

# Heaps Binomiais

Eduardo Camponogara

Departamento de Automação e Sistemas  
Universidade Federal de Santa Catarina

DAS-9003: Introdução a Algoritmos

## Heaps Binomiais

### Operações em Heaps Binomiais

# Sumário

Heaps Binomiais

Operações em Heaps Binomiais

## Introdução

Estudaremos *Heaps Binomiais* que são estruturas de dados para implementação de *Heaps Intercaláveis* (“*Mergeable Heaps*”).

### Operações

1. **Make\_Heap()**
2. **Insert\_Heap( $H, x$ )**: insere um nó  $x$ , cuja chave é  $key[x]$
3. **Minimum( $H$ )**: retorna um ponteiro para o nó de  $H$  cuja chave é mínima
4. **Extract\_Min( $H$ )**: remove de  $H$  o nó de chave mínima
5. **Union( $H_1, H_2$ )**: cria e retorna um novo heap que contém os nós de  $H_1$  e  $H_2$ .  $H_1$  e  $H_2$  são destruídos
6. **Decrease\_Key( $H, x, k$ )**: reduz a chave do nó  $x$  para  $k$
7. **Delete( $H, x$ )**: elimina o nó  $x$

## Análise Comparativa

Procedimento	Heap Binário (Pior Caso)	Heap Binomial (Pior Caso)	Heap Fibonacci (Amortizado)
Make_Heap	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Insert_Heap	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
Minimum	$\Theta(1)$	$\Theta(\lg n)$	$\Theta(1)$
Extract_Min	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$
Union	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(1)$
Decrease_Key	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
Delete	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$

# Definição de Heap Binomial

## Definition

Um Heap Binomial é uma coleção de árvore binomiais

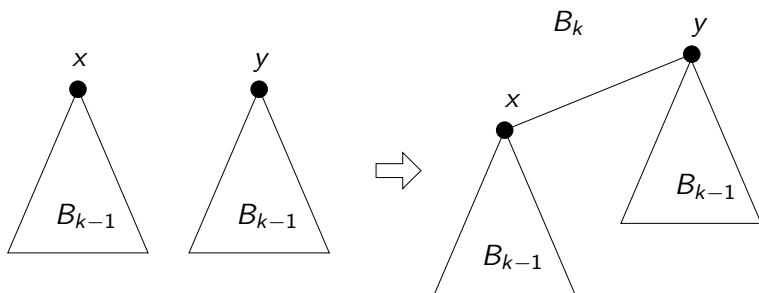
# Árvore Binomial

## Definition

Uma árvore binomial  $B_k$  é uma árvore ordenada definida recursivamente:

- Uma árvore binomial  $B_0$  consiste de apenas um vértice
- Uma árvore binomial  $B_k$  consiste de duas árvores binomiais  $B_{k-1}$  onde a raiz de uma é o filho mais à esquerda da outra

## Definição Recursiva



## Exemplos

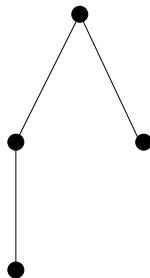
$B_0$



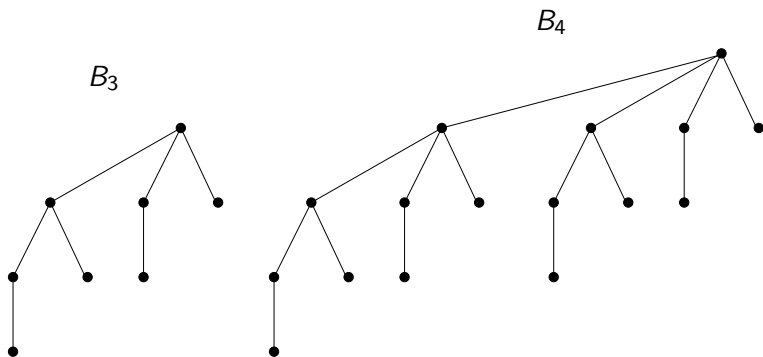
$B_1$



$B_2$



# Exemplos



## Propriedades

### Propriedades de Árvores Binomiais

Para uma árvore binomial  $B_k$ .

