Lakehead
UNIVERSITY

COMP 4433: Algorithm Design and Analysis

Dr. Y. Gu

March 22, 2023 (Lecture 17)

Fibonacci Heaps (Cont.) (Chapter 19)

– Another example of Amort. Analysis

Potential Function

For a given Fibonacci heap H, we indicate by $t(H)$ the number of trees in the root list of H and by $m(H)$ the number of marked nodes in H. We then define the potential $\Phi(H)$ of Fibonacci heap H by

$$\Phi(H) = t(H) + 2m(H) \qquad \text{(Book Ch19.1)}$$

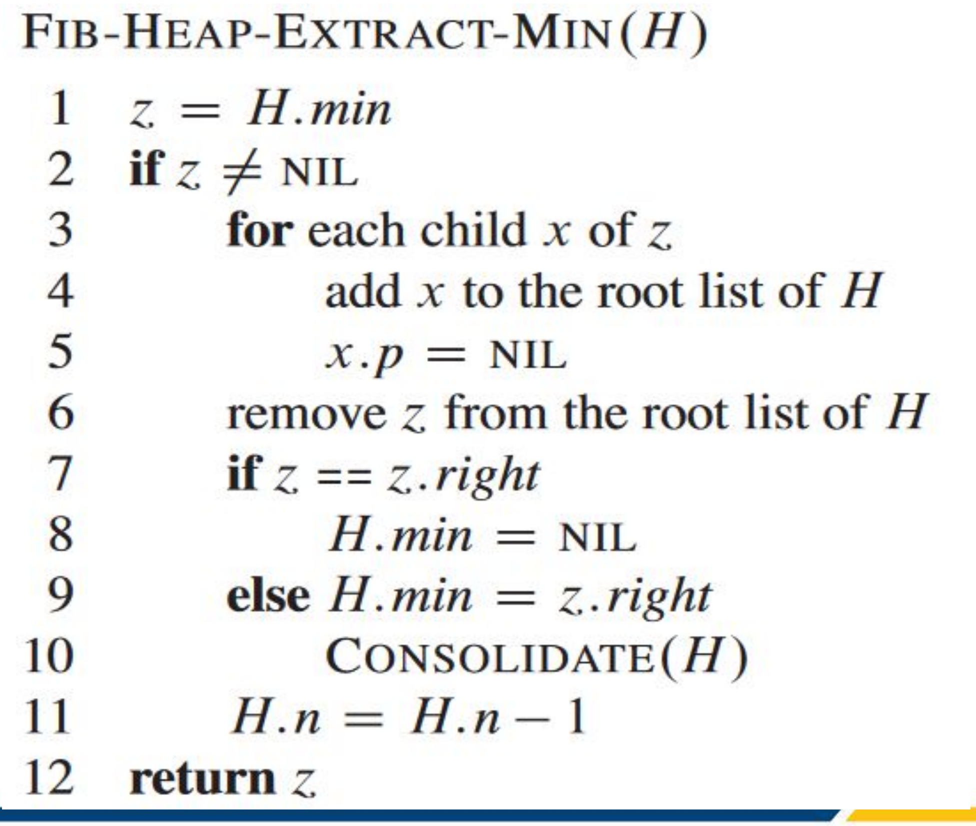
The intuition will be discussed later.

The potential of a set of Fibonacci heaps is the sum of the potentials of its constituent Fibonacci heaps. We shall assume that a unit of potential can pay for a constant amount of work, where the constant is sufficiently large to cover the cost of any of the specific constant-time pieces of work that we might encounter.

Mergeable-Heap Operations

Extracting the minimum node

It is where the delayed work of consolidating trees in the root list finally occurs.



```
FIB-HEAP-EXTRACT-MIN($H$)
$z = H.min$
if $z \neq$ NIL
    for each child $x$ of $z$
        add $x$ to the root list of $H$
        $x.p =$ NIL
    remove $z$ from the root list of $H$
    if $z == z.right$
        $H.min =$ NIL
    else $H.min = z.right$
        CONSOLIDATE($H$)
    $H.n = H.n - 1$
return $z$
```

