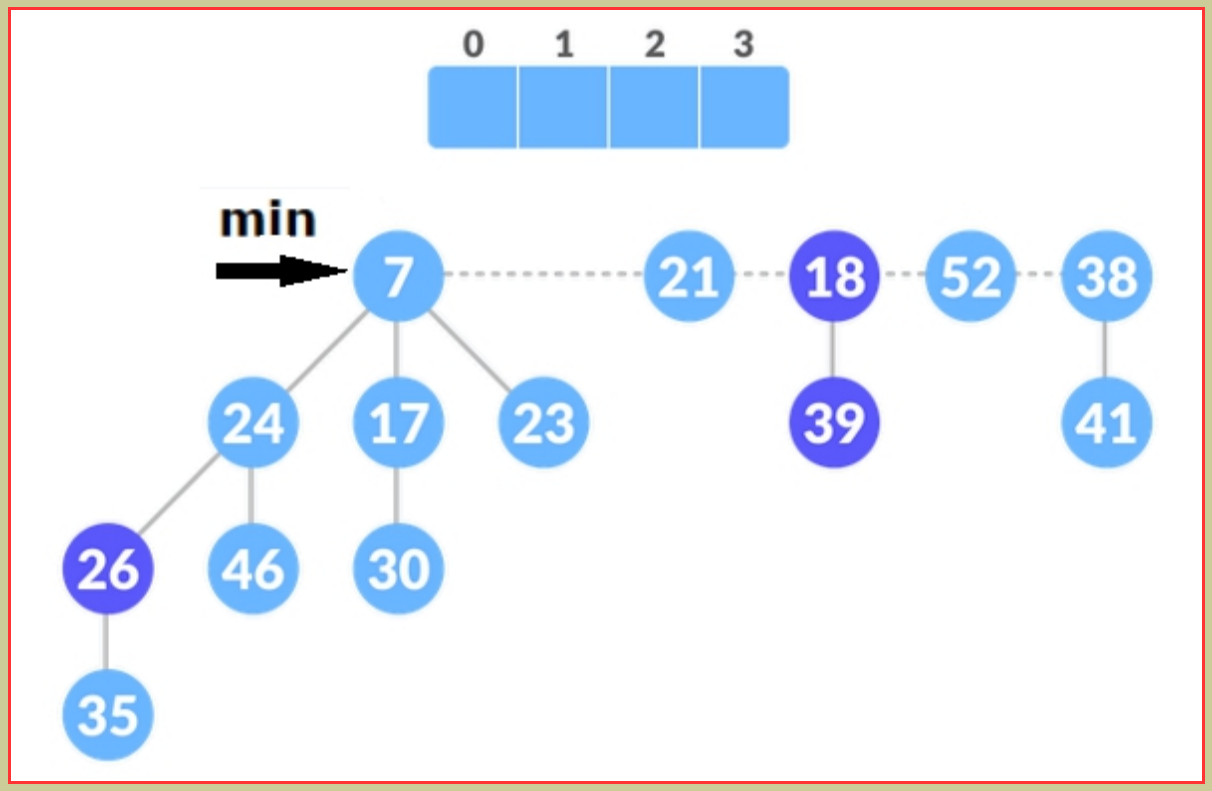
array de graus: [**23**; **17**; **24**; -]
como as árvores enraizadas nos nodos com **7** e **24** têm agora o mesmo grau (2) **então**
8. aplicar a operação **união** entre estas duas árvores



Operação principal remover(H) - minHeap

- Exemplo:

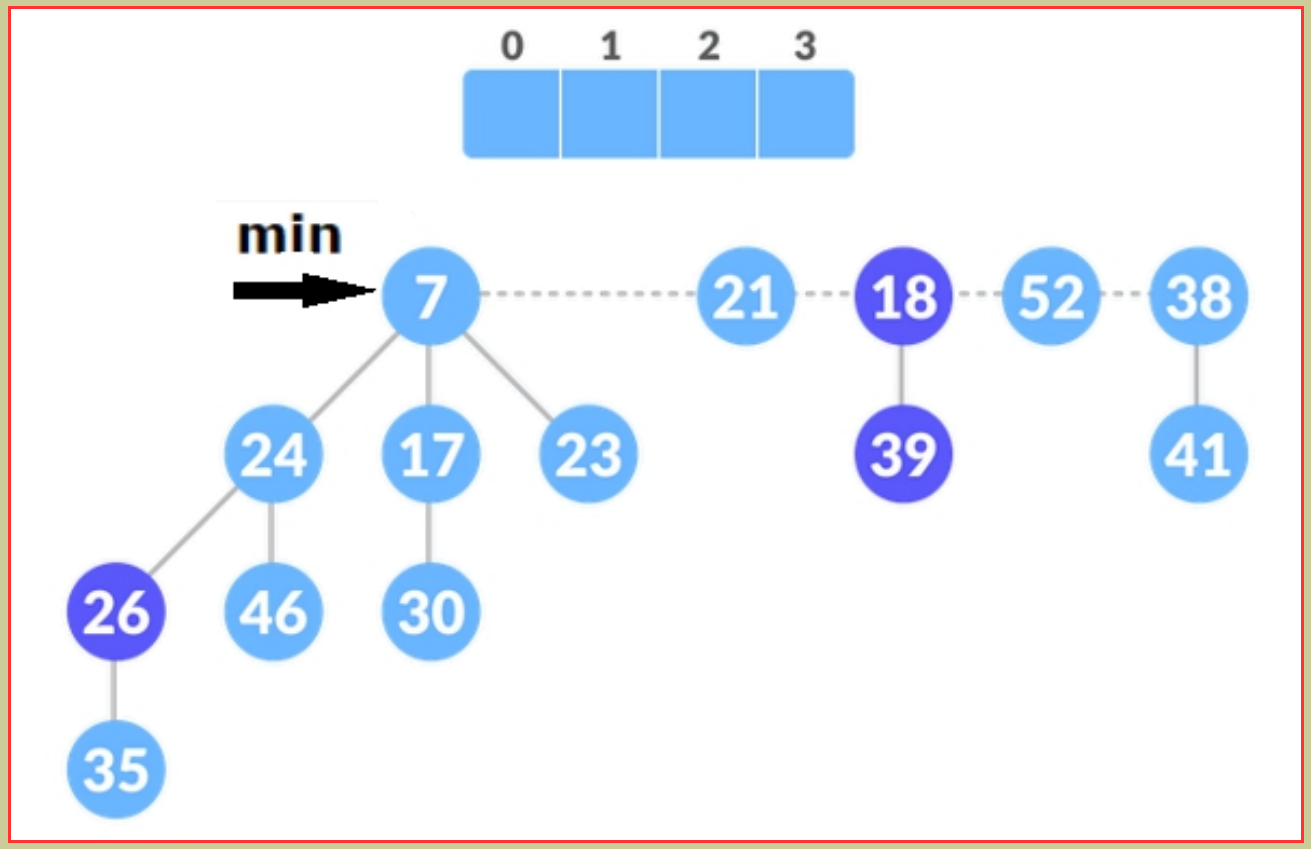
array de graus: [**23**; **17**; **24**; -]
como as árvores enraizadas nos nodos com **7** e **24** têm agora o mesmo grau (2) **então**
8. aplicar a operação **união** entre estas duas árvores
- a árvore resultante (minHeap) tem raiz com **7** e grau **3**



Operação principal remover(H) - minHeap

- Exemplo:

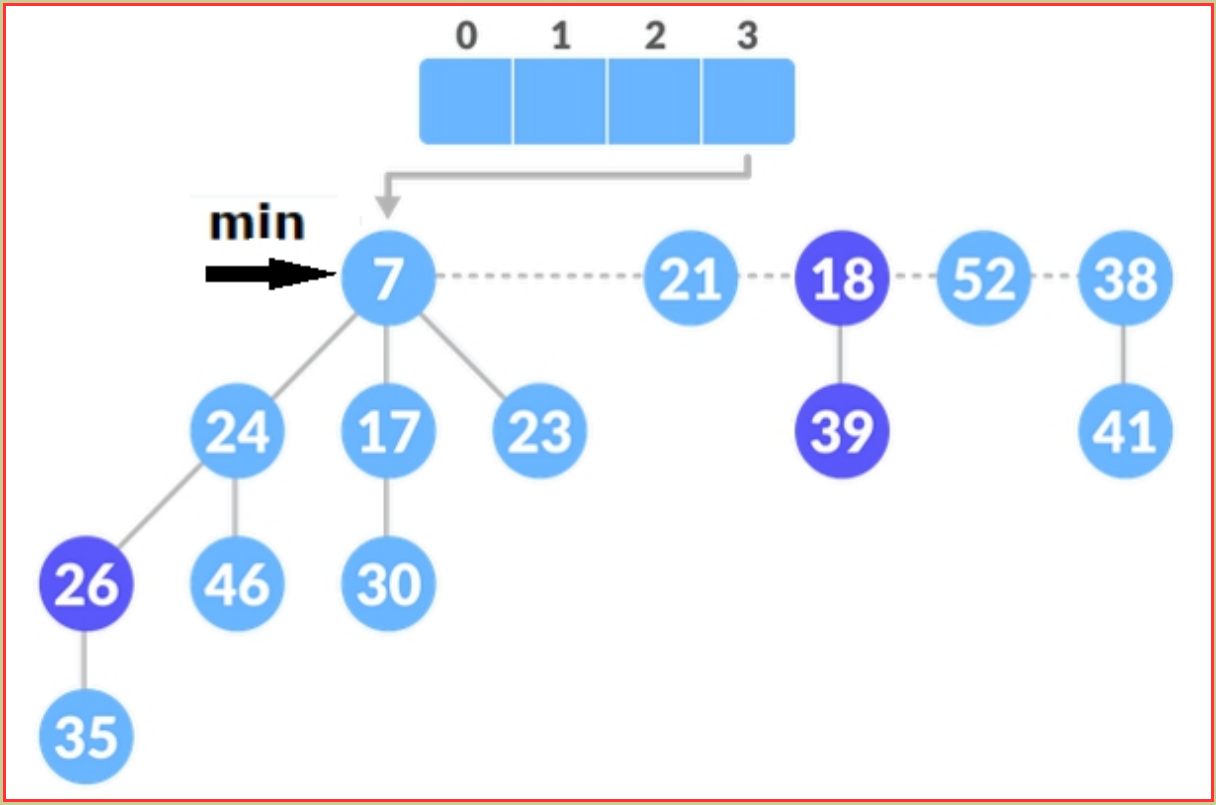
array de graus: [**23**; **17**; **24**; -]
como não há raizes com mapeamento no array (os nós do array já não são raizes) **então**
passar à iteração seguinte (criar um novo mapeamento do array)



