Fibbonacci heap

Yulia Burkatovskaya

Outline

- Mergeable heap
- Structure of Fibbonacci heap
- Mergeable-heap operations
- Fibbonacci-heap operations
- Bounding maximum degree

Mergeable heap

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 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(H1,H2) creates and returns a new heap that contains all the elements of heaps H1 and H2. Heaps H1 and H2 are "destroyed" by this operation.
- In addition to the mergeable-heap operations above, **Fibonacci heaps** also support the following two operations:
- DECREASE-KEY(H,x,k) assigns to element x within heap H the new key
- value k, which we assume to be no greater than its current key value.1
- DELETE(H; x) deletes element x from heap H.

Mergeable heap

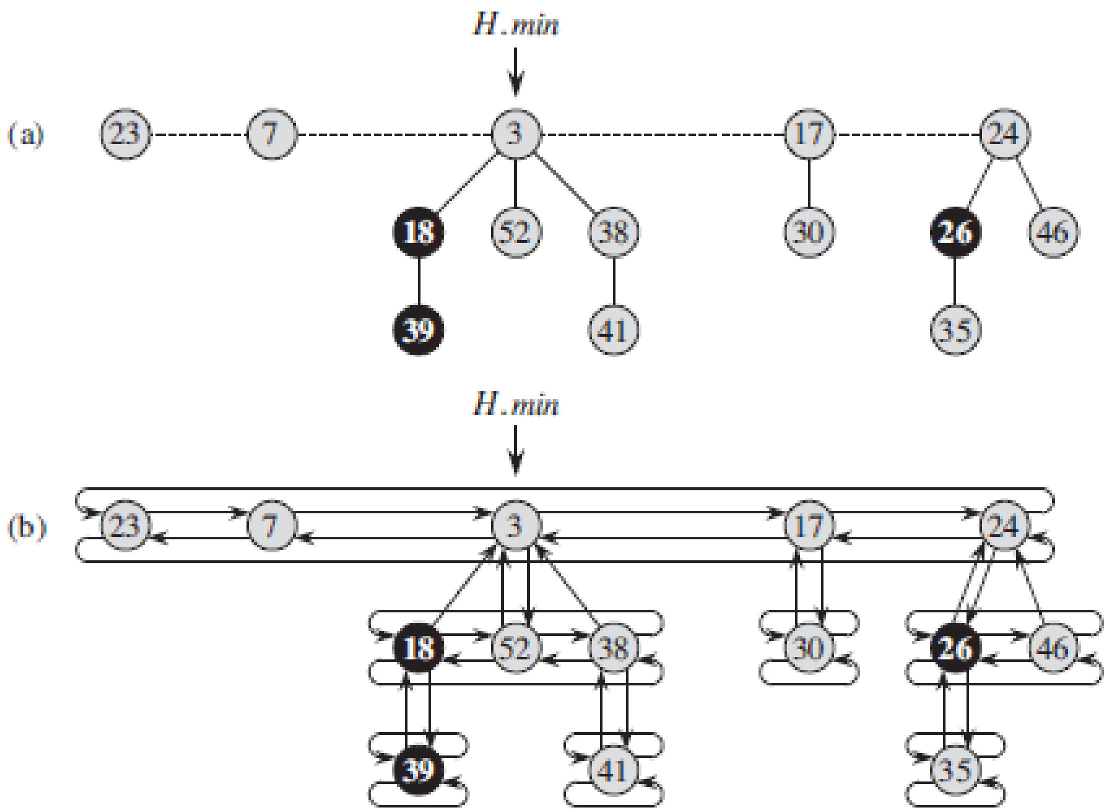
Procedure	Binary heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg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

Fibbonacci heaps

- Better to use when EXTRACT-MIN and DELETE operations are rare
- **Graphs** (single-source shortest path, shortest spanning tree)
- **Drawback**: not easy to understand and implement

Structure of Fibbonacci heap

- A Fibonacci heap is a collection of rooted trees that are min-heap ordered. That is, each tree obeys the min-heap property: the key of a node is greater than or equal to the key of its parent.