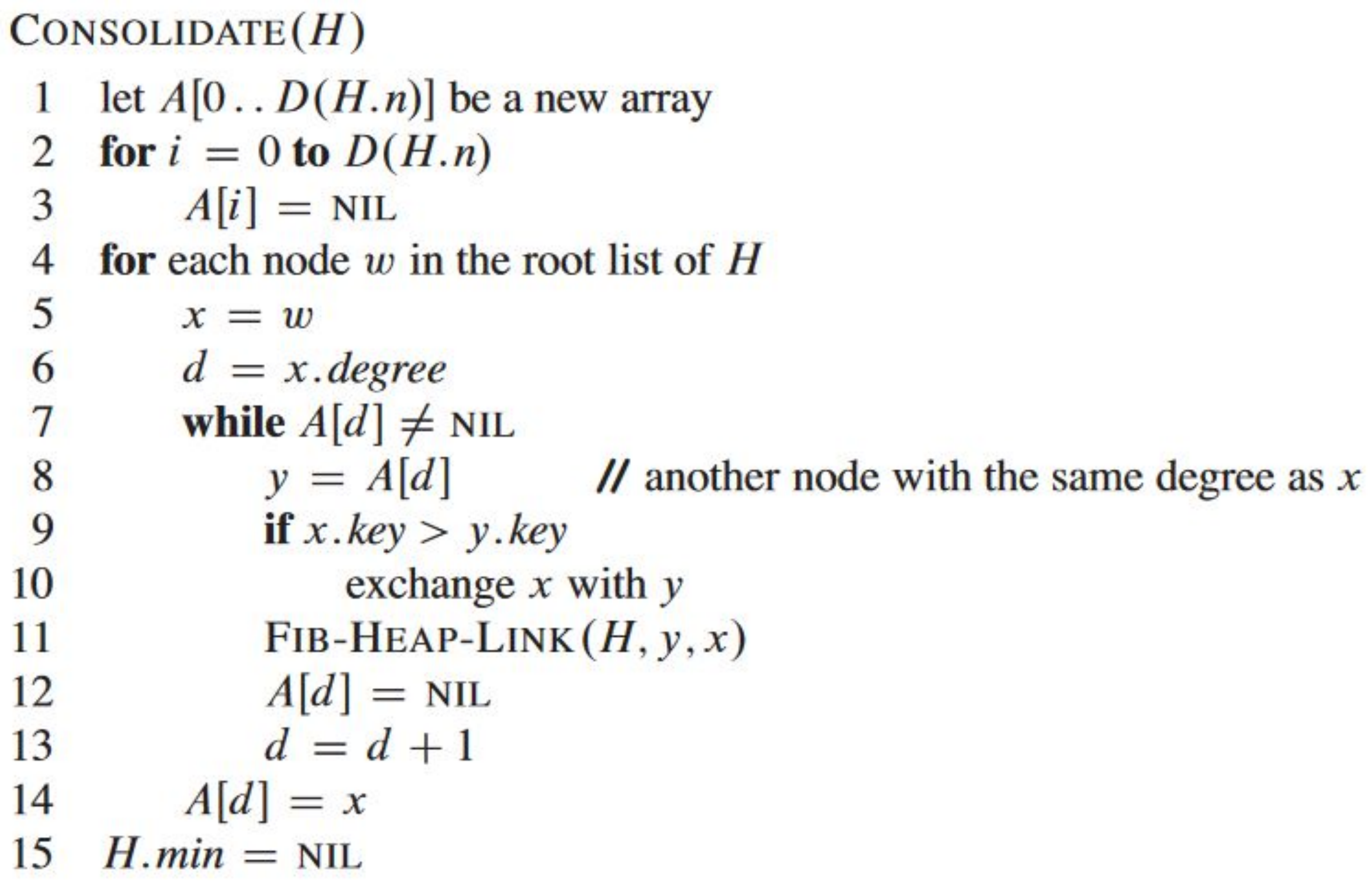
Mergeable-Heap Operations

The main steps of the procedure are as follows.

- Move all the children of z to the root list (lines 3 - 5).
- Remove z from the root list.
- If z is the only node in H, then set H as empty heap (lines 7-8).
- Otherwise, let H.min = z.right. But then H.min is not necessary the minimum node of H. So the procedure calls Consolidate to fix that problem.

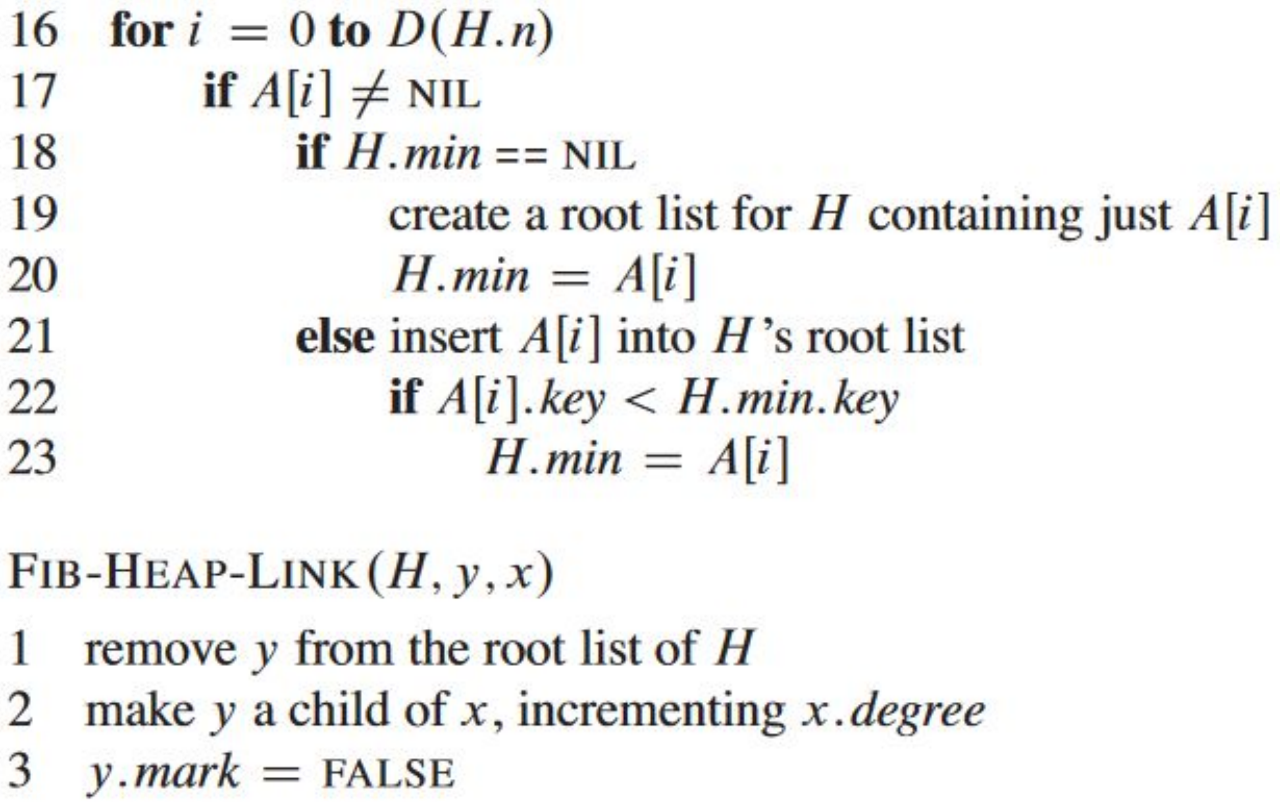
The consolidate procedure not only fix the minimum problem but also reduced the number of trees in the Fibonacci heap.

