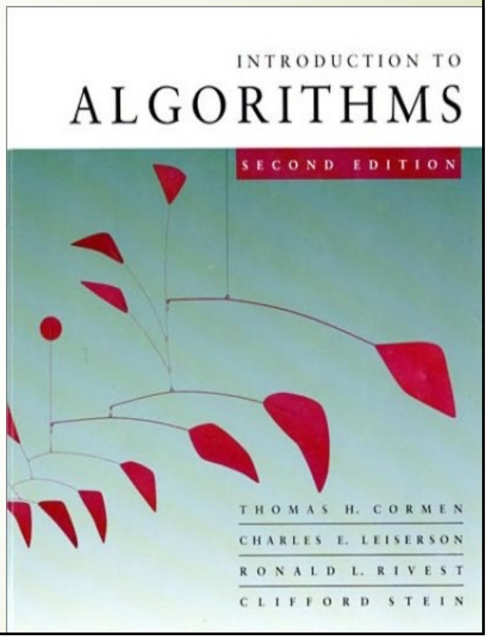
Fibonacci Heaps



SUBJECT-CODE : KCS-503

Dr. Ragini Karwayun

1

Fibonacci Heaps

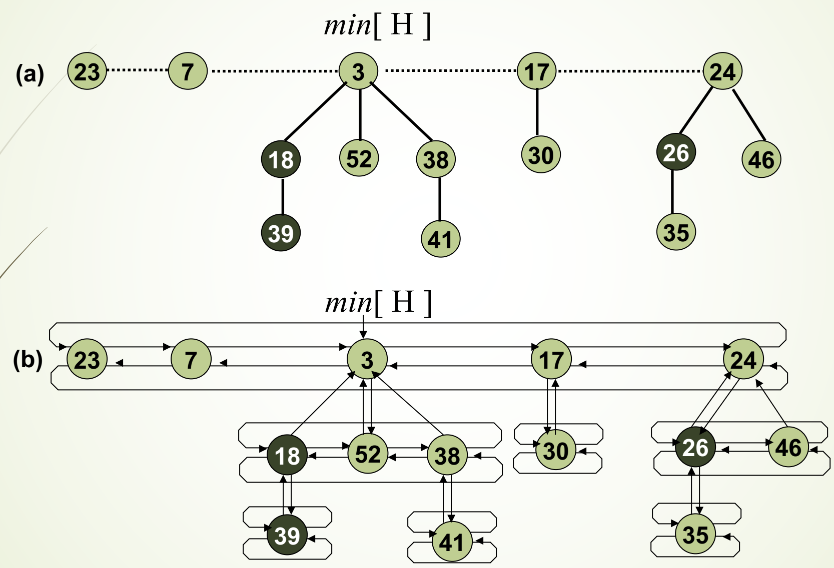
- Like Binomial Heap, a Fibonacci Heap is a collection of min-heap-ordered trees.
- Unlike trees within Binomial heaps, which are ordered, trees within Fibonacci heaps are rooted but unordered.
- Each node x contains:
  - P[x] - points to its parent
  - child[x] - points to any one of its children, children of x are linked together in a circular doubly linked list
  - degree[x] - number of children in the child list of x
  - mark[x] - indicate 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.

2

Fibonacci Heaps

- All mark fields are initially set to FALSE.
- min[H] - A given Fibonacci heap is accessed by a pointer min[H], that points to the root of the tree containing a minimum key.
- n[H] - number of nodes in H
- The roots of all the trees in a Fib-heap are linked together using their left and right pointers into a circular, doubly linked list called the root list of the Fib-Heap H.
- The order of the trees within a root list is arbitrary.

Fibonacci Heaps



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

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

- D(n): upper bound on the max degree of any node in an n-node Fibonacci heap
  - D(n) = O(lg n)

Amortized Analysis

- In an amortized analysis, the time required to perform a sequence of data structure operations is averaged over all the operations performed.
- It differs from the average case analysis in that probability of occurrence of data is not involved.
- It guarantees the average performance of each operation in the worst case.
- Amortized analysis is concerned with the overall cost of a sequence of operations.

Properties of Unordered Binomial Tree U_k

- Unordered binomial tree:

  U_0: a single node

  U_k: consists of 2 unordered binomial trees U_{k-1} for which the root of one is made into any child of the root of the other

Lemma: For an unordered binomial tree U_k,

1. There are 2^k nodes
2. height of the tree = k
3. There are exactly C(k,i) nodes at depth i for i=0,1,2,….k
4. deg(root) = k which is greater than that of any other node. The children of root are roots of subtrees U_{k-1}, U_{k-2}, …, U_0 in any order.

