Heaps binomiais

Gabriel Pedro de Castro

20 de setembro de 2007

Heaps binomiais são formados por uma lista ligada de *árvores binomiais*.

Definição

Árvores binomiais são definidas recursivamente da seguinte forma:

Árvores binomiais

Heaps binomiais são formados por uma lista ligada de *árvores binomiais*.

Definição

Árvores binomiais são definidas recursivamente da seguinte forma:

- B_0:

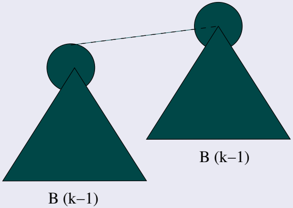
Heaps binomiais são formados por uma lista ligada de *árvores binomiais*.

Definição

Árvores binomiais são definidas recursivamente da seguinte forma:

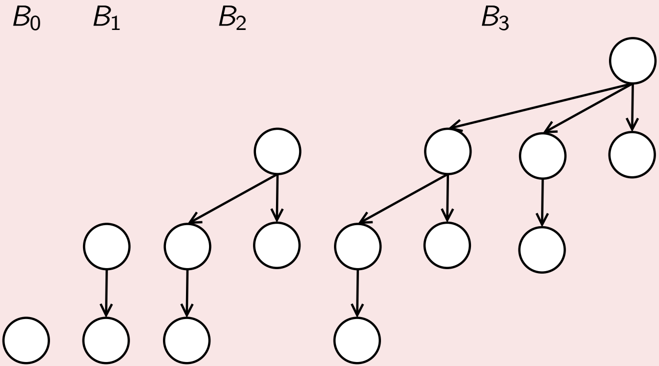
- B_0:

- B_k:



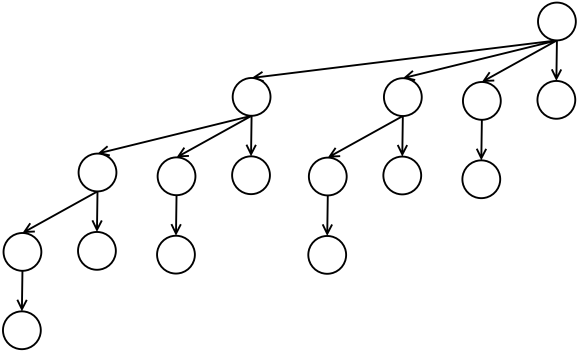
Árvores binomiais

Alguns exemplos de árvores binomiais



Árvores binomiais

Mais um exemplo: B_4



Propriedades das árvores binomiais

Uma árvore binomial B_k:

Propriedades das árvores binomiais

Uma árvore binomial B_k:

- Possui 2^k nós;
  Prova: Indução em k. Para B_0: $2^0 = 1$.
  B_k possui duas subárvores B_{k-1}:
  $B_k = B_{k-1} + B_{k-1} = 2^{k-1} + 2^{k-1} = 2^k$.

Árvores binomiais

Propriedades das árvores binomiais

Uma árvore binomial B_k:

- Possui 2^k nós;
  Prova: Indução em k. Para B_0: $2^0 = 1$.
  B_k possui duas subárvores B_{k-1}:
  $B_k = B_{k-1} + B_{k-1} = 2^{k-1} + 2^{k-1} = 2^k$.
- Tem altura k;
  Prova: Também por indução em k. Para B_0: $k = 0$.
  Para B_k: Uma árvore B_k é formada por duas subárvores B_{k-1}, sendo que a raíz de uma se torna filha da raíz da outra; portanto a altura da árvore é aumentada de 1 em relação as filhas. $\mathrm{h}(k) = \mathrm{h}(k-1) + 1 = k$.

Propriedades das árvores binomiais – continuação

Propriedades das árvores binomiais – continuação

- O nível i possui exatamente $\binom{k}{i}$, $i = 0, 1, \ldots, k$, nós.
  Prova: Seja $D(k, i)$ o número de nós na profundidade i da árvore B_k. Como B_k é composta de duas B_{k-1}, na profundidade i de B_k aparecem os nós da profundidade i de uma B_{k-1} e $i - 1$ da outra. Assim, $D(k, i) = D(k-1, i) + D(k-1, i-1)$. Pela hipótese de indução, $D(k, i) = \binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i}$.

Propriedades das árvores binomiais – continuação

- O nível i possui exatamente $\binom{k}{i}$, $i = 0, 1, \ldots, k$, nós.
  Prova: Seja $D(k, i)$ o número de nós na profundidade i da árvore B_k. Como B_k é composta de duas B_{k-1}, na profundidade i de B_k aparecem os nós da profundidade i de uma B_{k-1} e $i-1$ da outra. Assim, $D(k, i) = D(k-1, i) + D(k-1, i-1)$. Pela hipótese de indução, $D(k, i) = \binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i}$.
- A raíz tem grau k, maior que de todos os outros nós, e cada filho i, $i = k-1, k-2, \ldots, 0$, é raiz de uma subárvore B_i.
  Prova: A raíz de B_k tem o grau aumentado em um em relação a B_{k-1} justamente por estar ligada a outra B_{k-1}. Ainda por indução, como a raiz de B_{k-1} está ligada a subárvores $B_0, B_1, \ldots, B_{k-2}$, então B_k tambem o estará, assim como estará ligada a uma outra raíz B_{k-1}, pois é formada pela união das duas subárvores.

Definição

Um heap binomial H é um conjunto de árvores binomiais que satisfaz as seguintes propriedades:

Definição

Um heap binomial H é um conjunto de árvores binomiais que satisfaz as seguintes propriedades:

- Toda árvore binomial de H tem estrutura de heap, i.e., a chave de um nó é maior ou igual a chave de seu pai. Assim, sabemos que a raíz possui a menor chave da árvore.

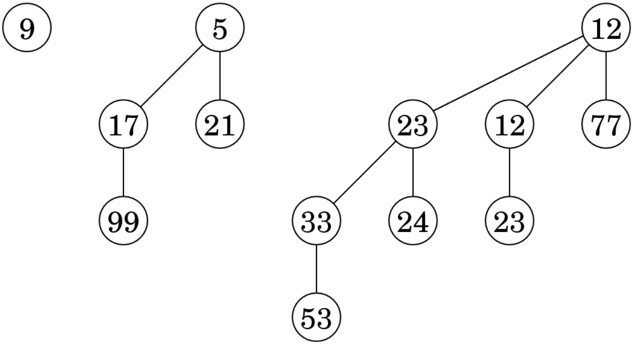
Definição

Um heap binomial H é um conjunto de árvores binomiais que satisfaz as seguintes propriedades:

- Toda árvore binomial de H tem estrutura de heap, i.e., a chave de um nó é maior ou igual a chave de seu pai. Assim, sabemos que a raíz possui a menor chave da árvore.
- Há no máximo uma árvore binomial em H com uma raíz de um determinado grau. Assim, para um heap de n nós há, no máximo, $\lfloor \lg n \rfloor + 1$ árvores binárias. Para ver isto basta pensar na representação binária do número de elementos do heap: $< b_{\lfloor \lg n \rfloor}, b_{\lfloor \lg n \rfloor - 1}, \ldots, b_0 >$, com $n = \sum_{i=0}^{\lfloor \lg n \rfloor} b_i 2^i$.

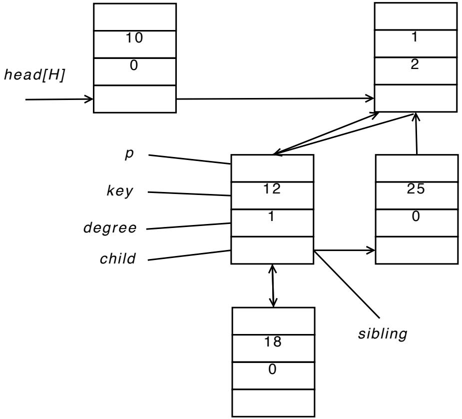
Exemplo

A figura é um heap binomial com as árvores B_0, B_2 e B_3, com $(1101)_2 = 13$ elementos:



Representação

Representamos um heap binomial com uma lista de árvores binomiais. Cada nó possui um apontador para o nó pai, uma para seu filho esquerdo e um para uma lista ligada de seus irmãos.



Algoritmos

- Criando um novo heap. Para criar um novo heap apenas alocamos e retornamos uma estrutura H tal que $head[h] =$ NIL. Este algoritmo tem complexidade $\theta(1)$.

Algoritmos

- Criando um novo heap. Para criar um novo heap apenas alocamos e retornamos uma estrutura H tal que $head[h] =$ NIL. Este algoritmo tem complexidade $\theta(1)$.
- Encontrando a menor chave. Para encontrar o menor elemento basta percorrer as raízes das árvores buscando o menor elemento. Como vimos, há no máximo $\lfloor \lg n \rfloor + 1$ raízes para checarmos – o que nos dá um algoritmo de complexidade $O(\lg n)$.

Busca pela menor chave

Algoritmo para busca o menor elemento

```
Binomial-Heap-Minimum(H)
y ← NIL
x ← head[H]
min ← ∞
while x ≠ NIL do
    if key[x] < min then
        min ← key[x]
        y ← x
    x ← sibling[x]
return y
```

União

União

- Uma vantagem dos heaps binomiais em relação aos heaps binários é a união. Esta operação pode ser feita em tempo $O(\lg n)$.

União

- Uma vantagem dos heaps binomiais em relação aos heaps binários é a união. Esta operação pode ser feita em tempo $O(\lg n)$.
- Nesta operação vamos utilizar uma função auxiliar que junta duas árvores B_{k-1}. A raíz z será também raíz da nova árvore B_k.

  Binomial-Link(y, z)
  1. $p[y] \leftarrow z$
  2. $sibling[y] \leftarrow child[z]$
  3. $child[z] \leftarrow y$
  4. $degree[z] \leftarrow degree[z] + 1$

União

- Uma vantagem dos heaps binomiais em relação aos heaps binários é a união. Esta operação pode ser feita em tempo $O(\lg n)$.
- Nesta operação vamos utilizar uma função auxiliar que junta duas árvores B_{k-1}. A raíz z será também raíz da nova árvore B_k.

  Binomial-Link(y, z)

  1. $p[y] \leftarrow z$
  2. $sibling[y] \leftarrow child[z]$
  3. $child[z] \leftarrow y$
  4. $degree[z] \leftarrow degree[z] + 1$
- Precisamos também de um procedimento Binomial-Heap-Merge, que junta dois heaps binomiais em ordem monotonicamente crescente do grau das raízes.

União

Algoritmo de união

Binomial-Heap-Union(H_1, H_2)

$H \leftarrow$ Make-Binomial-Heap()

$head[H] \leftarrow$ Binomial-Heap-Merge(H_1, H_2)

if $head[H] =$ NIL **then**

 return H

$prev_x \leftarrow$ NIL

$x \leftarrow head[H]$

$next_x \leftarrow sibling[x]$

while $next_x \neq$ NIL **do**

 if $(degree[x] \neq degree[next_x])$ or

  $(sibling[next_x] \neq$ NIL

  and $degree[sibling[next_x]] = degree[x])$ **then**

  $prev_x \leftarrow x$ /* Casos 1 e 2 */

  $x \leftarrow next_x$

União

Algoritmo de união - continuação

```
    else if key[x] ≤ key[next_x] then
        sibling[x] ← sibling[next_x] /* Caso 3 */
        Binomial-Link(next_x, x)
    else
        if prev_x = NIL then /* Caso 4 */
            head[H] ← next_x
        else
            sibling[prev_x] ← next_x
        Binomial-Link(x, next_x)
        x ← next_x
    next_x ← sibling[x]
return H
```

União

União

- O caso 1 ocorre quando não há árvores de mesmo grau consecutivas.

União

- O caso 1 ocorre quando não há árvores de mesmo grau consecutivas.
- No caso 2 há três árvores com o mesmo grau em seguida, formadas após a união de duas árvores. Exemplo: cada um dos heaps originais possuía uma B_1 e uma B_2. Ao unir-se as árvores B_1 ficamos com três B_2

União

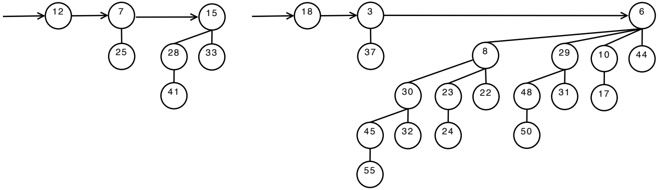
- O caso 1 ocorre quando não há árvores de mesmo grau consecutivas.
- No caso 2 há três árvores com o mesmo grau em seguida, formadas após a união de duas árvores. Exemplo: cada um dos heaps originais possuía uma B_1 e uma B_2. Ao unir-se as árvores B_1 ficamos com três B_2
- O Caso 3 as duas ávores B_{k-1} são somadas para formar uma B_k, sendo que a que possui a raíz com menor chave aparece primeiro na lista.

União

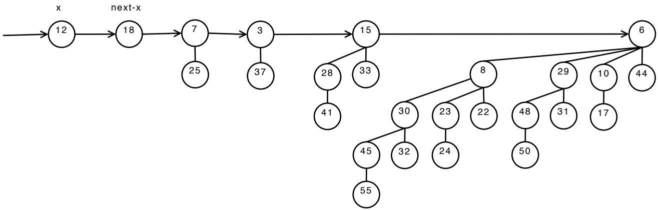
- O caso 1 ocorre quando não há árvores de mesmo grau consecutivas.
- No caso 2 há três árvores com o mesmo grau em seguida, formadas após a união de duas árvores. Exemplo: cada um dos heaps originais possuía uma B_1 e uma B_2. Ao unir-se as árvores B_1 ficamos com três B_2
- O Caso 3 as duas ávores B_{k-1} são somadas para formar uma B_k, sendo que a que possui a raíz com menor chave aparece primeiro na lista.
- Por ultimo, temos o caso análogo ao 3, porém quando a árvore que possui a menor chave aparece depois na lista.

União

Os dois heaps iniciais.

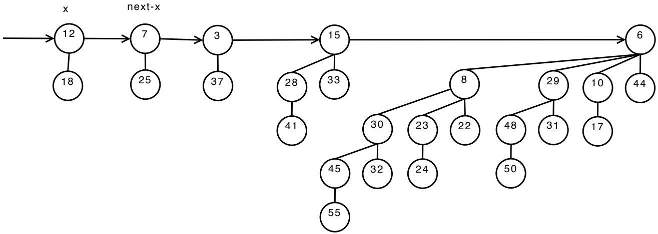


Após Binomial-Heap-Merge. Temos o caso 3.



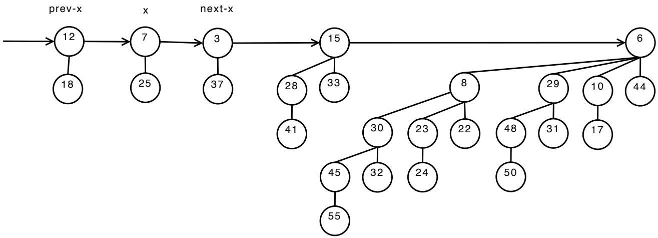
União

Caso 2


x
next-x
12
18
7
25
3
37
15
28
33
41
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55

União

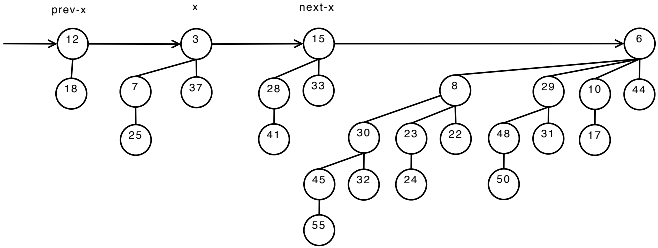
Caso 4



prev-x
x
next-x
12
7
3
15
6
18
25
37
28
33
8
29
10
44
41
30
23
22
48
31
17
45
32
24
50
55

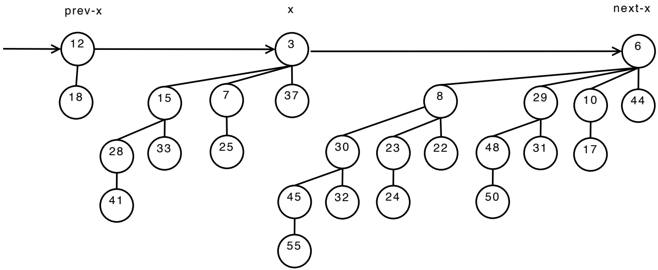
União

Caso 3



prev-x
x
next-x
12
3
15
6
18
7
37
28
33
8
29
10
44
25
41
30
23
22
48
31
17
45
32
24
50
55

Caso 1


prev-x
x
next-x
12
3
6
18
15
7
37
8
29
10
44
28
33
25
30
23
22
48
31
17
41
45
32
24
50
55

União

A complexidade do algoritmo de união é $O(\lg n)$

Prova. Se H_1 possui n_1 nós e H_2 possui n_2, então o número total de árvores binomiais é $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor + 2 \leq 2\lfloor \lg n \rfloor + 2 = O(\lg n)$, que é a complexidade de Binomial-Heap-Merge. Cada iteração do laço **while** consome tempo constante, e também é executado para cada árvore do heap, e portanto tem complexidade $O(\lg n)$.

Inserção

Inserção

- Para inserir um nó basta criarmos um novo heap contendo apenas este elemento e uni-lo ao heap em que queremos inserir-lo. Como vimos, a criar um novo heap consome tempo constante e a união é $O(\lg n)$. Portanto, inserir um novo elemento tem complexidade $O(\lg n)$.