- 1) Há  $2^k$  vértices
- 2) A altura da árvore é  $k$
- 3) Há exatamente  $\binom{k}{i}$  nós de profundidade  $i = 0, 1, \dots, k$
- 4) O vértice raiz tem grau  $k$  que é maior do que qualquer outro vértice
- 5) Se os filhos do vértice raiz são numerados por  $k - 1, k - 2, \dots, 0$ , da esquerda para a direita, então o filho  $i$  é raiz de uma sub-árvore  $B_i$

# Propriedades de Árvores Binomiais

## Propriedades

1) Há  $2^k$  vértices em  $B_k$

- ▶  $B_0$  (*base de indução*):  $|B_0| = 1 = 2^0$
- ▶  $B_k$  (*passo de indução*):  $|B_k| = 2|B_{k-1}| = 2 \times 2^{k-1} = 2^k$

2) A altura de  $B_k$  é  $k$

- ▶  $B_0$  (*base de indução*):  $\text{altura}(B_0) = 0$
- ▶  $B_k$  (*passo de indução*):  
 $\text{altura}(B_k) = \text{altura}(B_{k-1}) + 1 = k - 1 + 1 = k$

## Propriedades de Árvores Binomiais

### Propriedades

3) Há exatamente  $\binom{k}{i}$  nós de profundidade  $i = 0, 1, \dots, k$ .

Seja  $n(k, i)$  o número de vértices de profundidade  $i$  em uma árvore  $B_k$ .

$$\begin{aligned}n(k, i) &= n(k-1, i) + n(k-1, i-1) \\&= \binom{k-1}{i} + \binom{k-1}{i-1} \\&= \frac{(k-1)!}{i!(k-1-i)!} + \frac{(k-1)!}{(i-1)!(k-1-i+1)!} \\&= \frac{k!(k-i)}{ki!(k-i)!} + \frac{k!i}{ki!(k-i)!}\end{aligned}$$

## Propriedades de Árvores Binomiais

### Propriedades

3) Há exatamente  $\binom{k}{i}$  nós de profundidade  $i = 0, 1, \dots, k$ .

Seja  $n(k, i)$  o número de vértices de profundidade  $i$  em uma árvore  $B_k$ .

$$\begin{aligned}n(k, i) &= n(k-1, i) + n(k-1, i-1) \\ &= \vdots \\ &= \frac{k!}{i!(k-i)!} \left[ \frac{k-i}{k} + \frac{i}{k} \right] \\ &= \frac{k!}{i!(k-i)!} \left[ \frac{k}{k} \right] = \binom{k}{i}\end{aligned}$$

# Propriedades de Árvores Binomiais

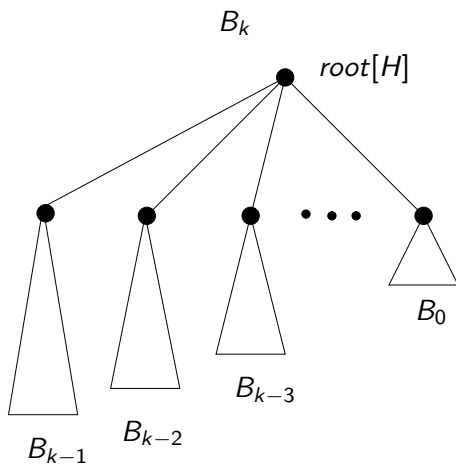
## Propriedades

4) O vértice raiz tem grau  $k$

O grau da raiz é precisamente o número de vértices de profundidade 1, logo:

$$\text{degree}[\text{root}[B_k]] = n(k, 1) = \binom{k}{1} = k$$

## Propriedades de Árvores Binomiais



## Propriedades de Árvores Binomiais

### Corolário

*O grau máximo de um nó qualquer de uma árvore binomial com  $n$  vértices é  $\lg n$*