Lemma

- The maximum degree D(n) of any node in an n-node fib heap is O(lg n)

  - Let x be any node in an n-node fib heap.
  - Let k = degree[x]
  - We have, $n \geq size[x] \geq \phi^k$ ϕ is referred as golden ratio
  - Taking base ϕ log
  - $k \leq \log_\phi n$
  - The maximum degree D(n) of any node is thus O(lg n)

Creating a new Fibonacci Heap

Make-Fib-Heap(H)
n[H]=0
min[H]=nil
return H

From
There are no trees in H therefore
t(H)=0 , m(H)=0 so Φ (H)=0

⇒ The amortized cost of Make-Fib-Heap is equal to its O(1) actual cost.

Inserting a node in Fibonacci Heap

Fib-Heap-Insert(H, x)

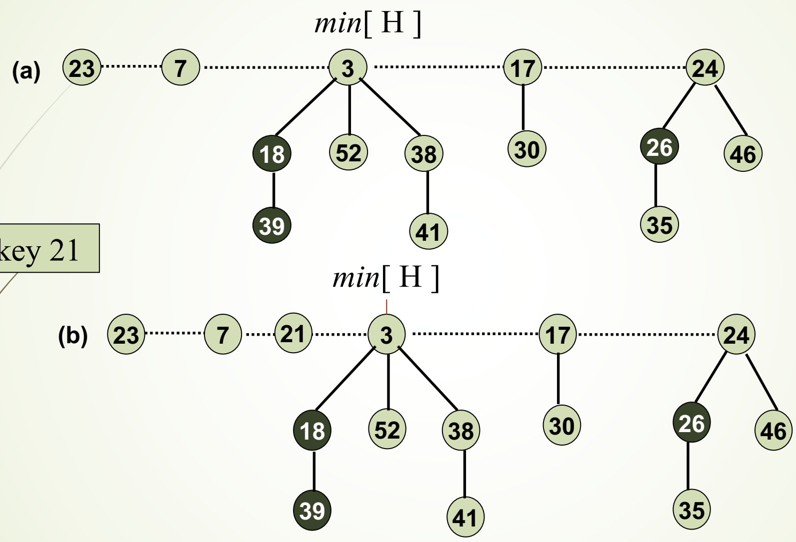
Assumes that the node x is already allocated and key[x] has already been filled in

```
{ 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]]
      min[H] ← x
  n[H] ← n[H]+1
}
```

Note : algo does not attempt to consolidate the trees within the fib heap.

Inserting a node in Fibonacci Heap

Insert key 21



11

Amortized Cost

- If k consecutive Fib-Heap_insert Operations occur then k single node trees are added to the root list.
- To deterine the amortized cost of insert procedure, let H be the Fib-Heap and H' be the resulting heap.

$$t(H') = t(H) + 1 \text{ and } m(h') = m(H)$$

Therefore increase in potential is :

((t(H) +1)+ 2m(H)) – (t(H) + 2m(H)) = 1 i.e. constant time

Therefore :

- Actual cost = O(1).
- Change in potential = +1.
- Amortized cost = O(1) + 1 = O(1)

12

Finding Minimum

- Finding the minimum node : given by a pointer min[H]
- So we can find the min node in O(1) actual time.
- Amortized cost O(1). Φ is not changed

Uniting two Fibonacci Heaps

<u>Fib-Heap-Union(H_1, H_2)</u>

{ H← Make-Fib-Heap[]

min[H] ← min[H_1]

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

if (min[H_1]=NIL) or (min[H_2] ≠ NIL and min[H_2]<min[H_1])

min[H] ← min[H_2]

n[H] ← n[H_1]+n[H_2]

free the objects H_1 and H_2

return H

}

Amortized Cost

- Change in potential

$= \Phi(H) - (\Phi(H_1)+\Phi(H_2))$

$= (t(H)+2m(H)) – ((t(H_1)+2m(H_1)) + (t(H_2)+2m(H_2)))$

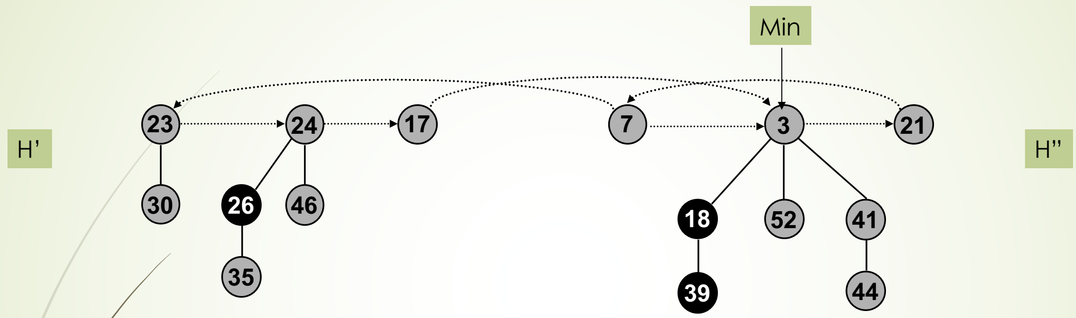
$= 0$

$\because$ **$t(H) = t(H_1) + t(H_2)$ and $m(H) = m(H_1) + m(H_2)$**

Thus the amortized cost of Fib-Heap-Union is therefore O(1) ~ actual cost

15

Fibonacci Heaps: Union



16

Fibonacci Heaps: Extracting the minimum

<u>Fib-Heap-Extract-Min(H)</u>

```
        z ← min[H]
        if z ≠ NIL
        {
                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]
                        min[H] ← NIL
                else
                        min[H] ← right[z]
                        Consolidate(H)
            n[H] ← n[H] – 1
        }
        return z
```


17

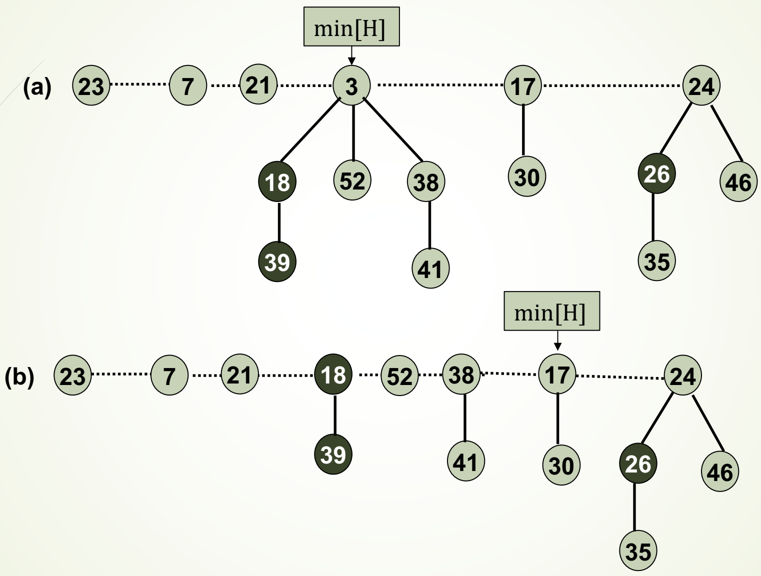
Consolidate(H)

```
for i ← 0 to D(n[H]) do A[i]=NIL
        for each node w in the root list of H
            x ← w
            d ← degree[x]
            while A[d] ≠ NIL
                y← A[d]
                if key[x]>key[y]
                    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])
            if A[i] ≠ NIL
                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]
```

```
Fib-Heap-Link(H, y, x)
 {
   remove y from the root list of H;
   make y a child of x;
   degree[x]←degree[x]+1;
   mark[y] ← FALSE;
 }
```