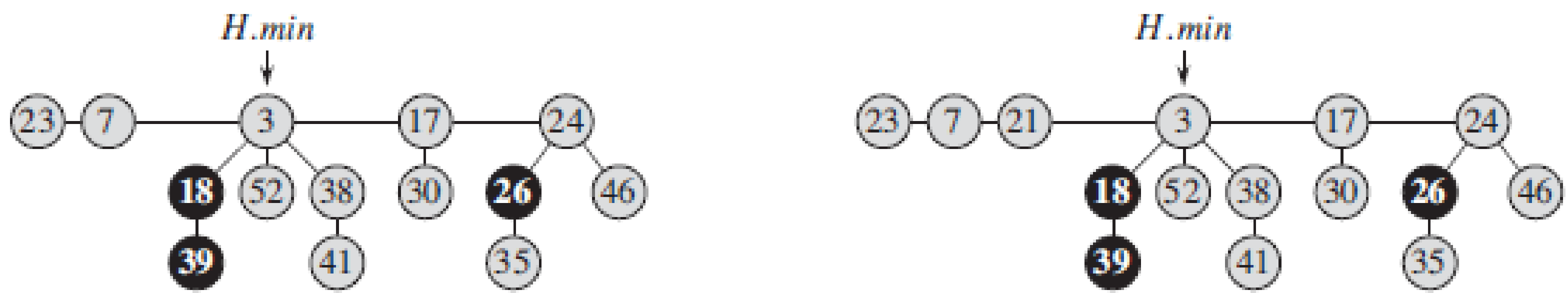
Structure of Fibbonacci heap

- **Attributes of nodes**
- **Key** (*x.key*)
- The **number of children** in the child list of node x (*x.degree*).
- The Boolean-valued attribute *x.mark* indicates whether node x has lost a child since the last time x was made the child of another node.
- Newly created nodes are unmarked, and a node x becomes unmarked whenever it is made the child of another node.
- A pointer *H*:*min* to the root of a tree containing the minimum key; we call this node the minimum node.
- The roots of the trees are connected with a doubly-linked list.

Mergeable-heap operations

- **Inserting a node**

```
FIB-HEAP-INSERT(H, x)
x.degree = 0
x.p = NIL
x.child = NIL
x.mark = FALSE
if H.min == NIL
    create a root list for H containing just x
    H.min = x
else insert x into H's root list
    if x.key < H.min.key
        H.min = x
H.n = H.n + 1
```



(a) (b)

Mergeable-heap operations

- **Uniting two Fibbonacci heaps**

FIB-HEAP-UNION(H_1, H_2)

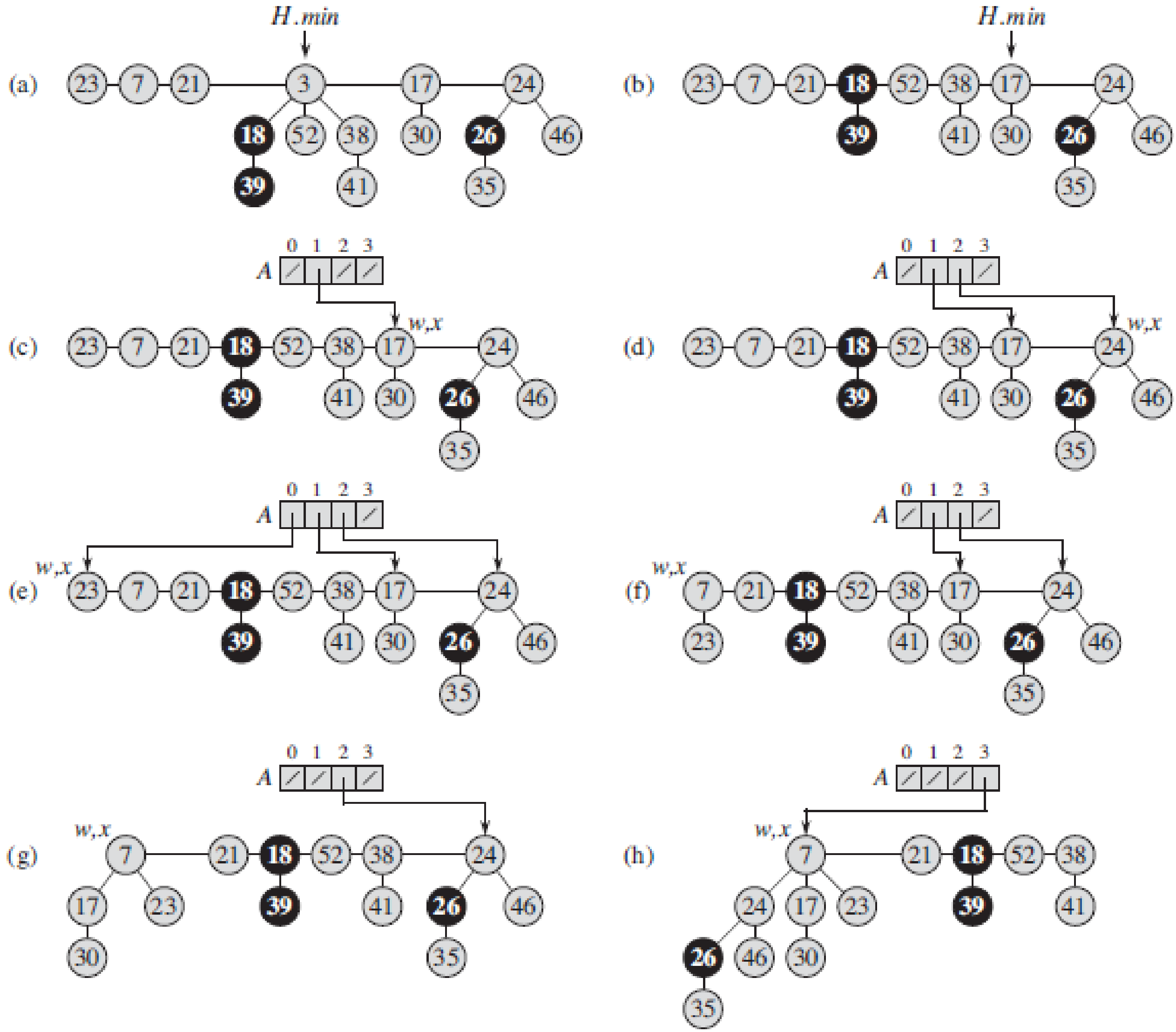
H = MAKE-FIB-HEAP()
$H.min = H_1.min$
concatenate the root list of H_2 with the root list of H
if ($H_1.min$ == NIL) or ($H_2.min \neq$ NIL and $H_2.min.key < H_1.min.key$)
    $H.min = H_2.min$
$H.n = H_1.n + H_2.n$
return H