### Prova

O grau máximo de uma árvore  $B_k$  é o grau da raiz. Portanto,

$$\begin{aligned} \max\_degree[B_k] &= degree[root[B_k]] \\ &= k \\ &= \lg 2^k \\ &= \lg n \end{aligned}$$

## Heaps Binomiais: Definição

### Definição de Heap Binomial

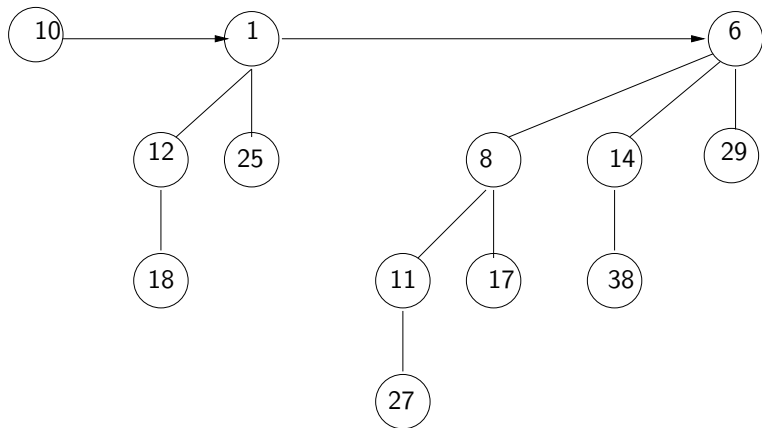
Um heap binomial  $H$  é uma coleção de árvores binomiais que satisfaz as propriedades:

- a) Cada árvore binomial de  $H$  é um heap ordenado  
A chave de um nó é maior ou igual do que a chave de seu pai
- b) Há no máximo uma árvore em  $H$  cuja raiz tem um certo grau

### Observações

- ▶ A primeira propriedade nos diz que a raiz de uma árvore contém a menor chave da árvore
- ▶ A segunda propriedade implica que um heap binomial  $H$  com  $n$  nós contém no máximo  $\lfloor \lg n \rfloor + 1$  árvores binomiais

## Exemplo de Heap Binomial



# Sumário

Heaps Binomiais

Operações em Heaps Binomiais

## Representação de Heaps

- ▶ Cada árvore binomial é armazenada utilizando-se uma representação de árvore filho à esquerda
- ▶ Cada nó  $x$  da árvore tem os atributos abaixo:
  - $key[x]$ : chave do item  $x$
  - $p[x]$ : ponteiro para o nó pai de  $x$
  - $child[x]$ : ponteiro para o filho mais à esquerda
  - $sibling[x]$ : ponteiro para o irmão imediatamente à direita

# Operações

## Criando um Heap

Make\_New\_Heap()

$H \leftarrow new$

$head[H] \leftarrow NIL$

return  $H$

# Operações

## Encontrando o Mínimo

BH\_Minimum( $H$ )

$y \leftarrow NIL$

$x \leftarrow head[H]$

$min \leftarrow \infty$

while  $x \neq NIL$

    if  $key[x] < min$

        then  $min \leftarrow key[x]$

$y \leftarrow x$

$x \leftarrow sibling[x]$

return  $y$

# Operações

## Encontrando o Mínimo

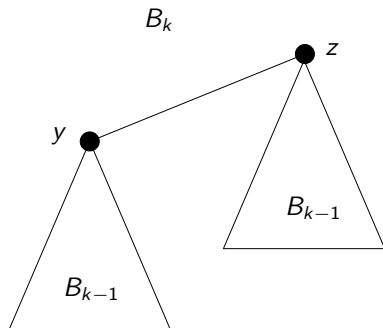
- ▶ O procedimento `BH_Minimum` verifica todas as raízes, cujo número é de no máximo  $\lfloor \lg n \rfloor + 1$
- ▶ Portanto, `BH_Minimum` roda em tempo  $O(\lg n)$

