Algorithms
Chapter 19
Fibonacci Heaps

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **Structure of Fibonacci heaps**
- Mergeable-heap operations
- Decreasing a key and deleting a node
- Bounding the maximum degree

Overview$_{1/2}$ mergeable heap ➡ 支援以下5种operators

- A **mergeable heap** is any data structure that supports the following five operations, in which each element has a key:
  - MAKE-HEAP() creates and returns a new heap containing no elements. 產生一個沒有任何元素的heap
  - INSERT(H, x) inserts element x, whose key field has already been filled in, into heap H. 將元素x放入heap H中
  - MINIMUM(H) returns a pointer to the element in heap H whose key is minimum. 回傳最小的元素
  - EXTRACT-MIN(H) deletes the element from heap H whose key is minimum, returning a pointer to the element. 回傳並刪除最小元素
  - UNION(H_1, H_2) creates and returns a new heap that contains all the elements of heaps H_1 and H_2. Heaps H_1 and H_2 are "destroyed" by this operation. 合併 H_1 和 H_2

Overview 2/2

Fibonacci heaps 支援 "5" + "2" 种 operations

- **Fibonacci heaps** support the mergeable-heap operations and the following two operations. 將元素 x 的 key value 降成 k
  - DECREASE-KEY(H, x, k) assigns to element x the new key value k, which is assumed to be no greater than its current key value.
  - DELETE(H, x) deletes node x from heap H. 將元素 x 從 H 中刪除

Procedure	Binary heap (**worst case**)	Binomial heap (**worst case**)	Fibonacci heap (**amortized**)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
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	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$

刪除

Fibonacci heaps in theory and practice

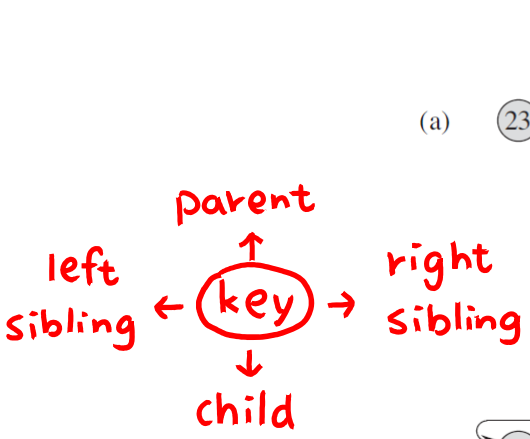
- From a theoretical standpoint, Fibonacci heaps are especially desirable when the number of EXTRACT-MIN and DELETE operations is small relative to the number of other operations performed. 理論上：若Extra-Min和DELETE的次数不多，Fibonacci不錯
- From a practical point of view, the constant factors and programming complexity of Fibonacci heaps make them less desirable than ordinary binary (or *k*-ary) heaps for most applications, except for certain applications that manage large amounts of data. 實際上：因為常数大和coding的難度
  还是 binary heaps 好，除非
  资料量很大

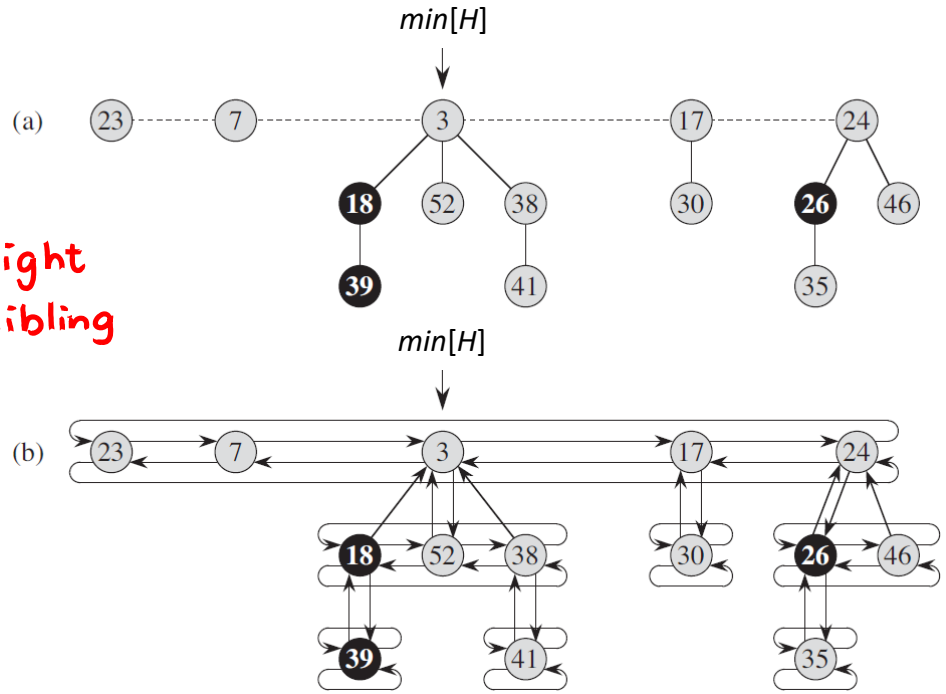
binomial heap = binomial trees + binomial heap properties