18

Example

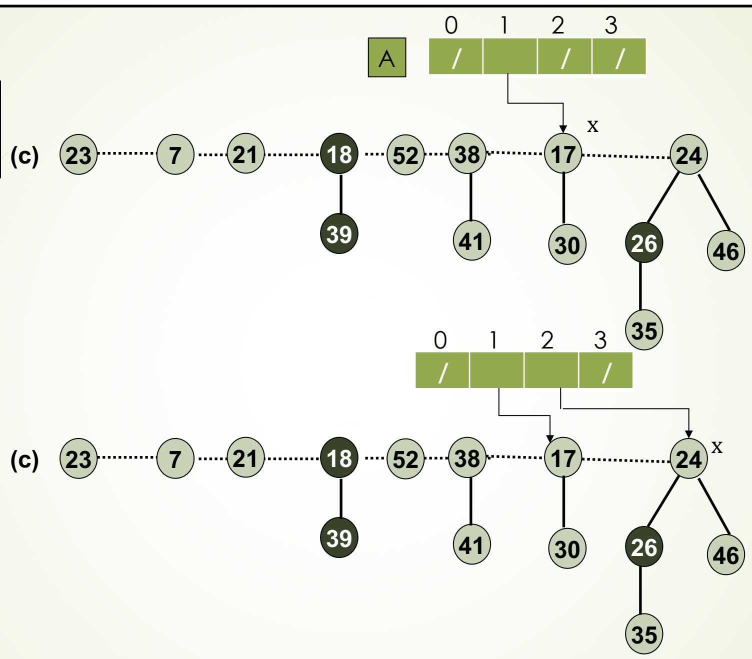


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

19

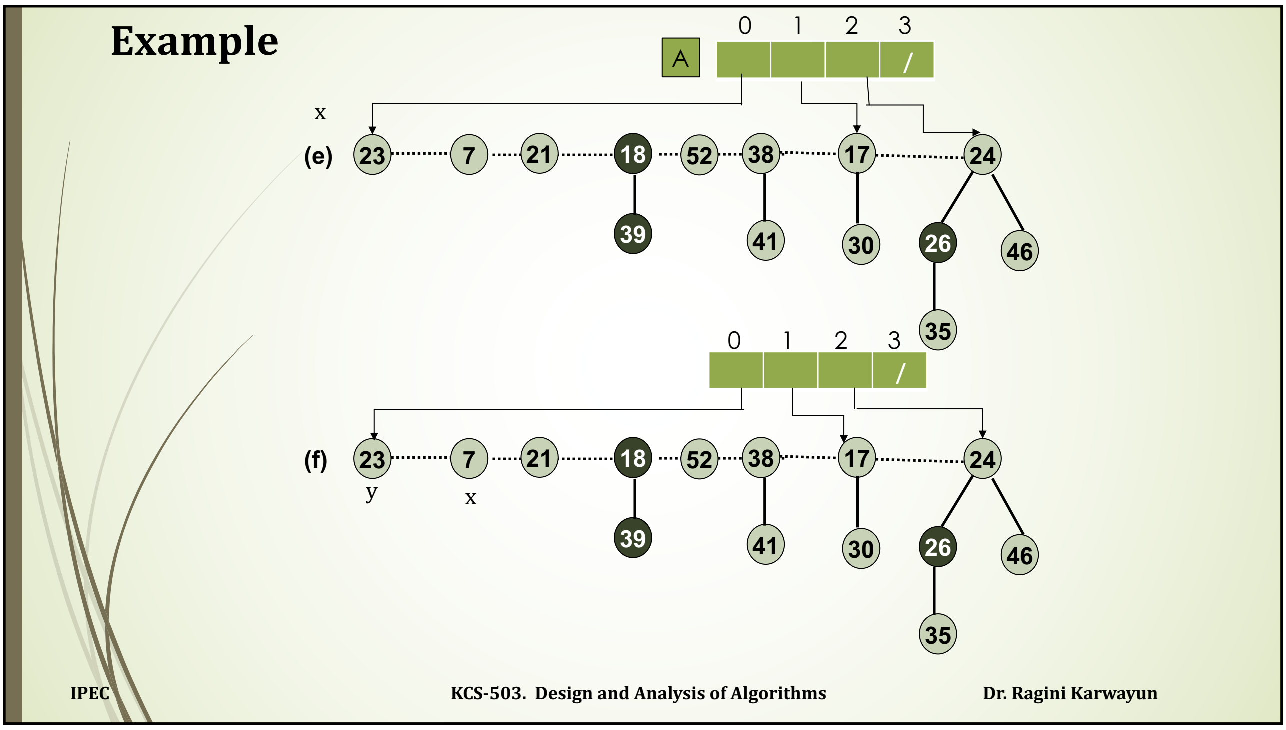
Example

A = [0.... D[n]]
 = [0.... ≤⌊log 15⌋
 = [0...3]

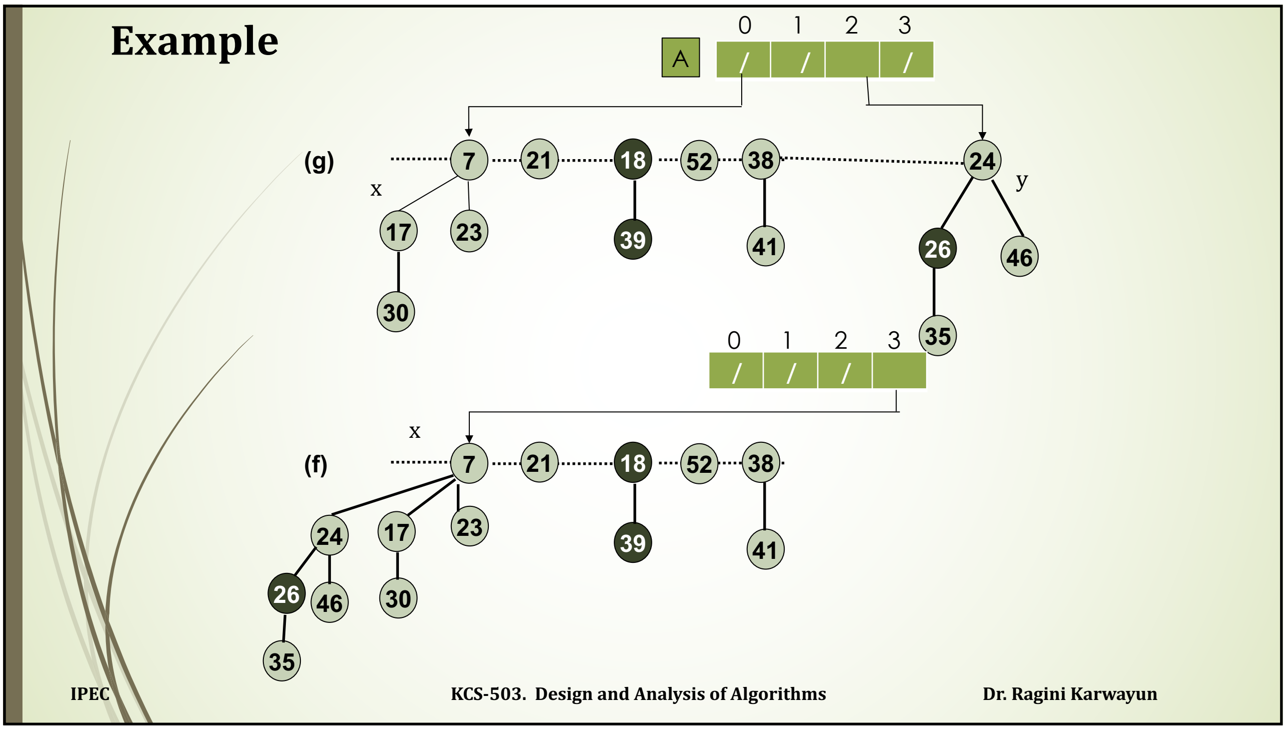


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

20

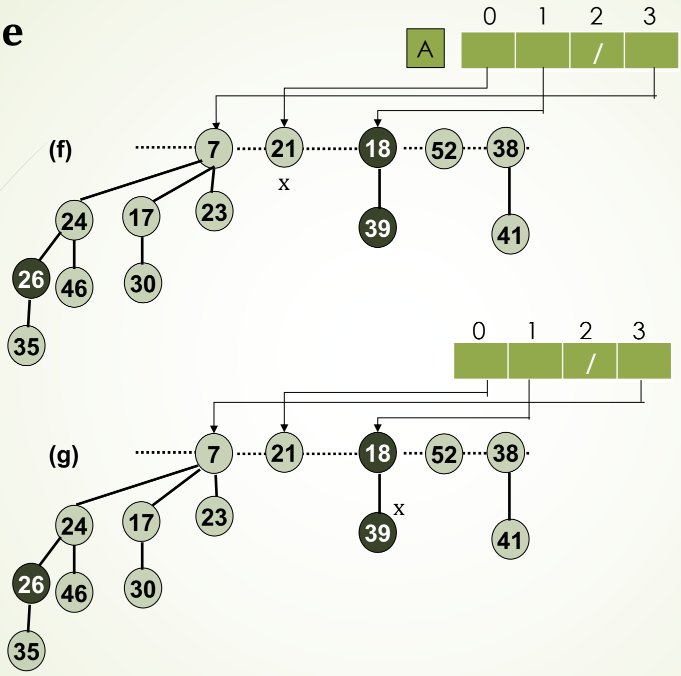

Example
0 1 2 3
A
/
x
(e)
23 7 21 18 52 38 17 24
39 41 30 26 46
35
0 1 2 3
/
(f)
23 7 21 18 52 38 17 24
y x
39 41 30 26 46
35
IPEC
KCS-503. Design and Analysis of Algorithms
Dr. Ragini Karwayun

21


Example
0 1 2 3
A
/ / /
(g)
7 21 18 52 38 24
x y
17 23 39 41 26 46
30 35
0 1 2 3
/ / /
x
(f)
7 21 18 52 38
24 17 23 39 41
26 46 30
35
IPEC
KCS-503. Design and Analysis of Algorithms
Dr. Ragini Karwayun

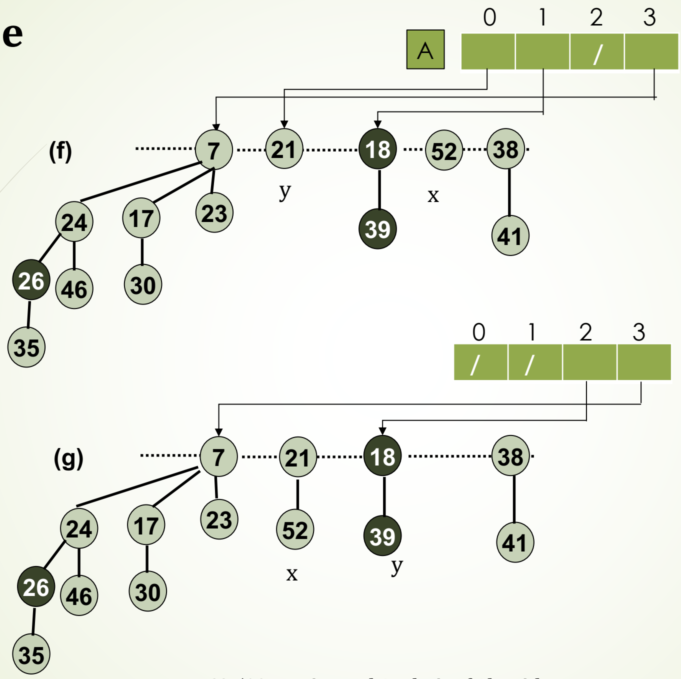
22

Example



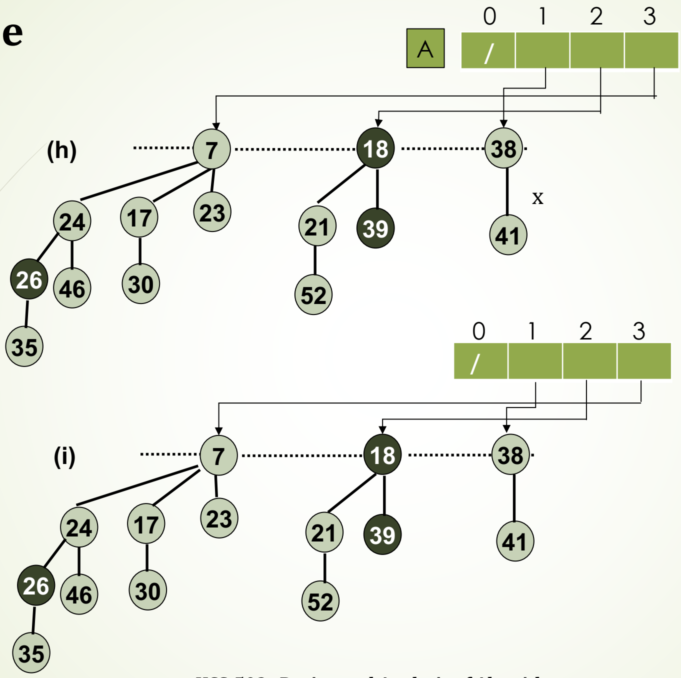
23

Example



24

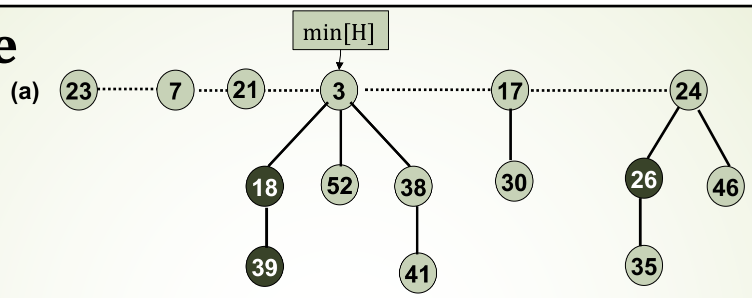
Example



IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

25

Example

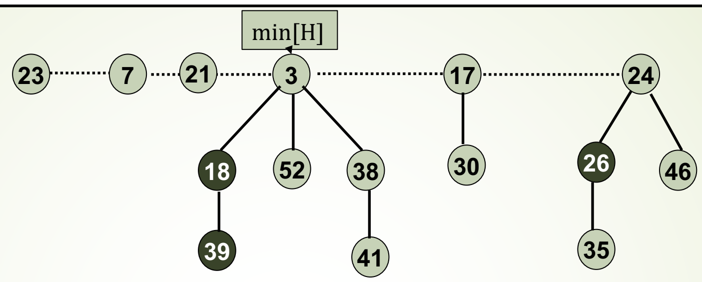


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

26

Example

(a)



27

Analysis of Fib-Heap-Extract-Min

H : n-node Fib-Heap

Note : MaxDeg(n) >= max deg of a node in final heap

Actual cost :

O(D(n)) : for-loop in Fib-Heap-Extract-Min

D(n)+t(H)-1 : size of the root list

Total actual cost:

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

∴ At most D(n)+1 nodes remain on the list and no nodes become marked

Potential *before* extracting : t(H)+2m(H)

Potential *after* extracting : ≤ D(n)+1+2m(H)

Thus the amortized cost is at most:
O(D(n))+t(H)+[(D(n)+1+2m(H)) – (t(H)+2m(H))]
= O(D(n)+t(H)-t(H))
= O(D(n))

28

Decreasing a key and deleting a node

do not preserve the property that all trees in the Fibonacci heap are unordered binomial trees.
Roughly, we mark a node if it has lost a child

```
Fib-Heap-Decrease-key(H, x, k)
if k>key[x]
    error "new key is greater than current key"
key[x] ← k
y ← P[x]
if y≠NIL and key[x]<key[y]
    CUT(H, x, y)
    CASCADING-CUT(H, y)
if key[x]<key[min[H]]
        min[H] ← x
```

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

29

Decreasing a key and deleting a node

```
CUT(H, x, y)
    remove x from the child list of y, decrease degree[y]
    add x to the root list of H
    P[x] ← NIL
    mark[x] ← FALSE
```

```
CASCADING-CUT(H, y)
    z← P[y]
    if z≠NIL
        if mark[y]=FALSE
            mark[y] ← TRUE
        else
            CUT(H, y, z)
            CASCADING-CUT(H, z)
```

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

30

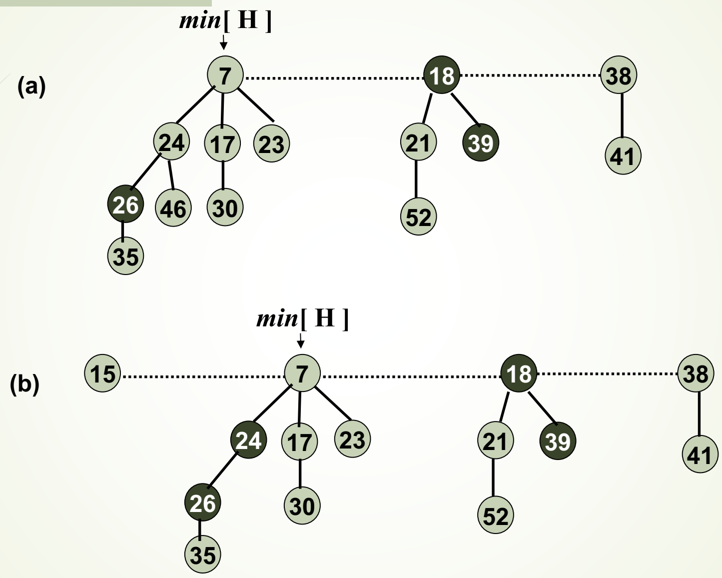
Decreasing a key and deleting a node

- Let x be any node in a Fib-heap and y=p(x)
- Decrease key of x to k.
- If k > key(y) leave.
- else
  1. If k < key(y) then cut x from the tree and add it to the root list of H.
  2. Mark(x) = FALSE
  3. If node y (parent of (x)) is unmarked, mark it and leave.
  4. Else, if it is marked , cut it from the tree, unmark it and add it to the root list.
  5. Repeat the process until you reach the root.