Mergeable-heap operations

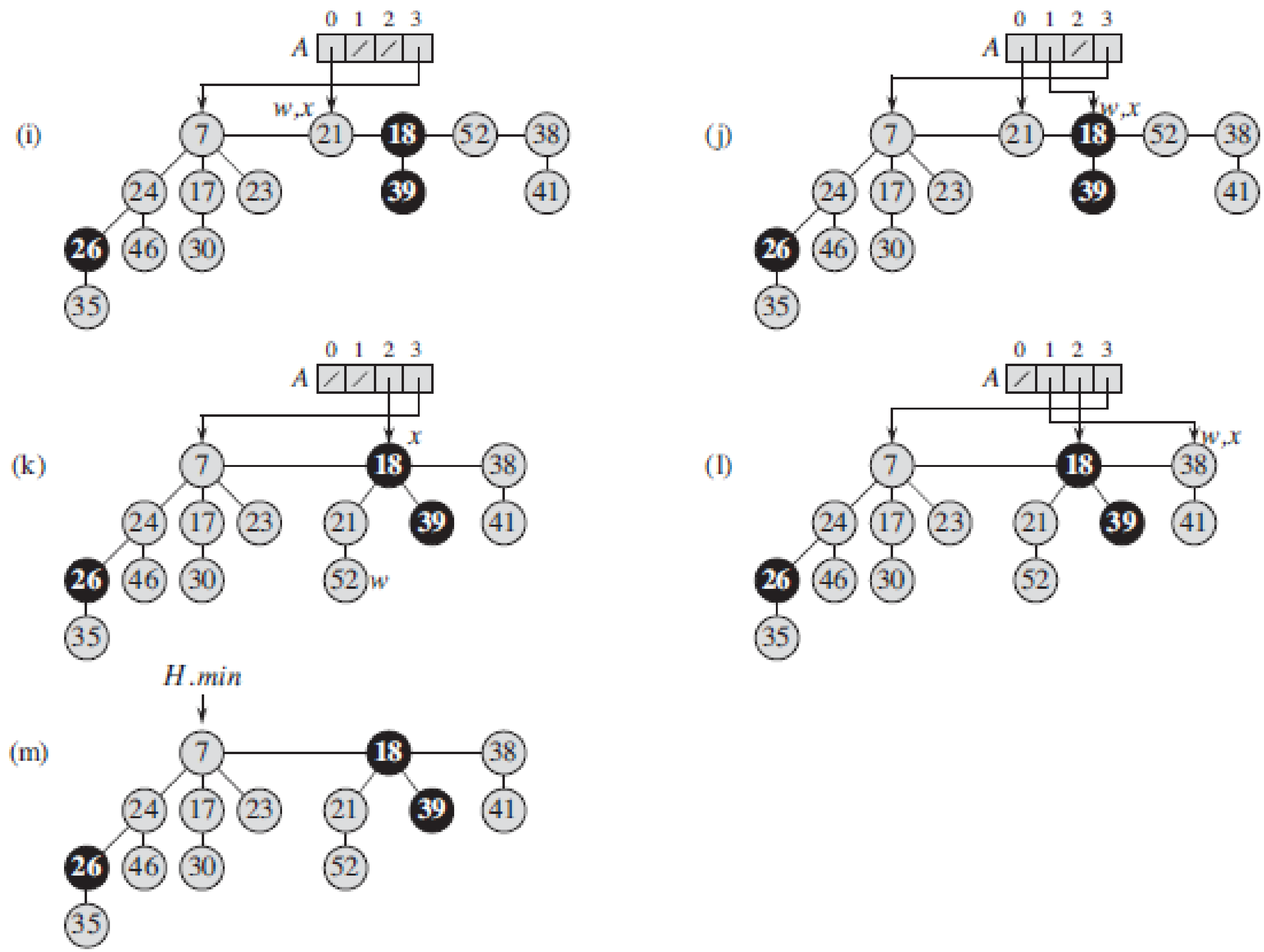
- **Extracting the minimum node**

```
FIB-HEAP-EXTRACT-MIN(H)
z = H.min
if z ≠ 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



Mergeable-heap operations



Mergeable-heap operations

- The procedure CONSOLIDATE uses an auxiliary array A[0..D(H,n)] to keep track of roots according to their degrees. If A[i]=y, then y is currently a root with y:degree=i.

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] ≠ 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
for i = 0 to D(H.n)
    if A[i] ≠ NIL
        if H.min == NIL
            create a root list for H containing just A[i]
            H.min = A[i]
        else insert A[i] into H's root list
            if A[i].key < H.min.key
                H.min = A[i]
```

```
FIB-HEAP-LINK(H, y, x)
remove y from the root list of H
make y a child of x, incrementing x.degree
y.mark = FALSE
```

