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$_{1/2}$

- 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.
  - INSERT(H, x) inserts element x, whose key field has already been filled in, into heap H.
  - MINIMUM(H) returns a pointer to the element in heap H whose key is minimum.
  - EXTRACT-MIN(H) deletes the element from heap H whose key is minimum, returning a pointer to the element.
  - 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.

Overview$_{2/2}$

- **Fibonacci heaps** support the mergeable-heap operations and the following two operations.
  - 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.

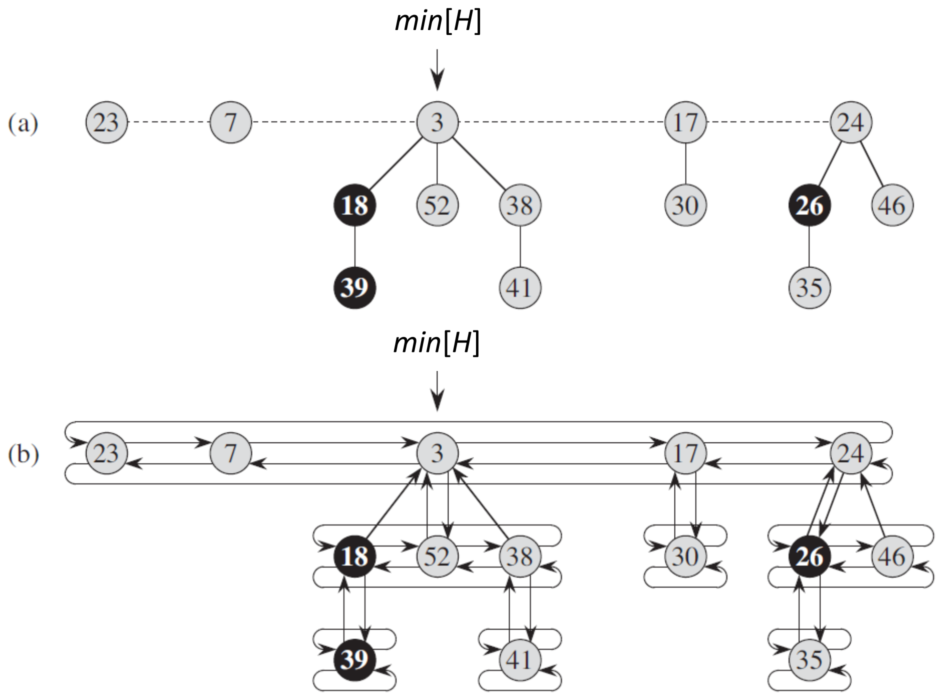
Procedure	Binary heap (**worst case**)	Binomial heap (**worst case**)	Fibonacci heap (**amortized**)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(\lg n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$
UNION	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$

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.
- 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.

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.



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.
  - *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.
- The potential of Fibonacci heap H is then defined by

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

  - $t(H)$: 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.

Maximum degree

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

Outline

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

Creating a new Fibonacci heap

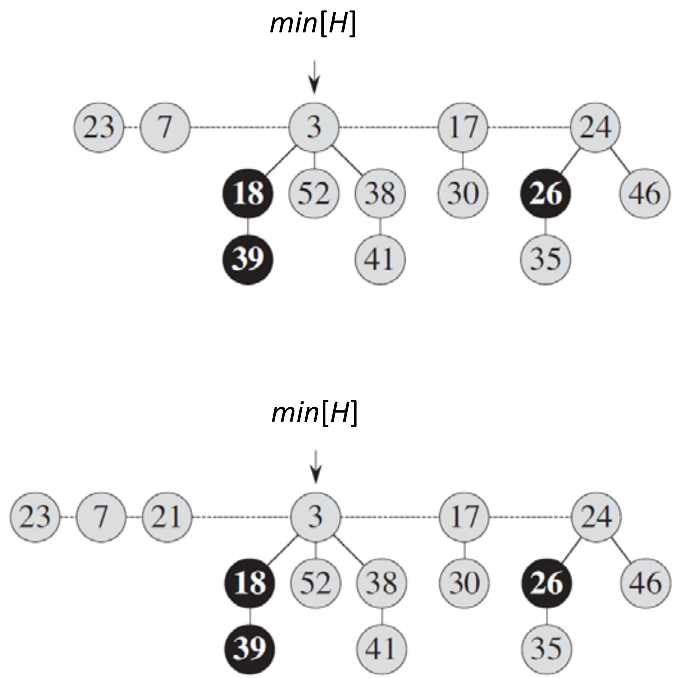
- We describe and analyze the mergeable-heap operations as implemented for Fibonacci heaps.
- The mergeable-heap operations on Fibonacci heaps delay work as long as possible.
- The MAKE-FIB-HEAP procedure allocates and returns the Fibonacci heap object H, where $n[H] = 0$ and $min[H] = \text{NIL}$.
- The potential of the empty Fibonacci heap is $\Phi(H) = 0$.
  - Because $t(H) = 0$ and $m(H) = 0$.
- The amortized cost of MAKE-FIB-HEAP is thus equal to its $O(1)$ actual cost.