31

EXAMPLE

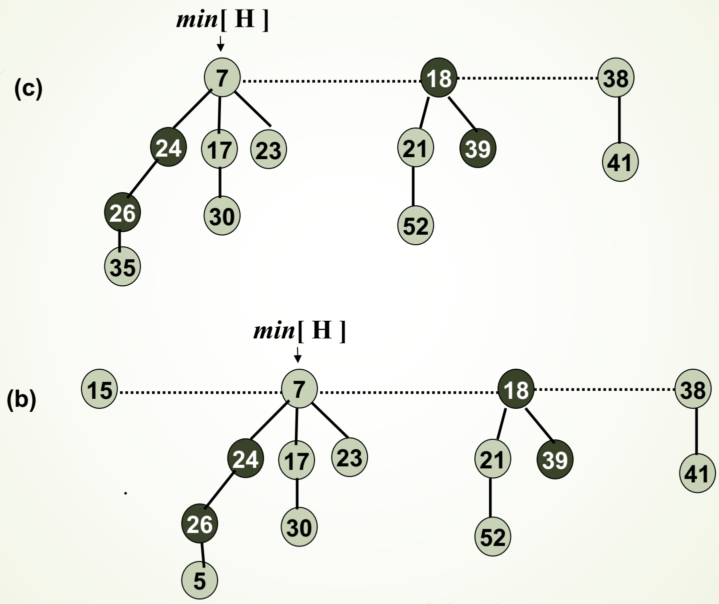
Decreasing x=46 with k=15



32

Decreasing x=35 with k=5

EXAMPLE

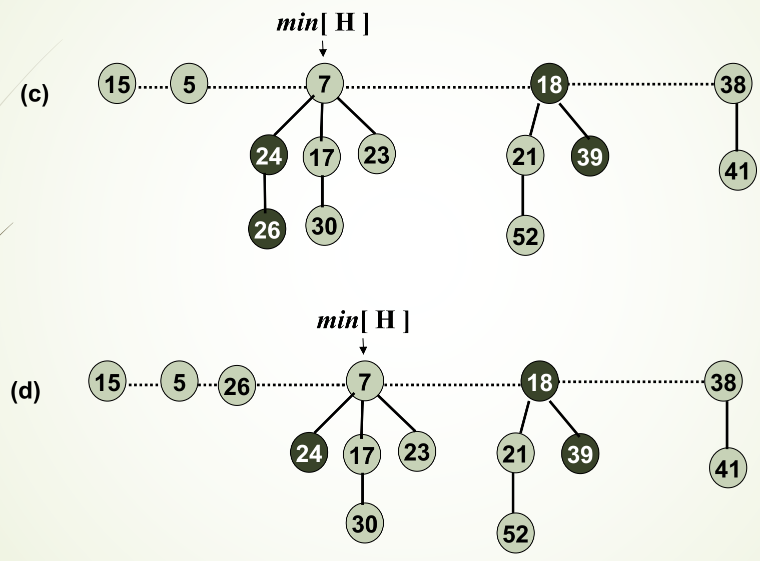


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

33

Decreasing x=35 with k=5

EXAMPLE

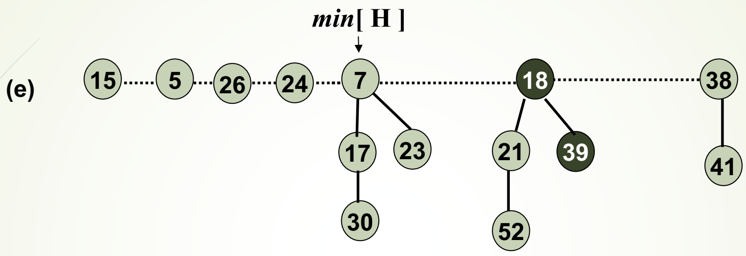


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

34

Decreasing x=35 with k=5

EXAMPLE



Analysis of Decrease-key

Actual cost : O(c) suppose CASCADING-CUT is recursively called c times

Each recursive call of CASCADING-CUT except for the last one, cuts a marked node and clears the mark bit, takes O(1) time.

Last call of CASCADING-CUT may have marked a node

After Decrease-key, there are at most t(H)+c trees, and at most m(H)-c+2 marked nodes.

Thus the potential change is : [t(H)+c+2(m(H)-c+2)] - [t(H)+2m(H)]
= 4-c

Amortized cost: O(c)+4-c = O(1)

Fibonacci Heaps: Delete

- Delete node x.

```
Fib-Heap-Delete(H, x)
   Fib-Heap-Decrease-key(H, x, -∞)
   Fib-Heap-Extract-Min(H)
```

- Amortized cost. O(D(n))
  - O(1) for decrease-key.
  - O(D(n)) for delete-min.
  - D(n) = max degree of any node in Fibonacci heap.
  - Since D(n) = O(lg n)
  - Therefore amortized time of Fib-Heap-Delete = O(lg n)

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

37

Fibonacci Heaps

- Bounding the maximum degree:

$$\text{Goal}: D(n) \le \lfloor \log_\phi n \rfloor,$$

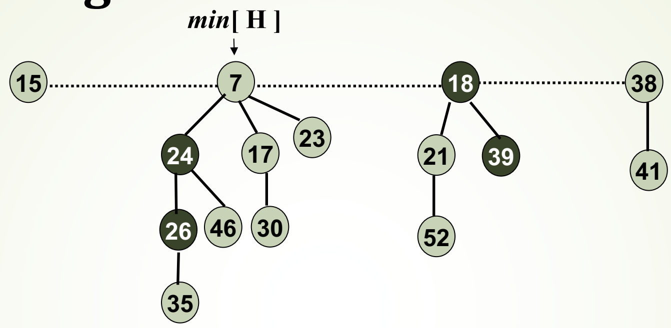
$$\phi = (1+\sqrt{5})/2$$

Golden Ratio

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

38

Assignment on Fibonacci Heaps



1. Perform Extract_Min on the Fib Heap given above.
2. Decrease 52 to 25 in the heap given.
3. Decrease 30 to 10 in the heap given.
4. Decrease 46 to 5 in the heap given.
5. Delete 35.
6. Suppose that a root x in a Fibonacci heap is marked. Explain how x came to be a marked root.