Mergeable-Heap Operations



```
CONSOLIDATE($H$)
let $A[0..D(H.n)]$ be a new array
for $i = 0$ to $D(H.n)$
    $A[i] = $ NIL
for each node $w$ in the root list of $H$
    $x = w$
    $d = x.degree$
    while $A[d] \neq$ NIL
        $y = A[d]$            // another node with the same degree as $x$
        if $x.key > y.key$
            exchange $x$ with $y$
        FIB-HEAP-LINK($H, y, x$)
        $A[d] = $ NIL
        $d = d + 1$
    $A[d] = x$
$H.min = $ NIL
```

Mergeable-Heap Operations

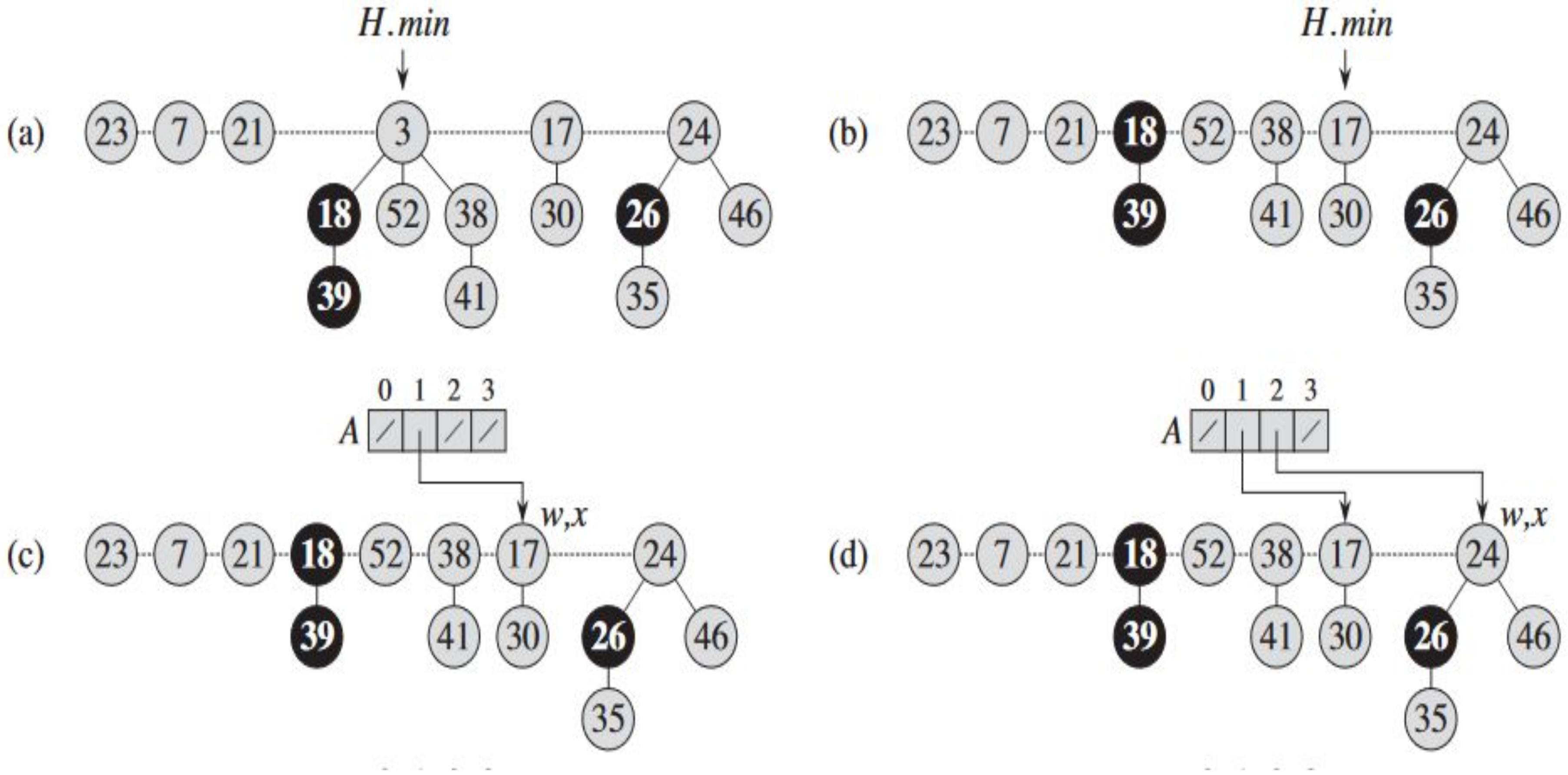


```
16  for $i = 0$ to $D(H.n)$
17      if $A[i] \neq$ NIL
18          if $H.min ==$ NIL
19              create a root list for $H$ containing just $A[i]$
20              $H.min = A[i]$
21          else insert $A[i]$ into $H$'s root list
22              if $A[i].key < H.min.key$
23                  $H.min = A[i]$
```

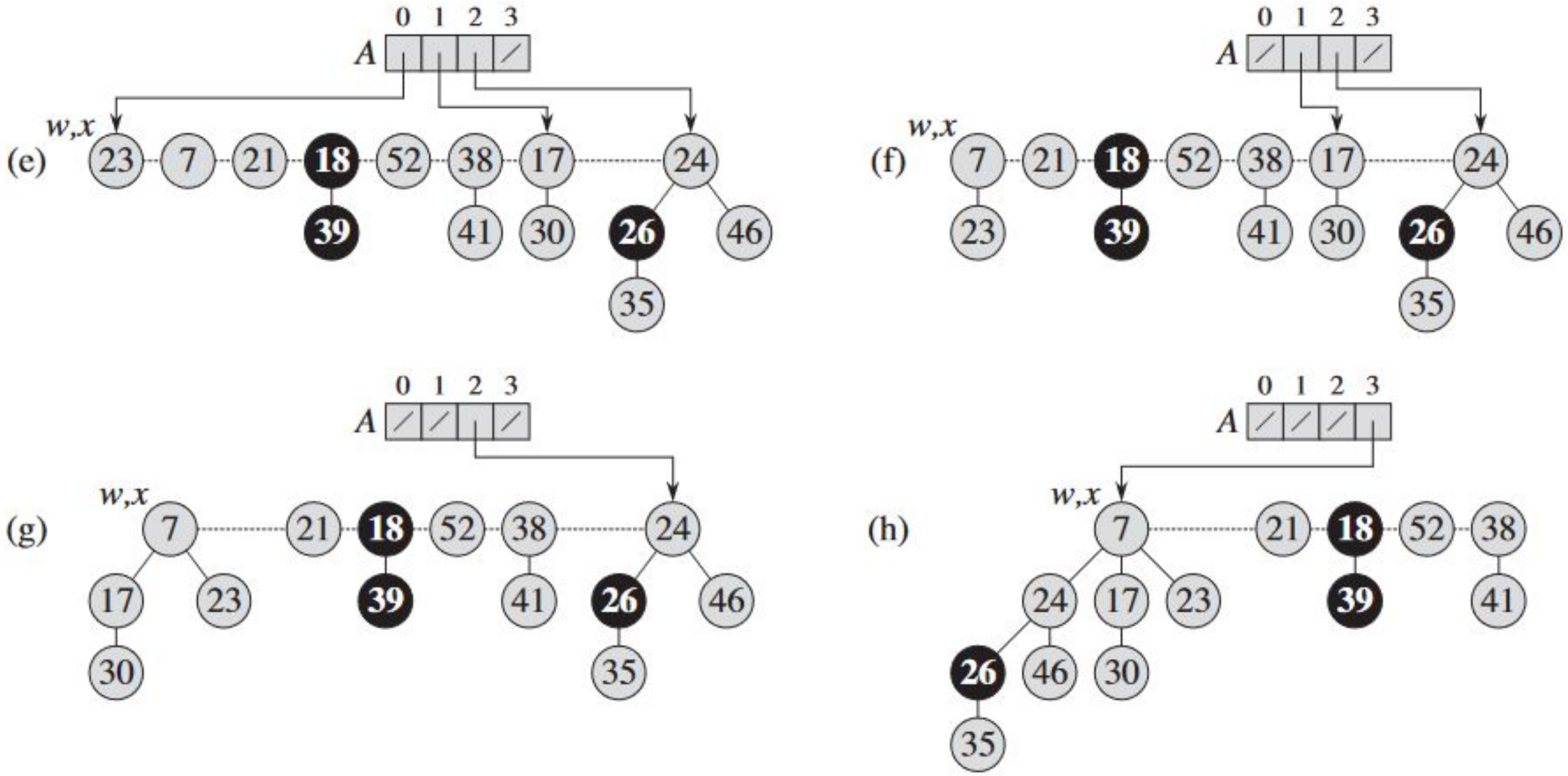
FIB-HEAP-LINK(H, y, x)

```
1  remove $y$ from the root list of $H$
2  make $y$ a child of $x$, incrementing $x.degree$
3  $y.mark =$ FALSE
```

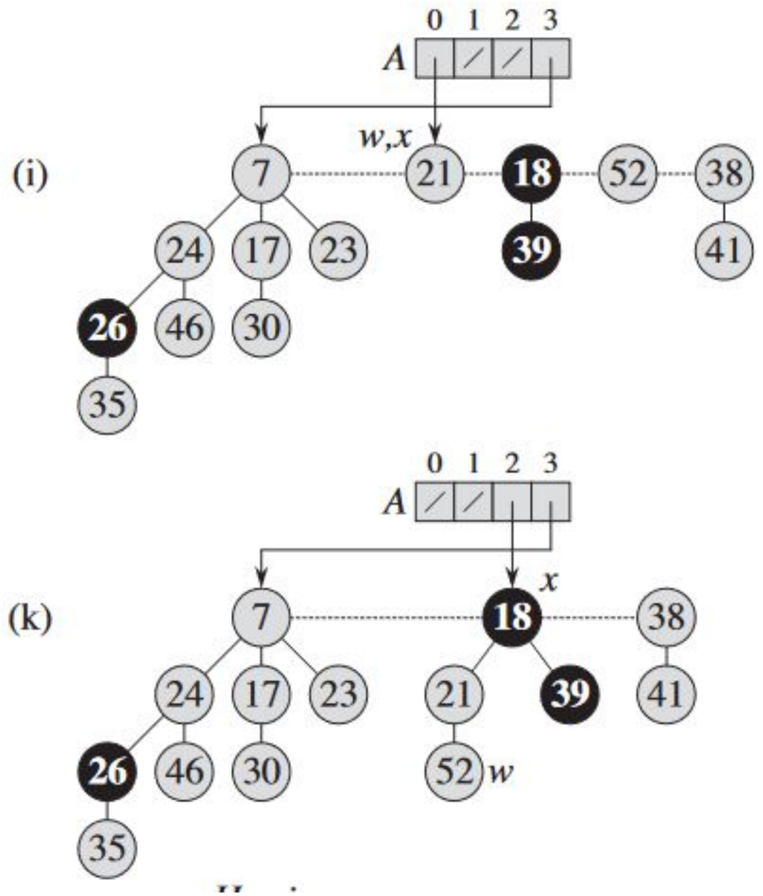
Delete Min Example (1)

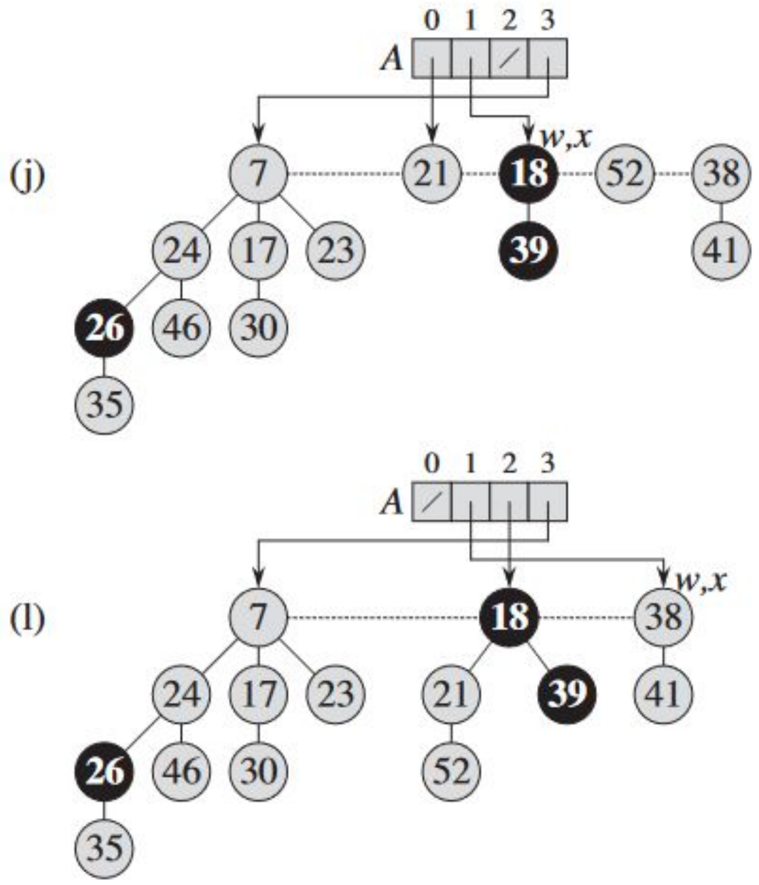


Delete Min Example (2)

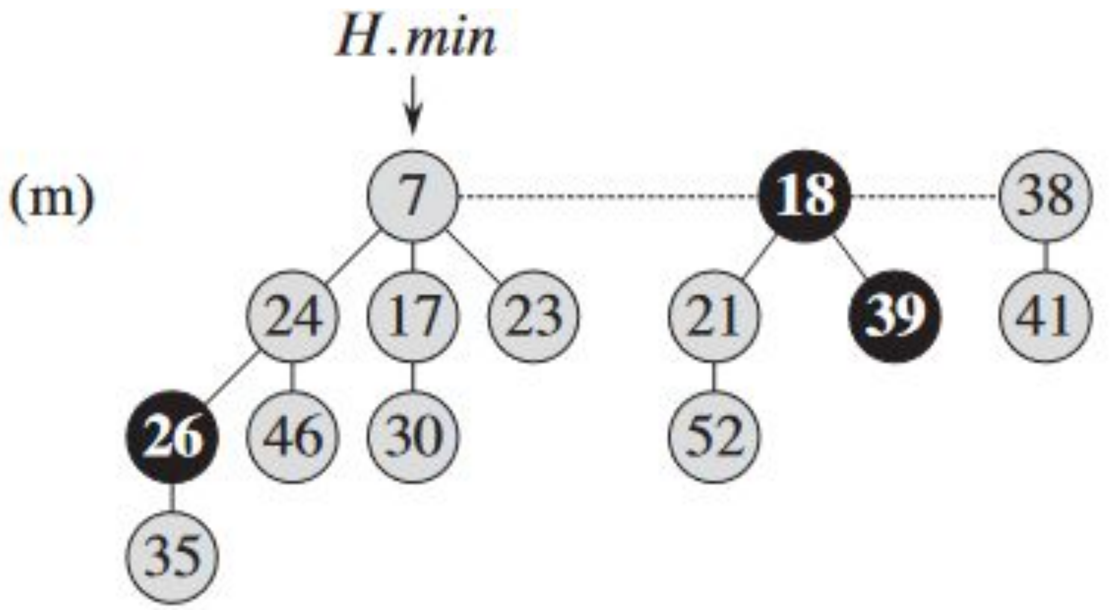


Delete Min Example (3)





Delete Min Example (4)



Mergeable-Heap Operations

The procedure Consolidate uses an auxiliary array A[0 . . . D(H.n)] to keep track of roots according to their degree. The main steps of the procedure are as follows.

- From line 4, let A[i] point to the node in the root list, that has degree i. If there are two nodes in the root list having same degree, then we put one node as a child of another node, that is done by the while loop in line 7. The if block in line 9 is used to keep the node with smaller key to remain in the root list.
- The Fib-Heap-Link is used to link one node as a child of another node.
- Line 15 empties the root list, and the remains are construct the renewed root list.

Mergeable-Heap Operations

Now we consider the amortized cost of extracting the minimum node of an n-node Fibonacci heap. Let H denote the Fibonacci heap just prior to the Fib-Heap-Extract-Min operation. We start by accounting for the actual cost of extracting the minimum node. An O(D(n)) contribution comes from Fib-Heap-Extract-Min processing at most D(n) children of the minimum node and from the work in lines 2-4 and 15-23 of Consolidate. It remains to analyze the contribution from the for loop of lines 4 -14 in Consolidate, for which we use an aggregate analysis. The size of the root list upon calling Consolidate is at most D(n) + t(H) − 1, since it consists of the original t(H) root-list nodes, minus the extracted root node, plus the children of the extracted node, which number is at most D(n).

Mergeable-Heap Operations

Within a given iteration of the for loop of lines 4 - 14, the number of iterations of the while loop of lines 7 - 13 depends on the root list.

But we know that every time through the while loop, one of the roots is linked to another, and thus the total number of iterations of the while loop over all iterations of the for loop is at most the number of roots in the root list.

Hence, the total amount of work performed in the for loop is at most proportional to D(n) + t(H). Thus, the total actual work in extracting the minimum node is O(D(n) + t(H)).

Mergeable-Heap Operations

The potential before extracting the minimum node is t(H)+2m(H), and the potential afterward is at most (D(n)+1)+2m(H), since at most D(n)+1 roots remain and no nodes become marked during the operation. The amortized cost is thus at most

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

since we can scale up the units of potential to dominate the constant hidden in O(t(H)). We shall see later that D(n) = $O(lg\ n)$, so that the amortized cost of extracting the minimum node is $O(lg\ n)$.

Fibonacci Heap Operations

Decreasing a key

We assume as before that removing a node from a linked list does not change any of the structural attributes in the removed node.

```
FIB-HEAP-DECREASE-KEY$(H, x, k)$
if $k > x.key$
    error "new key is greater than current key"
$x.key = k$
$y = x.p$
if $y \neq$ NIL and $x.key < y.key$
    CUT$(H, x, y)$
    CASCADING-CUT$(H, y)$
if $x.key < H.min.key$
    $H.min = x$
```

Fibonacci Heap Operations

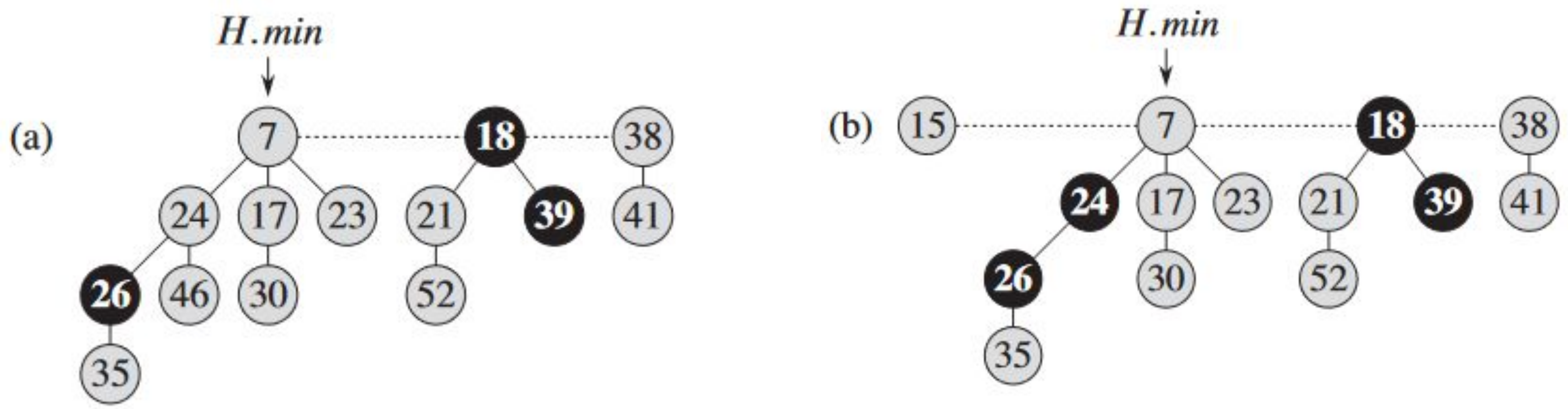
```
CUT($H, x, y$)
remove $x$ from the child list of $y$, decrementing $y.degree$
add $x$ to the root list of $H$
$x.p = \text{NIL}$
$x.mark = \text{FALSE}$

CASCADING-CUT($H, y$)
$z = y.p$
if $z \neq \text{NIL}$
    if $y.mark == \text{FALSE}$
        $y.mark = \text{TRUE}$
    else CUT($H, y, z$)
        CASCADING-CUT($H, z$)
```