Inserting a node

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

```
FIB-HEAP-INSERT(H, x)
    degree[x] ← 0
    p[x] ← NIL
    child[x] ← NIL
    left[x] ← x
    right[x] ← x
    mark[x] ← FALSE
    concatenate the root list containing x
    with root list H
    if min[H] = NIL or key[x] < key[min[H]]
        then min[H] ← x
    n[H] ← n[H] + 1
```

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



Inserting a node

- $t(H') = t(H)+1$ and $m(H') = m(H)$.
  - H: the input Fibonacci heap.
  - H': the resulting Fibonacci heap.
- The increase in potential is

$$((t(H) + 1) + 2m(H)) - (t(H) + 2m(H)) = 1.$$

- Since the actual cost is $O(1)$, the amortized cost is $O(1) + 1 = O(1)$.

- **Finding the minimum node**
  - The minimum node of H is given by the pointer $min[H]$.
  - Because the potential of H does not change, the amortized cost of this operation is equal to its $O(1)$ actual cost.

Uniting two Fibonacci heaps

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

FIB-HEAP-UNION(H_1, H_2)

$H \leftarrow$ MAKE-FIB-HEAP()
$min[H] \leftarrow min[H_1]$
concatenate the root list of H_2 with the root list of H_1
if ($min[H_1]$ = NIL) or ($min[H_2] \neq$ NIL and $min[H_2] < min[H_1]$)
  then $min[H] \leftarrow min[H_2]$
$n[H] \leftarrow n[H_1] + n[H_2]$
free the objects H_1 and H_2
return H

- The change in potential is

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

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

Extracting the minimum node

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

```
FIB-HEAP-EXTRACT-MIN(H)
    z ← min[H]
    if z ≠ NIL
       then for each child x of z
                do add x to the root list of H
                   p[x] ← NIL
            remove z from the root list of H
            if z = right[z]
               then min[H] ← NIL
               else  min[H] ← right[z]
                     CONSOLIDATE(H)
            n[H] ← n[H] – 1
    return z
```

Consolidating the root list

- Repeatedly executing the following steps until every root in the root list has a distinct *degree* value.
  - Find two roots *x* and *y* in the root list with the same degree, where *key*[*x*] ≤ *key*[*y*].
  - **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.
  - We will see how to calculate the upper bound $D(n[H])$ in Section 19.4.

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

CONSOLIDATE procedure

```
CONSOLIDATE(H)
    for i ← 0 to D(n[H])
        do A[i] ← NIL
    for each node w in the root list of H
        do x ← w
           d ← degree[x]
           while A[d] ≠ NIL
               do y ← A[d]
                  if key[x] > key[y]
                     then exchange x ↔ y
                  FIB-HEAP-LINK(H, y, x)
                  A[d] ← NIL
                  d ← d + 1
           A[d] ← x
    min[H] ← NIL
    for i ← 0 to D(n[H])
        do if A[i] ≠ NIL
              then add A[i] to the root list of H
                   if min[H] = NIL or key[A[i]] < key[min[H]]
                      then min[H] ← A[i]
```