Fibonacci heap = rooted trees + min-heap property

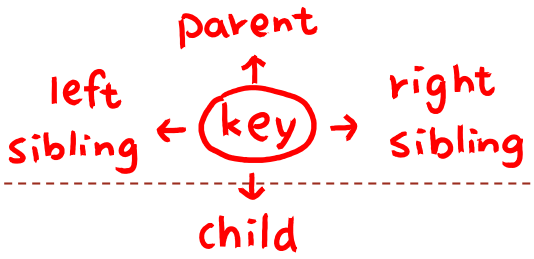
Structure of Fibonacci heaps 1/2

- A **Fibonacci heap** is a collection of rooted trees that are **min-heap ordered**, i.e., each tree obeys the min-heap property.
  - In a min-heap, the **min-heap property** is that the key of a node is greater than or equal to the key of its parent. 父≤子





Structure of Fibonacci heaps 2/2



- In a Fibonacci heap:
  - The children of *x* are linked together in a circular, doubly linked list, which we call the **child list** of *x*.
  - *p*[*x*]: parent; *child*[*x*]: any one of its children.
  - *left*[*x*]: right sibling; *right*[*x*]: right sibling.
  - *degree*[*x*]: the number of children; *n*[*H*]: the number of nodes in *H.*
  - *mark*[*x*]:indicates whether node *x* has lost a child since the last time *x* was made the child of another node. mark[x]: 自從當了別人的兒子後. 有沒有失去過兒子
  - *min*[*H*]: a pointer to the root of a tree containing a minimum key; this node is called the **minimum node** of the Fibonacci heap.
    - If a Fibonacci heap *H* is empty, then *min*[*H*] = NIL.
  - The roots of all the trees are linked together in a circular, doubly linked list, which we call the **root list**.

Potential function

- We shall use the potential method to analyze the performance of Fibonacci heap operations. 用 potential method 去分析 Fibonacci heap
- The potential of Fibonacci heap H is then defined by

$$\Phi(H) = t(H) + 2m(H).$$

建保局存款：root 的個数 + 2倍失去兒子的桌数

  - $t(H)$: the number of trees in the root list of H.
  - $m(H)$: the number of marked nodes in H.
- Example: The potential of the Fibonacci heap in the previous slide is $5 + 2 \cdot 3 = 11$.
- We shall assume that a unit of potential can pay for a constant amount of work, where the constant is sufficiently large.

假設建保局存款的單位很大

Maximum degree

假設 maximum degree D(n) 已知

- Assume that there is a known upper bound $D(n)$ on the maximum degree of any node in an n-node Fibonacci heap.
- Problem 19-2(d) shows that when only the mergeable-heap operations are supported, $D(n) \le \lfloor \lg n \rfloor$.

  如果支援 mergeable heap operators：$D(n) \le \lfloor \lg n \rfloor$
- In Sections 19.3 and 19.4, we shall show that when we support DECREASE-KEY and DELETE as well, $D(n) = O(\lg n)$.

  如果再加上支援 DECREASE-KEY 和 DELETE
  ⇒ $D(n) = O(\lg n)$

Outline

- Structure of Fibonacci heaps
- **Mergeable-heap operations**
- Decreasing a key and deleting a node
- Bounding the maximum degree

Creating a new Fibonacci heap

- We describe and analyze the mergeable-heap operations as implemented for Fibonacci heaps.
- The mergeable-heap operations on Fibonacci heaps delay work as long as possible. 盡量讓工作延遲,只有在非做不可時才工作
- The MAKE-FIB-HEAP procedure allocates and returns the Fibonacci heap object H, where $n[H] = 0$ and $min[H] = \text{NIL}$.
- The potential of the empty Fibonacci heap is $\Phi(H) = 0$.
  - Because $t(H) = 0$ and $m(H) = 0$. MAKE-FIB-HEAP(): n[H] = 0 且 min[H] = NIL
- The amortized cost of MAKE-FIB-HEAP is thus equal to its $O(1)$ actual cost. 花費 = 真正的花費 + 增加的存款
  = O(1) + 0
  = O(1)

Inserting a node 將元素放入 heap中

- The following procedure inserts node *x* into Fibonacci heap *H*.

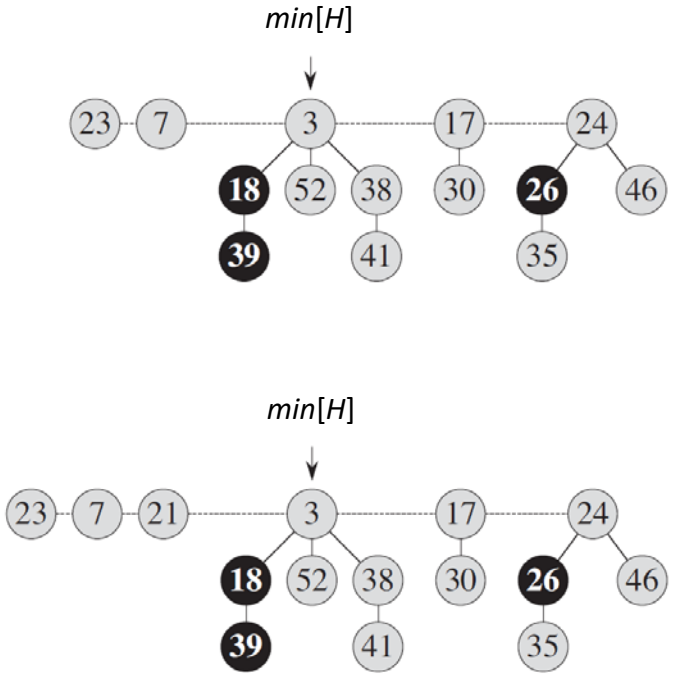
FIB-HEAP-INSERT(*H*, *x*)

1. *degree*[*x*] ← 0
2. *p*[*x*] ← NIL
3. *child*[*x*] ← NIL
4. *left*[*x*] ← *x*
5. *right*[*x*] ← *x*
6. *mark*[*x*] ← FALSE

建只包含 x 的 Heap Hx

7. concatenate the root list containing *x* with root list *H* 將H和Hx連接在一起
8. **if** *min*[*H*] = NIL or *key*[*x*] < *key*[*min*[*H*]]
9. **then** *min*[*H*] ← *x* 小的當min
10. *n*[*H*] ← *n*[*H*] + 1 個數加1

Example: FIB-HEAP-INSERT(*H*, 21)



Inserting a node

- $t(H') = t(H)+1$ and $m(H') = m(H)$. ⇒ root個枝加1，失去兒子的枝枝不變
  - H: the input Fibonacci heap.
  - H': the resulting Fibonacci heap.
- The increase in potential is

  $((t(H) + 1) + 2m(H)) - (t(H) + 2m(H)) = 1$. 增加的存款：$\Phi(H') - \Phi(H) = 1$
- Since the actual cost is $O(1)$, the amortized cost is $O(1) + 1 = O(1)$.

  Insert的花费：真正花费 + 增加的存款
  $= O(1) + 1 = O(1)$
- **Finding the minimum node**
  - The minimum node of H is given by the pointer $min[H]$. 直接return pointer
  - Because the potential of H does not change, the amortized cost of this operation is equal to its $O(1)$ actual cost.

    因為存款沒變，所以minimum的花费 $= O(1) + 0$
    $= O(1)$

Uniting two Fibonacci heaps

- The following procedure unites Fibonacci heaps H_1 and H_2, destroying H_1 and H_2 in the process. 做 H_1 和 H_2 的 Union

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_1
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

① 將 H_1 和 H_2 的 root list 連在一起

② min [H_1] 和 min [H_2] 中小的當 min

因為 root list 是双向鏈結，所以連在一起只需 O(1) 時間

- The change in potential is

$$\Phi(H) - (\Phi(H_1) + \Phi(H_2))$$
$$= (t(H) + 2m(H)) - ((t(H_1) + 2\,m(H_1)) + (t(H_2) + 2\,m(H_2))) = 0.$$

存款沒增加

- The amortized cost is therefore equal to its $O(1)$ actual cost.

Extracting the minimum node 回傳並刪除最小元素

- The process of extracting the minimum node
  - is the most complicated, and Extract min 最複雜
  - is also where the delayed work of consolidating occurs.

因為要做合併的動作

```
FIB-HEAP-EXTRACT-MIN(H)
1.   z ← min[H]                                  z: 最小的元素
2.   if z ≠ NIL
3.      then for each child x of z
4.                do add x to the root list of H      ] 將z的兒子都放到 root list
5.                   p[x] ← NIL
6.           remove z from the root list of H        將 z 從 root list 移走
7.           if z = right[z]
8.             then min[H] ← NIL                     ) 只有z一個人, min[H]=NIL
9.             else min[H] ← right[z]                ) 做合併的動作
10.                  CONSOLIDATE(H)
11.          n[H] ← n[H] – 1
12.  return z
```

root list 的合併

Consolidating the root list

目標：讓每一個 root 的 degree 都不相同

- Repeatedly executing the following steps until every root in the root list has a distinct *degree* value. ① 找兩個 degree 相同的 root
  - Find two roots *x* and *y* in the root list with the same degree, where *key*[*x*] ≤ *key*[*y*]. ② 將它們合併，小的當 root
  - **Link** *y* to *x*: Calling FIB-HEAP-LINK to make *y* a child of *x*.
- An auxiliary array *A*[0..*D*(*n*[*H*])] is used to finding two roots with the same degree. 用一個陣列 A[0…D(n[H])] 來幫助找相同 degree 的 root x 和 y
  - We will see how to calculate the upper bound *D*(*n*[*H*]) in Section 19.4.

之前有假設已知最大 degree D(n[H])

FIB-HEAP-LINK(*H*, *y*, *x*) 假設：key[x] ≤ key[y]

1. remove *y* from the root list of *H*
2. make *y* a child of *x*, incrementing *degree*[*x*]
3. *mark*[*y*] ← FALSE 將 y 設成白色

CONSOLIDATE procedure

如果有相同的 degree，就 merge 小的當父親
⇒ 每個 root 最多做一次

```
CONSOLIDATE(H)
1.  for i ← 0 to D(n[H])                    } O(D(n))
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]
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]
```

Another node with the same degree as *x*.

$O(D(n)+t(H))$

Time complexity = $O(D(n)+t(H))$

Since there are at most $D(n) + t(H) - 1$ roots

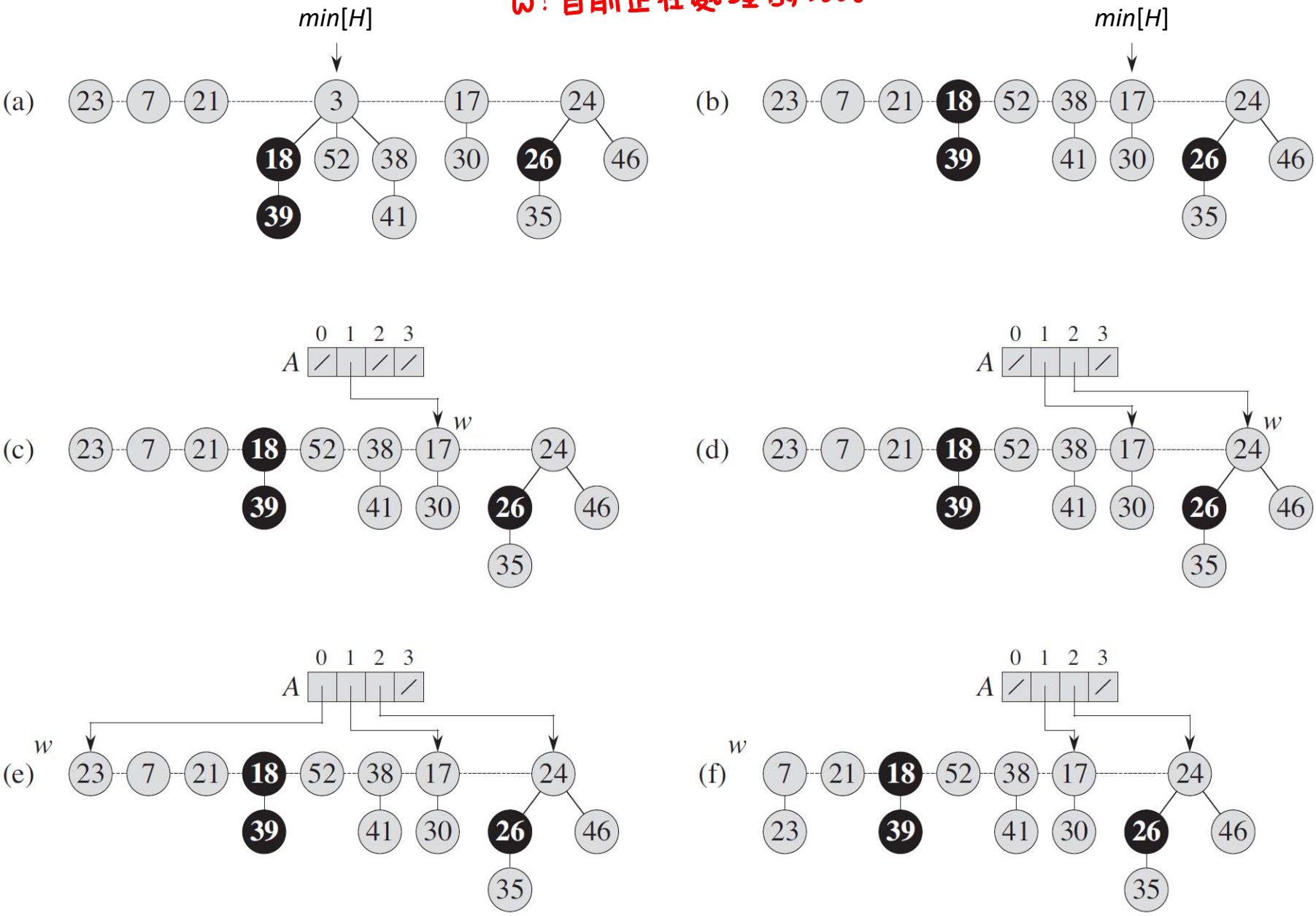
$D(n)$：z 的兒子 ≤ $D(n)$

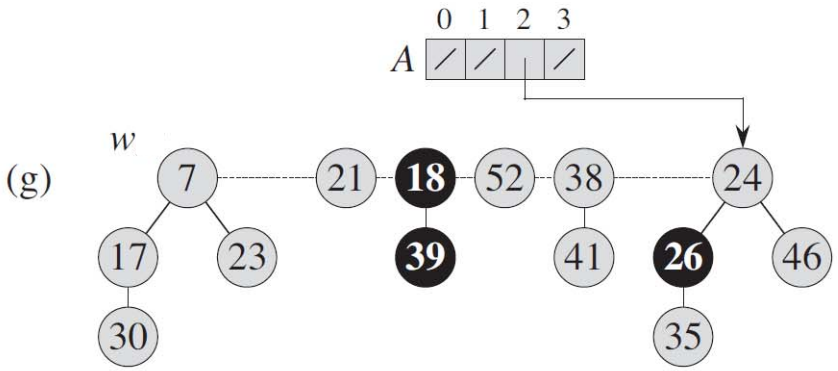
$t(H)$：root list

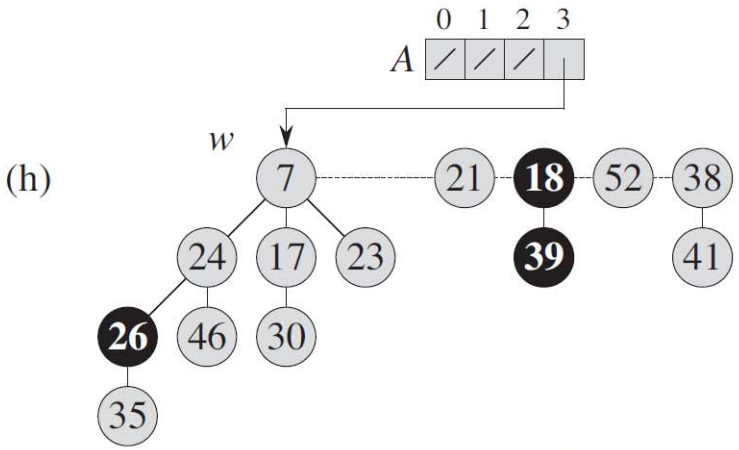
$O(D(n))$

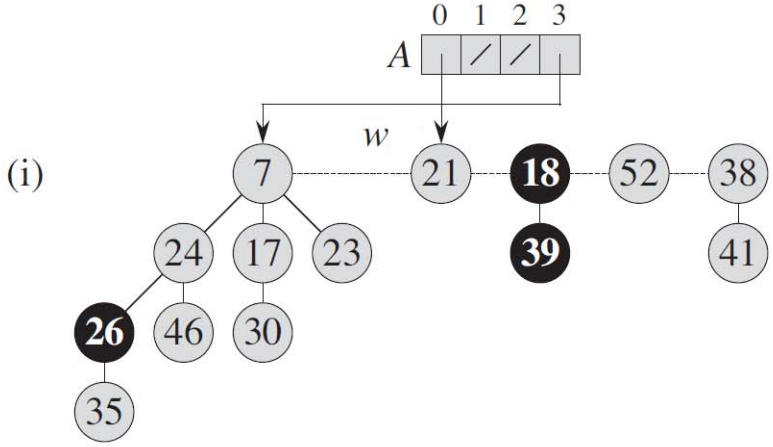
找最小的 root list

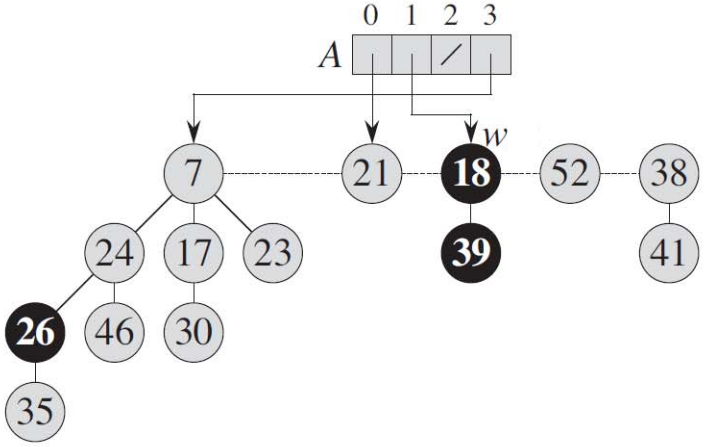
w：目前正在處理的 root

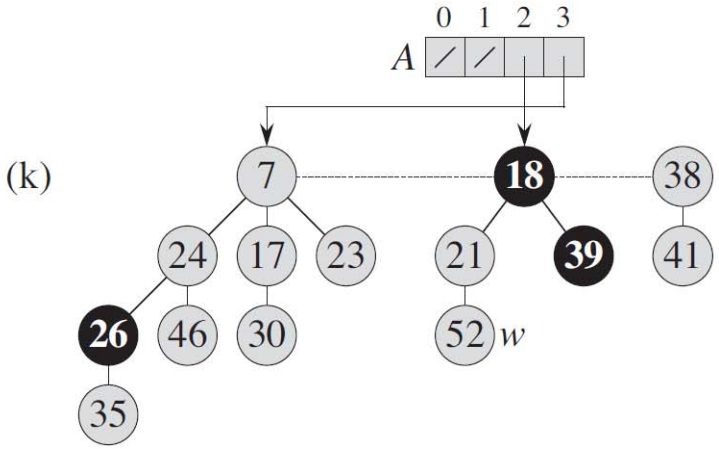


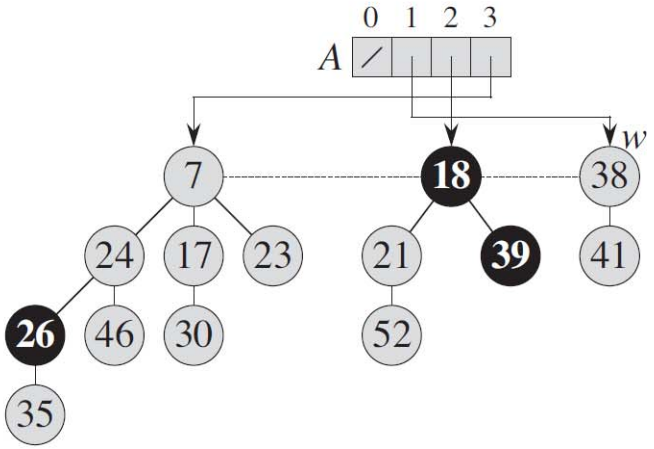

(g)
0 1 2 3
A
w
7
21
18
52
38
24
17
23
39
41
26
46
30
35


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


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

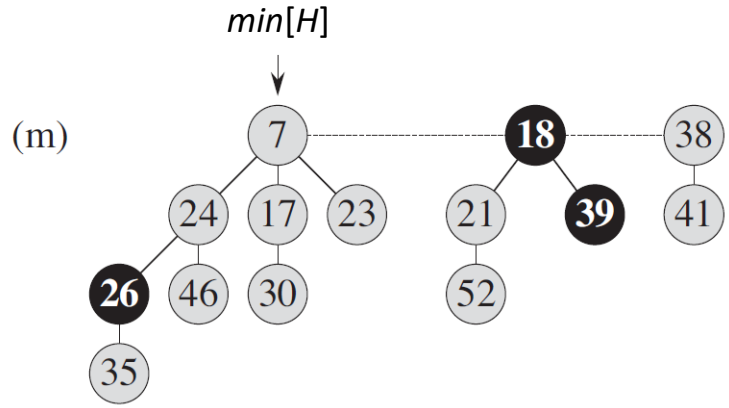

(j)
0 1 2 3
A
w
7
21
18
52
38
24
17
23
39
41
26
46
30
35


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


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

使用 root list 的存款

Amortized cost of extracting the minimum



① root 個数最多為 D(n)+1
② 沒人失去兒子且 root list 的合併可能 unmark node

原來的存款　　現在的存款

- $\Phi(H) = t(H) + 2m(H)$, $\Phi(H') \le (D(n) + 1) + 2m(H)$.
- The amortized cost is thus at most

$$O(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H))$$

真正花費 $= O(D(n)) + O(t(H)) - t(H)$　　增加的存款

$= O(D(n))$.　Extra-Min 的花費：真正的花費 + 增加的存款

- since we can scale up the units of potential to dominate the constant hidden in $O(t(H))$. 可以將單位存款變大 ⇒ $O(t(H)) - t(H) \le 0$

Outline

- Structure of Fibonacci heaps
- Mergeable-heap operations
- **Decreasing a key and deleting a node**
- Bounding the maximum degree

使用黑色的存款

Decreasing a key and deleting a node

- We show
  - decreasing a key in $O(1)$ amortized time, and
  - deleting a node in $O(D(n))$ amortized time.

$D(n) = O(\lg n)$ will show in Section 19.4

FIB-HEAP-DECREASE-KEY(H, x, k) 將 key 值變小

```
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
```

$O(1)$ (lines 1–6)

key[x] < key[y], 將 x 移到 root list

"cut" the link between x and its parent y, making x a root.

$O(c)$ (line 7)

Suppose that CASCADING-CUT is recursively called c times.

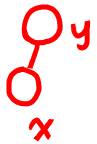
$O(1)$ (lines 8–9)

- Actual cost of FIB-HEAP-DECREASE-KEY = $O(1) + O(c)$. 真正的花費

Decreasing a key 2/2

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. $p[x] \leftarrow$ NIL
4. $mark[x] \leftarrow$ FALSE



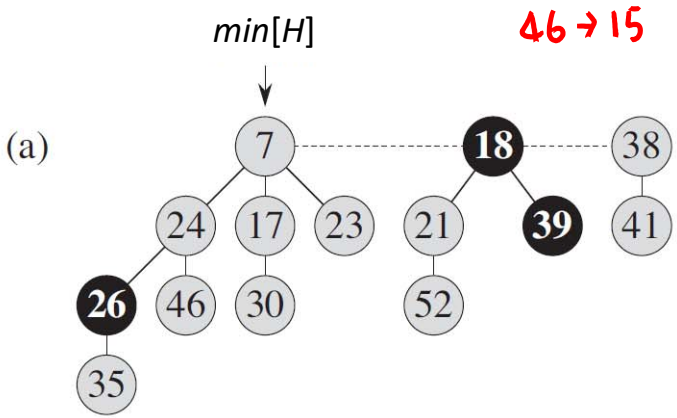
將x移到root list,並將x設成白色
(沒有失去過孩子)

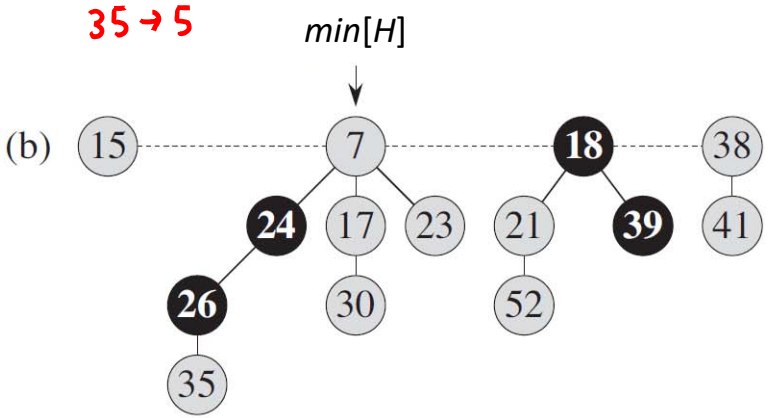
CASCADING-CUT(H, y)

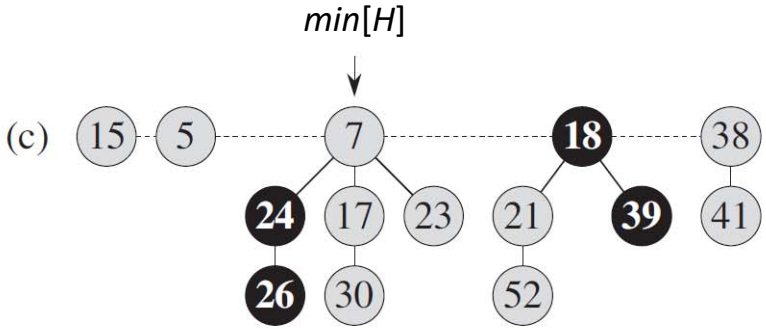
1. $z \leftarrow p[y]$
2. **if** $z \neq$ NIL
3. **then if** $mark[y]$ = FALSE
4. **then** $mark[y] \leftarrow$ TRUE
5. **else** CUT(H, y, z)
6. CASCADING-CUT(H, z)

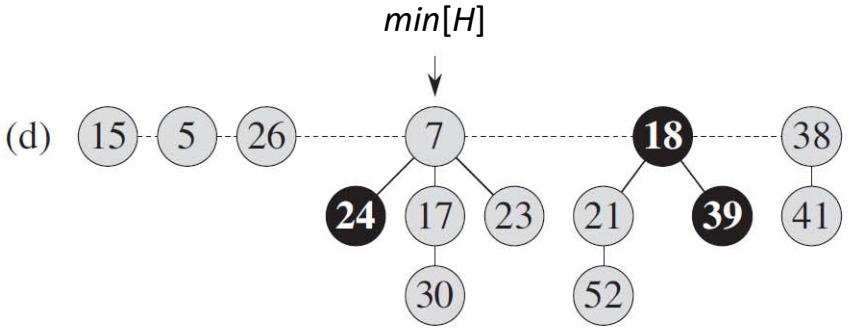
As soon as the second child has been lost, we cut y from its parent, making it a new root.

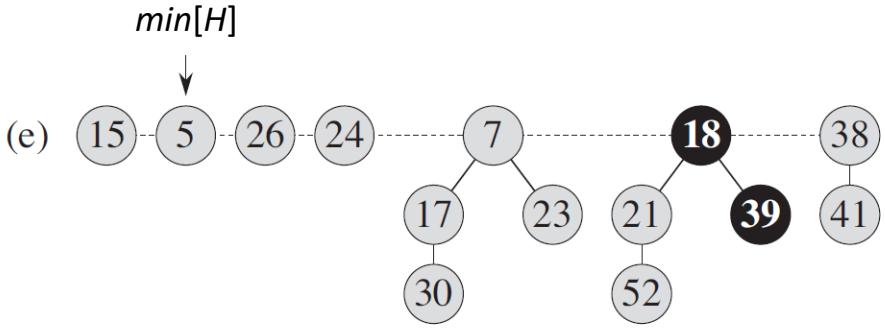
一直往上檢查.每個人最多可失去一個孩子,如果失去第2個,移到root list











(a): The initial Fibonacci heap.

(b): The node with key 46 has its key decreased to 15.

(c)–(e): The node with key 35 has its key decreased to 5.

Amortized cost of decreasing a key

- Each recursive call of CASCADING-CUT, except for the last one, cuts a marked node and clears the mark bit.
- $t(H') = t(H) + 1 + c - 1$.
  - $c - 1$ trees produced by cascading cuts, and 1 for the tree rooted at x.
- At most $m(H) - (c - 1) + 1$ marked nodes.
  - $c - 1$ nodes unmarked by cascading cuts, and the last call may mark a node.
- $\Phi(H') - \Phi(H) \le ((t(H) + c) + 2(m(H) - c + 2)) - (t(H) + 2m(H))$
  $= 4 - c$.
- The amortized cost is thus at most $O(c) + 4 - c = O(1)$, since we can scale up the units of potential to dominate the constant hidden in $O(c)$.

Deleting a node 刪除点

- We assume that there is no key value of $-\infty$ currently in the Fibonacci heap.

FIB-HEAP-DELETE(H, x)
1. FIB-HEAP-DECREASE-KEY(H, x, $-\infty$)
2. FIB-HEAP-EXTRACT-MIN(H)

- The amortized cost is $O(1) + O(D(n))$.

Outline

- Structure of Fibonacci heaps
- Mergeable-heap operations
- Decreasing a key and deleting a node
- **Bounding the maximum degree**

Bounding the maximum degree 證明最大 degree 為 $\lfloor \log_\phi n \rfloor$

- We shall show that $D(n) \le \lfloor \log_\phi n \rfloor$, where ϕ is the golden ratio, defined as $\phi = \frac{1+\sqrt{5}}{2} = 1.61803$.
- For $k = 0, 1, 2, \ldots$, the kth Fibonacci number is defined by the recurrence 費氏數列的第 k 個值

$$F_k = \begin{cases} 0 & \text{if } k = 0, \\ 1 & \text{if } k = 1, \\ F_{k-1} + F_{k-2} & \text{if } k \ge 2. \end{cases}$$

that explains the name "Fibonacci heaps"

- **Lemma 19.4** Let x be any node in a Fibonacci heap, and let $k = degree[x]$. Then, size(x) $\ge F_{k+2} \ge \phi^k$. ⇒ $k \le \lfloor \log_\phi n \rfloor$
  ↳ 以 x 為 root 的 subtree 點數
- **Corollary 19.5** The maximum degree $D(n)$ of any node in an n-node Fibonacci heap is $O(\lg n)$.