Fibonacci Heap Operations

The algorithm is working as follows.

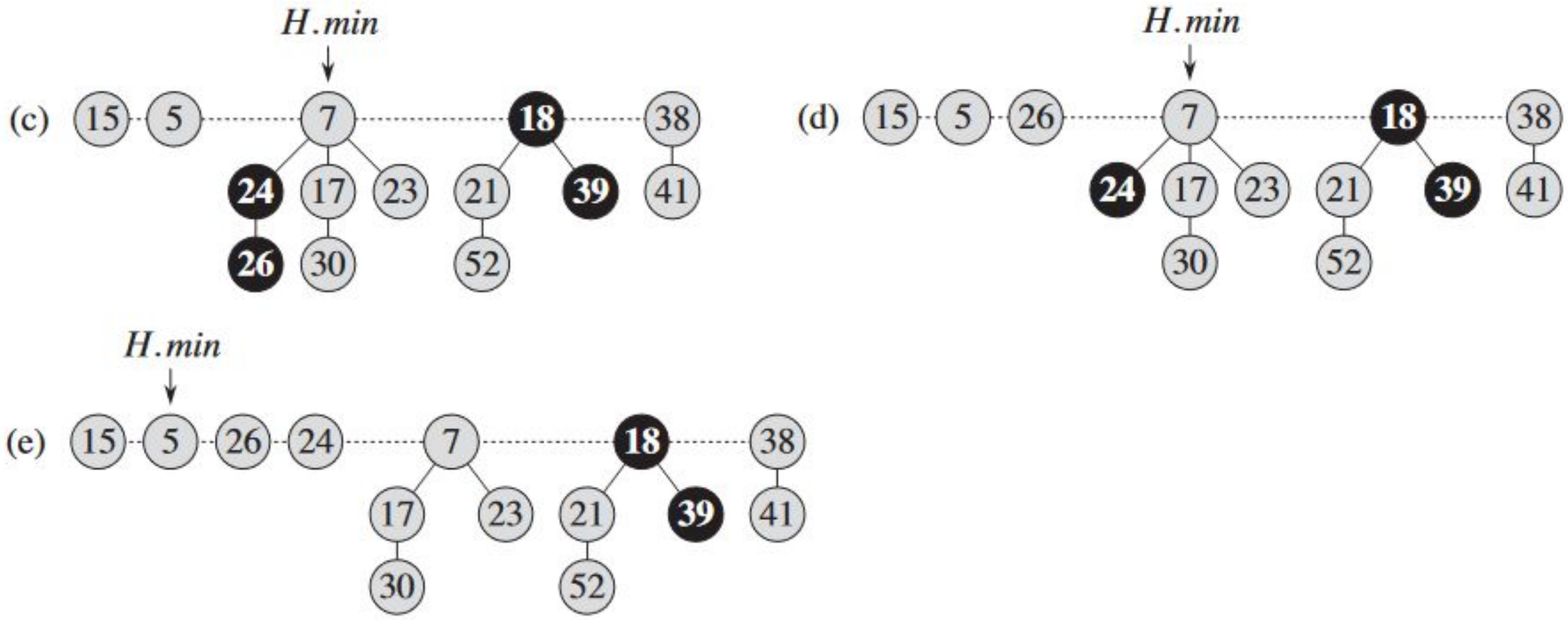
- First lines 1- 2 check if k is less than the key of x. Then change the key of x to k in lines 4 - 5.
- If x is in the root list, or x is not a root, but x.key is larger then its parent y's key, then the change is fine.
- If x.key < y.key, then the min-heap order is violated. In this case, we cut x from y and put x into the root list.
- Then we use the mark attribute to obtain the desired time bound. If a node x was linked to (made the child of) another node and then two children of x were removed by cuts, then we move x to the root list.

Decreasing Key Example



a) Original heap b) Decrease key 46 to 15 (24 is marked)

Decreasing Key Example



c) - e) Key 35 decresed to 5. In part (c), the node with key 5 becomes a root. Its parent, with key 26, is marked, so a cascading cut occurs. The node with key 26 is cut from its parent and made an unmarked root in (d). Another cascading cut occurs in (e), since the node with key 24 is marked as well.

Fibonacci Heap Operations

We now consider the amortized cost of the operation Fib-Heap-Decrease-Key.

We start by determining its actual cost. The Fib-Heap-Decrease-Key procedure takes O(1) time, plus the time to perform the cascading cuts. Suppose that a given invocation of Fib-Heap-Decrease-Key results in c calls of Cascading-Cut (recursively called). Each call of Cascading-Cut takes O(1) exclusive of recursive calls. Thus, the actual cost of Fib-Heap-Decrease-Key including all recursive calls, is O(c).