Fibbonacci-heap operations

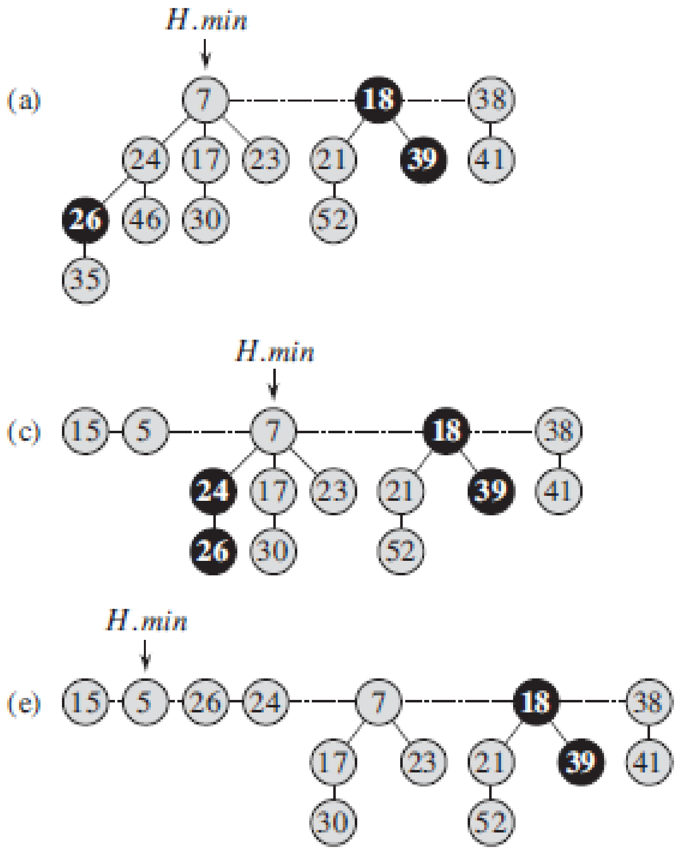
- Decreasing a key and deleting a node

```
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 ≠ NIL and x.key < y.key
    CUT(H, x, y)
    CASCADING-CUT(H, y)
if x.key < H.min.key
    H.min = x
```

```
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 = NIL
x.mark = FALSE
```

```
CASCADING-CUT(H, y)
z = y.p
if z ≠ NIL
    if y.mark == FALSE
        y.mark = TRUE
    else CUT(H, y, z)
        CASCADING-CUT(H, z)
```

Fibbonacci-heap operations



Fibbonacci-heap operations

- **Deleting a node**

FIB-HEAP-DELETE(H, x)

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

Fibbonacci-heap operations

- **Deleting a node**

FIB-HEAP-DELETE(H, x)

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

Bounding maximum degree

- For each node x within a Fibonacci heap, define size.x to be the number of nodes, including x itself, in the subtree rooted at x. (Note that x need not be in the root list—it can be any node at all.)
- Size(x) is exponential in *x:degree.*

- Let x be any node in a Fibonacci heap, and suppose that *x:degree =k.* Let y1, y2,…,yk denote the children of x in the order in which they were linked to x, from the earliest to the latest. Then, *y1:degree ≥ 0 ,…,yi :degree ≥ i–2.*
- Let x be any node in a Fibonacci heap, and let *k=x:degree*. Then *size.x* $\geq F(k-2) \geq \varphi^k$, where $\varphi=(1+\mathrm{sqrt}(5))/2$.
- The **maximum degree** D(n) of any node in an n-node Fibonacci heap is O(lg n).

Links

- https://mitpress.mit.edu/books/introduction-algorithms-third-edition
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill.