优先队列

汪小林

北大计算机系

主要内容

- 优先队列和二叉堆
- 二项树及二项堆
- 斐波那契堆

优先队列和二叉堆

优先队列

- 优先队列
  - 一种关于集合***S***的数据结构，集合中的元素都有键值***key***
  - 最大优先队列支持以下操作：
    - **INSERT(*S*, *x*)**：把元素***x***插入集合***S***，可表示为***S*←*S*∪{*x*}**
    - **MAXIMUM(*S*)**：返回集合***S***中具有最大键值的元素
    - **EXTRACT-MAX(*S*)**：去掉并返回集合*S*中具有最大键值的元素
    - **INCREASE-KEY(*S*, *x*, *k*)**：将元素***x***的键值增加到***k***，这里要求***k***不能小于***x***的原键值。
  - 类似的，最小优先队列支持以下操作：
    - **INSERT(*S*, *x*)**、**MININUM(*S*)**
    - **EXTRACT-MIN(*S*)**、**DECREASE-KEY(*S*, *x*, *k*)**

二叉堆

- 二叉堆是在一个数组上通过下标间关系维护父子结点关系的一棵几乎满的二叉树

PARENT(*i*)

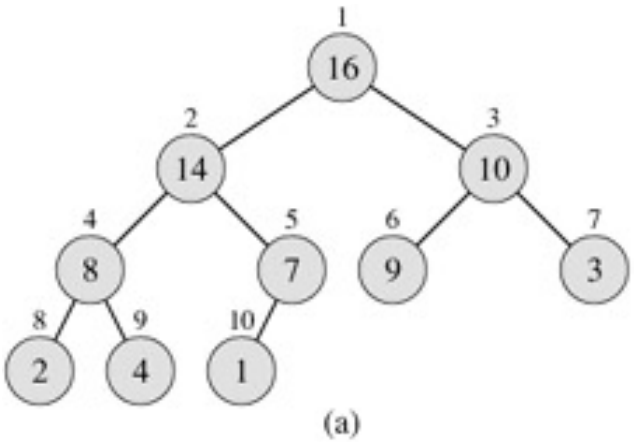
return $\lfloor i/2 \rfloor$

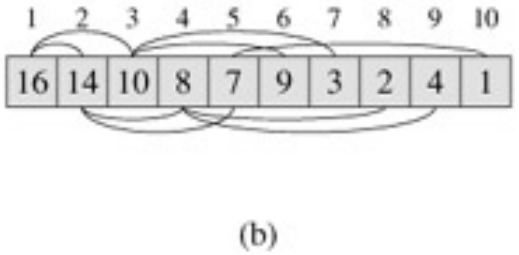
LEFT(*i*)

return $2i$

RIGHT(*i*)

return $2i + 1$





保持堆的性质

MAX-HEAPIFY(*A*, *i*)

1 *l* ← LEFT(*i*)

2 *r* ← RIGHT(*i*)

3 if *l* ≤ *heap-size*[*A*] and *A*[*l*] > *A*[*i*]

4 then *largest* ← *l*

5 else *largest* ← *i*

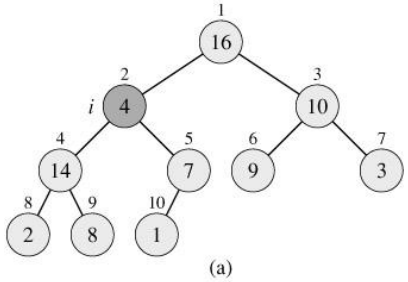
6 if *r* ≤ *heap-size*[*A*] and *A*[*r*] > *A*[*largest*]

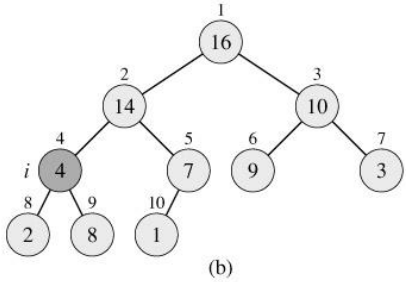
7 then *largest* ← *r*

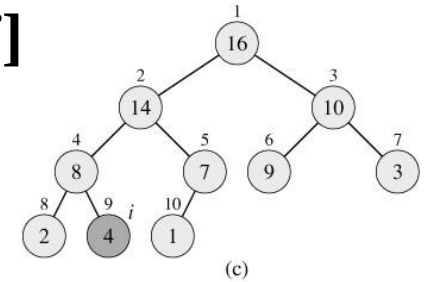
8 if *largest* ≠ *i*

9 then exchange *A*[*i*] ↔ *A*[*largest*]

10 MAX-HEAPIFY(*A*, *largest*)







建堆

BUILD-MAX-HEAP(*A*)

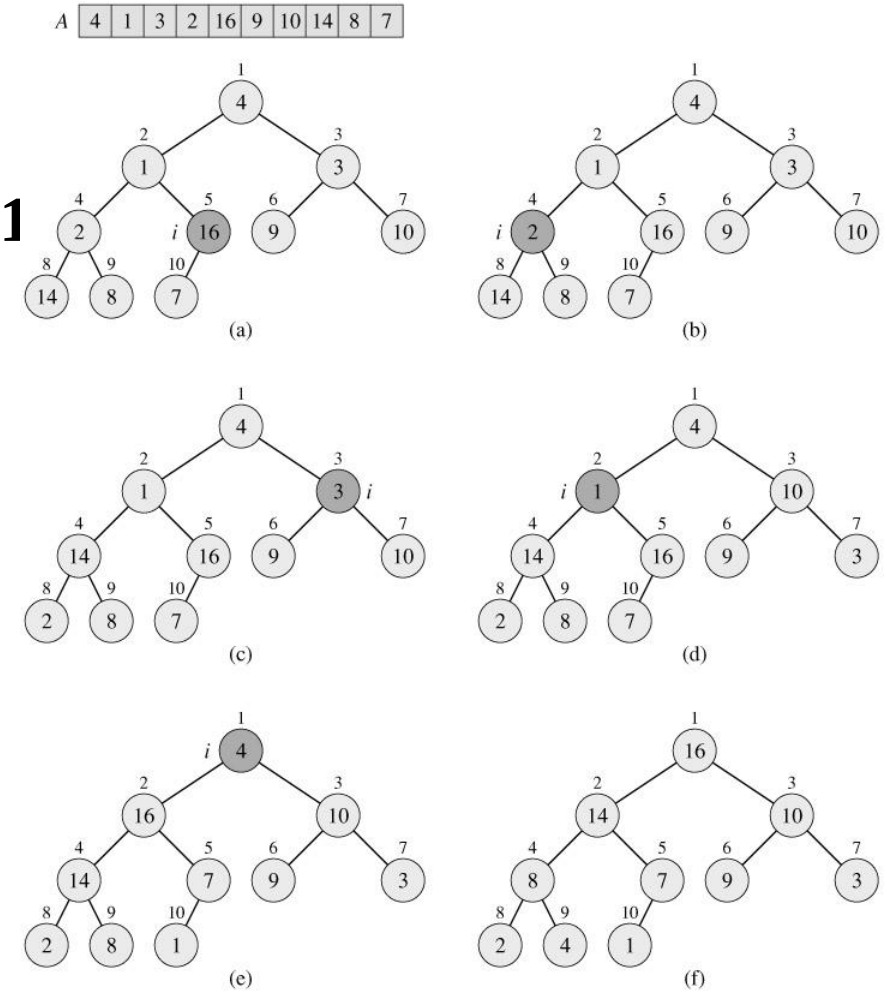
1 *heap-size*[*A*] ← *length*[*A*]

2 for *i* ← [*length*[*A*]/2] downto 1

3 do MAX-HEAPIFY(*A*, *i*)

总运行时间为**O(*n*)**。

合并堆的过程需要用重建堆来实现。



取最大元素 / 取出最大元素

HEAP-MAXIMUM(*A*)

1 return *A*[1]

HEAP-EXTRACT-MAX(*A*)

1 if *heap-size*[*A*] < 1

2 then error "heap underflow"

3 *max* ← *A*[1]

4 *A*[1] ← *A*[*heap-size*[*A*]]

5 *heap-size*[*A*] ← *heap-size*[*A*] - 1

6 MAX-HEAPIFY(*A*, 1)

7 return *max*

增加元素的键值 / 插入新元素

HEAP-INCREASE-KEY(*A*, *i*, *key*)

```
1 if key < A[i]
2     then error "new key is smaller than current key"
3 A[i] ← key
4 while i > 1 and A[PARENT(i)] < A[i]
5     do exchange A[i] ↔ A[PARENT(i)]
6        i ← PARENT(i)
```

MAX-HEAP-INSERT(*A*, *key*)

```
1 heap-size[A] ← heap-size[A] + 1
2 A[heap-size[A]] ← -∞
3 HEAP-INCREASE-KEY(A, heap-size[A], key)
```

二项堆

可合并堆

- 可合并堆(***mergeable heaps***)
  - **MAKE-HEAP()** 创建一个空的堆
  - **INSERT(*H*, *x*)** 把具有键值***key***的结点***x***插入堆***H***.
  - **MINIMUM(*H*)** 返回堆***H*** 中具有最小键值的结点
  - **EXTRACT-MIN(*H*)** 删除堆***H*** 中具有最小键值的结点并返回之
  - **UNION(H_1, H_2)** 创建一个包含了堆H_1和堆H_2所有结点的新堆，原来的堆H_1和堆H_2 被此操作“销毁”
- 其他两个堆上的操作
  - **DECREASE-KEY(*H*, *x*, *k*)** 给堆***H***中的结点***x***一个新键值***k***，新键值不应比结点***x***原来的键值大
  - **DELETE(*H*, *x*)** 删除堆*H*中的结点

不同堆上的操作开销比较

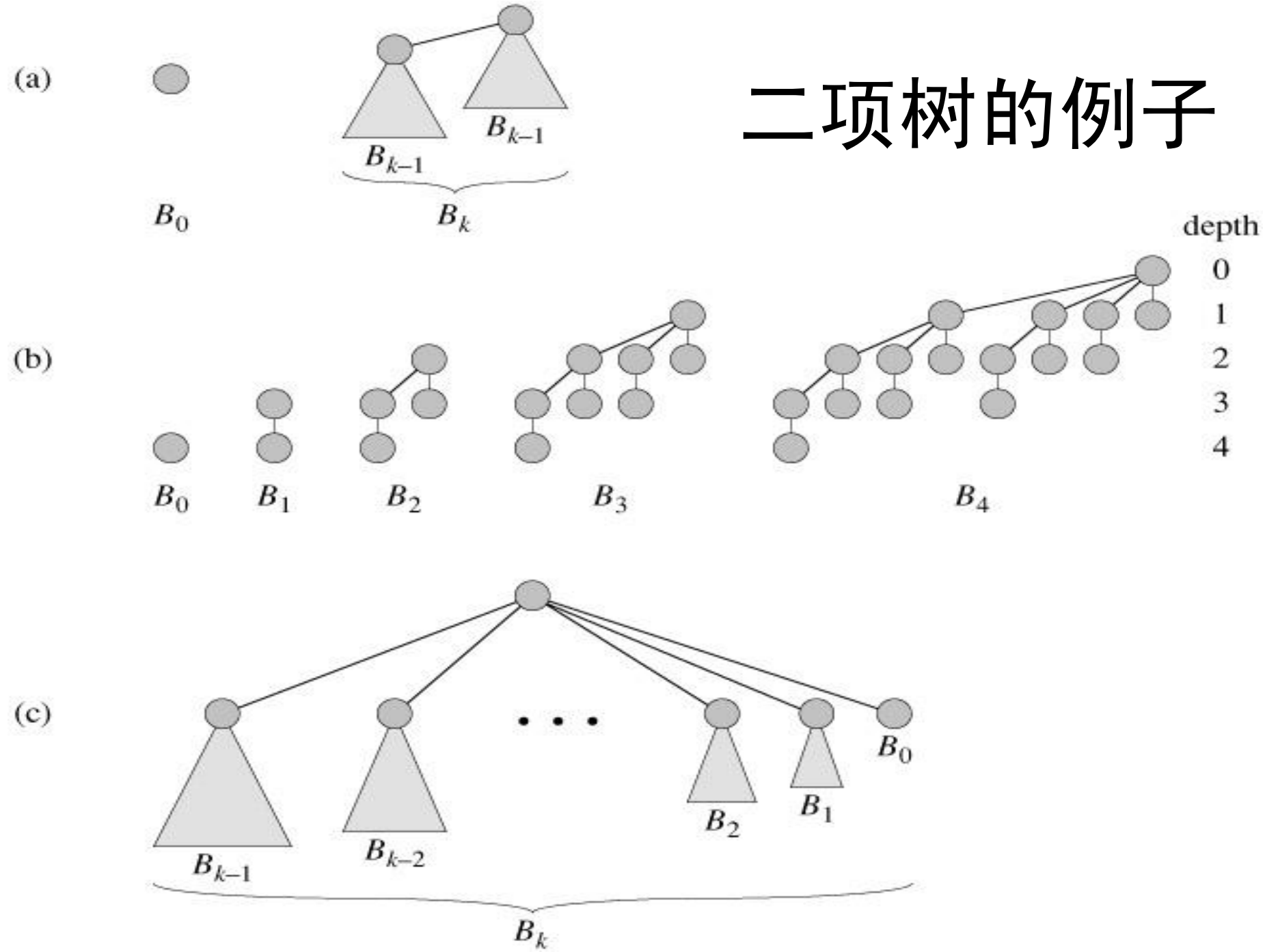
操作	二叉堆	二项堆	斐波那契堆*
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$O(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$O(\lg n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$O(\lg n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$

注：二叉堆和二项堆都是最差情形下的开销，斐波那契堆则是平摊开销

二项树

- 二项堆是一组二项树的集合
- 二项树是一个递归定义的有序树
  - 二项树B_0包含一个结点；二项树B_k 由两棵二项树B_{k-1}链接而成，其中一棵树的根是另一棵树的根的最左孩子
- 二项树B_k的性质
  1. 共有2^k个结点（设二项树B_k包含n个结点，则$k=\lg n$）
  2. 树的高度为k
  3. 在深度i处恰好有 $\binom{k}{i}$ 个结点，其中$i=0,1,2...k$
  4. 根的度数为k，它大于任何其他结点的度；并且根的子女从左到右编号$k-1,k-2,...,0$，则子女i是子树B_i的根

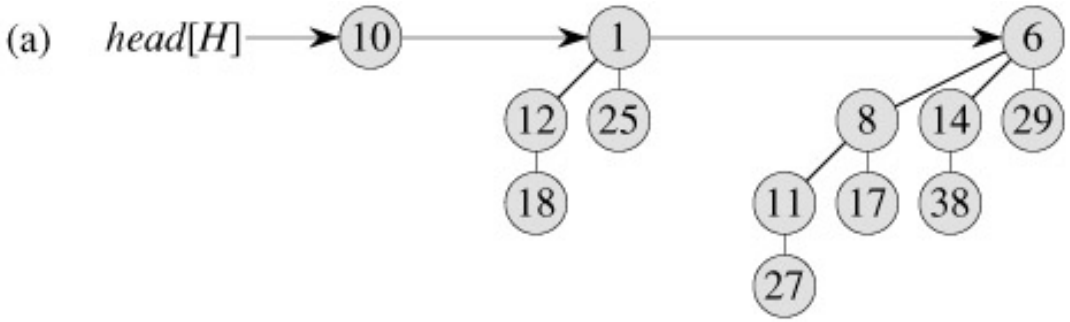
二项树的例子

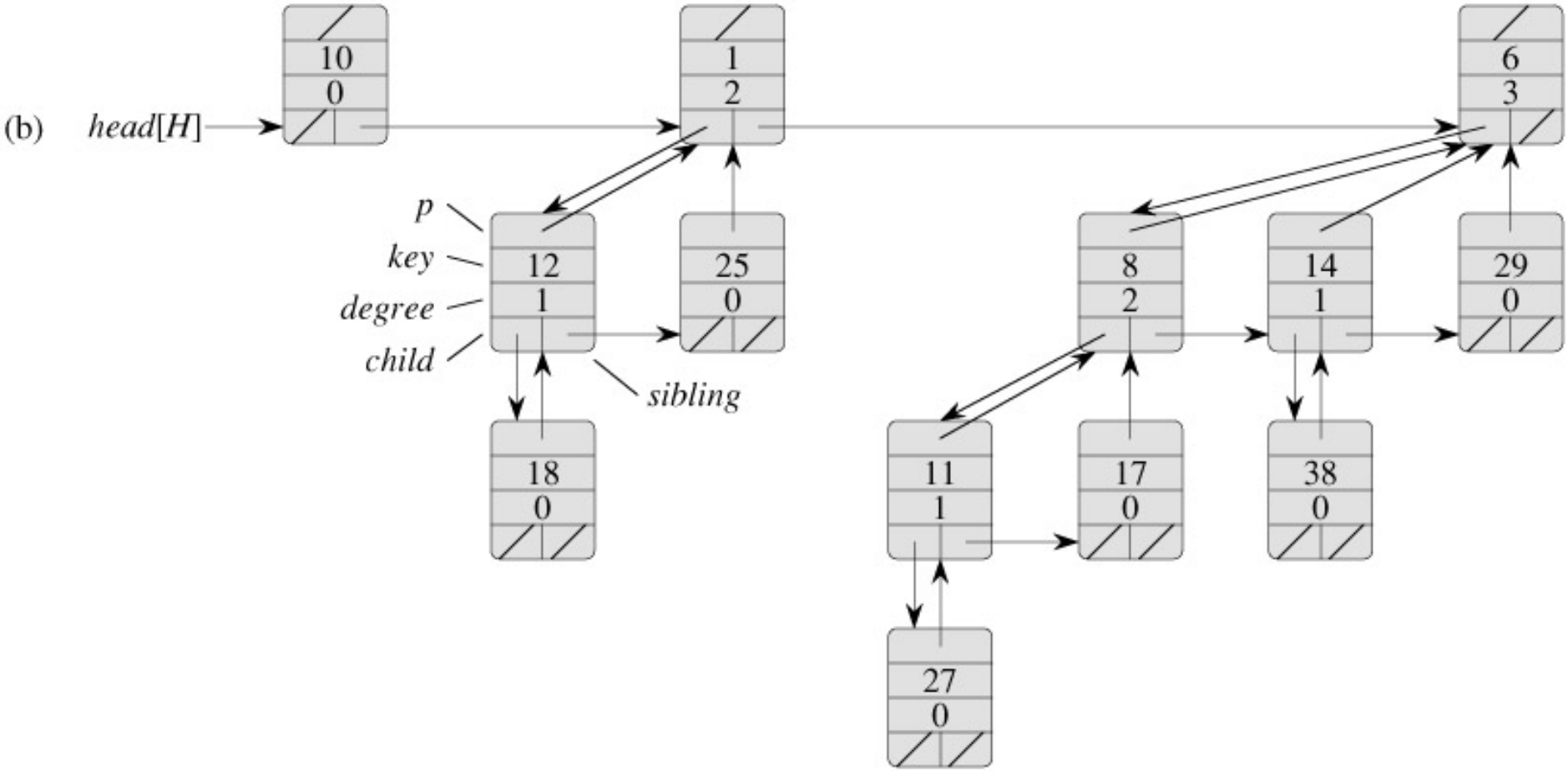


二项堆

- 二项堆H由一组满足二项堆性质的二项树组成
  1. H中的每个二项树遵循最小堆的性质；结点的键值大于等于其父结点的键值（这种树为最小堆有序的）
  2. 对任意非负整数k，在H中至多有一棵二项树的根具有度数k
- 由二项堆的性质，我们有
  - 一棵最小堆有序的二项树的根包含了树中最小的键值
  - 有n个结点的二项堆中，包含至多$\lfloor \lg n \rfloor + 1$棵二项树
    - 设$n = \sum_{0 \le i \le \lfloor \lg n \rfloor} b_i 2^i$，因二项树$B_k$共有$2^k$个结点
    - 所以，二项树B_i出现于H中当且仅当位$b_i = 1$

二项堆的表示


(a)
head[H]
10
1
6
12
25
8
14
29
18
11
17
38
27


(b)
head[H]
10
0
1
2
6
3
p
key
degree
child
sibling
12
1
25
0
8
2
14
1
29
0
18
0
11
1
17
0
38
0
27
0

对二项堆的操作

创建一个空的二项堆

MAKE-BINOMIAL-HEAP ()

1 *head*[*H*] ← NIL

2 return *H*

运行时间为**O(1)**

取最小键值的结点

BINOMIAL-HEAP-MINIMUM(*H*)

```
1 y ← NIL
2 x ← head[H]
3 min ← ∞
4 while x ≠ NIL
5     do if key[x] < min
6         then min ← key[x]
7              y ← x
8         x ← sibling[x]
9 return y
```

二项堆是最小堆有序的，所以最小键值必在某个二项树的根结点中，只需遍历所有的根即可找到最小键值的根结点

运行时间为**O(lg*n*)**

合并两个二项堆

BINOMIAL-LINK(*y*,*z*)

1 ***p*[*y*]** ← ***z***

2 ***sibling*[*y*]** ← ***child*[*z*]**

3 ***child*[*z*]** ← ***y***

4 ***degree*[*z*]** ← ***degree*[*z*]** + **1**

合并两个二项堆可通过反复用**BINOMIAL-LINK(*y*,*z*)** 链接根结点度数相同的两个二项树。

合并堆的运行时间为**O(lg*n*)**

BINOMIAL-LINK(*y*,*z*)把两个B_{k-1}的二项树链接成一个B_k的二项树，*z*成为*y*的父结点。

合并操作是后面各种操作的基础

合并两个二项堆

BINOMIAL-HEAP-UNION(H_1, H_2)

1 $H \leftarrow$ **MAKE-BINOMIAL-HEAP()**

2 *head*[H] $\leftarrow$ **BINOMIAL-HEAP-MERGE(H_1, H_2)**

3 free the objects H_1 and H_2 but not the lists they point to

4 if *head*[H] = **NIL**

5 **then return** H

6 *prev-x* $\leftarrow$ **NIL**

7 $x \leftarrow$ *head*[H]

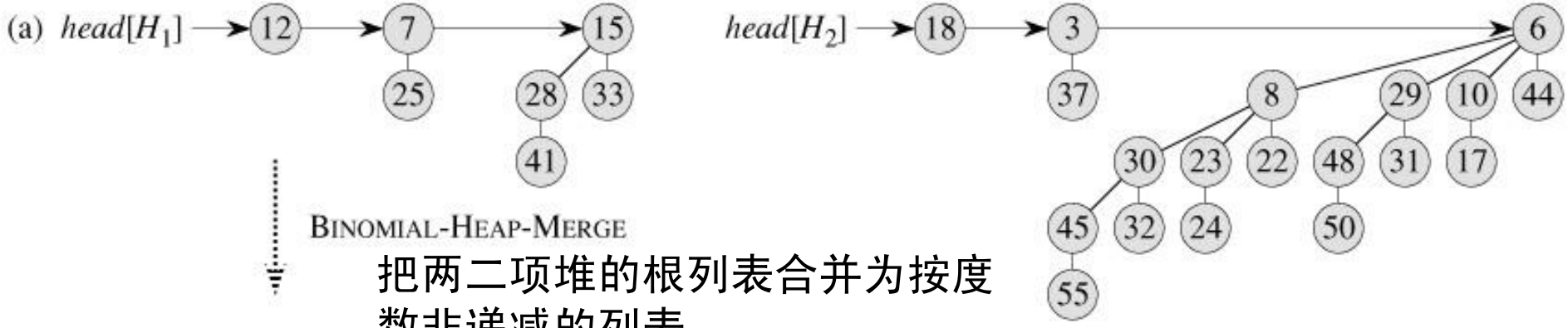
8 *next-x* $\leftarrow$ *sibling*[x]

9 while *next-x* $\neq$ **NIL**

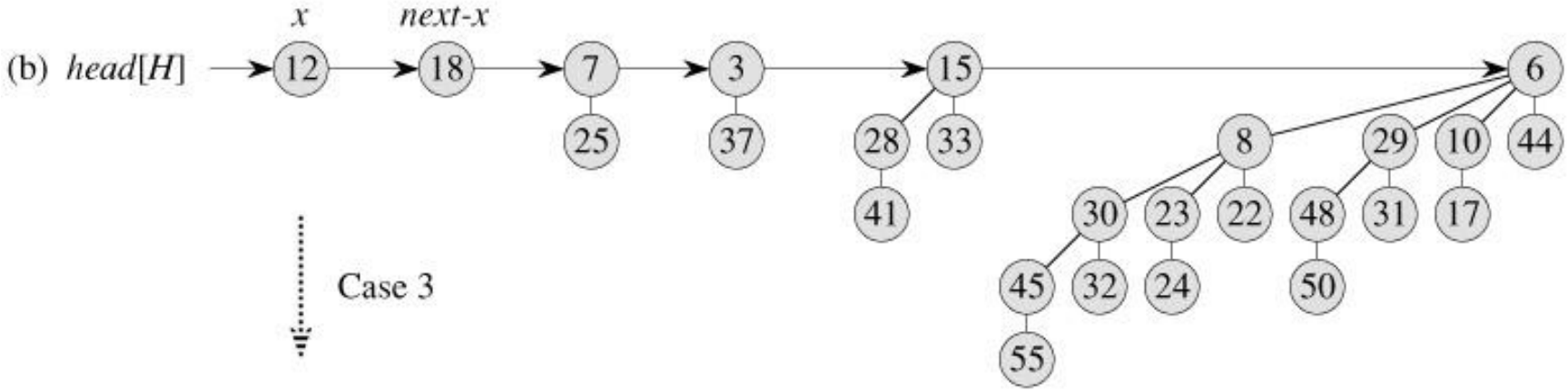
10 **do if (***degree*[x] $\neq$ *degree*[*next-x*]) **or**
(*sibling*[*next-x*] $\neq$ **NIL and** *degree*[*sibling*[*next-x*]] = *degree*[x])

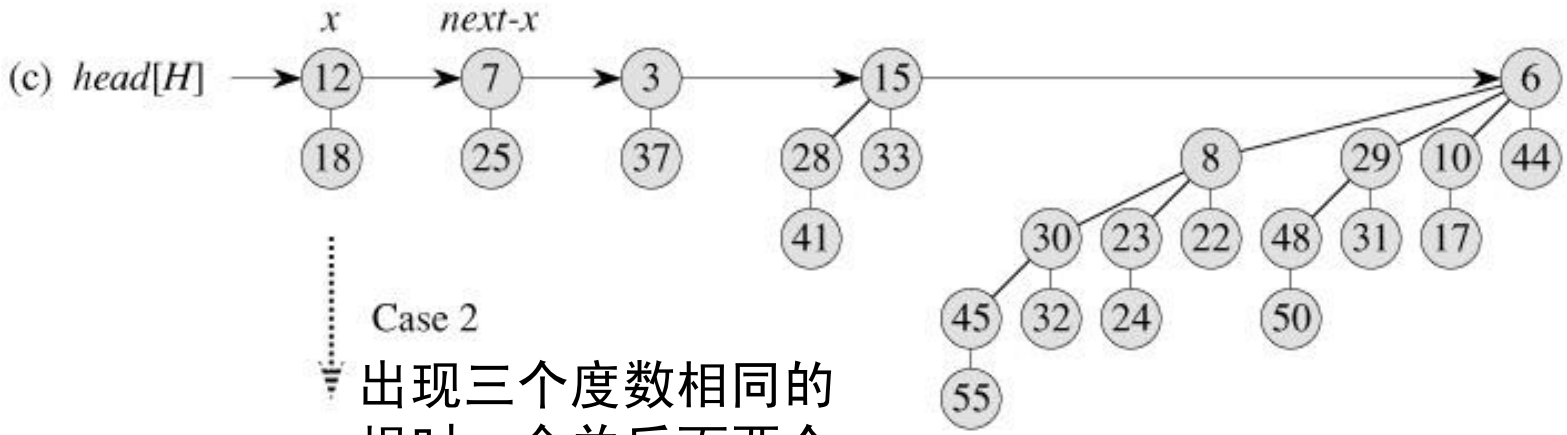
合并两个二项堆

```
11      then prev-x ← x                          ▷ Cases 1 and 2
12           x ← next-x                          ▷ Cases 1 and 2
13      else if key[x] ≤ key[next-x]
14           then sibling[x] ← sibling[next-x]   ▷ Case 3
15                BINOMIAL-LINK(next-x, x)       ▷ Case 3
16      else if prev-x = NIL                     ▷ Case 4
17           then head[H] ← next-x               ▷ Case 4
18           else sibling[prev-x] ← next-x       ▷ Case 4
19           BINOMIAL-LINK(x, next-x)            ▷ Case 4
20           x ← next-x                          ▷ Case 4
21     next-x ← sibling[x]
22 return H
```

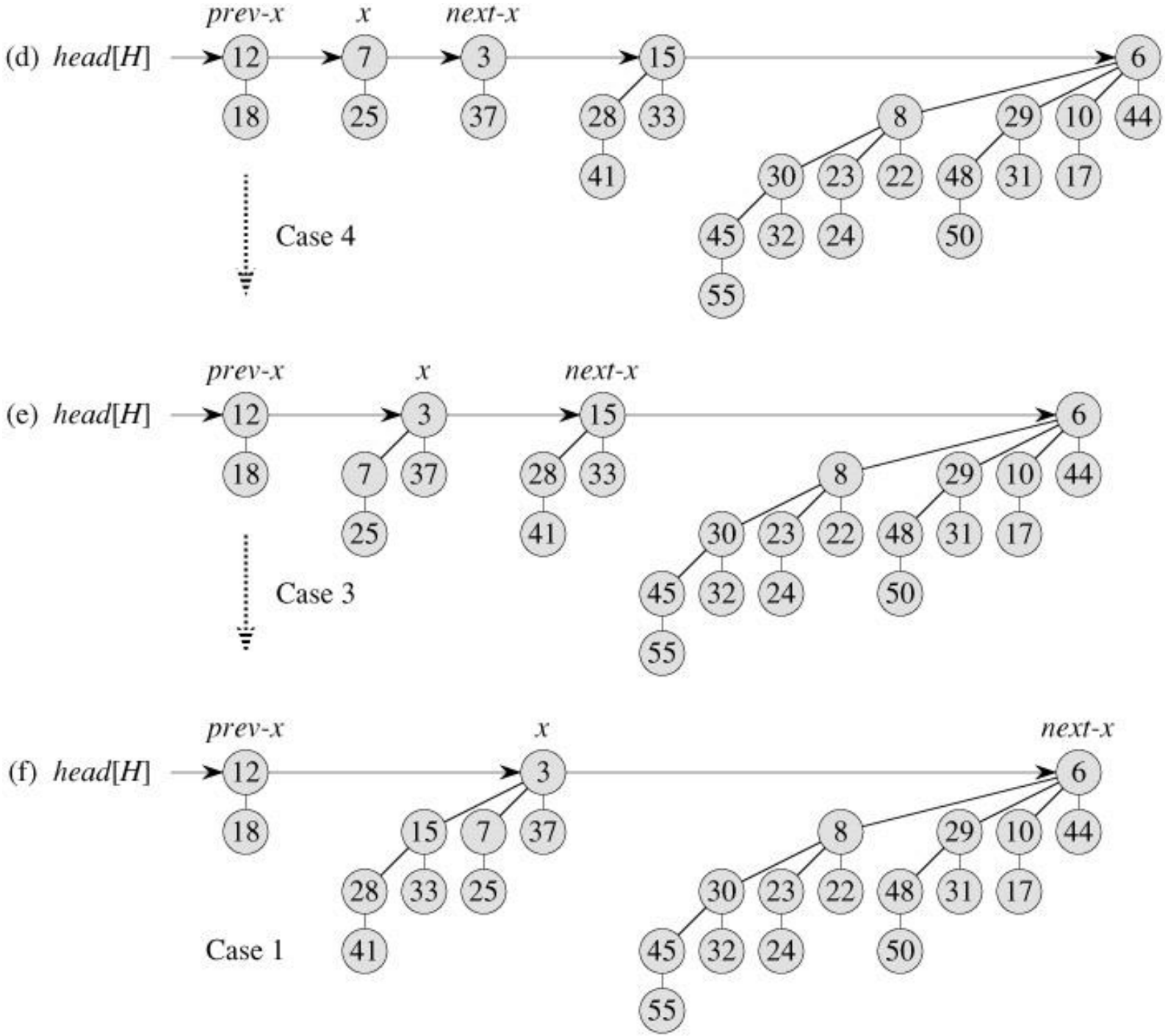


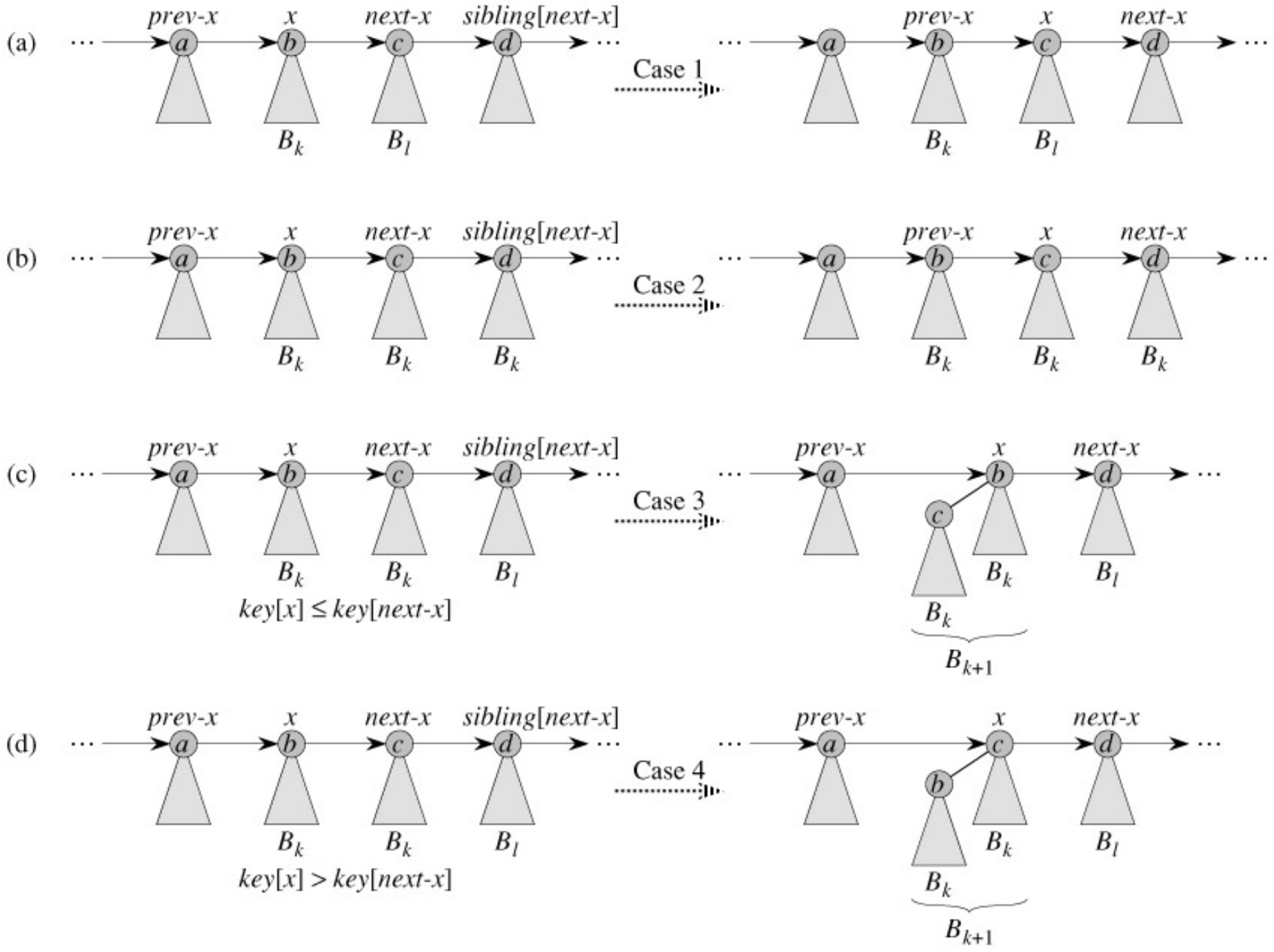
把两二项堆的根列表合并为按度数非递减的列表





出现三个度数相同的根时，合并后面两个


(d) head[H]
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
Case 4
(e) head[H]
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
Case 3
(f) head[H]
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
Case 1


(a)
prev-x
x
next-x
sibling[next-x]
a
b
c
d
B_k
B_l
Case 1
(b)
Case 2
(c)
Case 3
$key[x] \le key[next\text{-}x]$
B_{k+1}
(d)
Case 4
$key[x] > key[next\text{-}x]$

插入一个结点

BINOMIAL-HEAP-INSERT(*H*, *x*)

1 ***H′*** **←** **MAKE-BINOMIAL-HEAP()**

2 ***p*[*x*]** **←** **NIL**

3 ***child*[*x*]** **←** **NIL**

4 ***sibling*[*x*]** **←** **NIL**

5 ***degree*[*x*]** **←** **0**

6 ***head*[*H′*]** **←** ***x***

7 ***H*** **←** **BINOMIAL-HEAP-UNION(*H*, *H′*)**

把新结点作为一个二项堆***H′***和***H***合并，运行时间为**O(lg*n*)**

抽取最小键值结点

BINOMIAL-HEAP-EXTRACT-MIN(*H*)

1 find the root *x* with the minimum key in the root list of *H*, and remove *x* from the root list of *H*

2 *H′* ← MAKE-BINOMIAL-HEAP()

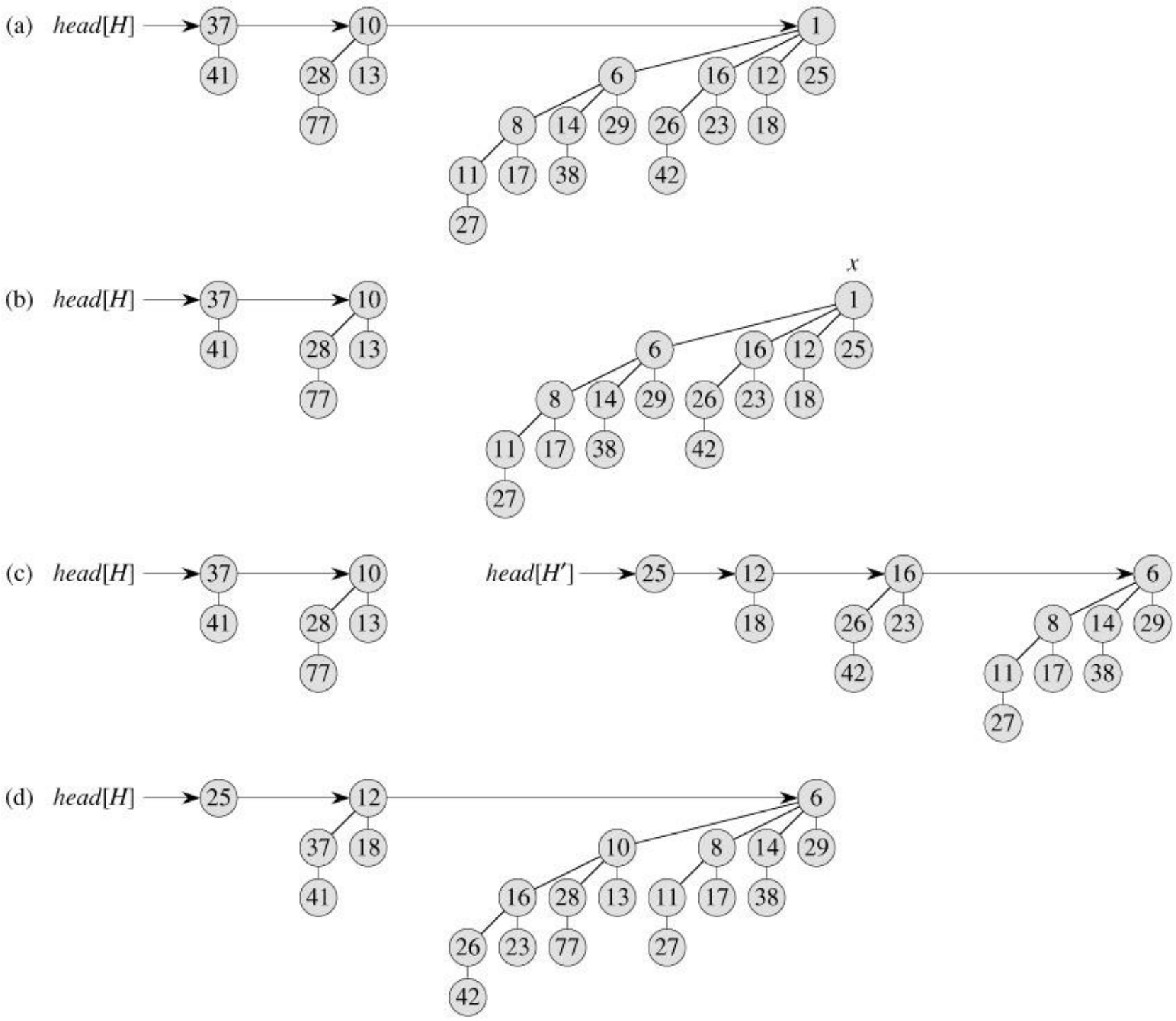
3 reverse the order of the linked list of *x*'s children, and set *head*[*H′*] to point to the head of the resulting list

4 *H* ← BINOMIAL-HEAP-UNION(*H*, *H′*)

5 return *x*

把最小键值所在的二项树的所有子树作为二项堆***H′***，和删除了该二项树后的堆***H***合并

运行时间为**O(lg*n*)**


(a) head[H]
37
10
1
41
28
13
6
16
12
25
77
8
14
29
26
23
18
11
17
38
42
27
(b) head[H]
37
10
41
28
13
77
x
1
6
16
12
25
8
14
29
26
23
18
11
17
38
42
27
(c) head[H]
37
10
41
28
13
77
head[H']
25
12
16
6
18
26
23
8
14
29
42
11
17
38
27
(d) head[H]
25
12
6
37
18
10
8
14
29
41
16
28
13
11
17
38
26
23
77
27
42

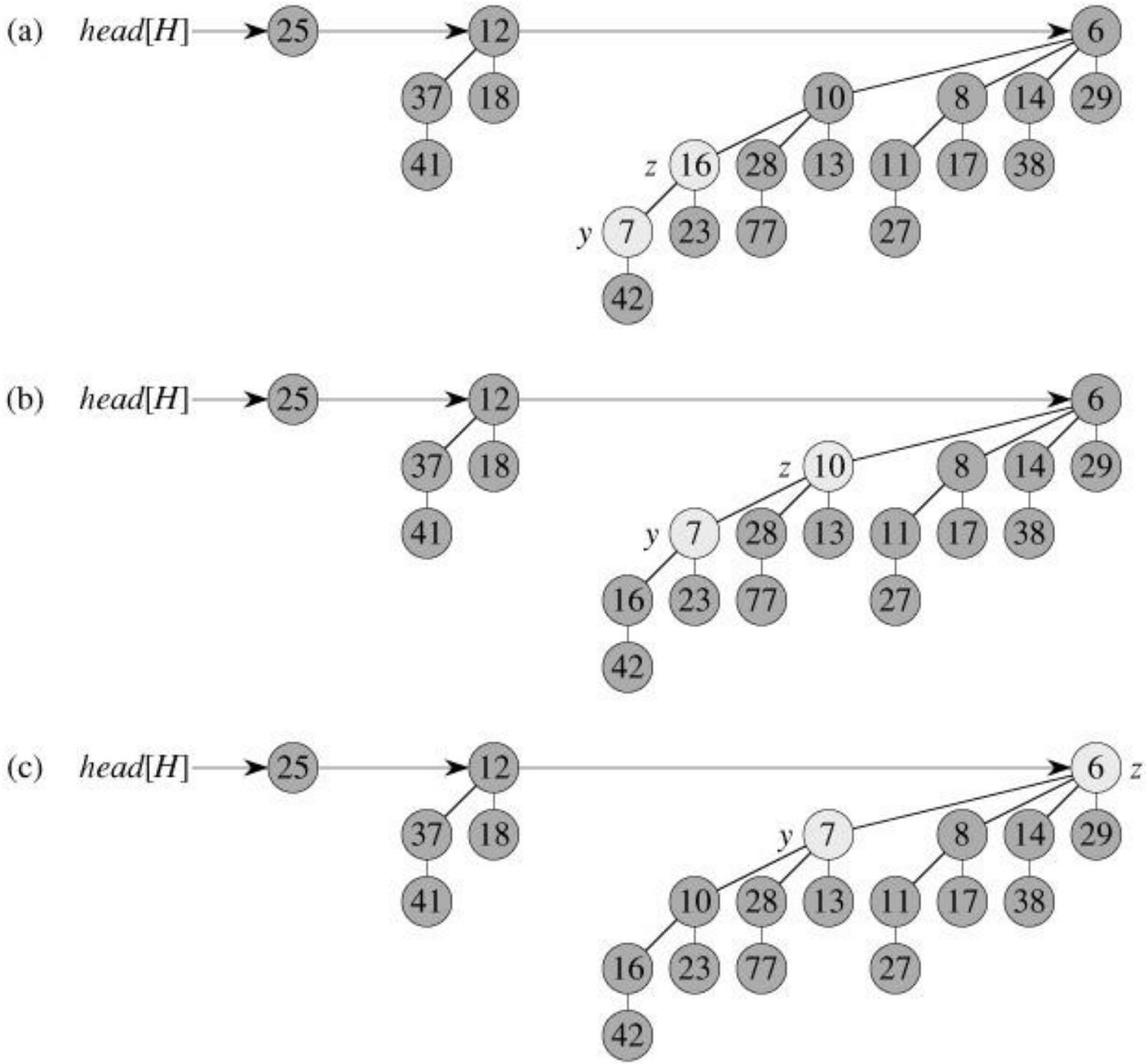
减小一个结点的键值

BINOMIAL-HEAP-DECREASE-KEY(*H*, *x*, *k*)

```
1 if k > key[x]
2    then error "new key is greater than current key"
3 key[x] ← k
4 y ← x
5 z ← p[y]
6 while z ≠ NIL and key[y] < key[z]
7      do exchange key[y] ↔ key[z]
8           ▸ If y and z have satellite fields, exchange them, too.
9           y ← z
10          z ← p[y]
```

把减小了键值的结点在满足最小堆有序的二项树上向根结点方向交换移动

运行时间为**O(lg*n*)**


(a)
head[H]
25
12
6
37
18
10
8
14
29
41
z
16
28
13
11
17
38
y
7
23
77
27
42
(b)
head[H]
25
12
6
37
18
z
10
8
14
29
41
y
7
28
13
11
17
38
16
23
77
27
42
(c)
head[H]
25
12
6
z
37
18
y
7
8
14
29
41
10
28
13
11
17
38
16
23
77
27
42

删除一个结点

BINOMIAL-HEAP-DELETE(*H*, *x*)

1 BINOMIAL-HEAP-DECREASE-KEY(*H*, *x*, -∞)

2 BINOMIAL-HEAP-EXTRACT-MIN(*H*)

把待删除结点的键值减至无穷小
通过抽取最小键值结点的方法删除该结点
运行时间为**O(lg*n*)**

斐波那契堆

斐波那契堆的特点

- 对不涉及删除的操作仅需**O(1)**的平摊运行时间
- 适用于涉及**DECREASE-KEY**操作较多，但**EXTRACT-MIN**和**DELETE**操作较少的应用
  - 最小生成树、单源最短路径
- 因为其上操作的算法较复杂，因而常数因子较大，实际应用中当***n***不太大时，二叉堆更好用
- 斐波那契堆松散的基于二项堆

斐波那契堆的结构

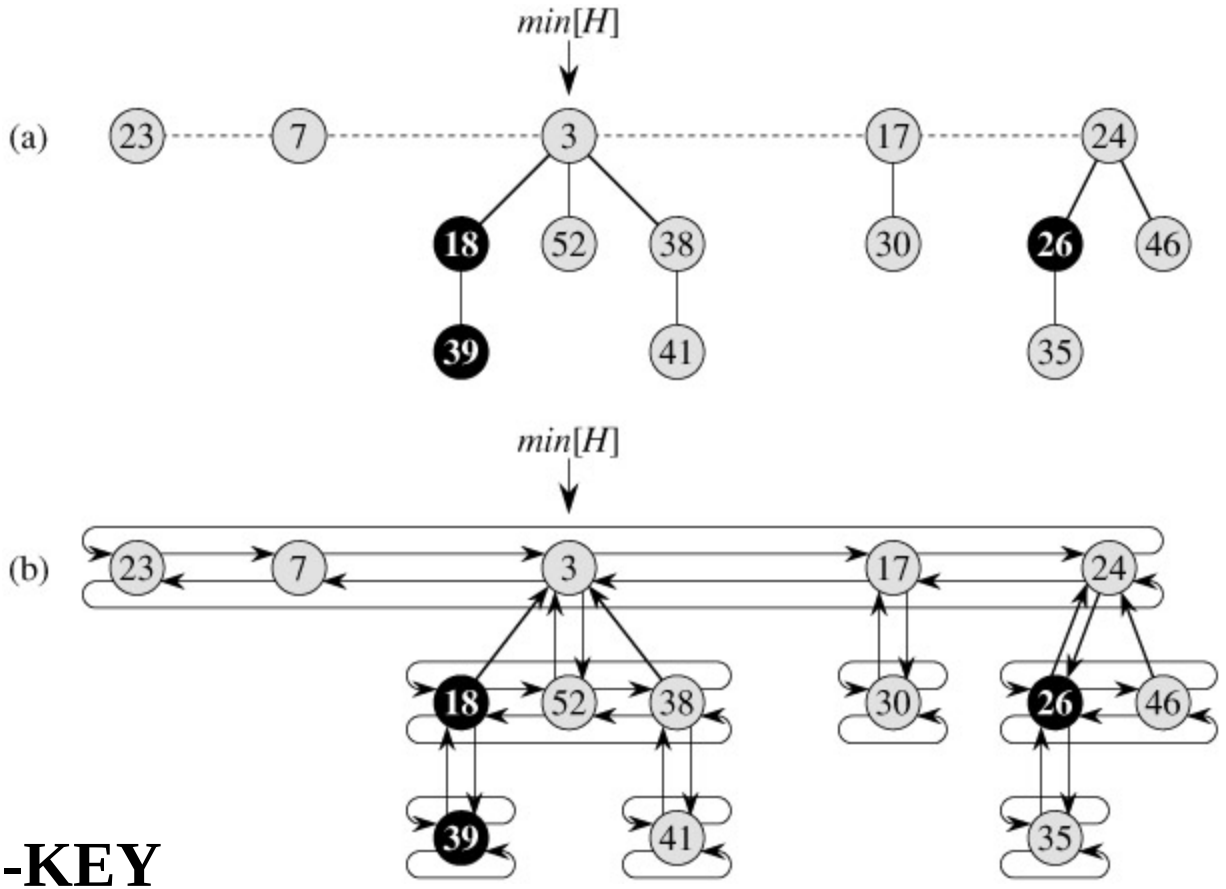
每个结点上

左右兄弟指针
形成双链表

记住每个结点
的度***degree***

标记结点成为另
一个结点的孩
子以来是否失
掉了孩子

仅在**DECREASE-KEY**
才置***mark*****(=TRUE)**



斐波那契堆上的势函数

- 将用势能法分析其上操作的平摊开销
- 斐波那契堆上的势定义为
  - $\Phi(H) = t(H) + 2m(H)$
    - $t(H)$：H中树的棵数
    - $m(H)$：H中有标记的结点的个数
- 假设初始堆都是空堆，初始势能为0，所以基于势能法的平摊开销之和就是总实际开销的上界

斐波那契堆上的结点的最大度数

- 在后面对斐波那契堆的平摊分析中，都假设包含n个结点的斐波那契堆中结点的最大度数有一个已知上界$D(n)$
  - 练习**20.2-3**说明，当斐波那契堆仅支持可合并堆操作时
    $$D(n) \leq \lfloor \lg n \rfloor$$
  - **20.3**节中说明，如果斐波那契堆还支持**DECREASE-KEY**和**DELETE**操作时
    $$D(n) = O(\lg n)$$

减小平摊开销的关键思想

- 尽可能久的将要做的工作推后完成。
- 在各种操作之间做性能权衡。
- 插入和合并两个斐波那契堆时，并不去合并树，而是将合并树的工作留给**EXTRACT-MIN**操作，因为只有**EXTRACT-MIN**操作希望树的棵数最少
- 这样就不需要其他的操作为保持树的棵数较小而花费较大的代价

创建一个新的斐波那契堆

MAKE-FIB-HEAP()

1 alloc an *H* object.

2 *n*[*H*] ← 0

3 *min*[*H*] ← NIL

4 return *H*

此时***H***中没有树，***t*(*H*)=0**，***m*(*H*)=0**，所以空斐波那契堆的势**Φ(*H*) =0**，因此**MAKE-FIB-HEAP**的平摊开销就等于其**O(1)**的实际开销。

插入一个结点

FIB-HEAP-INSERT(*H*, *x*)

1 ***degree*[*x*] ← 0**

2 ***p*[*x*] ← NIL**

3 ***child*[*x*] ← NIL**

4 ***left*[*x*] ← *x***

5 ***right*[*x*] ← *x***

6 ***mark*[*x*] ← FALSE**

7 concatenate the root list containing *x* with root list *H*

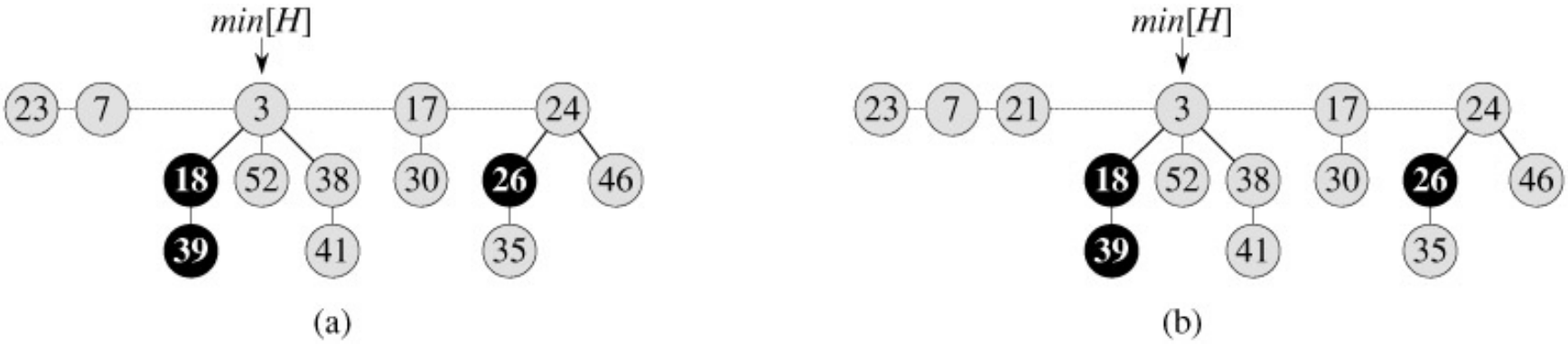
8 if *min*[*H*] = NIL or *key*[*x*] < *key*[*min*[*H*]]

9 then *min*[*H*] ← *x*

10 ***n*[*H*] ← *n*[*H*] + 1**

把插入的结点初始化后插入到堆的根表中，同时与最小键值结点比较，如果更小，则重新设置最小键值结点为插入的结点。

插入一个结点



- 这里插入结点时并不合并树（这与二项堆的插入要进行树的合并不同），插入的平摊代价
  - 设H插入一个结点后为H'，
    则$t(H')=t(H)+1$, $m(H')=m(H)$
  - 势差$\Phi(H') - \Phi(H) =1$
  - 实际代价为$O(1)$，故平摊代价为$O(1)+1= O(1)$。

寻找最小键值结点

FIB-HEAP-MINIMUM(*H*)

1 return *min*[*H*]

斐波那契堆结构中有***min*[*H*]**指向具有最小键值的结点，找最小键值结点的实际开销为**O(1)**，又因为势没有变化，因此平摊开销和实际开销一样，都是**O(1)**

合并两个斐波那契堆

FIB-HEAP-UNION(H_1, H_2)

1 $H \leftarrow$ **MAKE-FIB-HEAP()**

2 $min[H] \leftarrow min[H_1]$

3 concatenate the root list of H_2 **with the root list of** H

4 if ($min[H_1]$ = NIL) **or** ($min[H_2] \neq$ NIL **and** $min[H_2] < min[H_1]$)

5 **then** $min[H] \leftarrow min[H_2]$

6 $n[H] \leftarrow n[H_1] + n[H_2]$

7 free the objects H_1 **and** H_2

8 return H

合并操作仅把两个根表合并，实际开销为**O(1)**，又因为合并操作不改变势（新堆的势与原来两个堆的势之和相等），所以平摊开销为**O(1)**。

抽取最小键值结点

```
FIB-HEAP-EXTRACT-MIN(H)
1 z ← min[H]
2 if z ≠ NIL
3     then for each child x of z
4              do add x to the root list of H
5                 p[x] ← NIL
6          remove z from the root list of H
7          if z = right[z]
8             then min[H] ← NIL
9             else min[H] ← right[z]
10                 CONSOLIDATE(H)
11         n[H] ← n[H] - 1
12 return z
```

先使最小键值结点的每个子女都成为一个根，并将最小键值结点从根表中去掉。

然后，通过将度数相同的根链接起来，直至对应每个度数至多只有一个根，来调整根表。

CONSOLIDATE(*H*)

```
1 for i ← 0 to D(n[H])
2     do A[i] ← NIL
3 for each node w in the root list of H
4     do x ← w
5        d ← degree[x]
6        while A[d] ≠ NIL
7            do y ← A[d]      ▷ Another node with the same degree as x.
8               if key[x] > key[y]
9                  then exchange x ↔ y
10              FIB-HEAP-LINK(H, y, x)
11              A[d] ← NIL
12              d ← d + 1
13       A[d] ← x
14 min[H] ← NIL
15 for i ← 0 to D(n[H])
16     do if A[i] ≠ NIL
17           then add A[i] to the root list of H
18                if min[H] = NIL or key[A[i]] < key[min[H]]
19                   then min[H] ← A[i]
```

调整根表的过程

1）在根表中找出两个具有相同度数的根x和y，且$key[x] \le key[y]$；

2）将y链接到x；将y从根表中移出，成为x的一个孩子。这个操作有**FIB-HEAP-LINK**过程完成。域$degree[x]$增值，且如果y上有标记，则清除标记。

辅助度数数组$A[0...D(n[H])]$

抽取最小键值结点

FIB-HEAP-LINK(*H*, *y*, *x*)

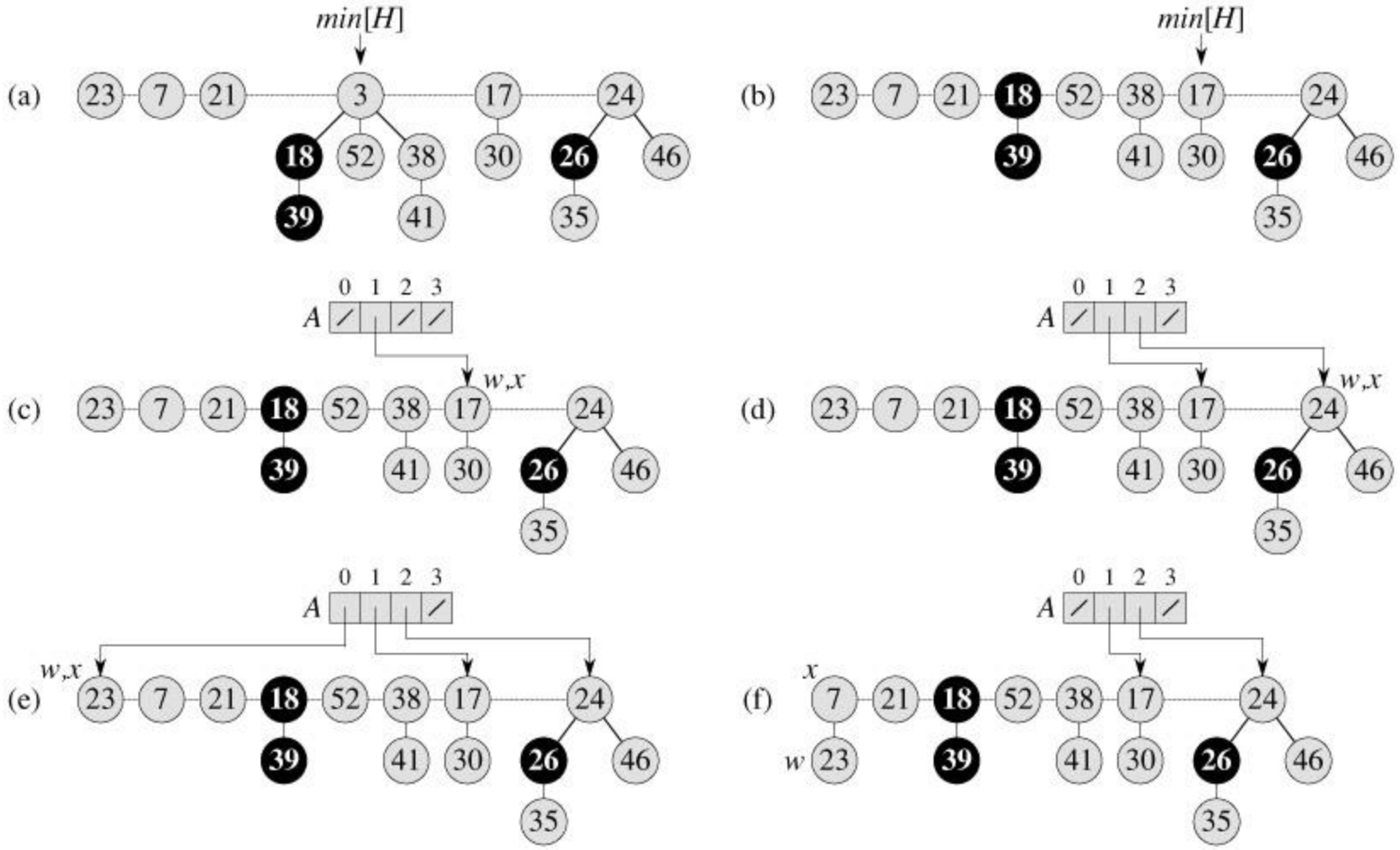
1 remove *y* from the root list of *H*

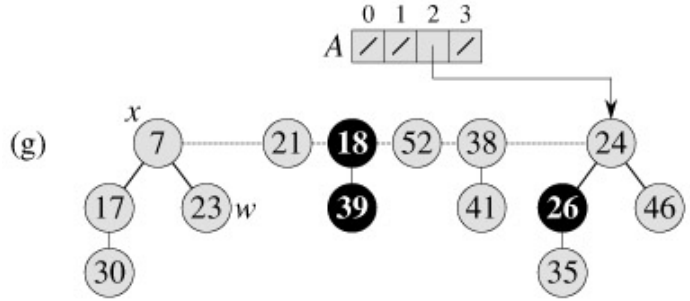
2 make *y* a child of *x*, incrementing *degree*[*x*]

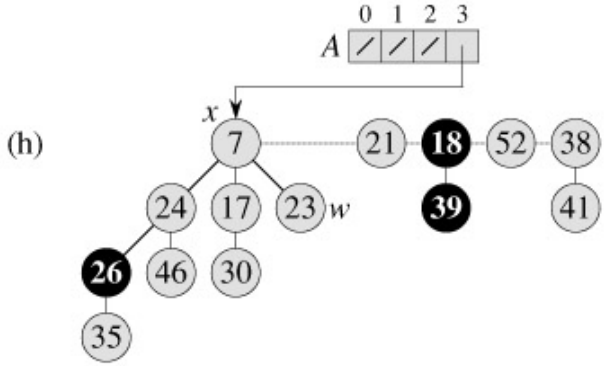
3 *mark*[*y*] ← FALSE

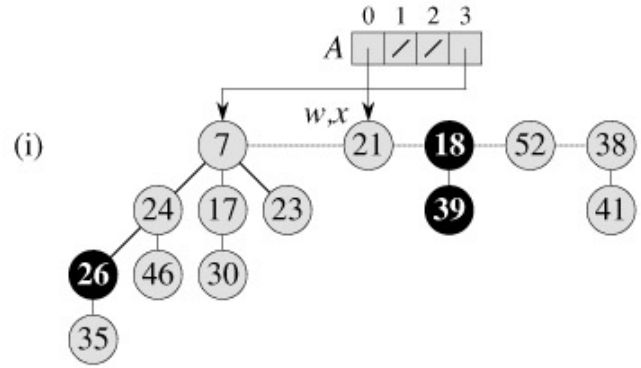
FIB-HEAP-EXTRACT-MIN(*H*)至多处理最小结点的**$D(n)$**个子女
CONSOLIDATE(*H*)遍历**A**数组的时间代价为**$O(D(n))$**
调用**CONSOLIDATE**时，根表大小至多为**$D(n)+t(H)-1$**
所以**FIB-HEAP-EXTRACT-MIN**的实际工作量为**$O(D(n)+t(H))$**

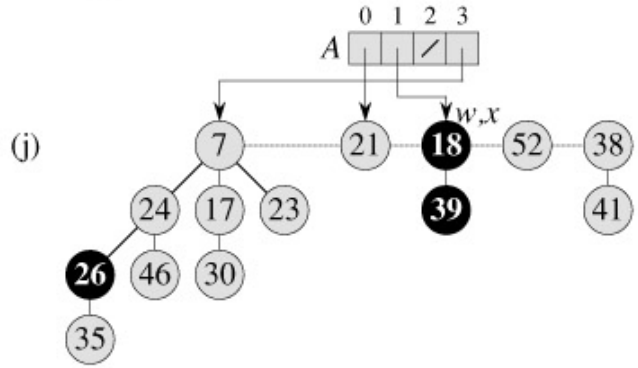
平摊开销为
$O(D(n)+t(H)) + ((D(n)+1)+2m(H)) - (t(H)+2m(H))$
$=O(D(n)) + O(t(H)) - t(H) = O(D(n))$ （$= O(\lg n)$ 后面将说明）
扩大势的单位可以支配$O(t(H))$ 中的隐藏常数

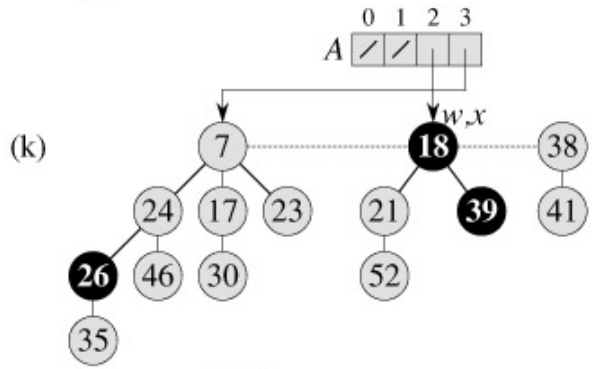

(a)
min[H]
23 7 21 3 17 24
18 52 38 30 26 46
39 41 35
(b)
min[H]
23 7 21 18 52 38 17 24
39 41 30 26 46
35
(c)
A
0 1 2 3
w,x
23 7 21 18 52 38 17 24
39 41 30 26 46
35
(d)
A
0 1 2 3
w,x
23 7 21 18 52 38 17 24
39 41 30 26 46
35
(e)
A
0 1 2 3
w,x
23 7 21 18 52 38 17 24
39 41 30 26 46
35
(f)
A
0 1 2 3
x
w
7 21 18 52 38 17 24
23 39 41 30 26 46
35

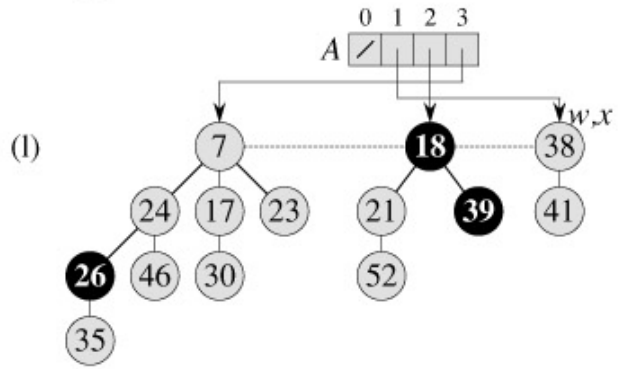

(g)
0 1 2 3
A
x
7
21
18
52
38
24
17
23
w
39
41
26
46
30
35

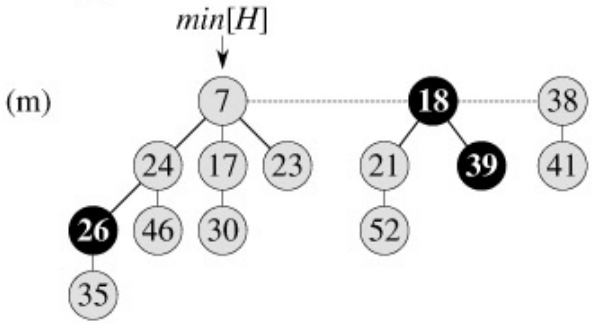

(h)
0 1 2 3
A
x
7
21
18
52
38
24
17
23
w
39
41
26
46
30
35


(i)
0 1 2 3
A
w,x
7
21
18
52
38
24
17
23
39
41
26
46
30
35


(j)
0 1 2 3
A
w,x
7
21
18
52
38
24
17
23
39
41
26
46
30
35


(k)
0 1 2 3
A
w,x
7
18
38
24
17
23
21
39
41
26
46
30
52
35


(l)
0 1 2 3
A
w,x
7
18
38
24
17
23
21
39
41
26
46
30
52
35


(m)
min[H]
7
18
38
24
17
23
21
39
41
26
46
30
52
35

减小一个结点的键值

FIB-HEAP-DECREASE-KEY(*H*, *x*, *k*)

```
1 if k > key[x]
2     then error "new key is greater than current key"
3 key[x] ← k
4 y ← p[x]
5 if y ≠ NIL and key[x] < key[y]
6     then CUT(H, x, y)
7          CASCADING-CUT(H, y)
8 if key[x] < key[min[H]]
9     then min[H] ← x
```

减小一个结点的键值

CUT(*H*, *x*, *y*)

1 remove *x* from the child list of *y*, decrementing *degree*[*y*]

2 add *x* to the root list of *H*

3 *p*[*x*] ← NIL

4 *mark*[*x*] ← FALSE

CASCADING-CUT(*H*, *y*)

1 *z* ← *p*[*y*]

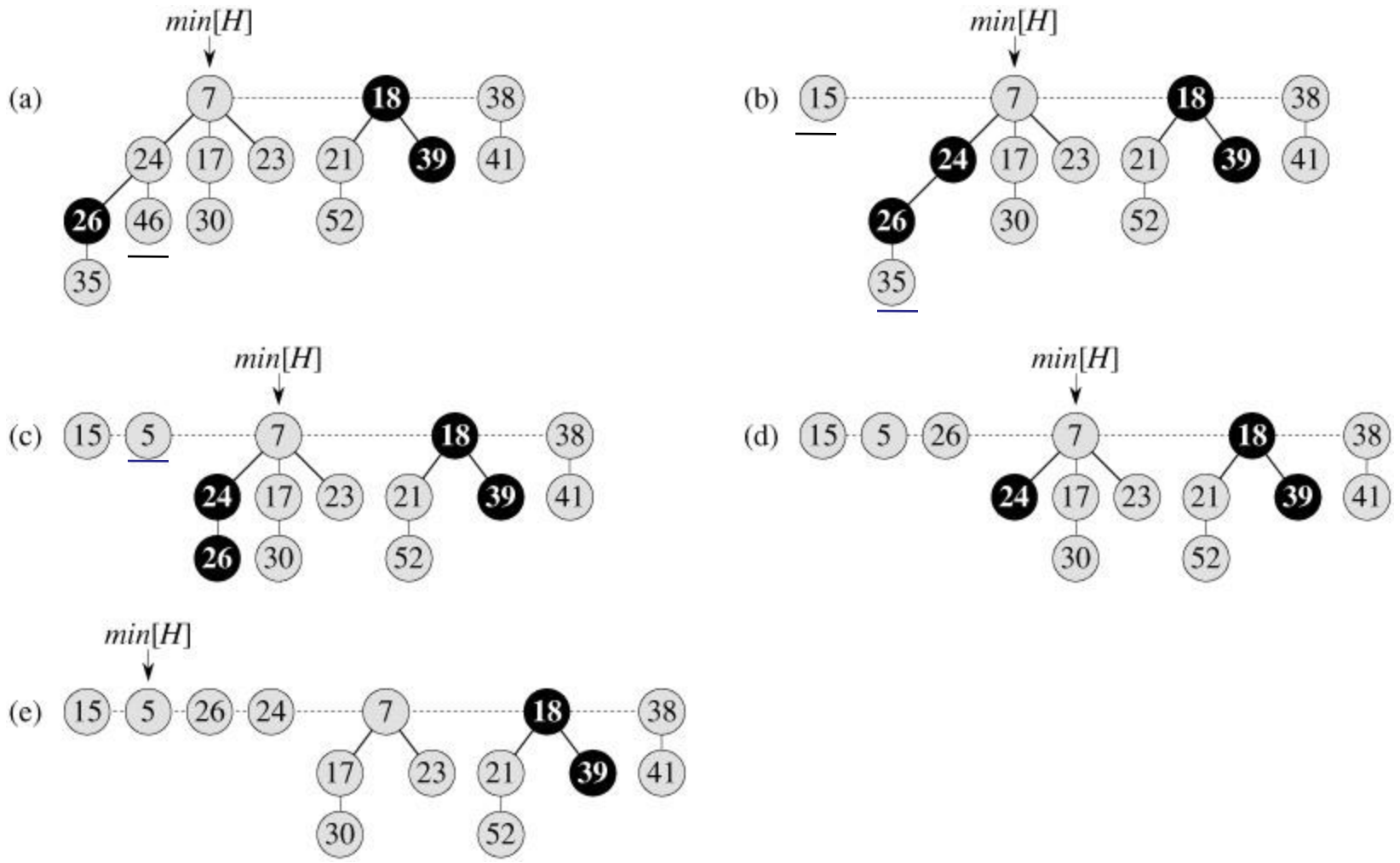
2 if *z* ≠ NIL

3 then if *mark*[*y*] = FALSE

4 then *mark*[*y*] ← TRUE

5 else CUT(*H*, *y*, *z*)

6 CASCADING-CUT(*H*, *z*)


(a)
min[H]
7
18
38
24
17
23
21
39
41
26
46
30
52
35
(b)
min[H]
15
7
18
38
24
17
23
21
39
41
26
30
52
35
(c)
min[H]
15
5
7
18
38
24
17
23
21
39
41
26
30
52
(d)
min[H]
15
5
26
7
18
38
24
17
23
21
39
41
30
52
(e)
min[H]
15
5
26
24
7
18
38
17
23
21
39
41
30
52

减小一个结点的键值的平摊开销

- 实际开销
  - 设过程中调用了c次**CASCADING-CUT**
  - 则减小键值的实际开销为$O(c)$
- 势的变化
  - 新堆中树的个数为$t(H)+(c-1)+1= t(H)+c$
  - 标记结点数为$m(H)-(c-1)+1=m(H)-c+2$
  - 势的变化$(t(H)+c)+2(m(H)-c+2)-(t(H)+2m(H))=4-c$
- 平摊代价
  - $O(c)+ 4 - c=O(1)$
  - 扩大势单位支配$O(c)$中的常数

清除标记，势减小2，一个用于支付切断与清除，一个用于补偿成为根的势的增加。

删除一个结点

FIB-HEAP-DELETE(*H*, *x*)

1 FIB-HEAP-DECREASE-KEY(*H*, *x*, -∞)

2 FIB-HEAP-EXTRACT-MIN(*H*)

与二项堆删除结点方法一样：
把待删除结点的键值减至无穷小
通过抽取最小键值结点的方法删除该结点
运行时间为**O(D(*n*)) = O(lg*n*)**

最大度数的界

- 在只支持可合并堆操作时，$D(n) \le \lfloor \lg n \rfloor$。
- 当还支持删除和减小键值操作时，我们要证明 $D(n) \le \lfloor \log_{\varphi} n \rfloor$，其中$\varphi=(1+5^{1/2})/2$
- 证明的关键在于，对斐波那契堆，定义*size*(*x*)为以*x*为根的子树中包括*x*在内的所有结点个数（*x*无需在根表中）。于是我们将证明*size*(*x*)为*degree*[*x*]的幂。这里*degree*[*x*]是*x*的度数的准确计数。

引理 20.1

- 设x为斐波那契对中的任意结点，且假设$degree[x]=k$。设$y_1, y_2,..., y_k$表示按与x链接的次序排列的x的子女，从最早的到最迟的，则对于$i=2,3,...,k$，有$degress[y_1] \geq 0$和$degress[y_i] \geq i-2$。

证明：很明显， $degress[y_1] \geq 0$。

- 对于$i \geq 2$，注意当y_i链接到x上时， $y_1, y_2,..., y_{i-1}$都是x的子女，故必有$degress[x] \geq i-1$。又仅当$degress[x] = degress[y_i]$时，才将结点$y_i$链接到$x$上，故这时又必有$degress[y_i] \geq i-1$。从此之后，结点$y_i$至多失去一个孩子，因为如果他失去两个孩子，它将被从x处切断，所以有$degress[y_i] \geq i-2$。 ■

引理20.2

- 关于第k个斐波那契数 $F_k = \begin{cases} 0 & \text{如果 } k = 0 \\ 1 & \text{如果 } k = 1 \\ F_{k-1} + F_{k-2} & \text{如果 } k \geq 2 \end{cases}$
- 对于所有整数$k \geq 0$, $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$

证明：对k进行归纳，当$k=0$是，

$$1 + \sum_{i=0}^{0} F_i = 1 + F_0 = 1 + 0 = 1 = F_2$$

假设对于 $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$ 有

$$1 + \sum_{i=0}^{k} F_i = 1 + \sum_{i=0}^{k-1} F_i + F_k = F_{k+1} + F_k = F_{k+2}$$

■

引理20.3

- 设x为斐波那契堆的任一结点，且$k=degree[x]$。那么，$size(x) \geq F_{k+2} \geq \varphi^k$

证明：设s_k为斐波那契堆中度数为k的任一结点z的最小可能的$size$。显然$s_0=1$以及$s_1=2$。s_k至多为$size(x)$，而且，s_k的值随着k单调递增。考虑某个结点z，在任意斐波那契堆中，有$degree[z]=k$以及$size(z)= s_k$。因为$s_k \leq size(x)$，通过计算s_k的下界来计算$size(x)$的下界。（可归纳法证明：$s_i \geq F_{i+2}$）

$$
\begin{aligned}
size(x) &\geq s_k \\
&= 2 + \sum_{i=2}^{k} s_{degree[y_i]} \\
&\geq 2 + \sum_{i=2}^{k} s_{i-2} ,
\end{aligned}
\qquad
\begin{aligned}
s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\
&\geq 2 + \sum_{i=2}^{k} F_i \\
&= 1 + \sum_{i=0}^{k} F_i \\
&= F_{k+2} \qquad \text{(by Lemma 20.2)} .
\end{aligned}
$$

推论20.4

- 在一个包含有n个结点的斐波那契堆中，结点的最大度数$D(n)$为$O(\lg n)$
- 由上述引理可直接证明。

优先队列小结

- 优先队列和二叉堆
- 二项堆和二项树
- 斐波那契堆