Operação principal remover(H) - minHeap

- Exemplo:

6. mapear o grau da árvore com raiz apontada por **min** no array de graus
  - a árvore enraizada no nodo com a chave 7 (**min**) tem grau 3

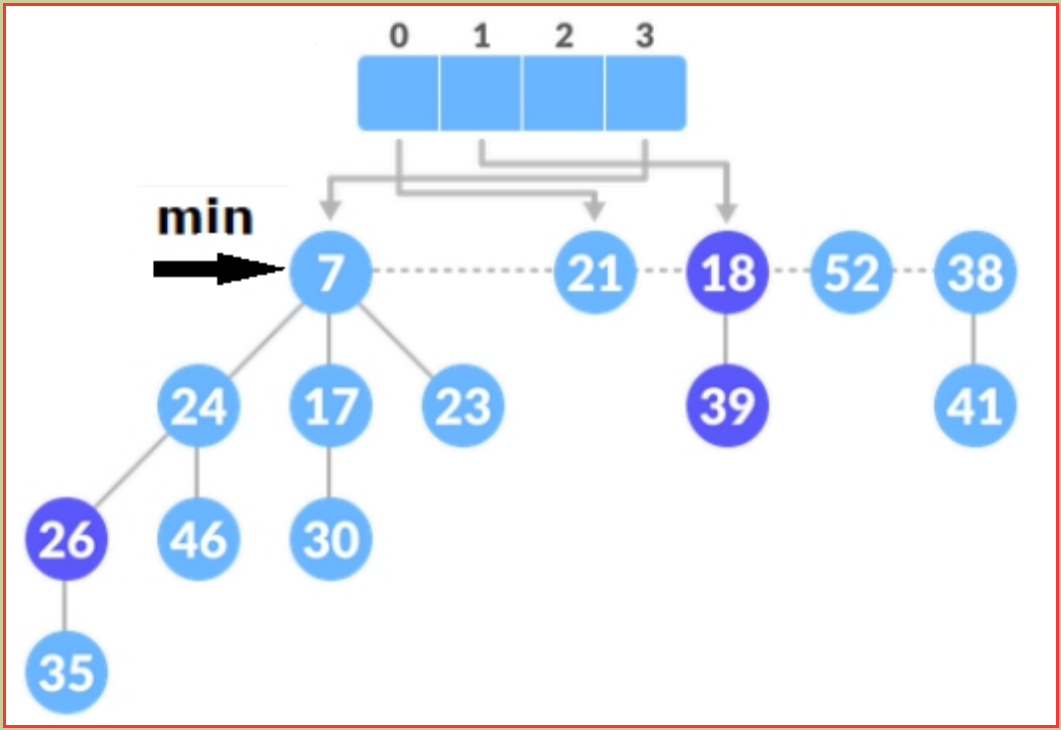


array de graus: [-; -; -; 7]

Operação principal remover(H) - minHeap

- Exemplo:

7. mapear o grau das árvores enraizadas nas próximas raizes da lista no array de graus
- a árvore enraizada no nodo com a chave 21 (**min→prox**) tem grau 0
- a árvore enraizada no nodo com a chave 18 (**min→prox→prox**) tem grau 1