$O(D(n))$

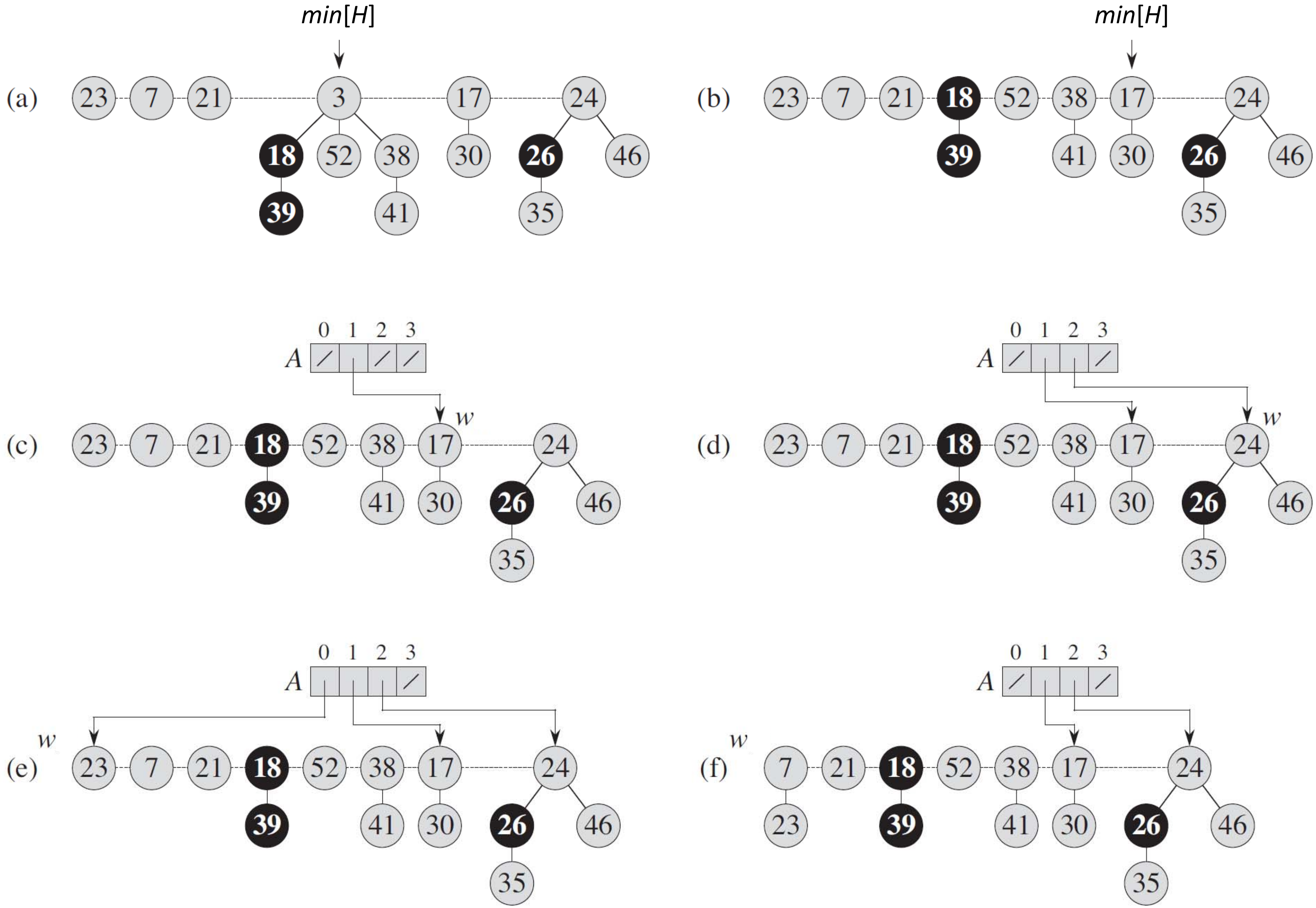
Another node with the same degree as x.

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

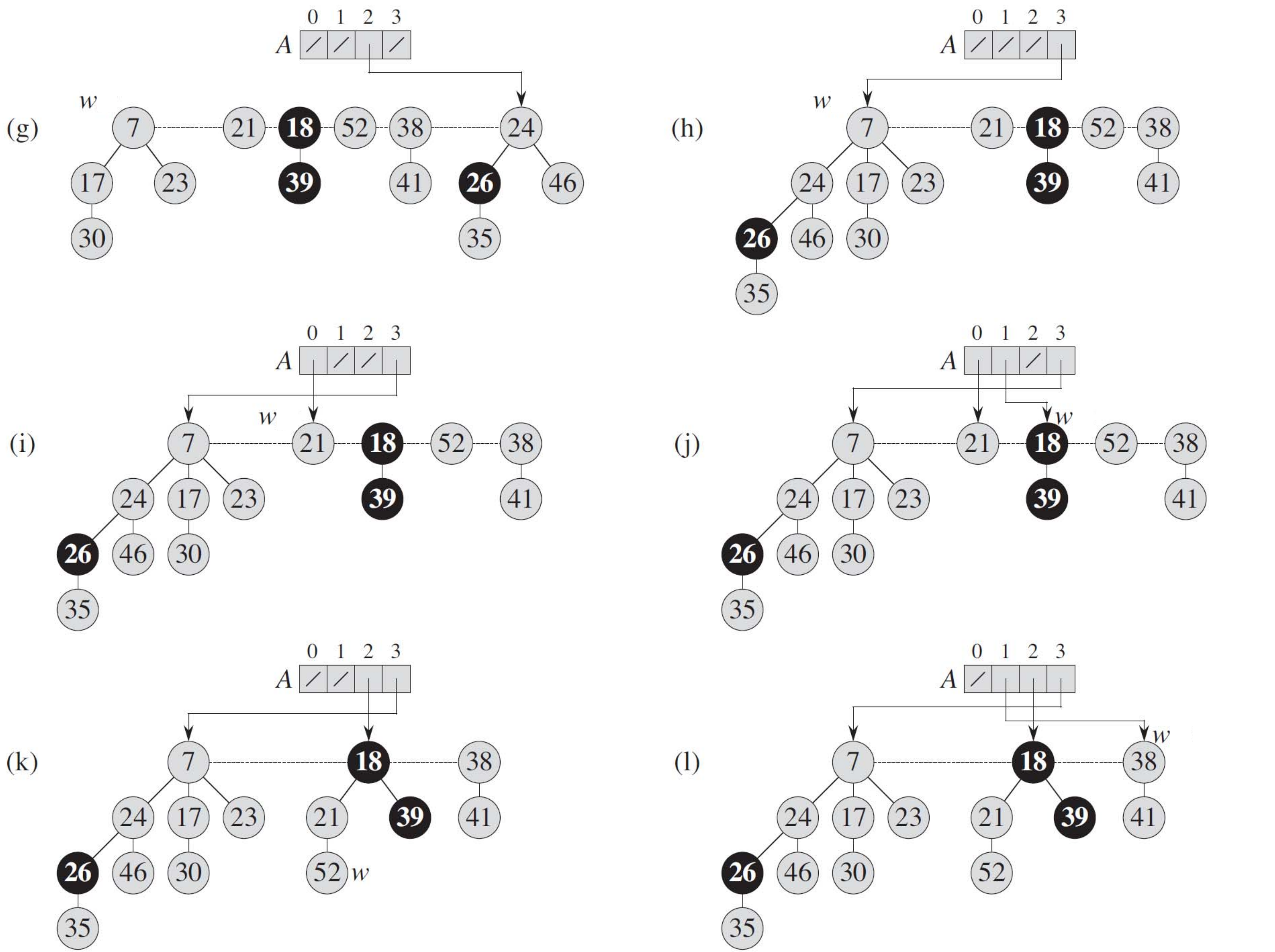
Since there are at most $D(n) + t(H) - 1$ roots

$O(D(n))$

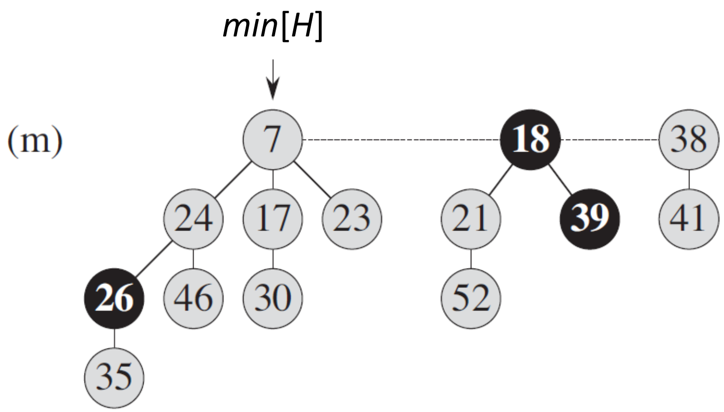
Time complexity = $O(D(n)+t(H))$



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


(g)
(h)
(i)
(j)
(k)
(l)

Amortized cost of extracting the minimum



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

$$
\begin{aligned}
&O(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H)) \\
&\qquad = O(D(n)) + O(t(H)) - t(H) \\
&\qquad = O(D(n)).
\end{aligned}
$$

- since we can scale up the units of potential to dominate the constant hidden in $O(t(H))$.

Outline

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

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)
    if k > key[x]
        then error "new key is greater than current key"
    key[x] ← k
    y ← p[x]
    if y ≠ NIL and key[x] < key[y]
        then CUT(H, x, y)
                CASCADING-CUT(H, y)    } O(c)
    if key[x] < key[min[H]]            } O(1)
    then min[H] ← x
```

$O(1)$

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

Suppose that CASCADING-CUT is recursively called c times.

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

Decreasing a key$_{2/2}$

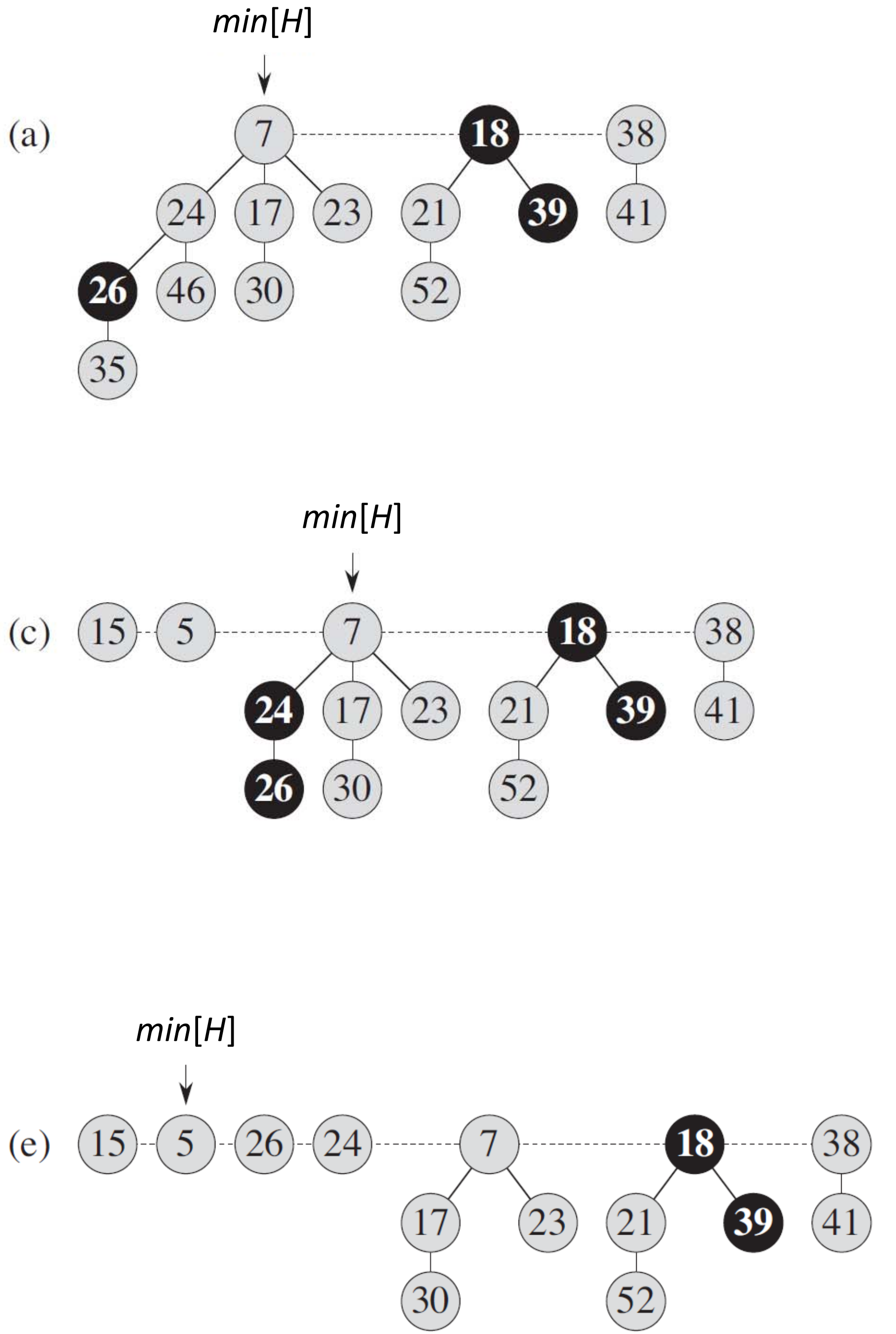
CUT(H, x, y)

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

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.



(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

- 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

- 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

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

that explains the name "Fibonacci heaps"

- **Lemma 19.4** Let x be any node in a Fibonacci heap, and let $k = degree[x]$. Then, $\text{size}(x) \ge F_{k+2} \ge \phi^k$.
- **Corollary 19.5** The maximum degree $D(n)$ of any node in an n-node Fibonacci heap is $O(\lg n)$.