## Unificação de Heaps Binomiais

- ▶ O procedimento  $BH\_Union(H_1, H_2)$  repetidamente concatena duas árvores binomiais cujas raízes tem o mesmo grau
- ▶ O procedimento abaixo concatena uma árvore binomial  $B_{k-1}$  com raiz em  $y$  na árvore binomial  $B_{k-1}$  com raiz em  $z$ , dessa forma obtendo uma árvore binomial  $B_k$  com raiz em  $z$

## Unificação de Heaps Binomiais

BH\_Link( $y, z$ )  
 $p[y] \leftarrow z$   
 $sibling[y] \leftarrow child[z]$   
 $child[z] \leftarrow y$   
 $degree[z] \leftarrow degree[z] + 1$



## Unificação de Heaps Binomiais

- ▶ O procedimento  $BH\_Link(y, z)$  roda em tempo  $O(1)$
- ▶ O procedimento a seguir unifica dois heaps,  $H_1$  e  $H_2$ , retornando o heap resultante
- ▶ Além da função  $BH\_Link$ , o procedimento faz chamadas a um procedimento auxiliar  $BH\_Merge$  que intercala a lista de raízes de  $H_1$  e  $H_2$  em uma lista única ordenada pelo grau das raízes em ordem monotonicamente crescente

## Procedimento $BH\_Union(H_1, H_2)$

$BH\_Union(H_1, H_2)$

- 1)  $H \leftarrow \text{Make\_Heap}()$
- 2)  $\text{head}[H] \leftarrow BH\_Merge(H_1, H_2)$
- 3) free the objects  $H_1$  and  $H_2$ , but the lists they point to
- 4) if  $\text{head}[H] = NIL$
- 5)     then return  $H$
- 6)  $\text{prev}_x \leftarrow NIL$
- 7)  $x \leftarrow \text{head}[H]$
- 8)  $\text{next}_x \leftarrow \text{sibling}[x]$

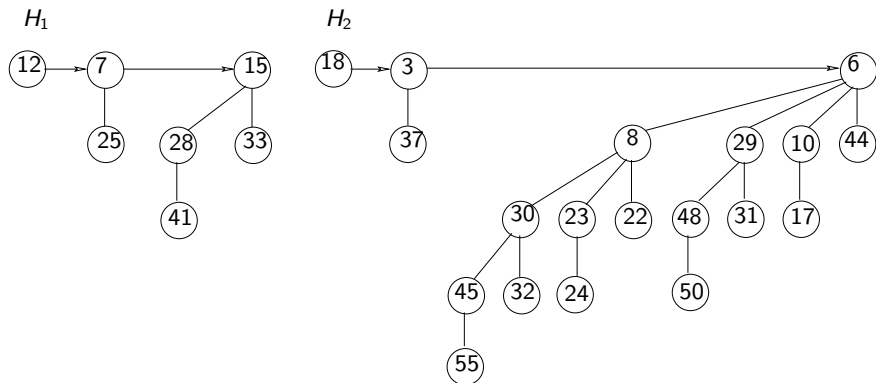
## Procedimento $BH\_Union(H_1, H_2)$

```

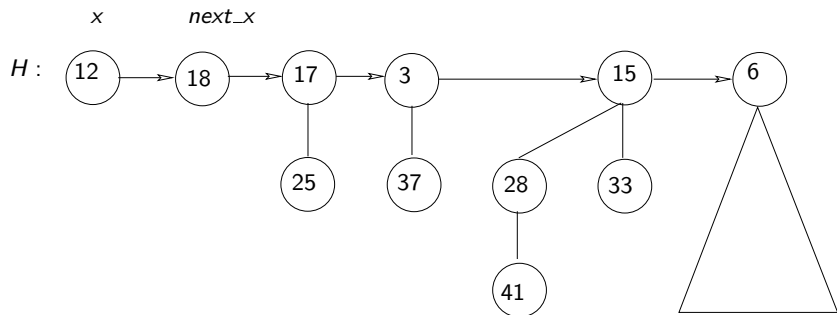
BH_Union( $H_1, H_2$ )
9)   while  $next\_x \neq NIL$ 
10)  if ( $degree[x] \neq degree[next\_x]$ ) or
      ( $sibling[next\_x] \neq NIL$  and
         $degree[sibling[next\_x]] = degree[next\_x]$ )
11)  then  $prev\_x \leftarrow x$ 
12)   $x \leftarrow next\_x$ 
13)  else if  $key[x] \leq key[next\_x]$ 
14)  then  $sibling[x] \leftarrow sibling[next\_x]$ 
15)   $BH\_Link(next\_x, x)$ 
16)  else if  $prev\_x = NIL$ 
17)  then  $head[H] \leftarrow next\_x$ 
18)  else  $sibling[prev\_x] \leftarrow next\_x$ 
19)   $BH\_Link(x, next\_x)$ 
20)   $x \leftarrow next\_x$ 
21)   $next\_x \leftarrow sibling[x]$ 

```

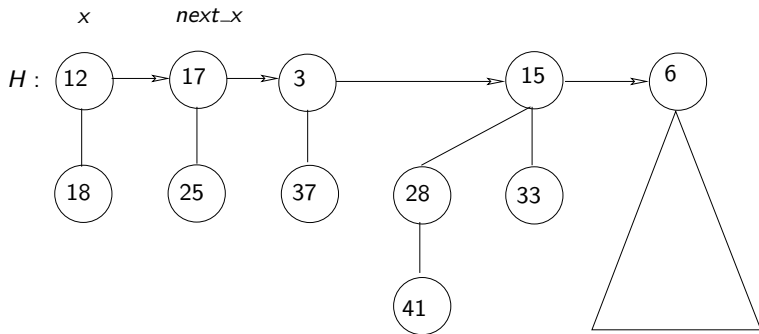
## Procedimento $BH\_Union(H_1, H_2)$



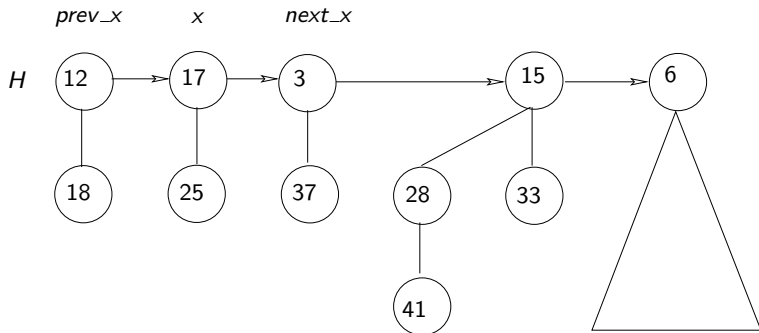
## Procedimento BH\_Union( $H_1, H_2$ )



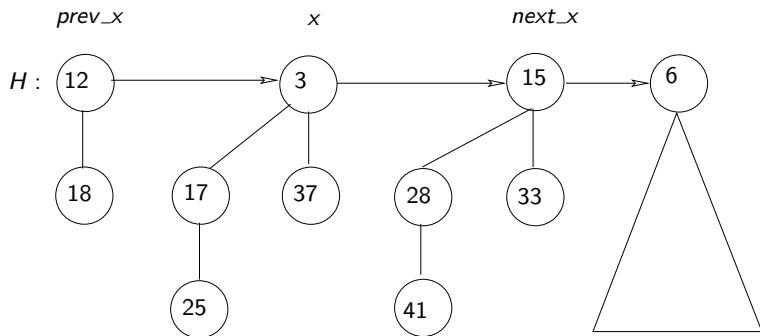
## Procedimento BH\_Union( $H_1, H_2$ )



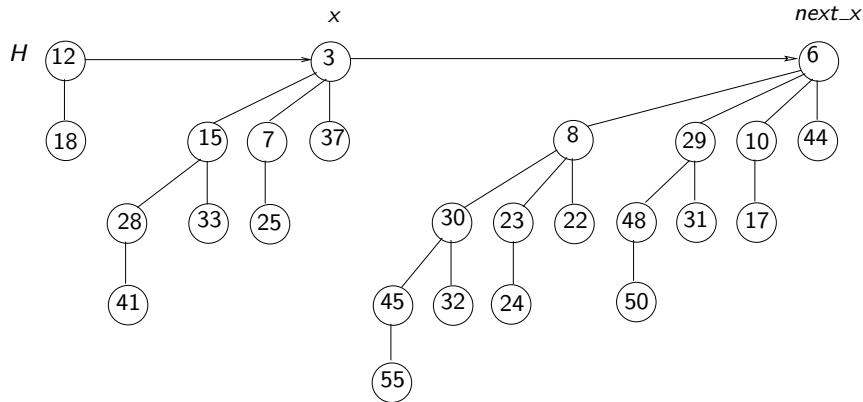
## Procedimento BH\_Union( $H_1, H_2$ )



## Procedimento BH\_Union( $H_1, H_2$ )



## Procedimento $BH\_Union(H_1, H_2)$



## Funcionamento do Procedimento de Unificação

1. Inicialmente o algoritmo intercala as listas de raízes de  $H_1$  e  $H_2$ , produzindo uma lista única para  $H$ , onde os elementos estão ordenados em ordem crescente de grau
2. Podem existir até no máximo duas raízes de um mesmo grau
3. Na segunda fase, o algoritmo combina árvores de mesmo grau até que reste no máximo uma raiz de cada grau
4. O procedimento *BH\_Merge* tem tempo de execução  $\Theta(m)$ , onde  $m$  é o número de raízes em  $H_1$  e  $H_2$

## Segunda Fase: Procedimento de Unificação

O procedimento mantém três ponteiros para raízes:

- a)  $x$  aponta para a raiz que está sendo examinada
- b)  $prev\_x$  aponta para a raiz precedente de  $x$

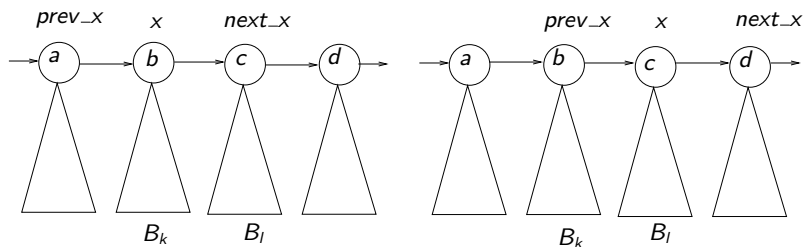
$$sibling[prev\_x] = x$$

- c)  $next\_x$  aponta para a raiz que segue  $x$

$$sibling[x] = next\_x$$

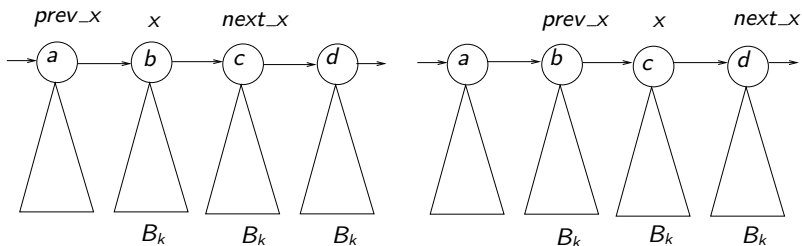
## Segunda Fase: Procedimento de Unificação

Caso 1:  $B_k$  e  $B_l$  com  $k < l$



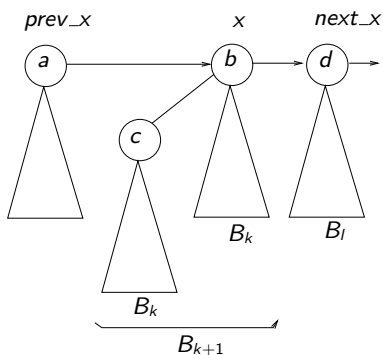
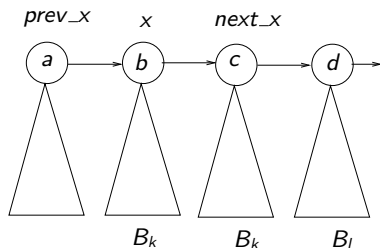
## Segunda Fase: Procedimento de Unificação

### Caso 2:



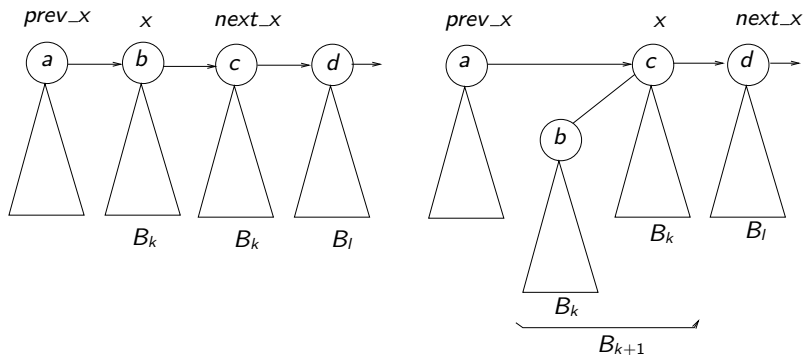
## Segunda Fase: Procedimento de Unificação

Caso 3:  $key[x] \leq key[next\_x]$



## Segunda Fase: Procedimento de Unificação

Caso 4:  $key[next\_x] \leq key[x]$



## Análise do Procedimento de Unificação

- ▶ Seja  $n_1$  o número de nós em  $H_1$ . Então sabemos que  $H_1$  tem no máximo  $\lfloor \lg n_1 \rfloor + 1$  árvores binomiais
- ▶ Seja  $n_2$  o número de nós em  $H_2$ . Então sabemos que  $H_2$  tem no máximo  $\lfloor \lg n_2 \rfloor + 1$  árvores binomiais
- ▶ Seja  $n = n_1 + n_2$  o número de nós em  $H$ . Então sabemos que  $H$  tem no máximo:

$$\lfloor \lg n_1 \rfloor + 1 + \lfloor \lg n_2 \rfloor + 1 \leq 2\lfloor \lg n \rfloor + 2 \in O(\lg n)$$

raízes imediatamente após a chamada  $\text{BH\_Merge}(H_1, H_2)$

- ▶ O tempo de execução de  $\text{BH\_Merge}$  é  $O(\lg n)$

## Análise do Procedimento de Unificação

- ▶ Note que o laço **while** executa no máximo  $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor + 1$  iterações
  - ▶ A cada iteração o ponteiro avança uma posição ou remove uma raiz da lista
- ▶ Podemos então concluir que BH\_Union executa em tempo  $O(\lg n)$

## Inserindo um Elemento

$BH\_Insert(H, x)$

- 1)  $H' \leftarrow Make\_Heap()$
- 2)  $p[x] \leftarrow child[x] \leftarrow sibling[x] \leftarrow NIL$
- 3)  $degree[x] \leftarrow 0$
- 4)  $head[H'] \leftarrow x$
- 5)  $H \leftarrow BH\_Union(H, H')$

### Análise

Claramente, o procedimento  $BH\_Insert$  executa em tempo  $O(\lg n)$

## Extraindo o Elemento de Menor Chave

### Princípios do Procedimento *BH\_Extract\_Min()*

1. Encontre a raiz  $x$  com a menor chave na lista de raízes de  $H$
2. Remova  $x$  da lista de raízes de  $H$
3.  $H' \leftarrow \text{Make_Heap}()$
4. Inverta a ordem da lista encadeada dos filhos de  $x$ , fazendo  $H'$  apontar para o início desta lista
5.  $H \leftarrow \text{BH_Union}(H, H')$
6. retorne  $x$

## Extraindo o Elemento de Menor Chave

### Corretude e Análise do Procedimento *BH\_Extract\_Min*

- ▶ Se  $x$  é a raiz de uma árvore  $B_k$ , então pelas propriedades de um Heap, os filhos de  $x$  da esquerda para a direita, são raízes de árvores,  $B_{k-1}, B_{k-2}, \dots, B_0$
- ▶ Ao invertermos a ordem da lista de filhos de  $x$ , passo 4 do algoritmo, obtemos um heap binomial  $H'$  que contém todos os nós da árvore com raiz em  $x$ , exceto  $x$
- ▶ Uma vez que todas as linhas 1–5 executam em tempo  $O(\lg n)$ , verificamos que o procedimento *BH\_Extract\_Min* tem tempo de execução  $O(\lg n)$

## Reduzindo a Chave de um Elemento

A tarefa é reduzir a chave de um nó  $x$  de um heap binomial para um valor  $k$

$BH\_Decrease\_Key(H, x, k)$

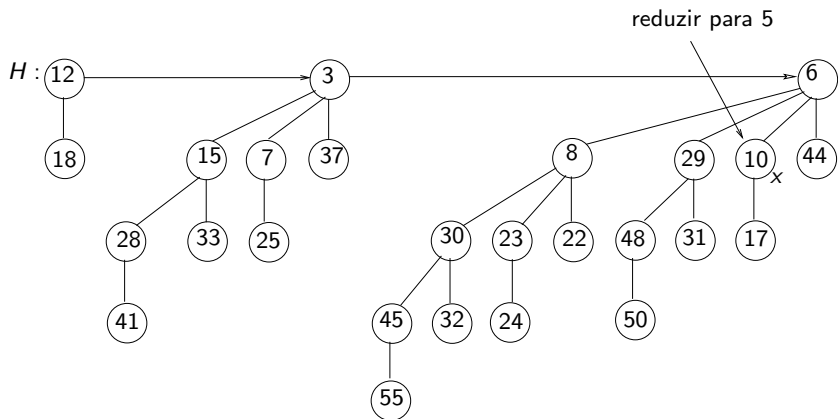
- 1) if  $k > key[x]$
- 2) then error "new key is invalid"
- 3)  $key[x] \leftarrow k$
- 4)  $y \leftarrow x$
- 5)  $z \leftarrow p[x]$
- 6) while  $z \neq NIL$  and  $key[y] < key[z]$
- 7) do exchange  $key[y] \leftrightarrow key[z]$
- 8)  $y \leftarrow z$
- 9)  $z \leftarrow p[y]$

## Reduzindo a Chave de um Elemento

### Análise

- ▶ O procedimento se comporta da mesma forma que no heap binário, movendo a chave para cima até que ela encontre seu nível.
- ▶ O tempo de execução de `BH_Decrease_Key` é no máximo a altura da maior árvore binomial em  $H$ .
- ▶ No pior caso, temos apenas uma árvore binomial  $B_k$  com  $n$  nós. Uma vez que a altura de  $B_k$ , neste caso, é  $k = \lg n$ , deduzimos que o tempo de execução é  $O(\lg n)$

## Procedimento BH\_Decrease\_Key)



## Removendo uma Chave

### Questão

Como você removeria uma chave qualquer?

# Heaps Binomiais

- ▶ Fim!
- ▶ Obrigado pela presença