Algoritmo de inserção

Binomial-Heap-Insert(H, x)
1. $H' \leftarrow$ Make-Binomial-Heap()
2. $p[x] \leftarrow$ NIL
3. $child[x] \leftarrow$ NIL
4. $sibling[x] \leftarrow$ NIL
5. $degree[x] \leftarrow 0$
6. $head[H'] \leftarrow x$
7. $H \leftarrow$ Binomial-Heap-Union(H, H')

Extraindo o menor elemento

Extraindo o menor elemento

- Para extrair o menor elemento do heap buscamos sua posição (em $O(\lg n)$) e extraímos a árvore em que ele é raíz, digamos B_k.

Extraindo o menor elemento

- Para extrair o menor elemento do heap buscamos sua posição (em $O(\lg n)$) e extraímos a árvore em que ele é raíz, digamos B_k.
- Criamos um novo heap H' a partir das subárvores deste elemento, $B_0, B_1, \ldots, B_{k-1}$ e fazemos a união de H e H'. Esta operação tem complexidade $O(\lg n)$.

Extraindo o menor elemento

- Para extrair o menor elemento do heap buscamos sua posição (em $O(\lg n)$) e extraímos a árvore em que ele é raíz, digamos B_k.
- Criamos um novo heap H' a partir das subárvores deste elemento, $B_0, B_1, \ldots, B_{k-1}$ e fazemos a união de H e H'. Esta operação tem complexidade $O(\lg n)$.
- Assim, concluímos que a operação de extrair o menor elemento do heap tem complexidade $O(\lg n)$.

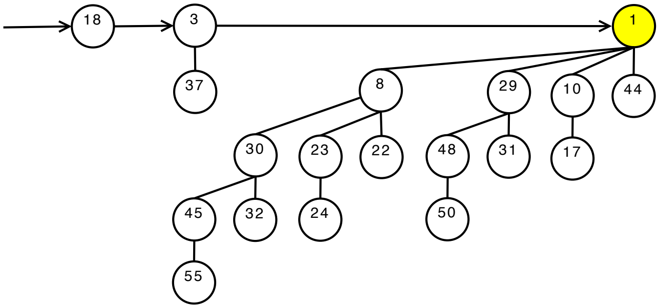
Extraindo o menor elemento

Algoritmo para extrair o menor elemento

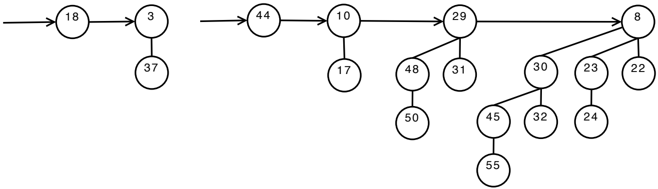
Binomial-Heap-Extract-Min()

1. Encontre a raíz x com a menor chave em H e a remova da lista de raízes de H
2. $H' \leftarrow$ Make-Binomial-Heap()
3. Inverta a ordem da lista ligada de filhos de x e a atribua a H'
4. $H \leftarrow$ Binomial-Heap-Union(H,H')
5. **return** x

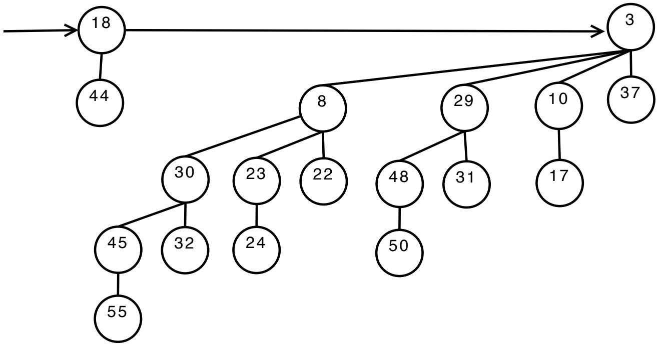
Extraindo o menor elemento



Extraindo o menor elemento



Extraindo o menor elemento



Conclusão

- Os heaps binomiais são eficientes na operação de união.

Complexidades dos algoritmos para três tipos de heap

Procedimento	Heap binário (pior caso)	Heap binomial (pior caso)	Heap de Fibonacci (amortizado)
Make-heap	$\theta(1)$	$\theta(1)$	$\theta(1)$
Insert	$\theta(\lg n)$	$O(\lg n)$	$\theta(1)$
Minimum	$\theta(1)$	$O(\lg n)$	$\theta(1)$
Extract-Min	$\theta(\lg n)$	$\theta(\lg n)$	$O(\lg n)$
Union	$\theta(n)$	$O(\lg n)$	$\theta(1)$
Decrease-key	$\theta(\lg n)$	$\theta(\lg n)$	$\theta(1)$
Delete	$\theta(\lg n)$	$\theta(\lg n)$	$O(\lg n)$