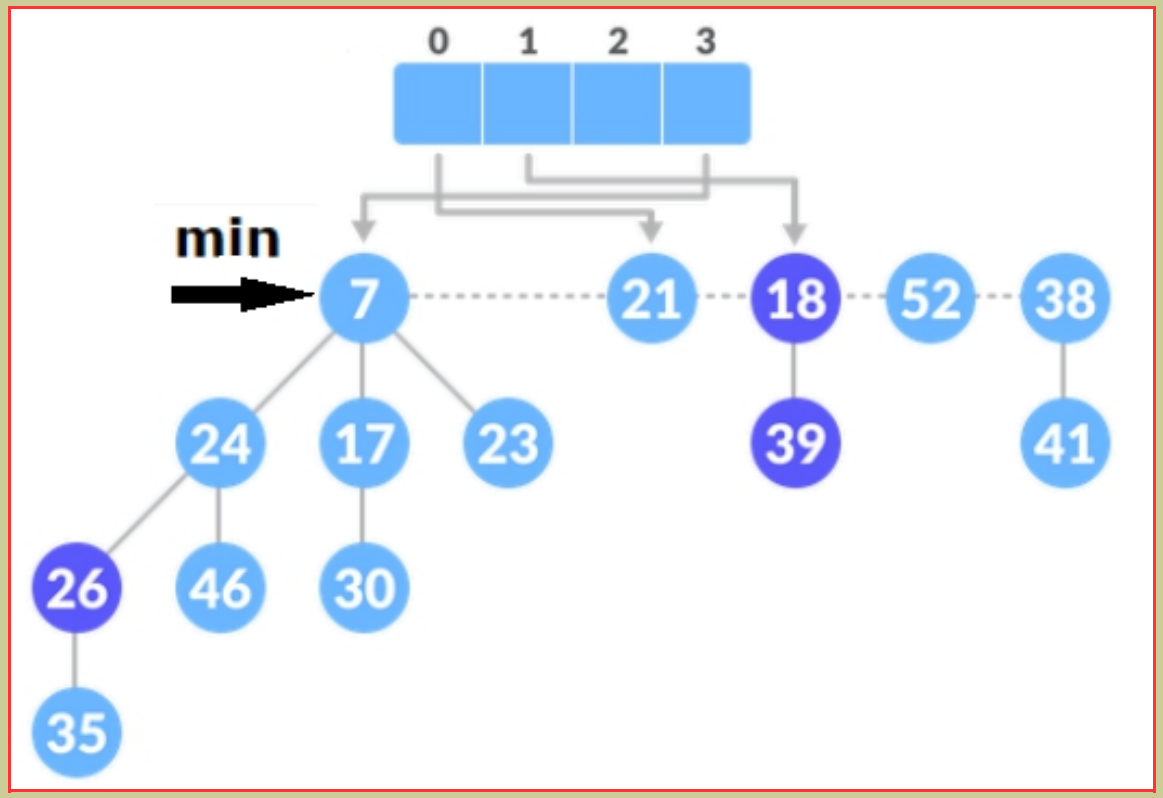


array de graus: [21; 18; -; 7]

Operação principal remover(H) - minHeap

- Exemplo:

array de graus: [**21**; 18; -; 7]
como as árvores enraizadas nos nodos com **21** e **52** têm o mesmo grau (0) **então**
8. aplicar a operação **união** entre estas duas árvores



Operação principal remover(H) - minHeap

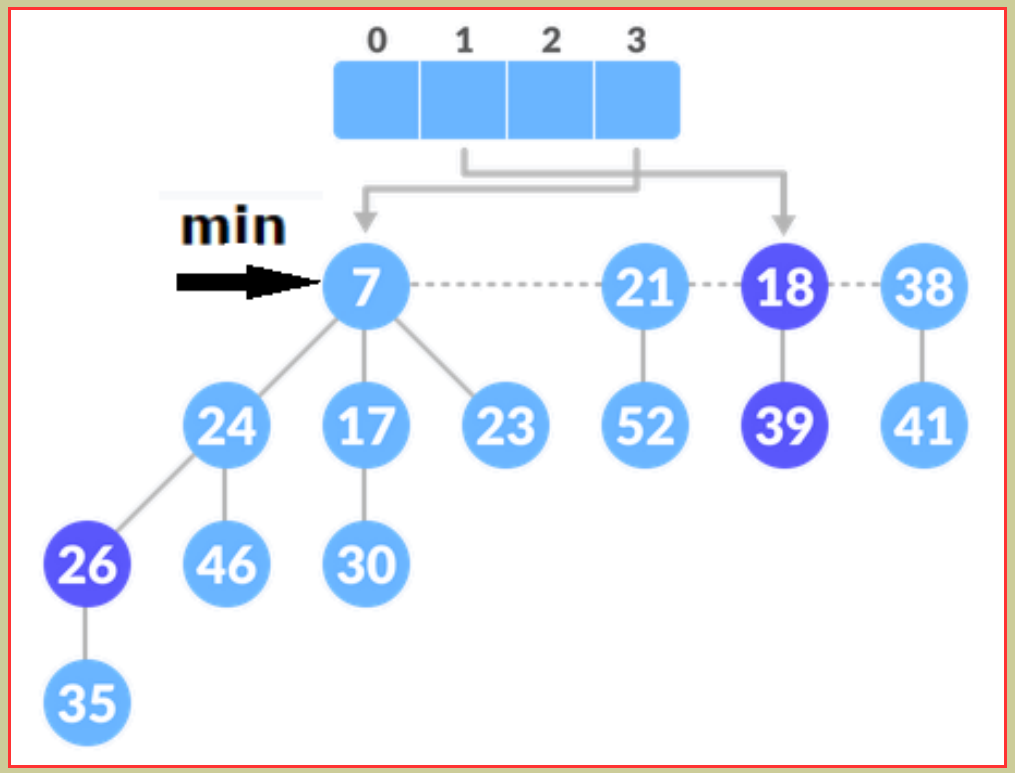
- Exemplo:

array de graus: [**21**; 18; -; 7]

como as árvores enraizadas nos nodos com **21** e **52** têm o mesmo grau (0) **então**

8. aplicar a operação **união** entre estas duas árvores

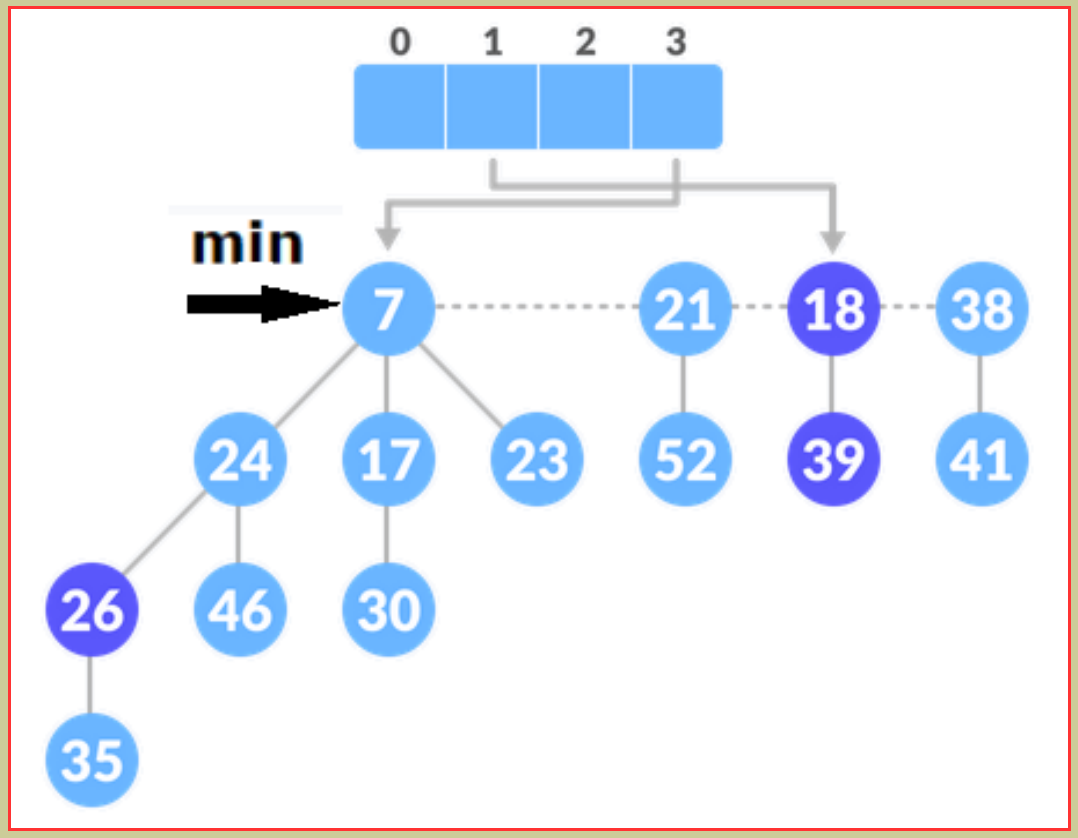
- a árvore resultante (minHeap) tem raiz com **21** e grau **1**



Operação principal remover(H) - minHeap

- Exemplo:

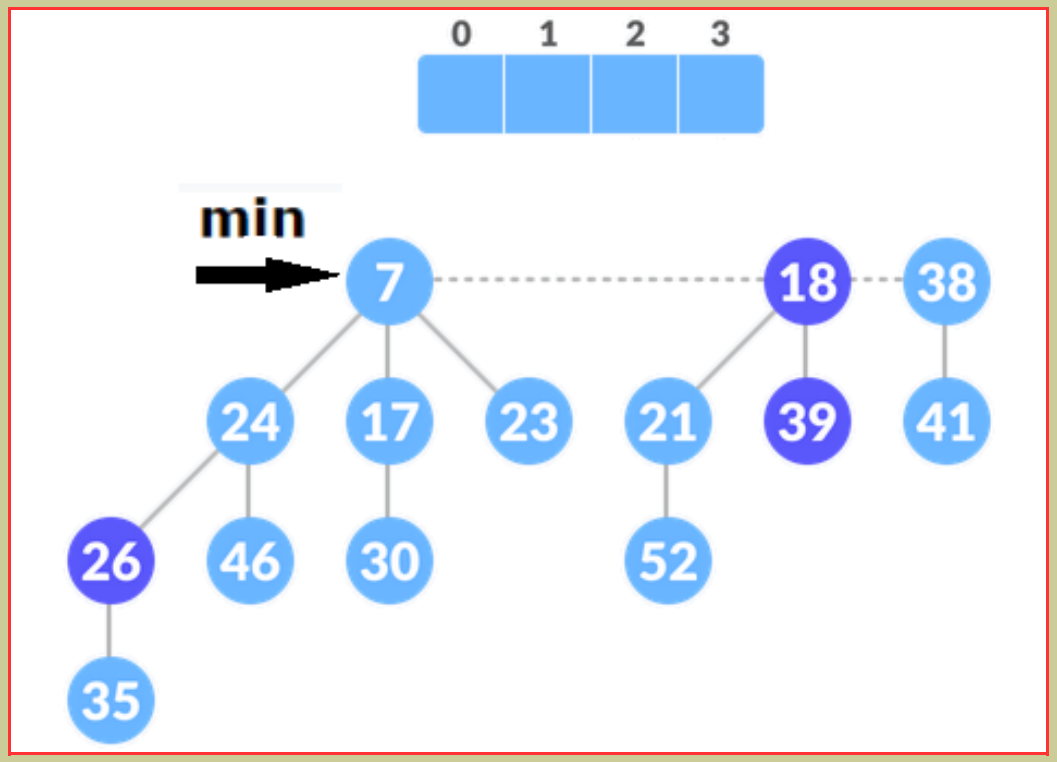
array de graus: [**21**; **18**; -; 7]
como as árvores enraizadas nos nodos com **21** e **18** têm **agora** o mesmo grau (1) **então**
8. aplicar a operação **união** entre estas duas árvores



Operação principal remover(H) - minHeap

- Exemplo:

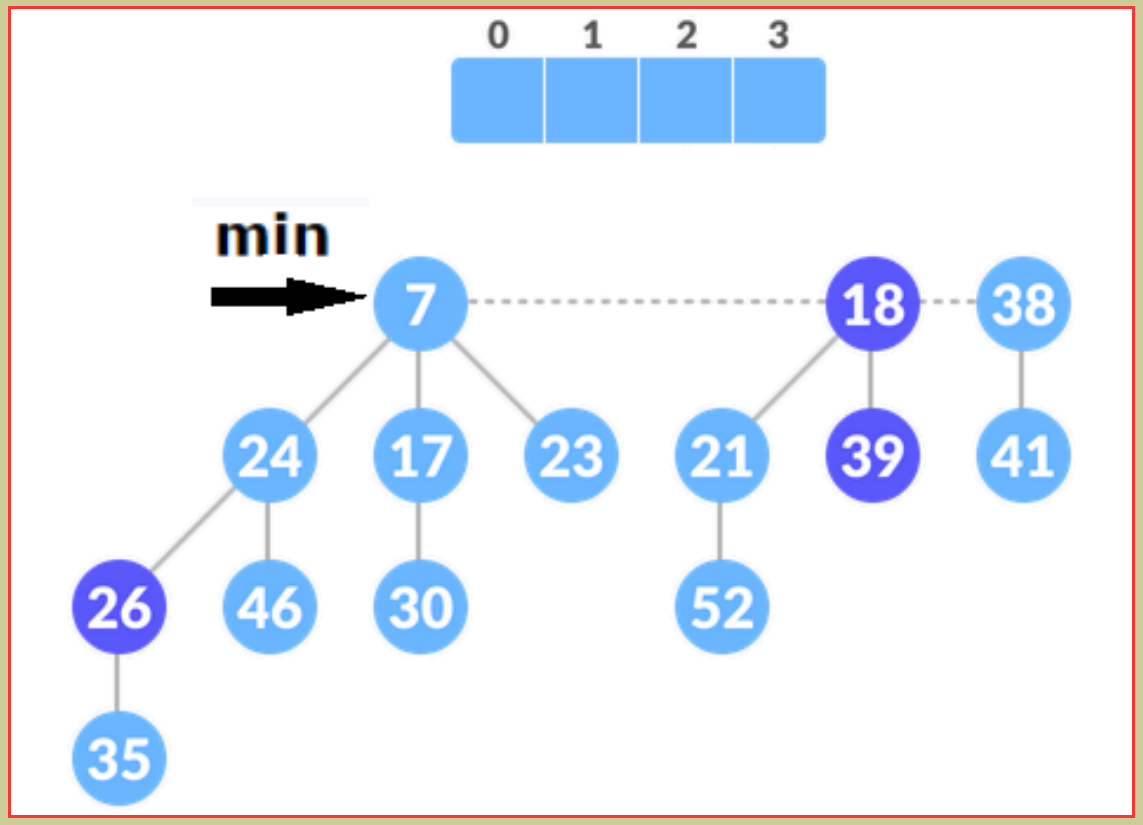
array de graus: [**21**; **18**; -; 7]
como as árvores enraizadas nos nodos com **21** e **18** têm **agora** o mesmo grau (1) **então**
8. aplicar a operação **união** entre estas duas árvores
- a árvore resultante (minHeap) tem raiz com **18** e grau **2**



Operação principal remover(H) - minHeap

- Exemplo:

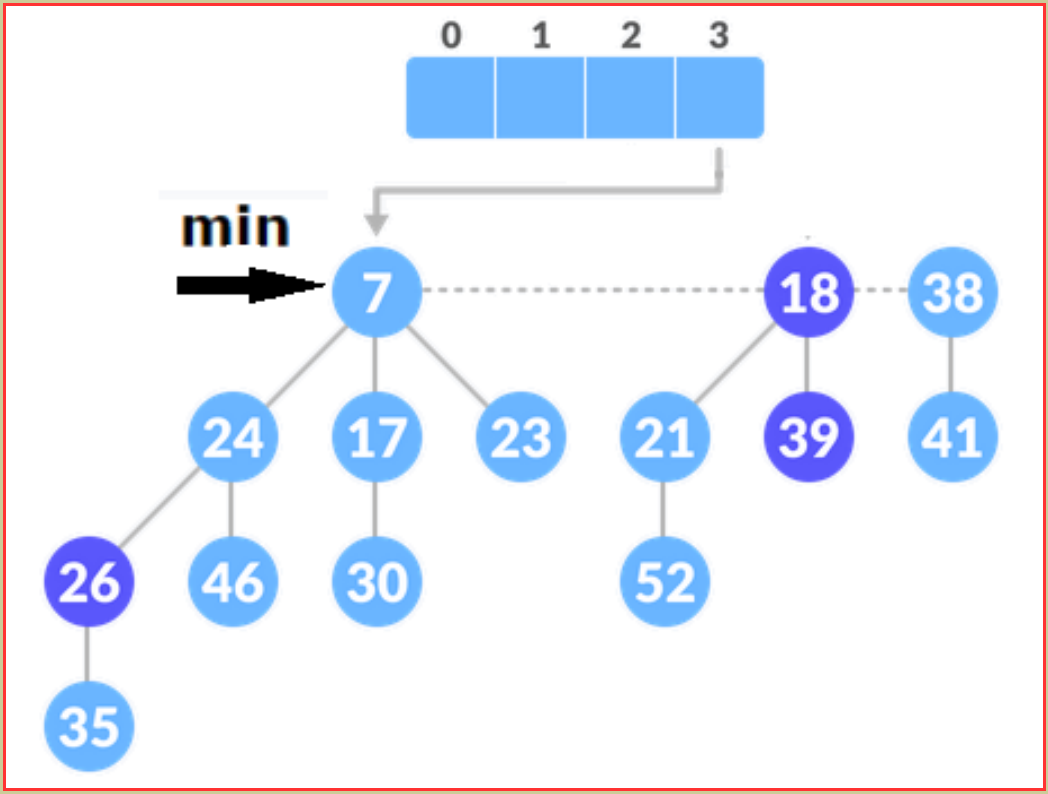
array de graus: [**21**; **18**; -; **7**]
como não há árvores com o mesmo grau da árvore enraizada no nodo com **7 então**
passar à iteração seguinte (criar um novo mapeamento do array)



Operação principal remover(H) - minHeap

- Exemplo:

6. mapear o grau da árvore com raiz apontada por **min** no array de graus
  - a árvore enraizada no nodo com a chave 7 (**min**) tem grau 3

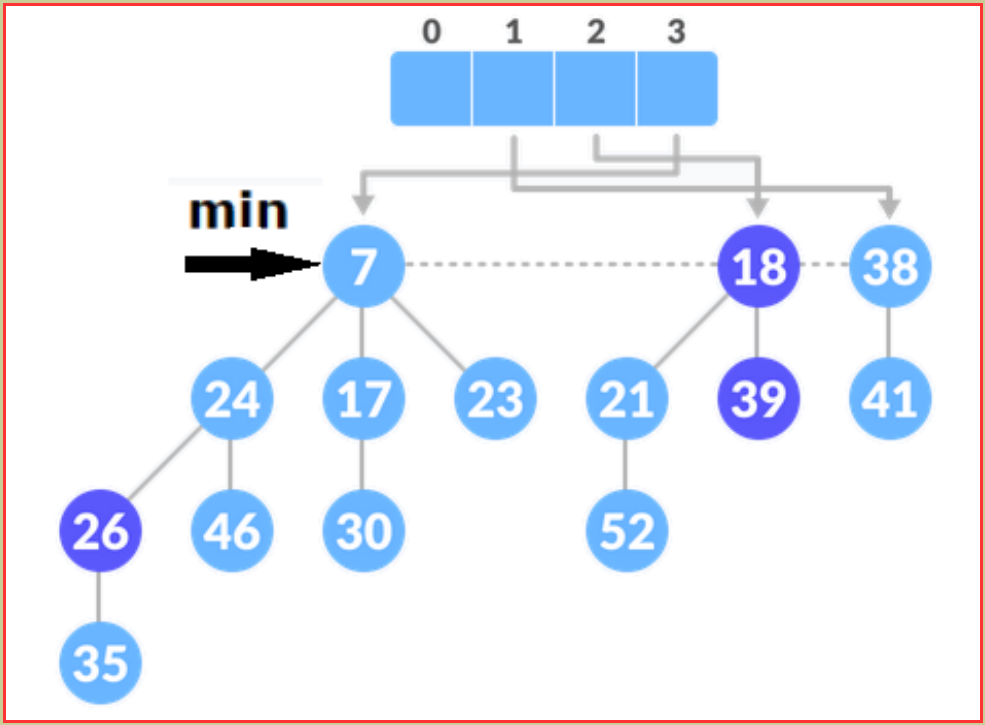


array de graus: [-; -; -; 7]

Operação principal remover(H) - minHeap

- Exemplo:

7. mapear o grau das árvores enraizadas nas próximas raizes da lista no array de graus
- a árvore enraizada no nodo com a chave 18 (**min→prox**) tem grau 2
- a árvore enraizada no nodo com a chave 38 (**min→prox→prox**) tem grau 1



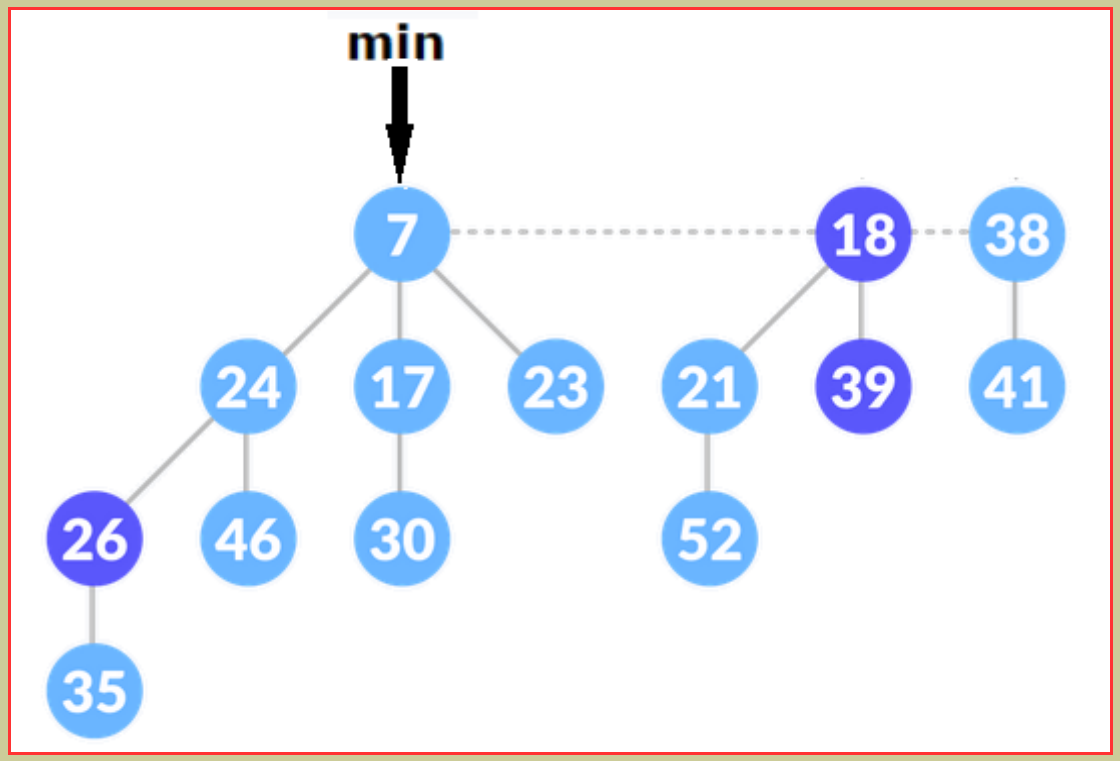
array de graus: [-; 38; 18; 7]

Operação principal remover(H) - minHeap

- Exemplo:

array de graus: [-; 38; 18; 7]
como não há árvores com os mesmos graus **então**
terminar

- o heap resultante da remoção do elemento com menor chave



Eficiência computacional

Ordens de complexidade das operações principais

Estrutura Abstrata de Dados	inserir	remover	consultar
Lista não ordenada (array)	O(1)	O(n)	O(n)
Lista ordenada (array)	O(n)	O(1)	O(1)
Lista não ordenada (listas ligadas)	O(1)	O(n)	O(n)
Lista ordenada (listas ligadas)	O(n)	O(1)	O(1)
Árvore binária pesquisa	O(n)	O(n)	O(n)
Árvore binária pesquisa balanceada	O(log n)	O(log n)	O(log n)
Heap binário	O(log n)	O(log n)	O(1)
Heap binomial	O(log n)	O(log n)	O(1)
Heap de Fibonacci	O(1)	O(log n)	O(1)

- O **Heap de Fibonacci** é mais eficiente do que os **Heaps Binário** e **Binomial**