Fibonacci Heap Operations

We next compute the change in potential. Let H denote the Fibonacci heap just prior to the Fib-Heap-Decrease-Key operation. The call to Cut in Fib-Heap-Decrease-Key creates a new tree rooted at node x and clears x's mark bit (which may have already been FALSE). Each call of Cascading-Cut, except for the last one, cuts a marked node and clears the mark bit. Afterward, the Fibonacci heap contains t(H) + c trees and at most m(H) − c + 2 marked nodes (c − 1 were unmarked by cascading cuts and the last call of Cascading-Cut may have marked a node). The change in potential is therefore at most

$$((t(H) + c) + 2(m(H) - c + 2)) - (t(H) + 2m(H)) = 4 - c .$$

Fibonacci Heap Operations

So the amortized cost of Fib-Heap-Decrease-Key is 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).

When a marked node y is cut by a cascading cut, its mark bit is cleared, which reduces the potential by 2. One unit of potential pays for the cut and the clearing of the mark bit, and the other unit compensates for the unit increase in potential due to node y becoming a root.

That is why we defined the potential function to include a term that is twice the number of marked nodes.

Fibonacci Heap Operations

Deleting a node

To delete a node from a Fibonacci heap, we use the following.

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

The amortized time of Fib-Heap-Delete is the sum of the O(1) amortized time of Fib-Heap-Decrease-Key and the O(D(n)) amortized time of Fib-Heap-Extract-Min. Since we shall see later that D(n) = O(lg n), the amortized time of Fib-Heap-Delete is O(lg n).

Bounding the Maximum Degree

We need to prove that D(n) = O(lg n). In fact, we will show that D(n) ≤ $\lfloor \log_{\phi} n \rfloor$, where ϕ is the golden ratio

$$\phi = (1 + \sqrt{5})/2 = 1.61803 \ldots$$

For each node x within a Fibonacci heap, we define size(x) to be the number of nodes, including x itself, in the subtree rooted at x.

Bounding the Maximum Degree

Lemma 1 Let x be any node in a Fibonacci heap, and suppose that x.degree = k. Let $y_1, y_2, \ldots, y_k$ denote the children of x in the order in which they were linked to x, from the earliest to the latest. Then y_1.degree ≥ 0 and y_i .degree ≥ i − 2 for i = 2, 3, . . . , k.

Proof. For i ≥ 2, we note that when y_i was linked to x, all of y_1, $y_2, \ldots, y_{i-1}$ were children of x, and so we must have had x.degree ≥ i − 1. Because node y_i is linked to x (by Consolidate) only if x.degree = y_i .degree, we must have also had y_i .degree ≥ i − 1 at that time. Since then, node y_i has lost at most one child, since it would have been cut from x (by Cascading-Cut) if it had lost two children. We conclude that y_i .degree ≥ i − 2.

Bounding the Maximum Degree

Lemma 2 For all integers $k \geq 0$, the $(k + 2)$nd Fibonacci number satisfies $F_{k+2} \geq \phi^k$.

Proof. The proof is by induction on k. For $k = 0$ and $k = 1$, we have $F_2 = 1 = \phi^0$ and $F_3 = 2 > \phi^1$.

The inductive steps is for $k \geq 2$, and we assume that $F_{i+2} \geq \phi^i$ for $i = 0, 1, . . . , k -1$. We have

$$F_{k+2} = F_{k+1} + F_k \geq \phi^{k-1} + \phi^{k-2}$$

$$= \phi^{k-2} (\phi + 1) = \phi^{k-2} \cdot \phi^2$$

$$= \phi^k$$

Bounding the Maximum Degree

Lemma 3 Let x be any node in a Fibonacci heap, and let k = x.degree. Then size(x) ≥ $F_{k+2} \geq \phi^k$, where $\phi = (1 + \sqrt{5})/2$.

Proof. Let s_k denote the minimum possible size of any node of degree k in any Fibonacci heap. Trivially, $s_0 = 1$ and $s_1 = 2$. The number s_k is at most size(x) (s_k is minimum possible) and, because adding children to a node cannot decrease the node's size, the value of s_k increases monotonically with k. Consider some node z, in any Fibonacci heap, such that z.degree = k and size(z) = s_k. Because $s_k \leq$ size(x), we compute a lower bound on size(x) by computing a lower bound on s_k.

Bounding the Maximum Degree

Let $y_1, y_2, \ldots, y_k$ denote the children of z in the order in which they were linked to z. To bound s_k, we count one for z itself and one for the first child y_1 (for which size(y_1) ≥ 1), giving

$$\begin{aligned} s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\ &\geq 2 + \sum_{i=2}^{k} F_i \\ &= 1 + \sum_{i=0}^{k} F_i \\ &= F_{k+2} \end{aligned}$$

From Lemma 2, $F_{k+2} \geq \phi^k$.

Bounding the Maximum Degree

Corollary 4. The maximum degree D(n) of any node in an n-node Fibonacci heap is O(lg n).

Proof. Let x be any node in an n-node Fibonacci heap, and let k = x.degree. By the above lemma, we have $n \geq \text{size}(x) \geq \phi^{k}$. Taking base-$\phi$ logarithms gives us $k \leq \log_{\phi} n$. The maximum degree D(n) of any node is thus O(lg n).

After Class

After class reading: Part V 19.2-19.3.