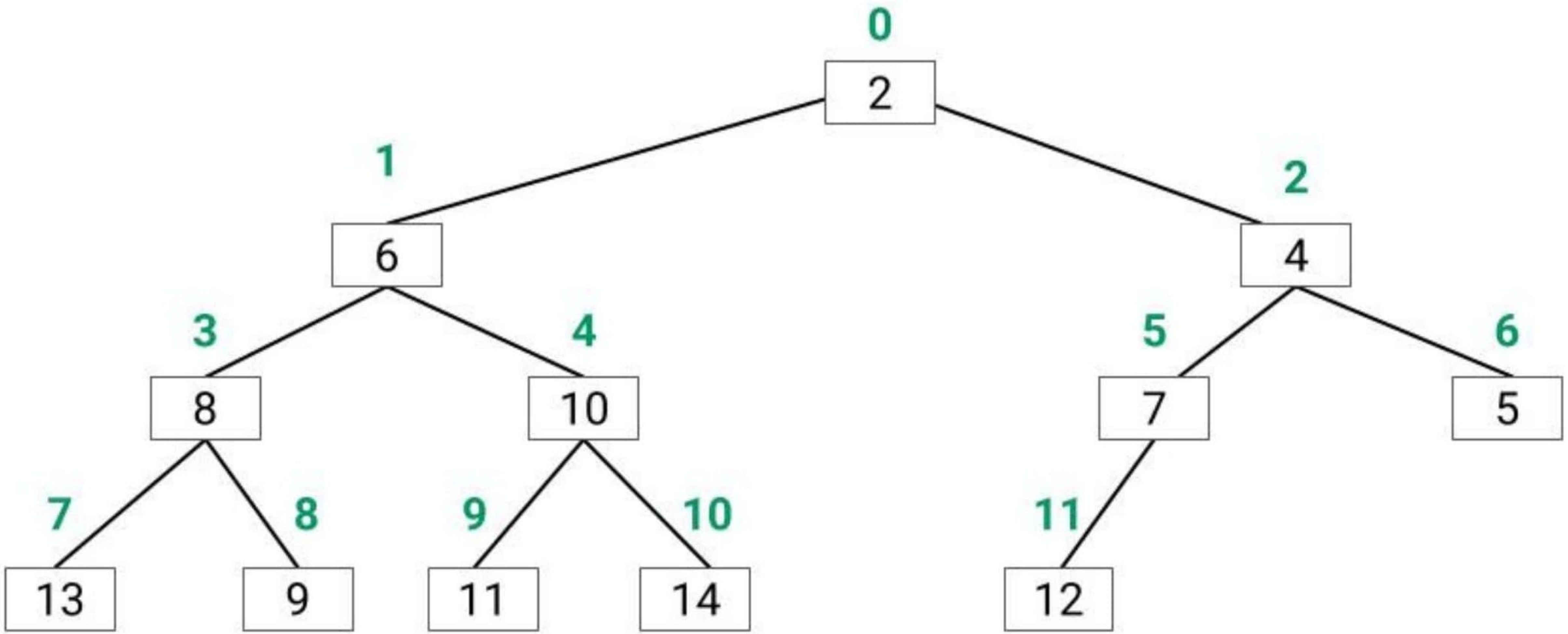
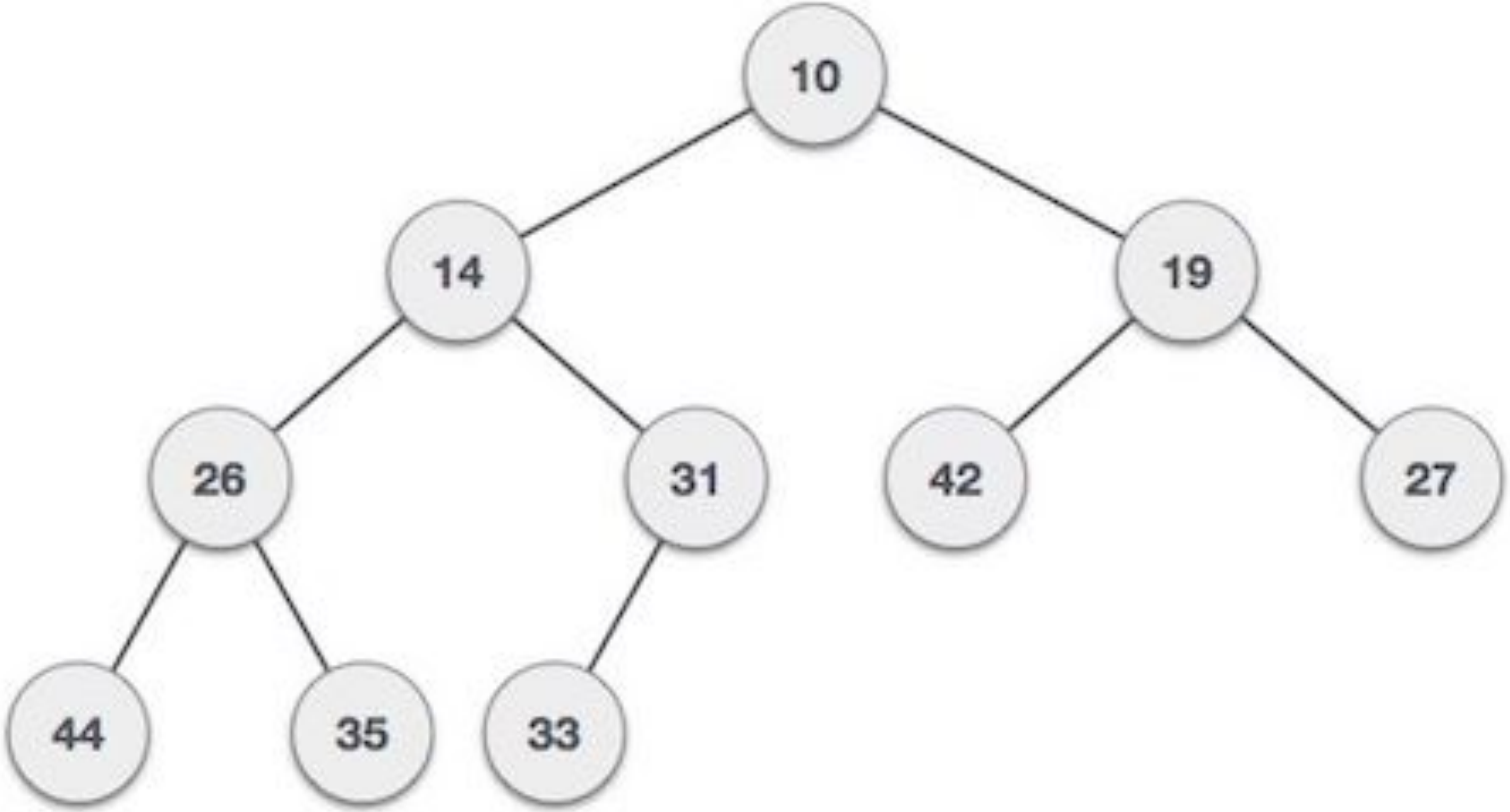
Heap

Heap is a complete binary tree where each element satisfies a heap property. In a complete binary tree all levels are full except the last level, ie nodes in all levels except the last level will have 2 children. All levels will be filled from left to right.

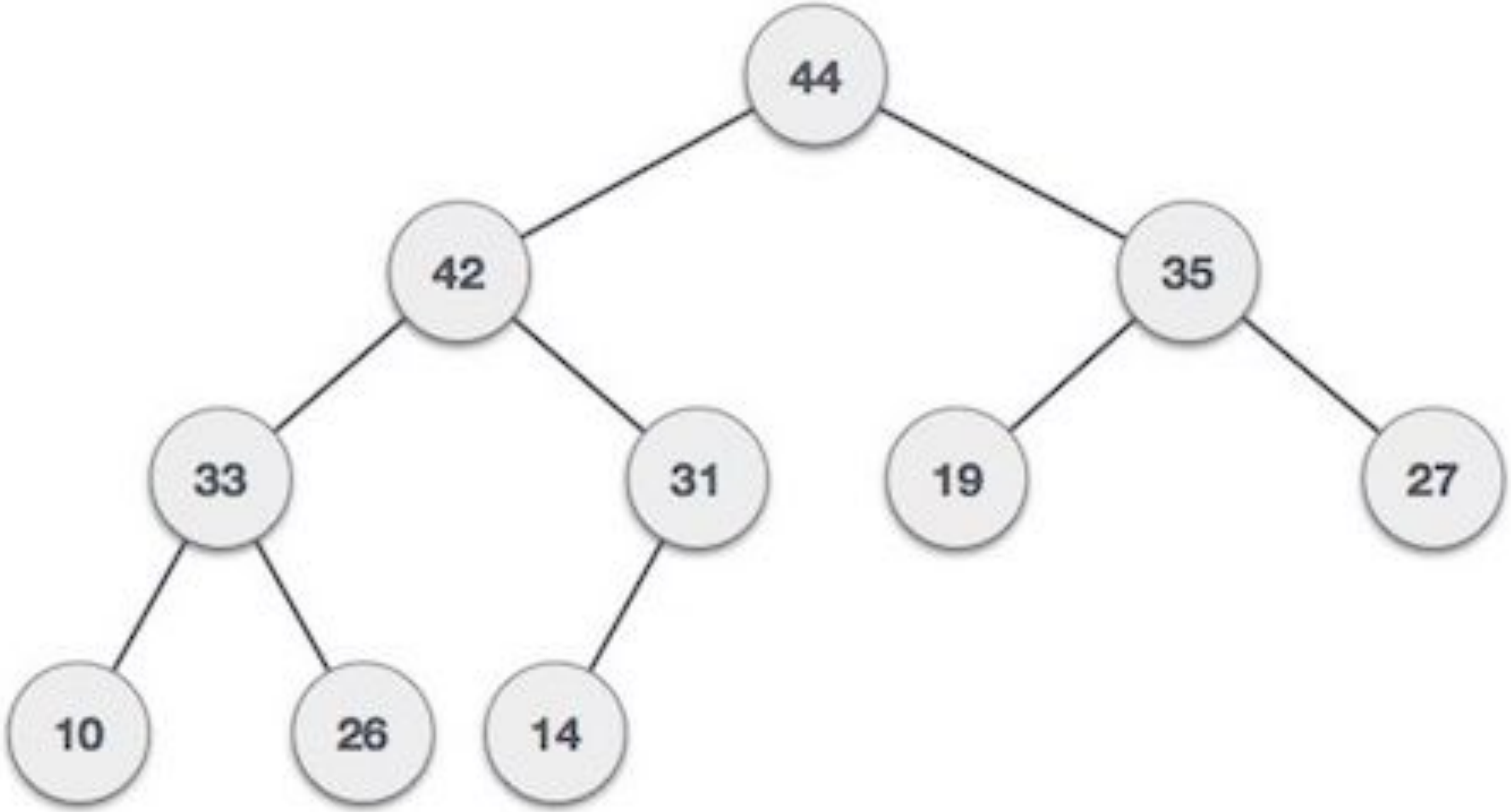
Heap Property: All nodes are either **greater than or equal to** or **less than or equal to** each of its children. If the parent nodes are greater than their child nodes, heap is called a **Max-Heap**, and if the parent nodes are smaller than their child nodes, heap is called **Min-Heap**.


0
2
1
6
2
4
3
8
4
10
5
7
6
5
7
13
8
9
9
11
10
14
11
12

Min-Heap − Where the value of the root node is less than or equal to either of its children.



Max-Heap − Where the value of the root node is greater than or equal to either of its children.



Mergeable Heaps

Mergeable heap : Data structure that supports the following 5 operations:

- Make-Heap( ) : return an empty heap
- Insert(H, x, k) : insert an item x with key k into a heap H
- Find-Min(H) : return item with min key
- Extract-Min(H) : return and remove
- Union(H1 , H2) : merge heaps H1 and H2

Examples of **Mergeable heap** -

Binomial Heap and Fibonacci Heap

Binomial Tree

A Binomial Tree of order 0 has 1 node. A Binomial Tree of order k can be constructed by taking two binomial trees of order k-1 and making one as leftmost child or other.

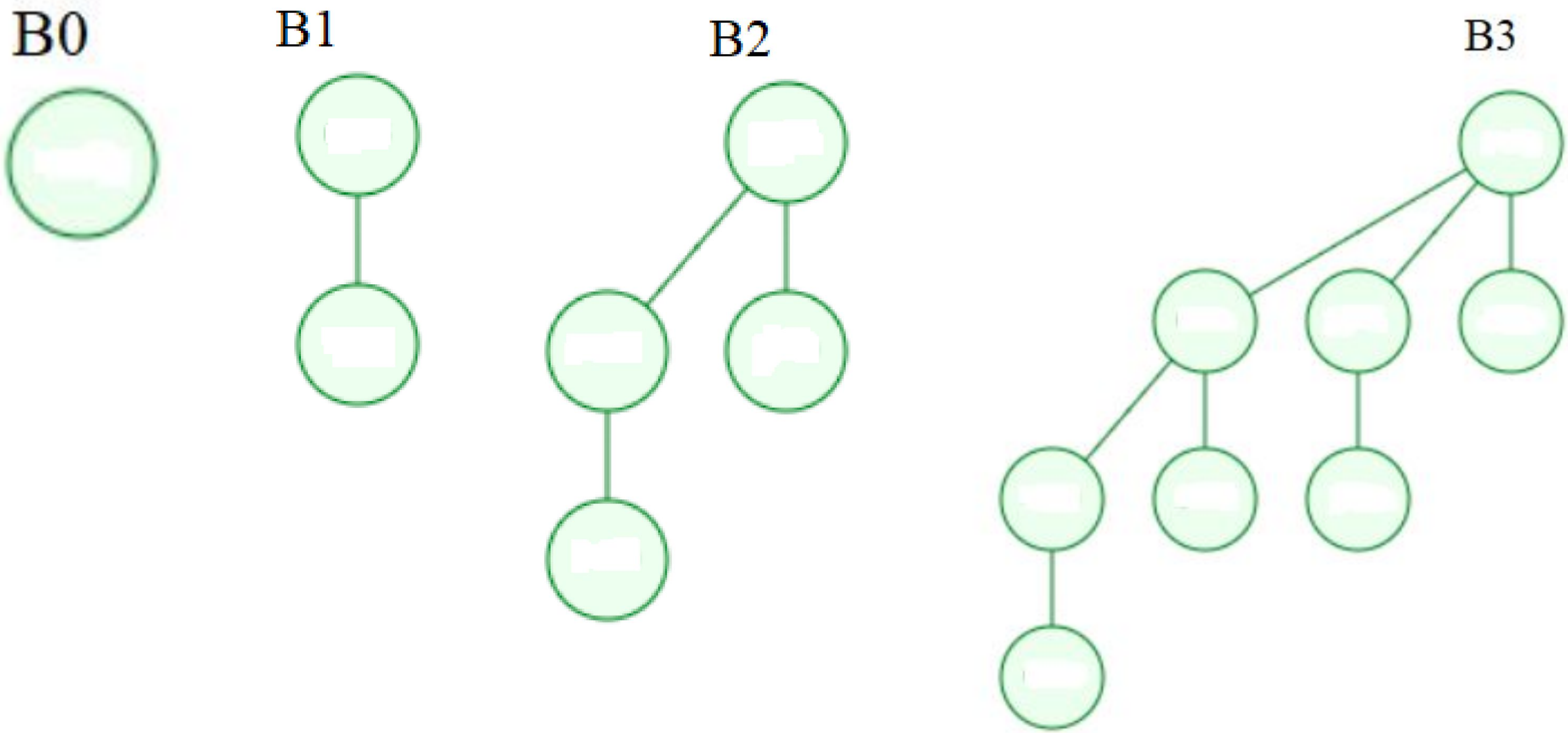
A Binomial Tree of order k has following properties.

a) It has exactly 2^k nodes.

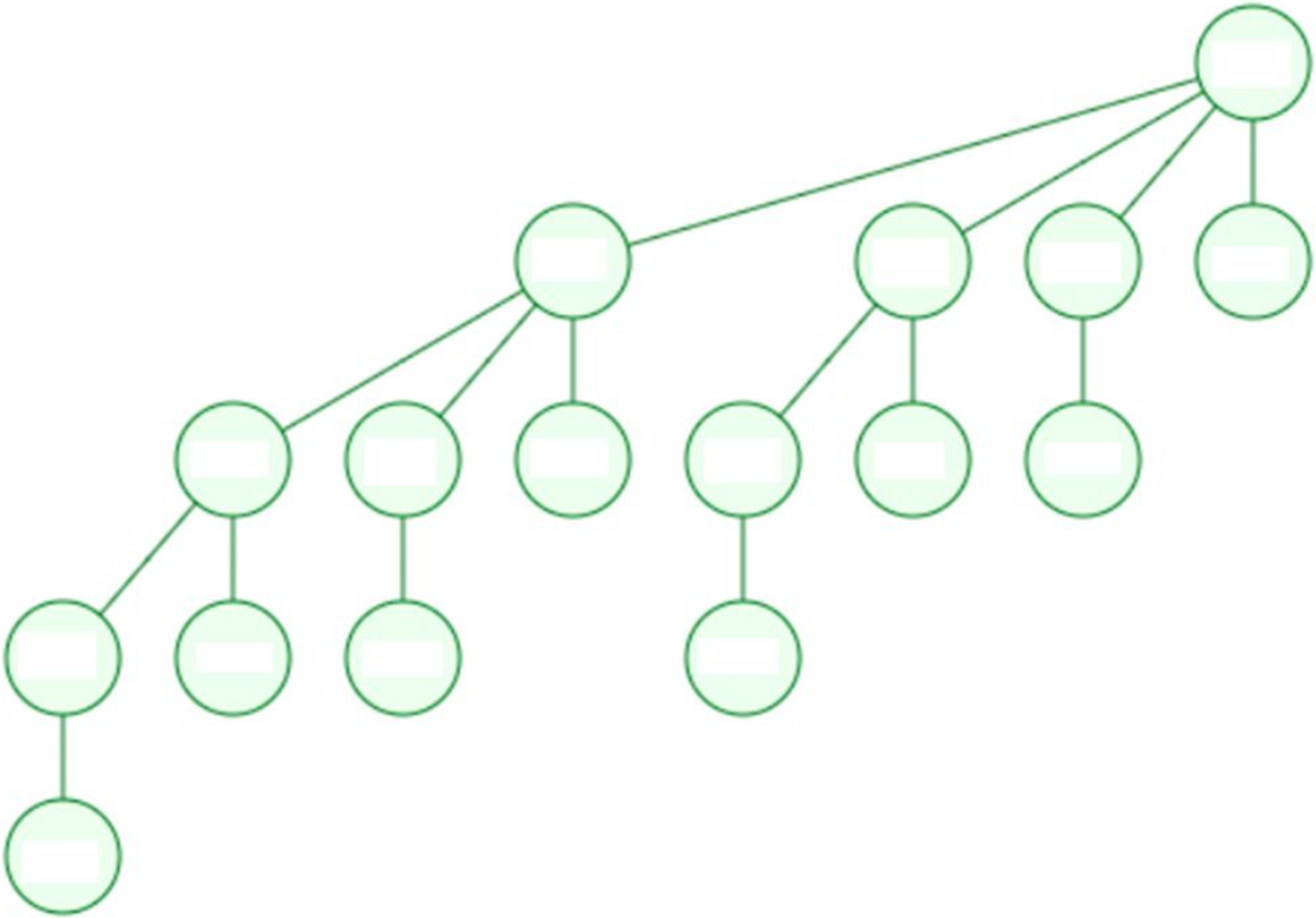
b) It has depth as k.

c) There are exactly ${}^kC_i$ nodes at depth i for i = 0, 1, . . . , k.

d) The root has degree k and children of root are themselves Binomial Trees with order k-1, k-2,.. 0 from left to right.


B0
B1
B2
B3

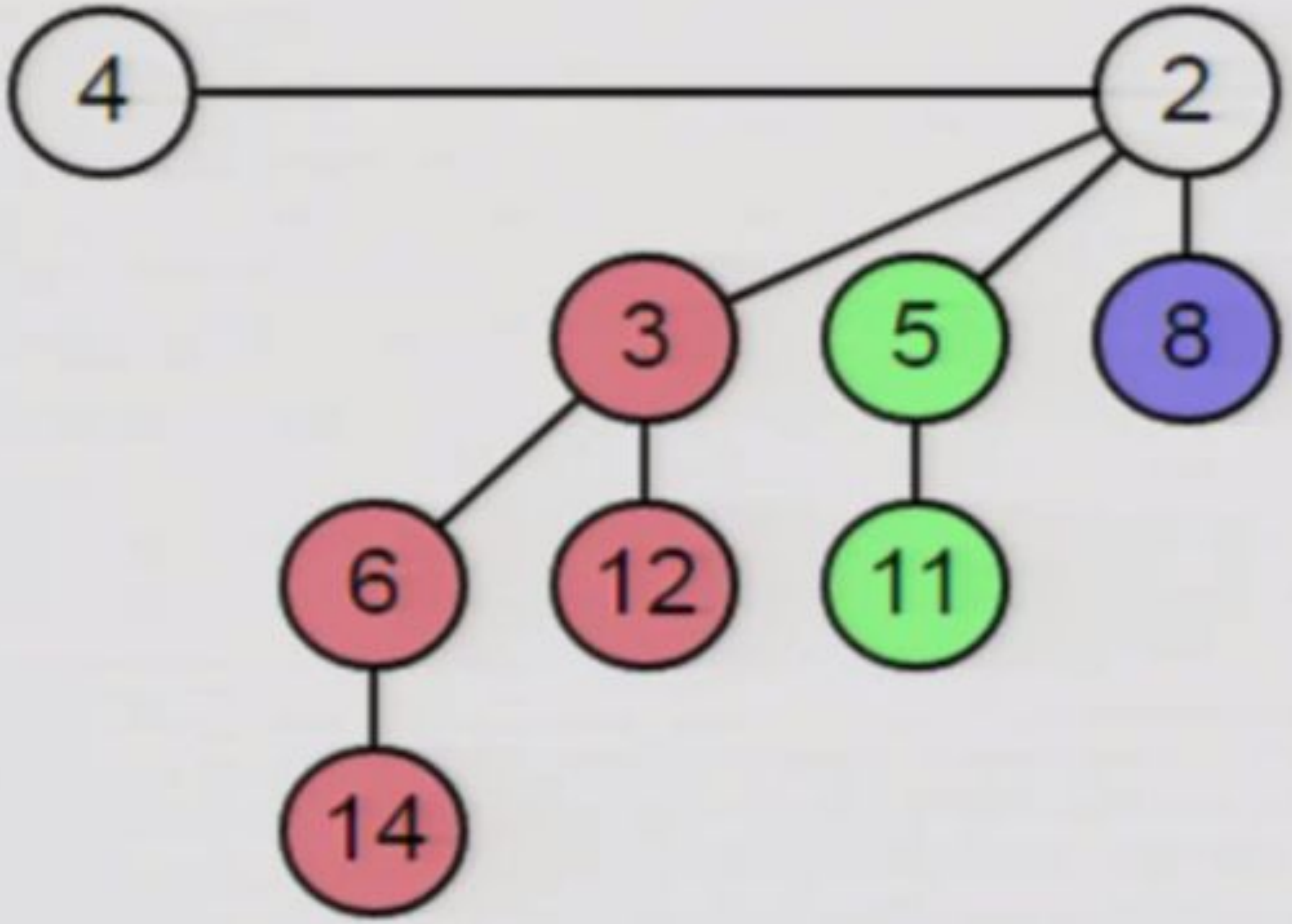
B4



Binomial Heap and Properties

- Binomial Heap is a collection of Binomial trees that satisfy the following Binomial Heap properties:
    - Each binomial tree in a heap H obeys min-heap property.
    - For any non-negative integer k, there must be at most 1 binomial tree whose degree is k.
    - Binomial trees will be joined by the linked lists of the roots.

Binomial Heap- Example

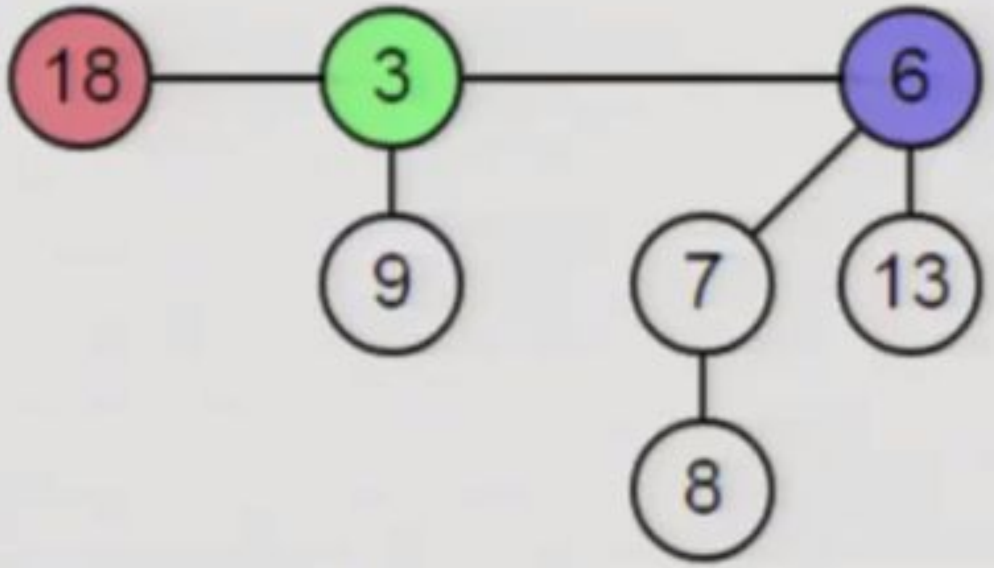


Various Operations on Binomial Heap

- Creating a new Binomial Heap.
- Finding the minimum key in a Binomial Heap.
- Union of two Binomial Heaps.
- Inserting a node in a Binomial Heap.
- Extracting the minimum node from Binomial Heap.
- Decreasing a key Value of a node in a Binomial Heap.
- Delete a node from Binomial Heap.

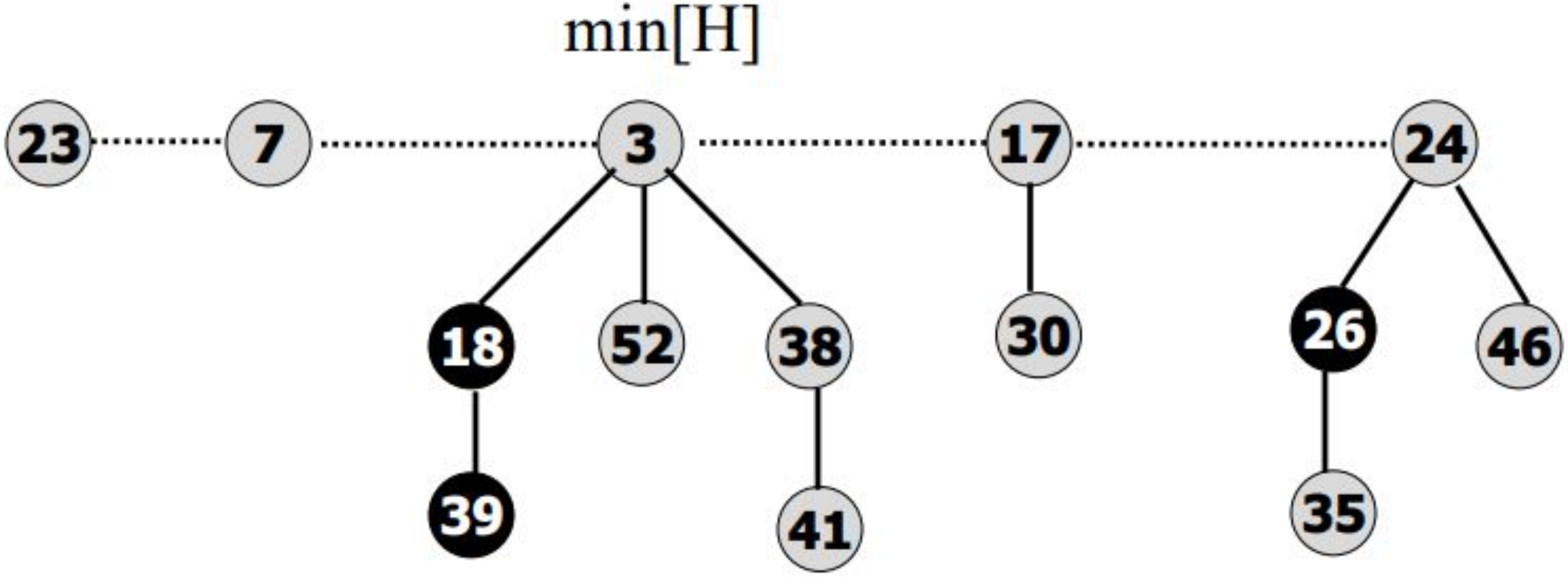
Finding the minimum key- Example

Find minimum iterates through the roots of each binomial tree in the heap.



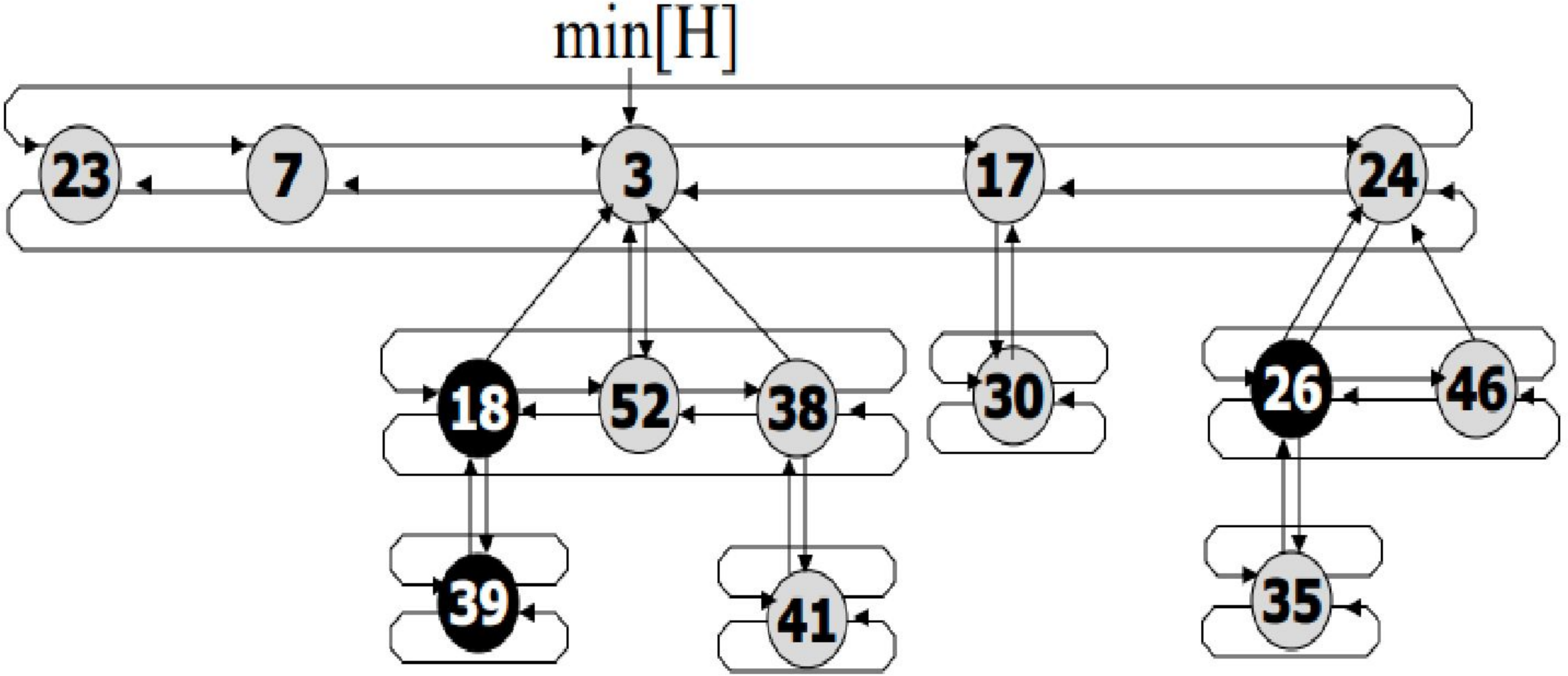
Fibonacci Heap

- A collection of min-heap ordered trees
- Fibonaaci heap can have many trees of same degree and a tree does not have exactly 2^k nodes.
- Each tree is rooted but “unordered” , meaning there is no order between the child nodes of a node and, roots



Each node x has

- One parent pointer p[x]
- One child pointer child[x] which points to an arbitrary child of x
- The children of x are linked together in a circular, doubly linked list
- Each node y has pointers left[y] and right[y] to its left and right node in the list
- The root of the trees are again connected with a circular, doubly linked list using their left and right pointers
- A pointer min[H] which points to the root of a tree containing the minimum element (minimum node of the heap)
- A variable n[H] storing the number of elements in the heap

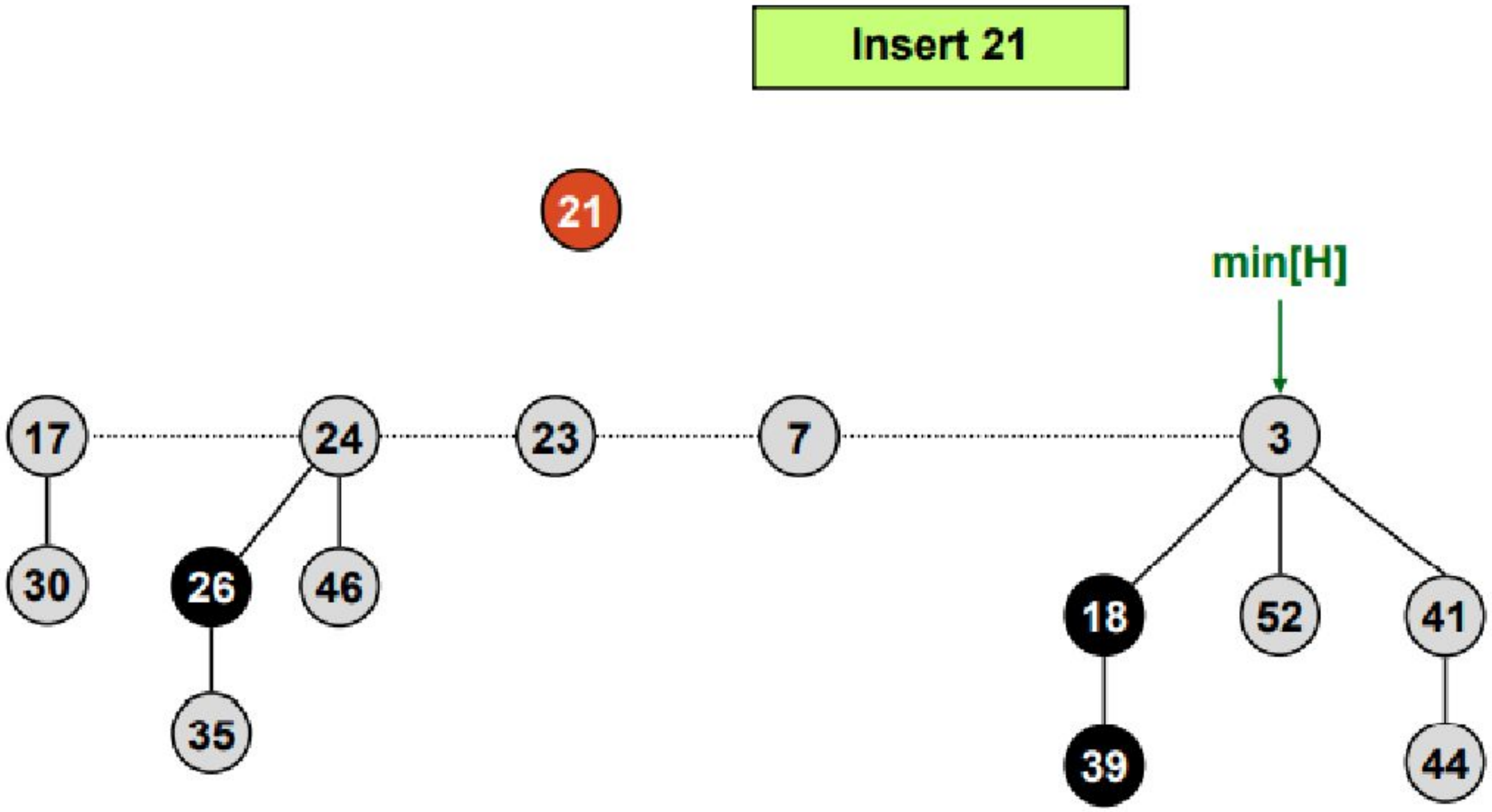

min[H]
23
7
3
17
24
18
52
38
30
26
46
39
41
35

Operations

- Create an empty Fibonacci heap
- Insert an element in a Fibonacci heap
- Merge two Fibonacci heaps (Union)
- Extract the minimum element from a Fibonacci heap
- Decrease the value of an element in a Fibonacci heap
- Delete an element from a Fibonacci heap

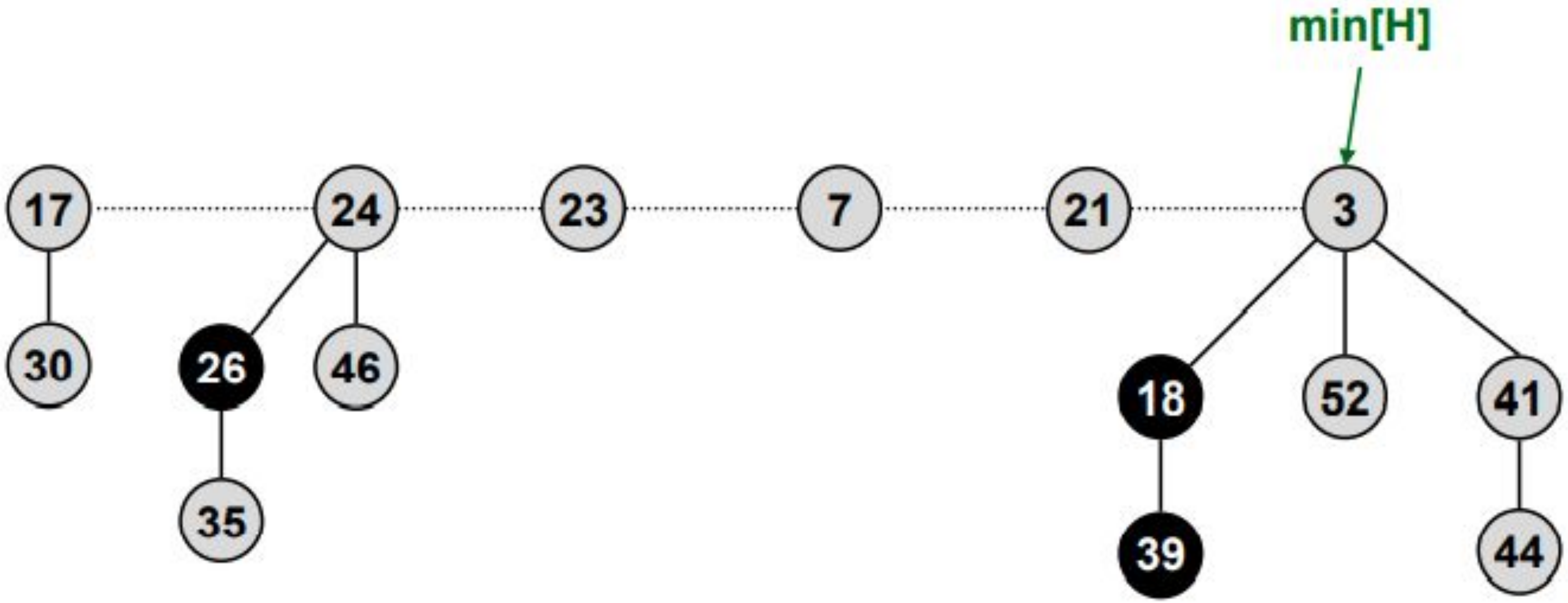
Inserting an Element

- Add the element to the left of min[H]
- Update min[H] if needed



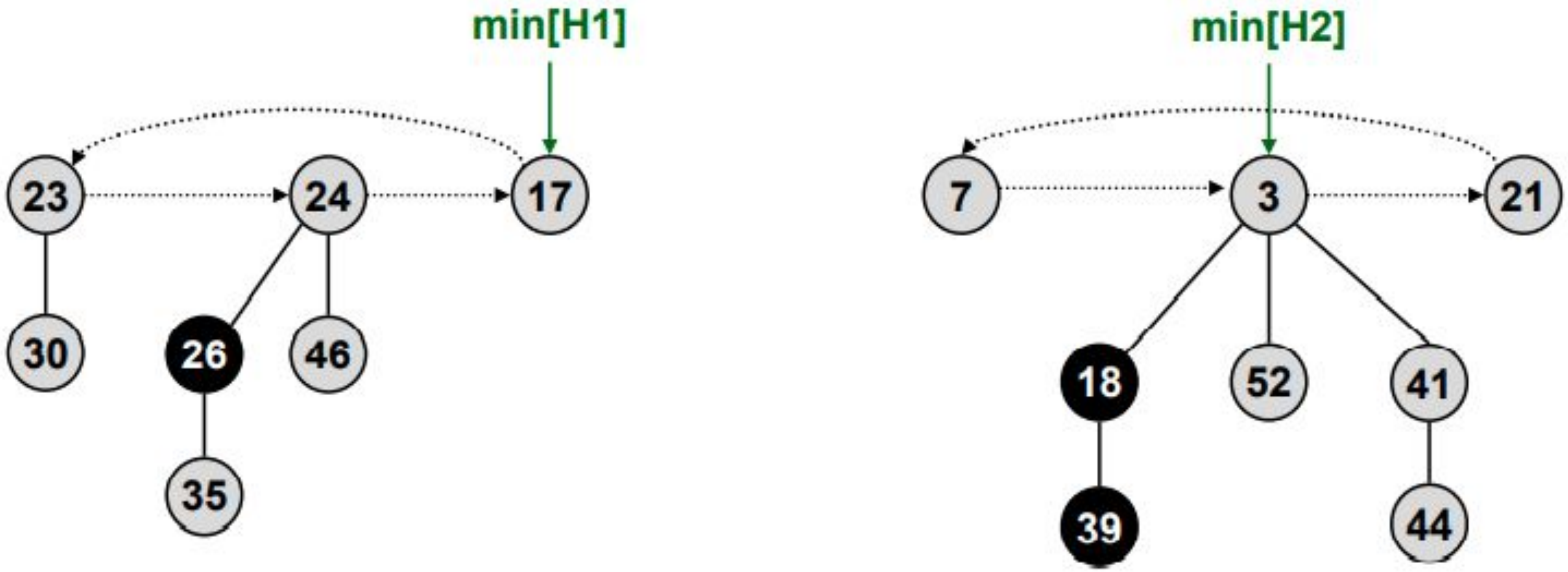
Inserting an Element (contd.)

- Add the element to the left of node pointed to by min[H]
- Update min[H] if needed



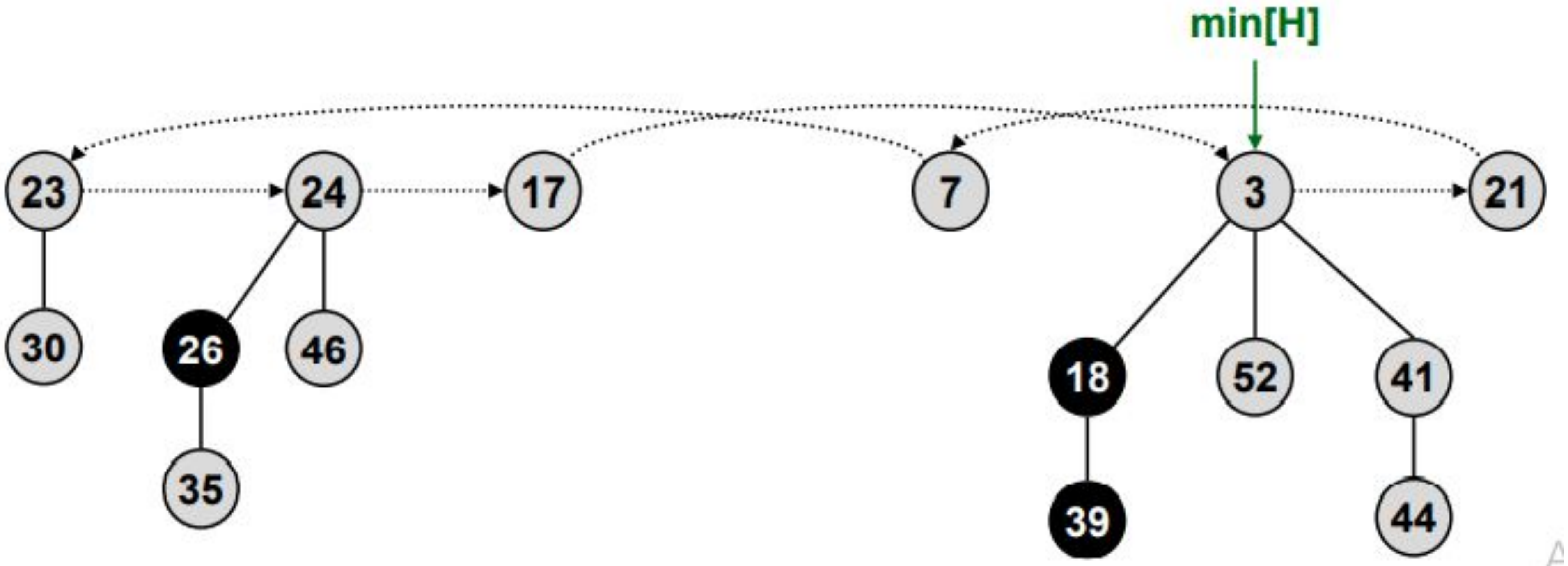
Merging Two Heaps (Union)

- Concatenate the root lists of the two Fibonacci heaps
- Root lists are circular, doubly linked lists, so can be easily concatenated



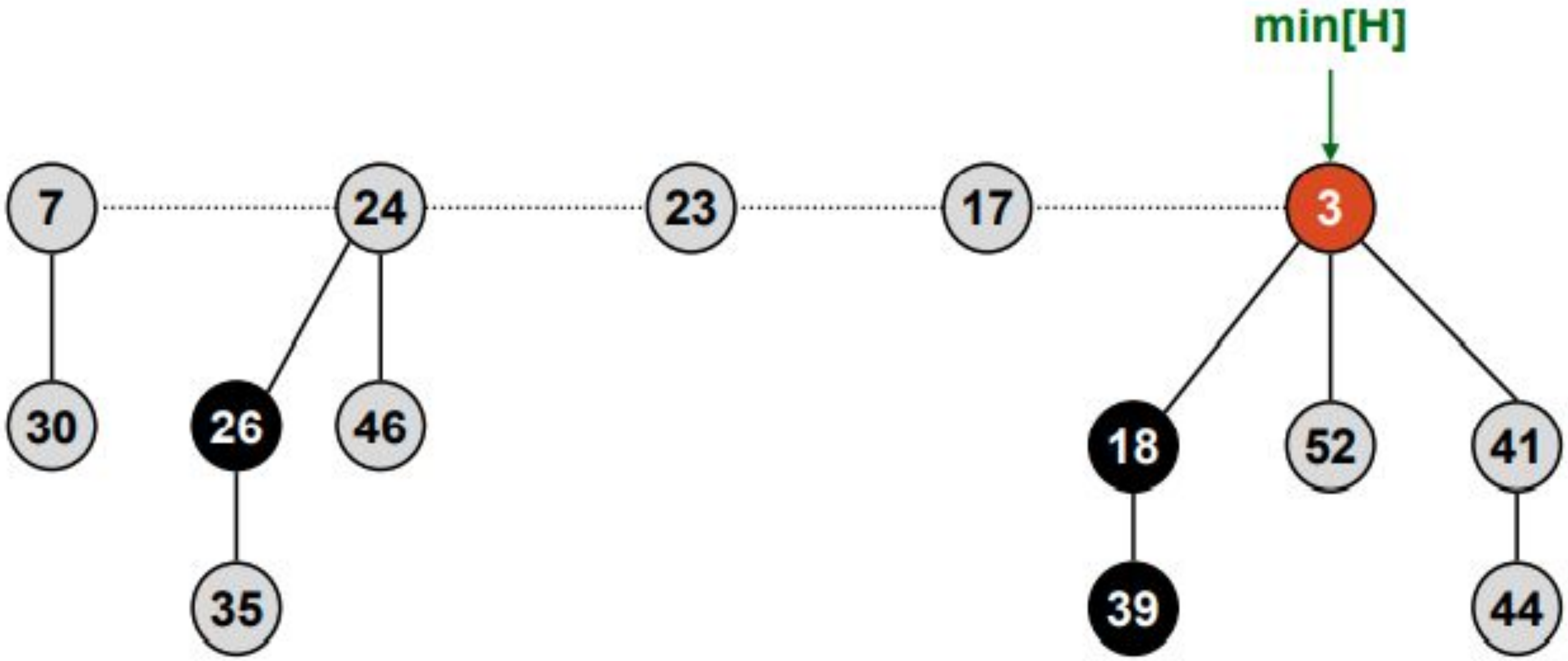
Merging Two Heaps (contd.)

- Concatenate the root lists of the two Fibonacci heaps
- Root lists are circular, doubly linked lists, so can be easily concatenated



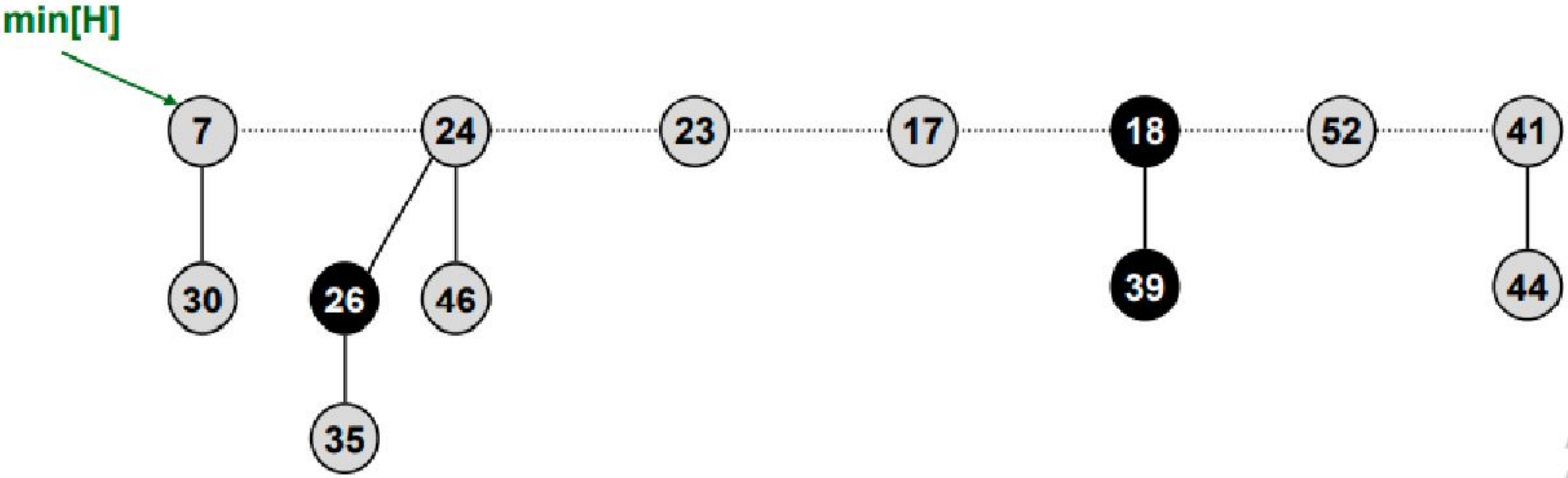
Extracting the Minimum Element

- **Step 1:**
  - Delete the node pointed to by min[H]
  - Concatenate the deleted node's children into root list



Extracting the Minimum (contd.)

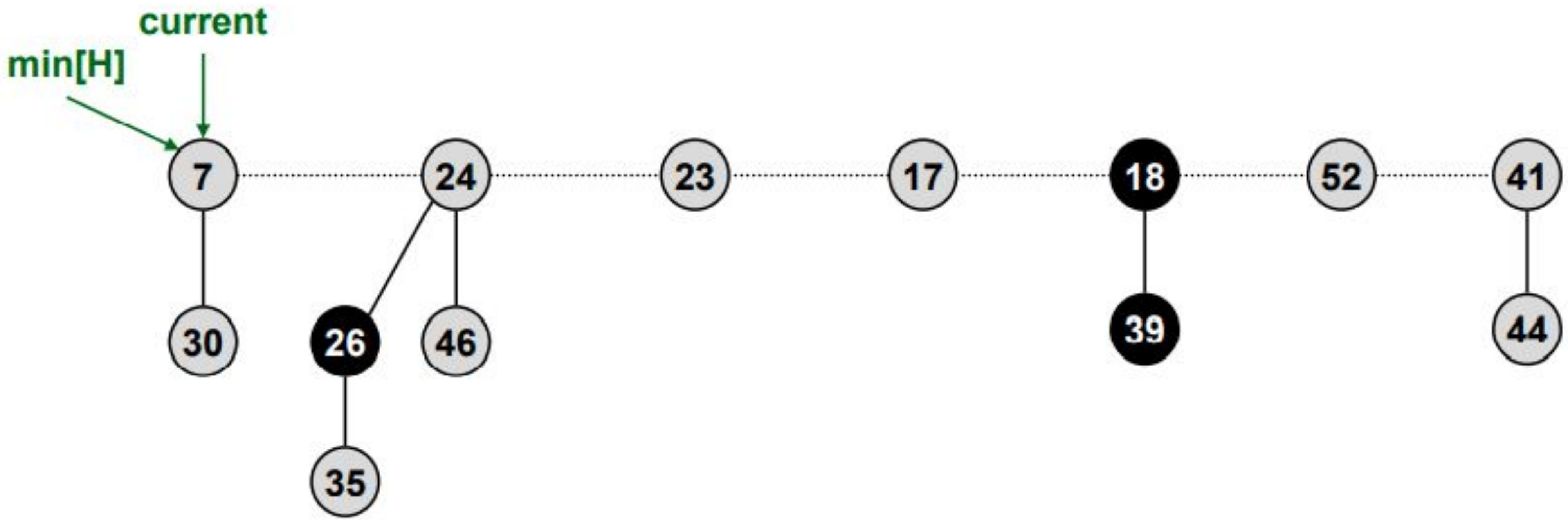
- **Step 1:**
  - Delete the node pointed to by min[H]
  - Concatenate the deleted node's children into root list



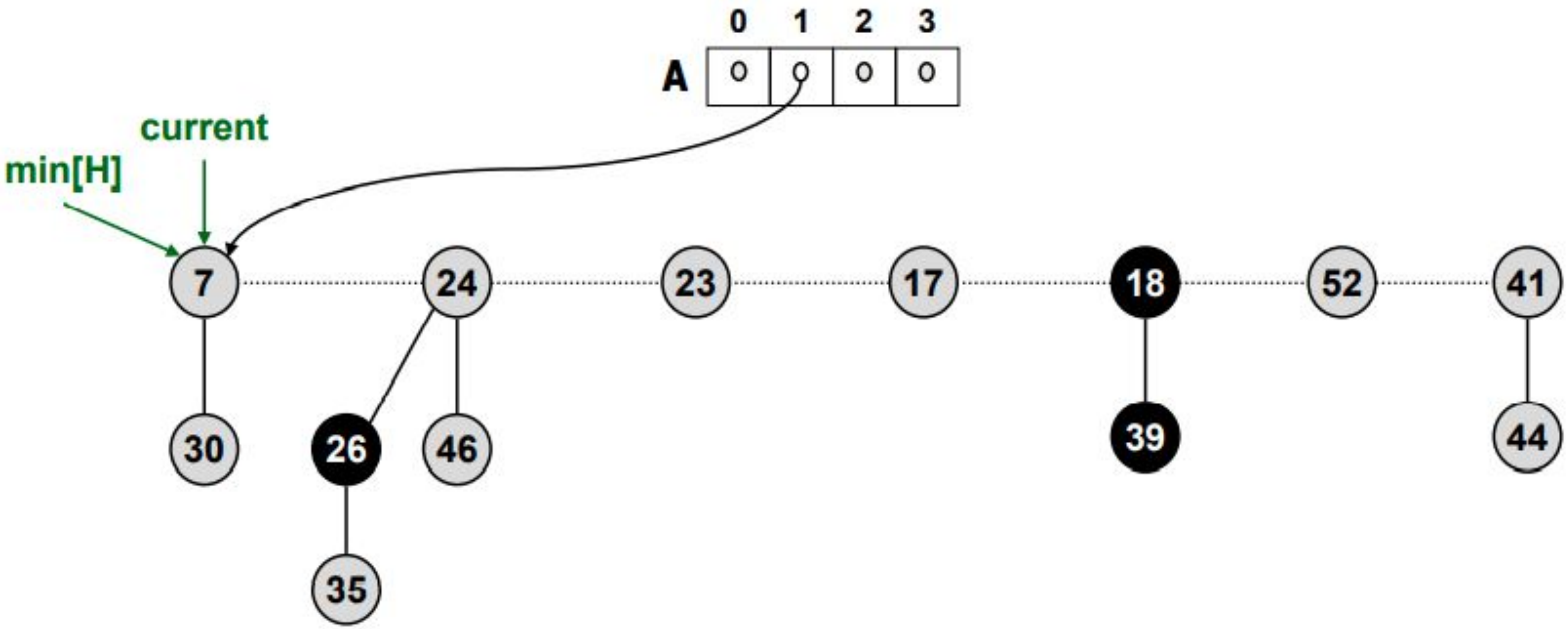
- **Step 2:** Consolidate trees so that no two roots have same degree
  - Traverse the roots from min towards right
  - Find two roots x and y with the same degree, with key[x] ≤ key[y]
  - Remove y from root list and make y a child of x
  - Increment degree[x]
  - Unmark y if marked
- We use an array A[0..D(n)] where D(n) is the maximum degree of any node in the heap with n nodes, initially all NIL
  - If A[k] = y at any time, then degree[y] = k

Extracting the Minimum (contd.)

- **Step 2:** Consolidate trees so that no two roots have same degree. Update min[H] with the new min after consolidation.

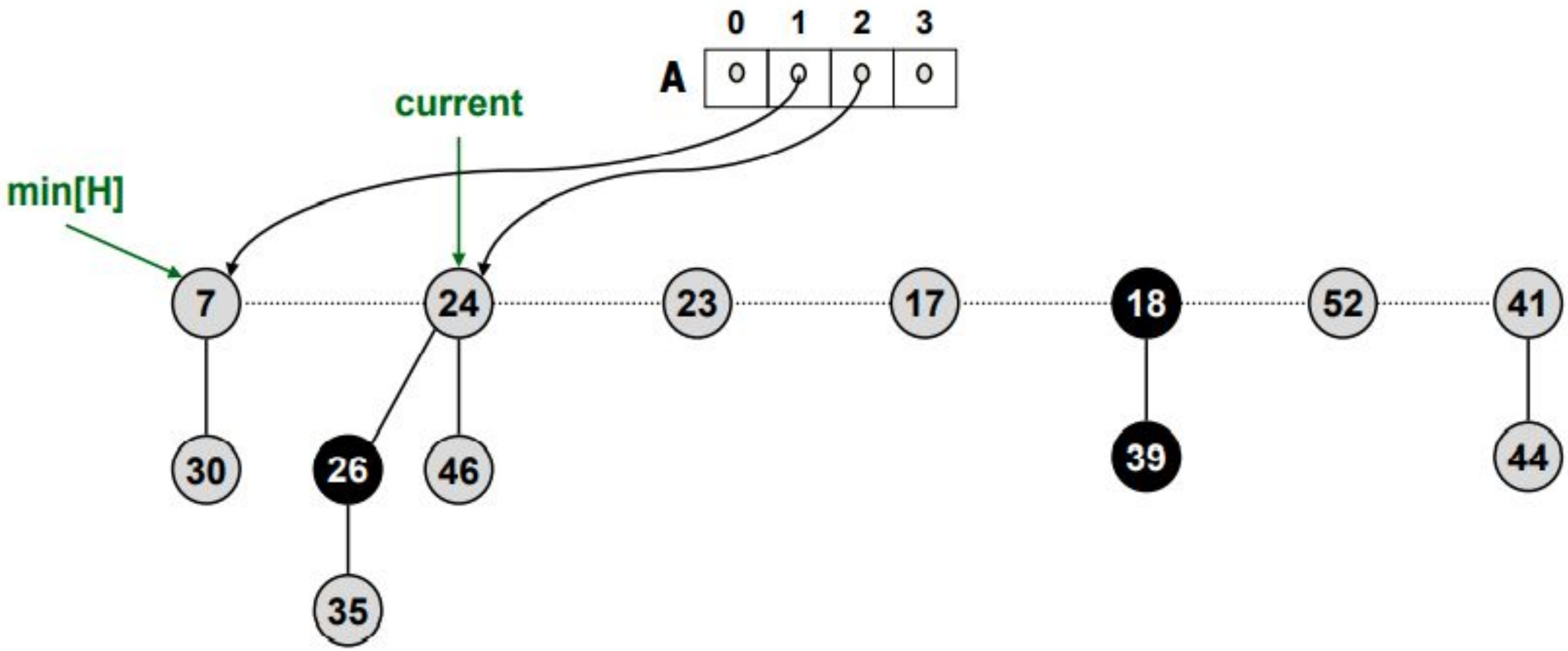


Extracting the Minimum (contd.)

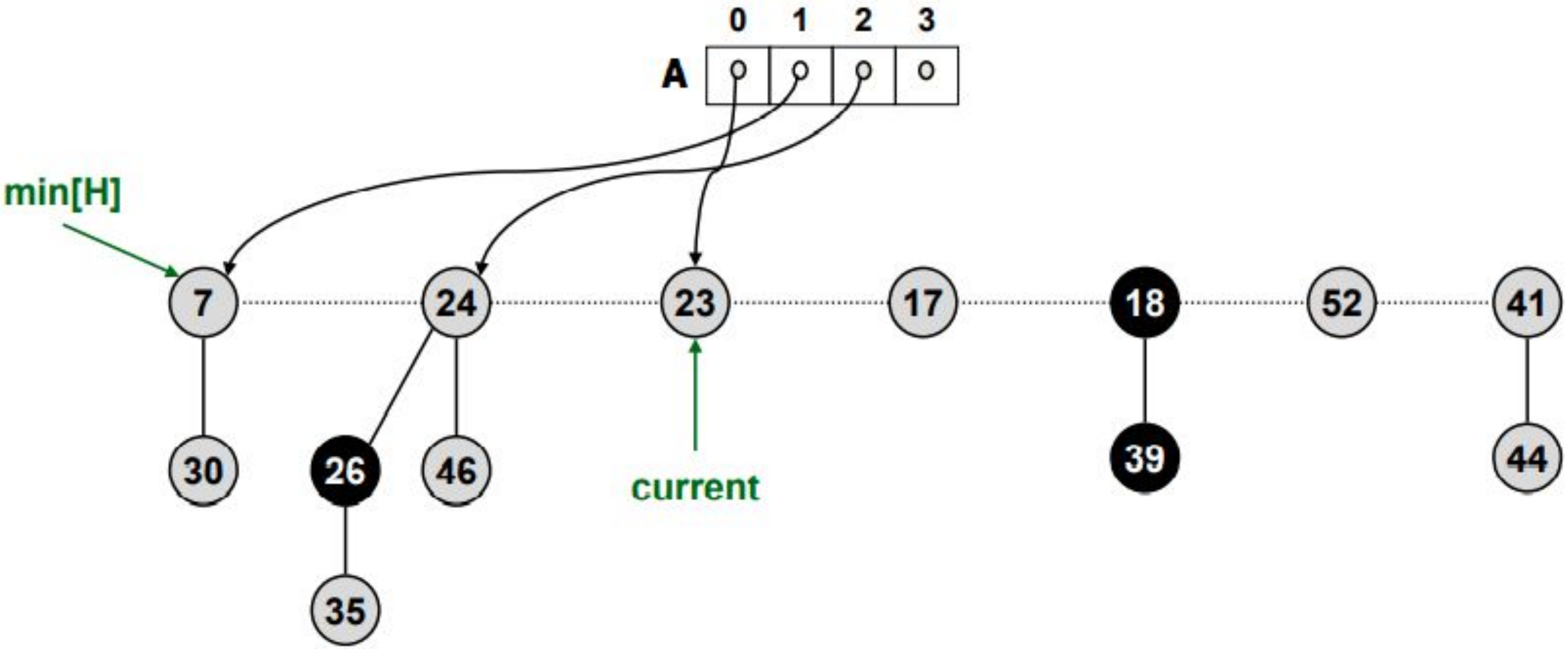


Act

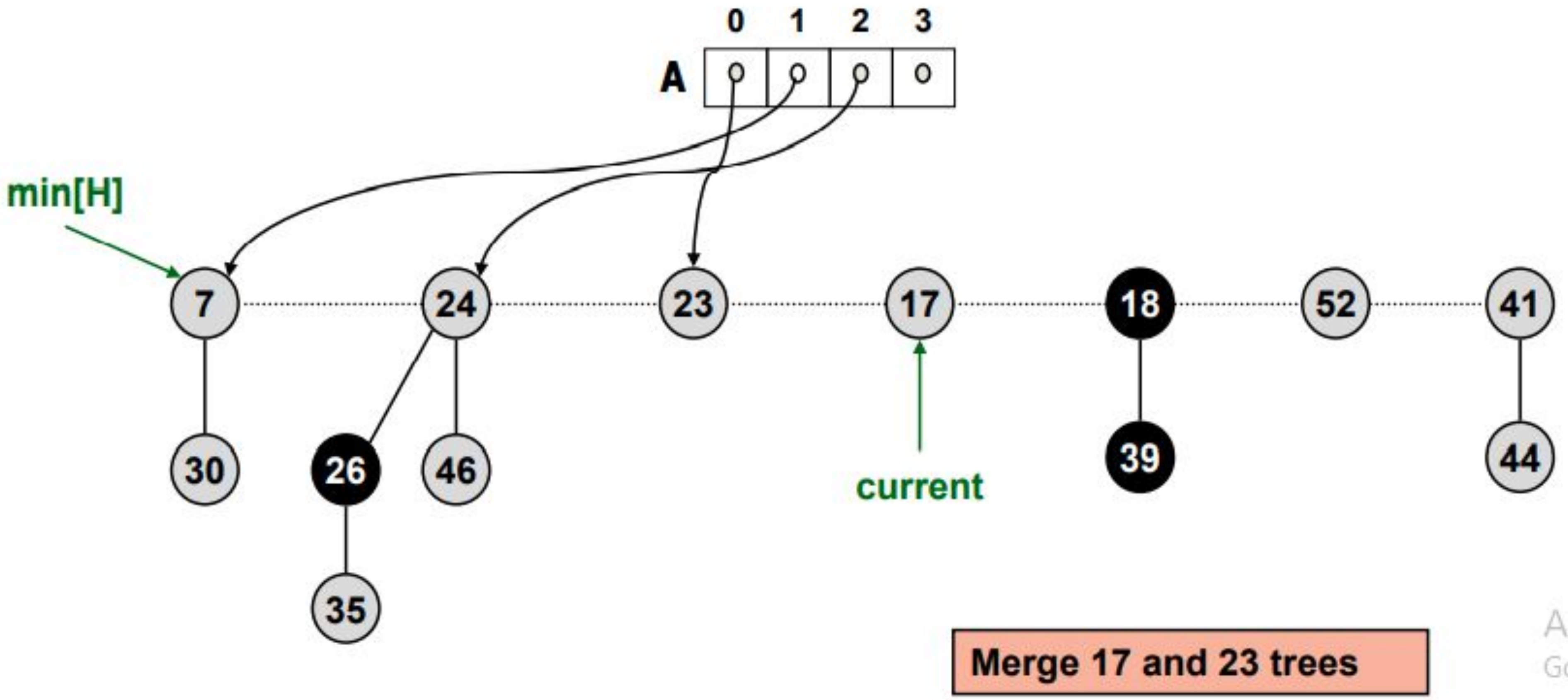
Extracting the Minimum (contd.)



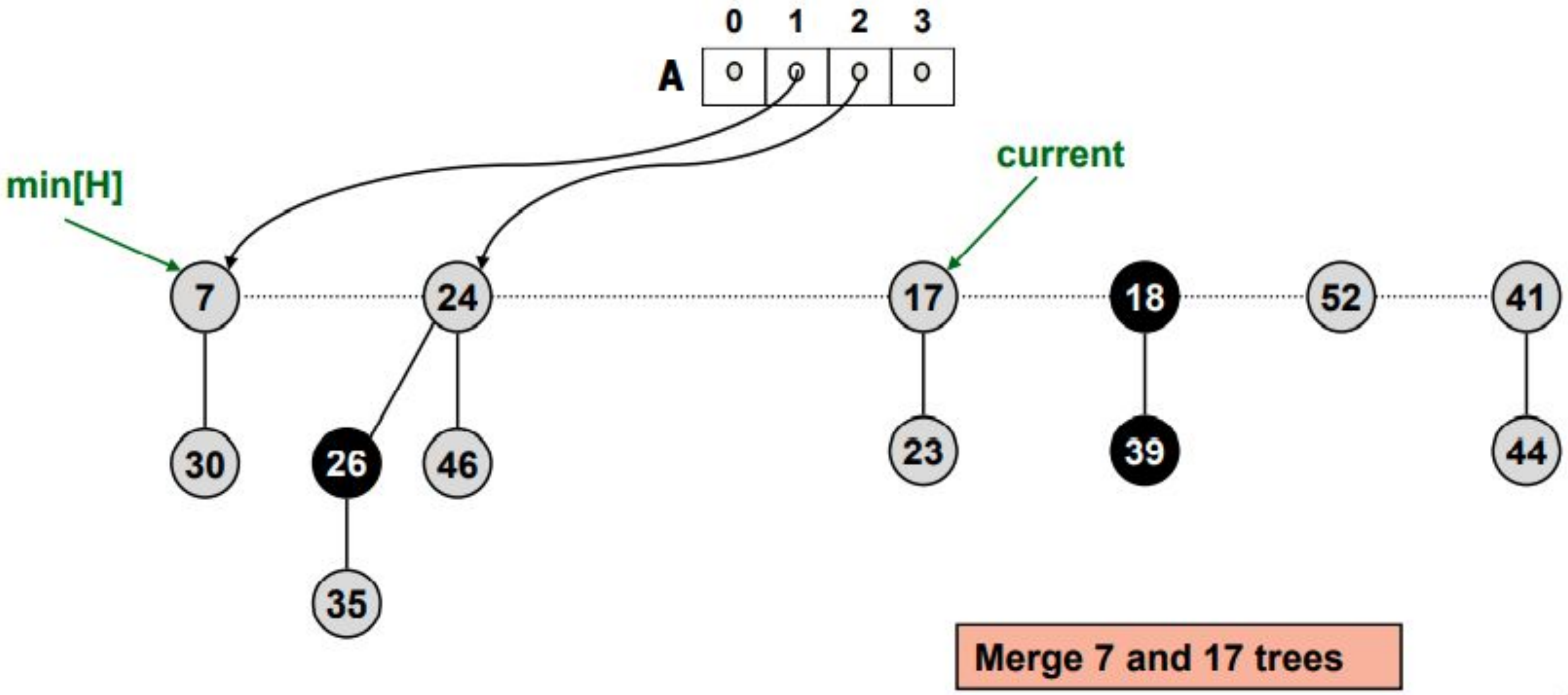
Extracting the Minimum (contd.)



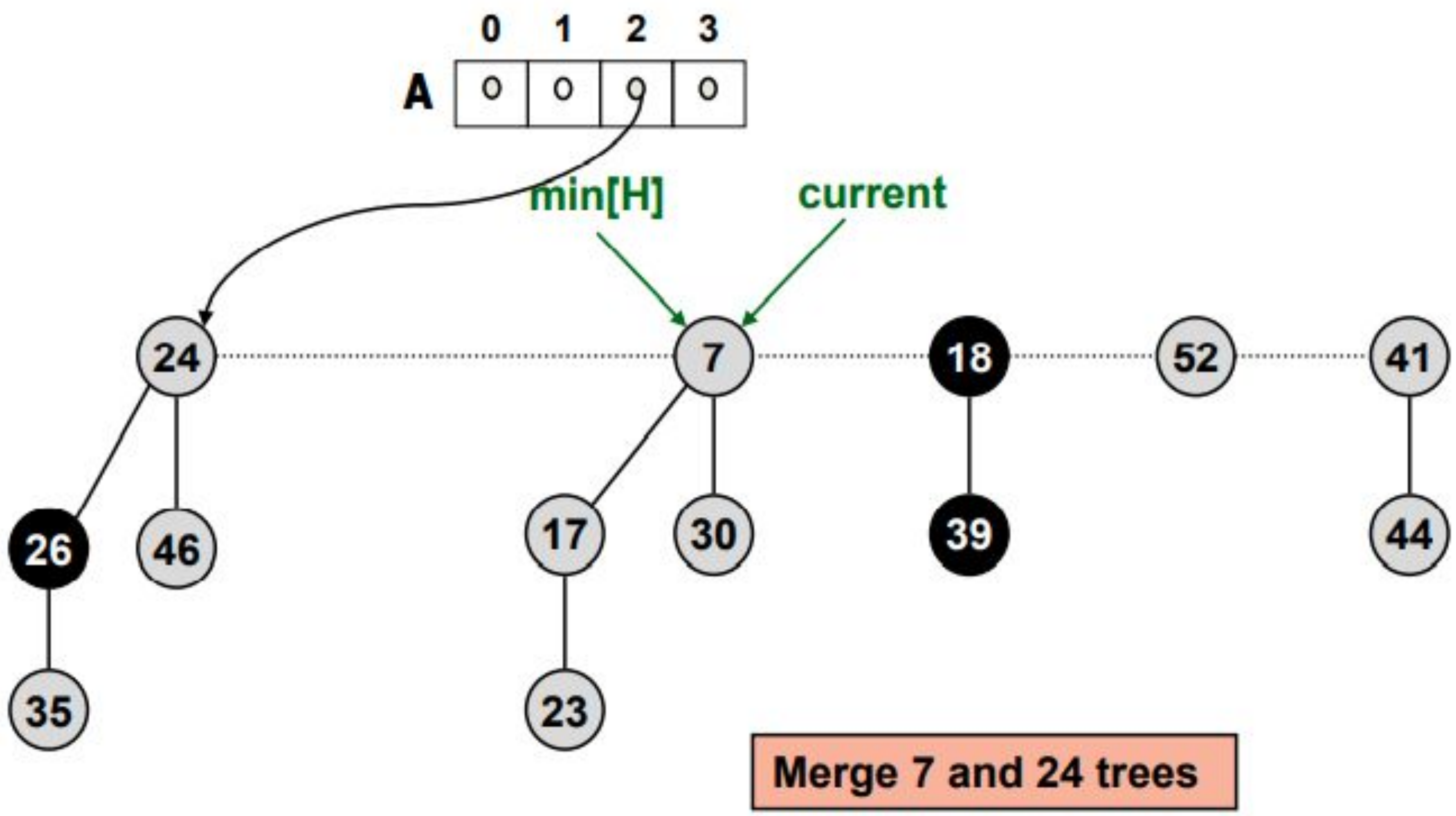
Extracting the Minimum (contd.)



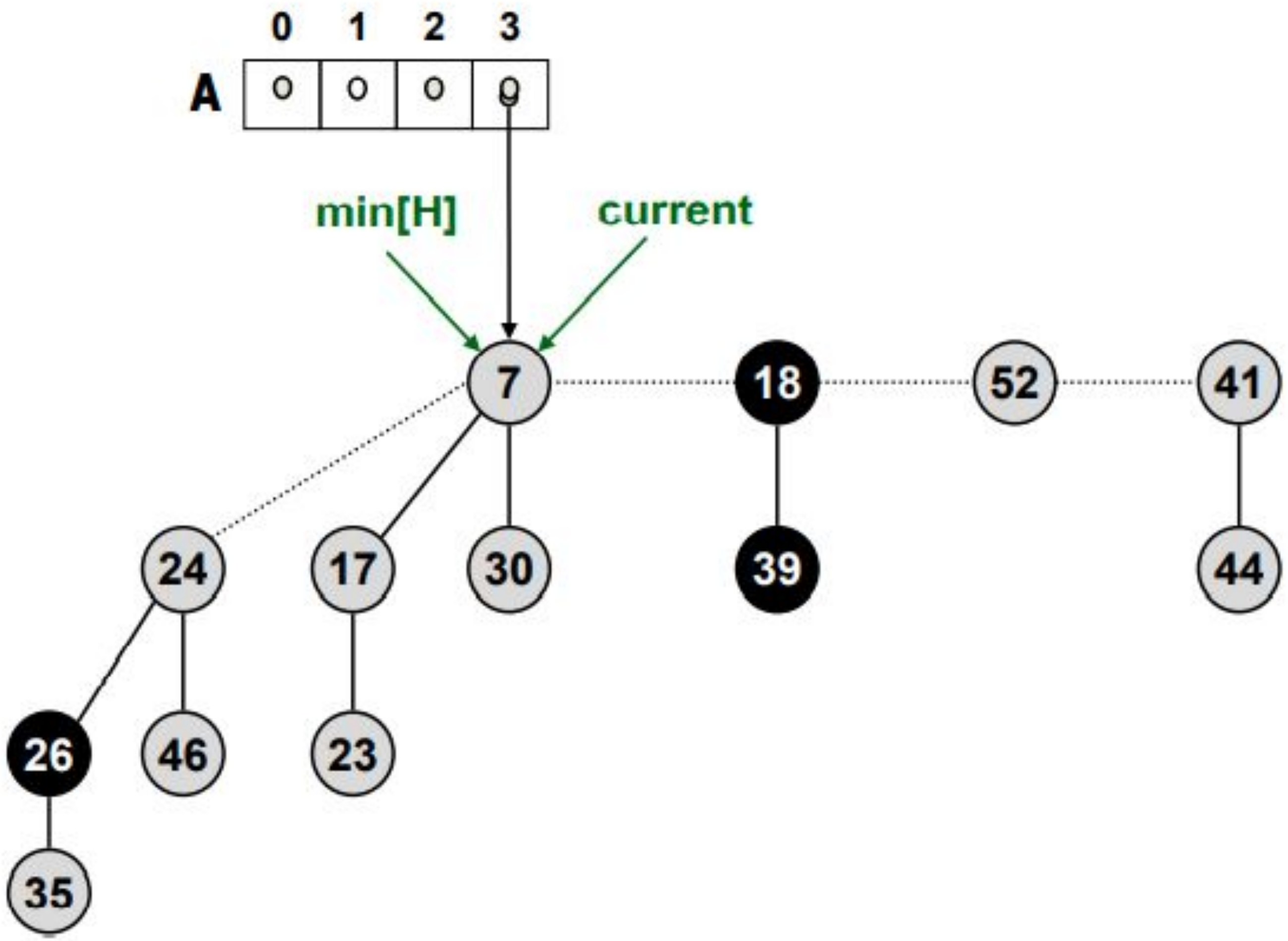
Extracting the Minimum (contd.)



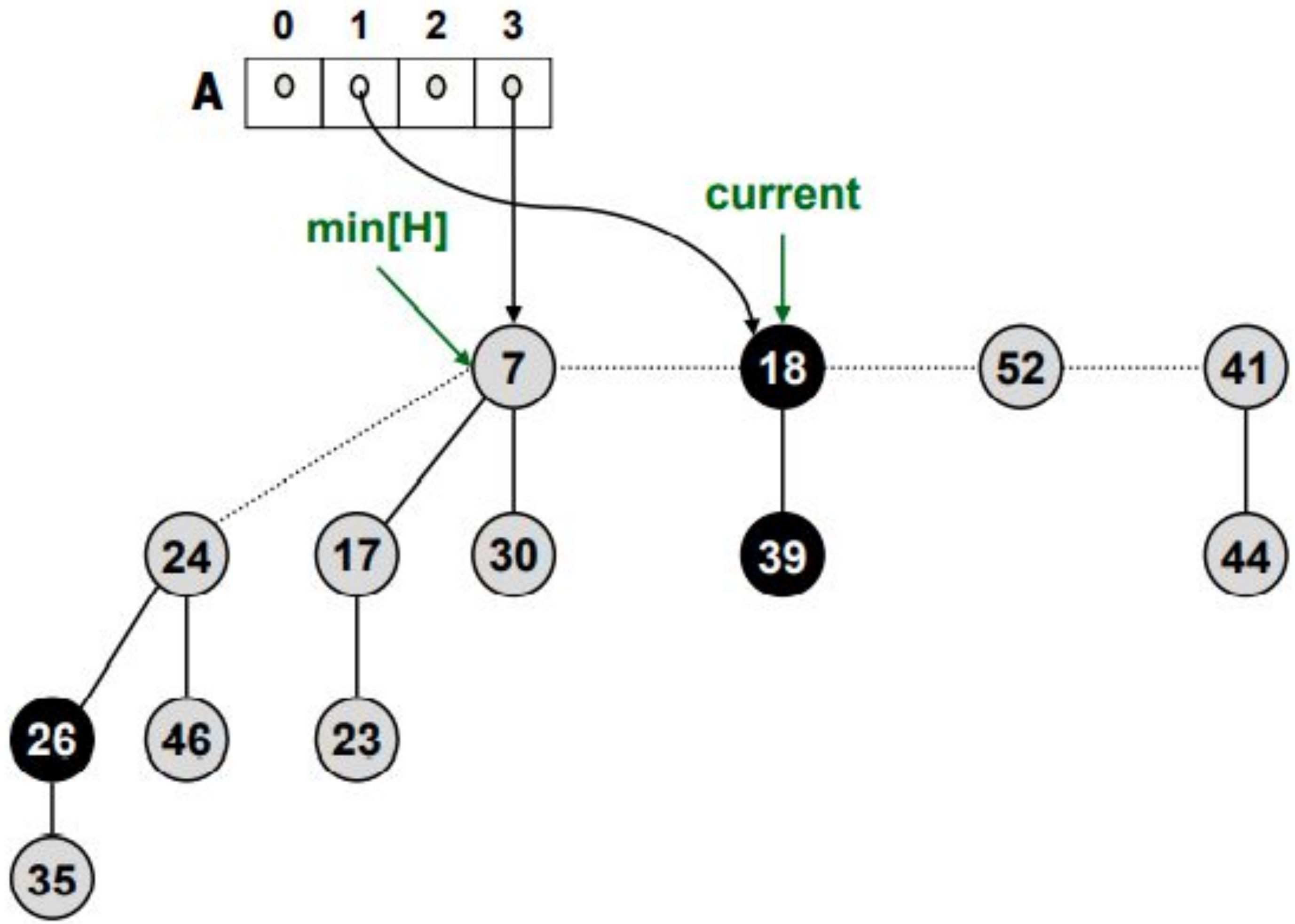
Extracting the Minimum (contd.)



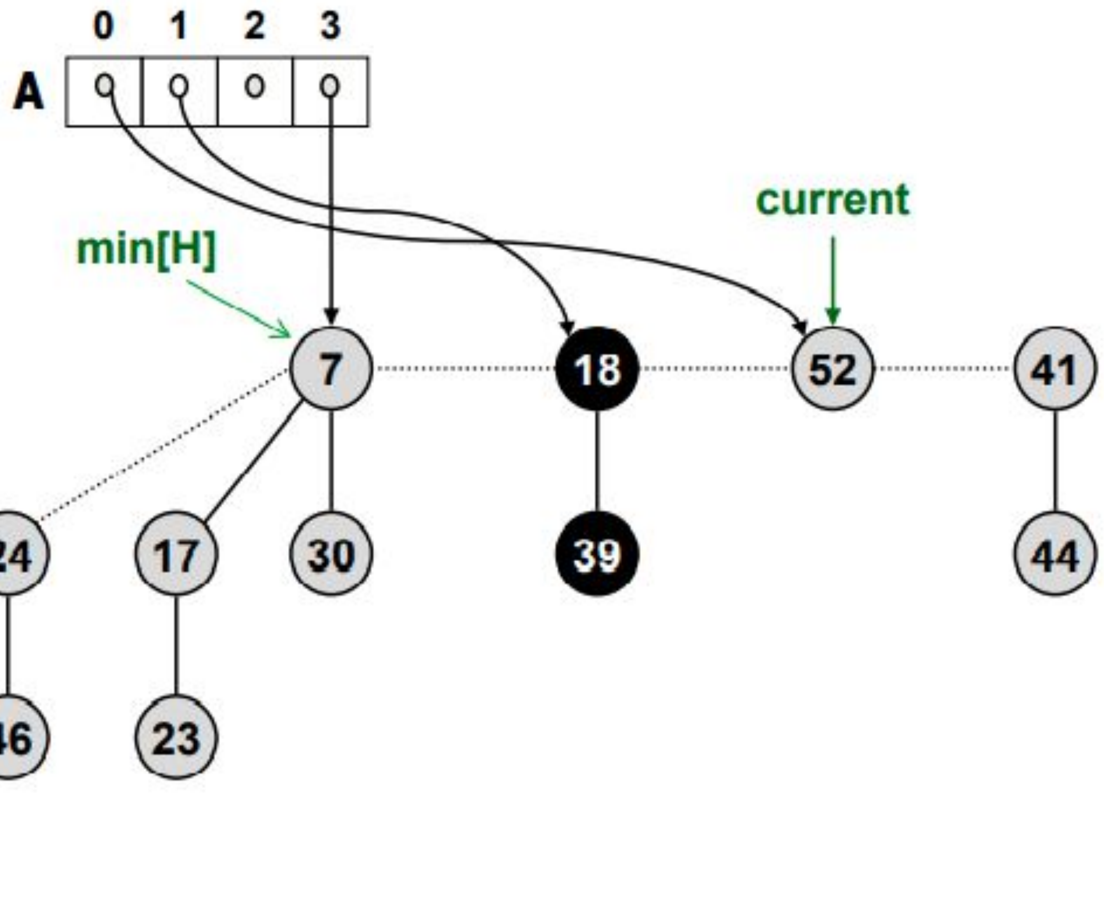
Extracting the Minimum (contd.)



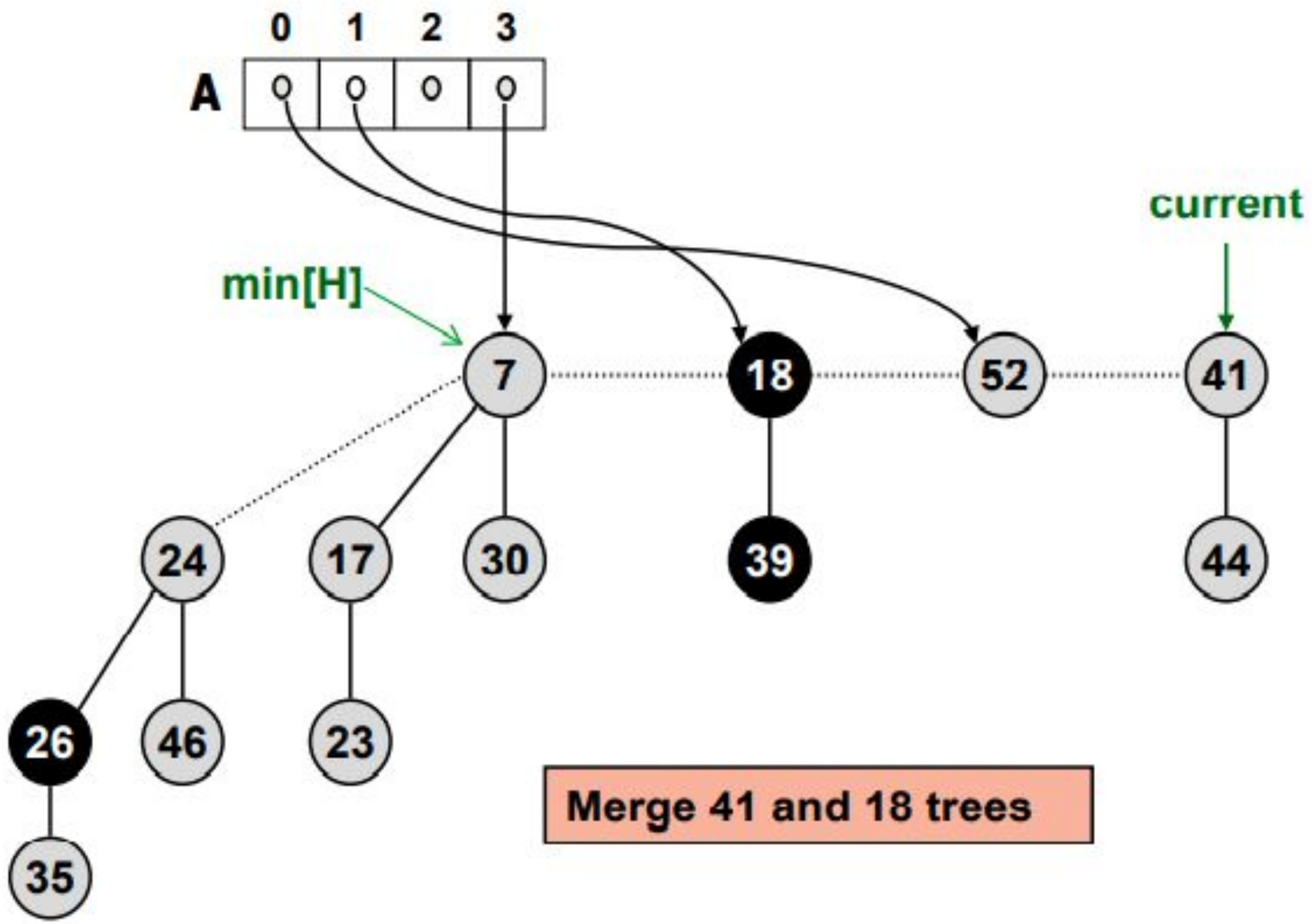
Extracting the Minimum (contd.)



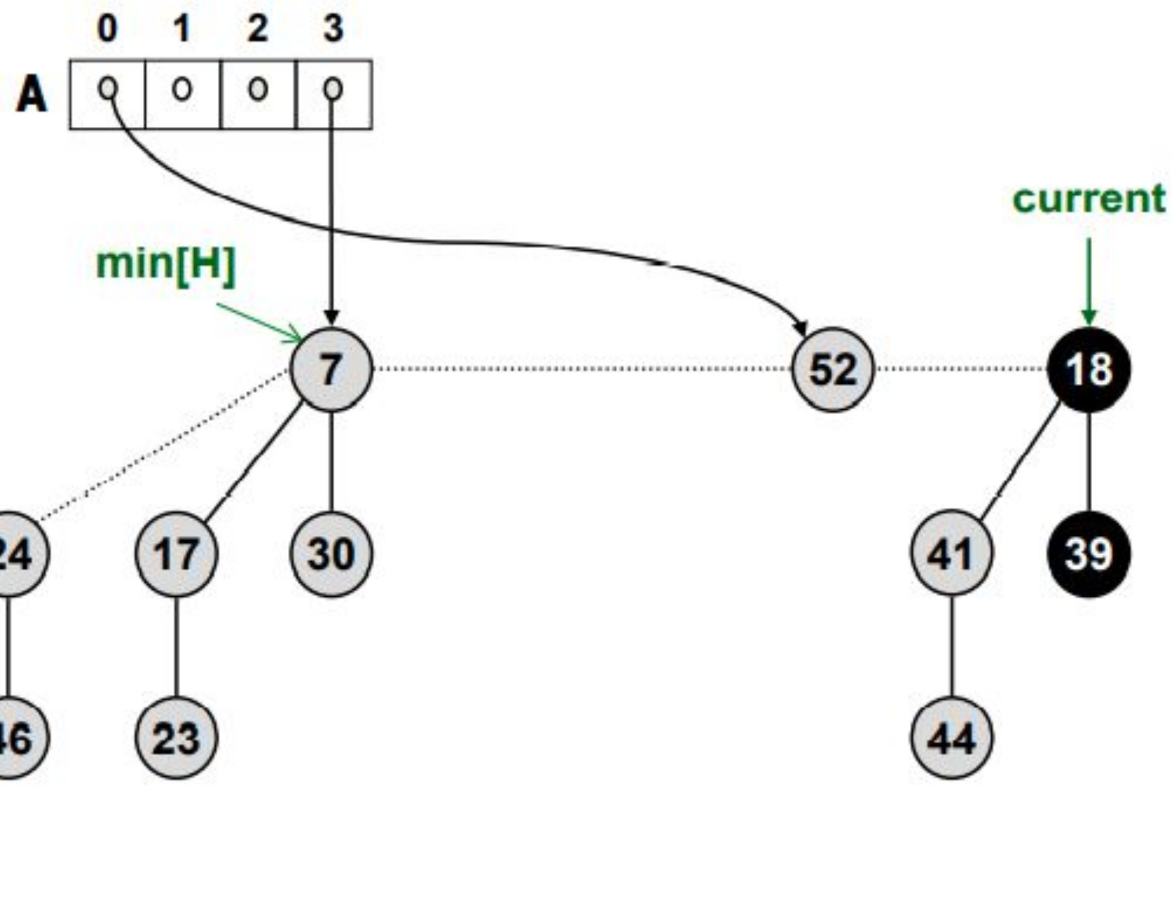
Extracting the Minimum (contd.)



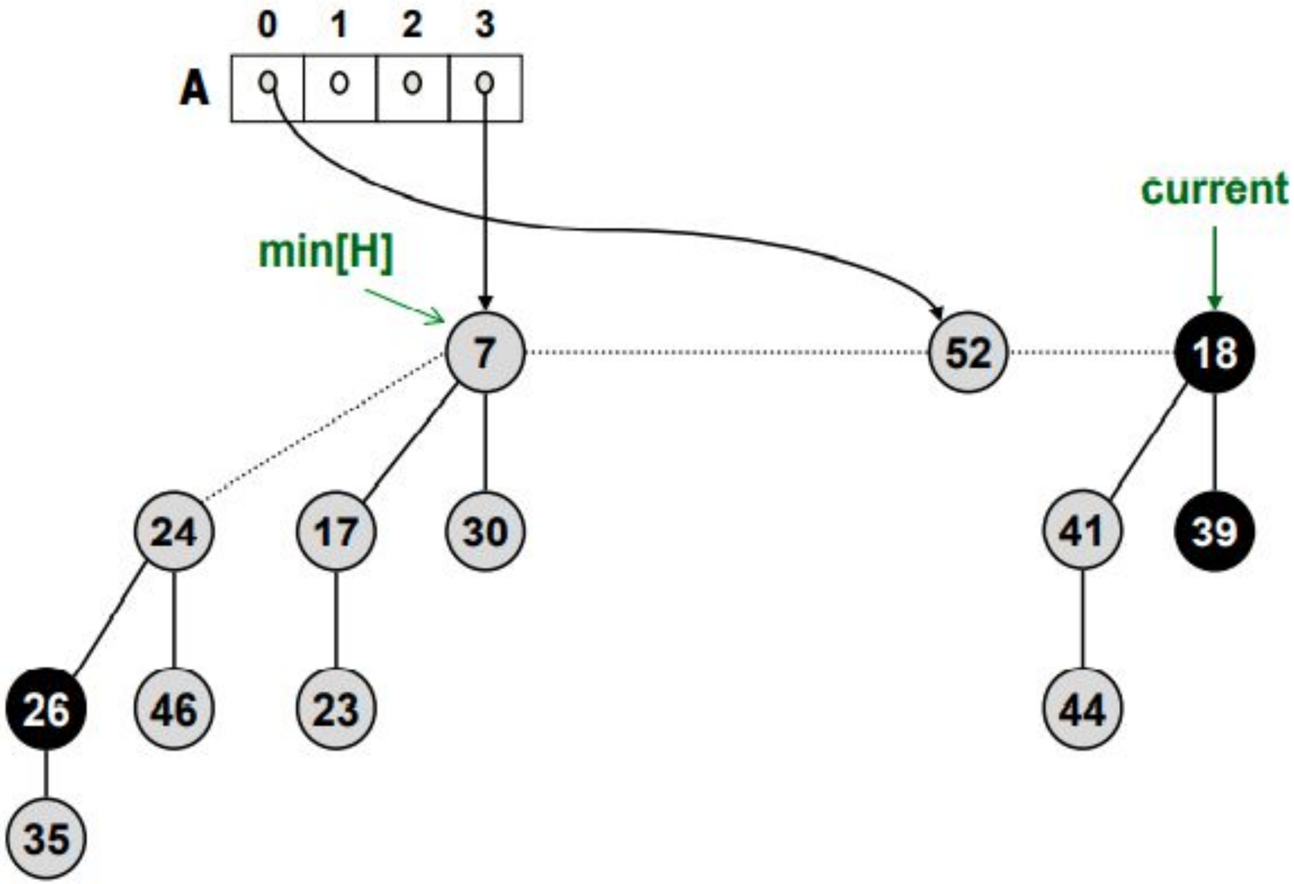
Extracting the Minimum (contd.)



Extracting the Minimum (contd.)

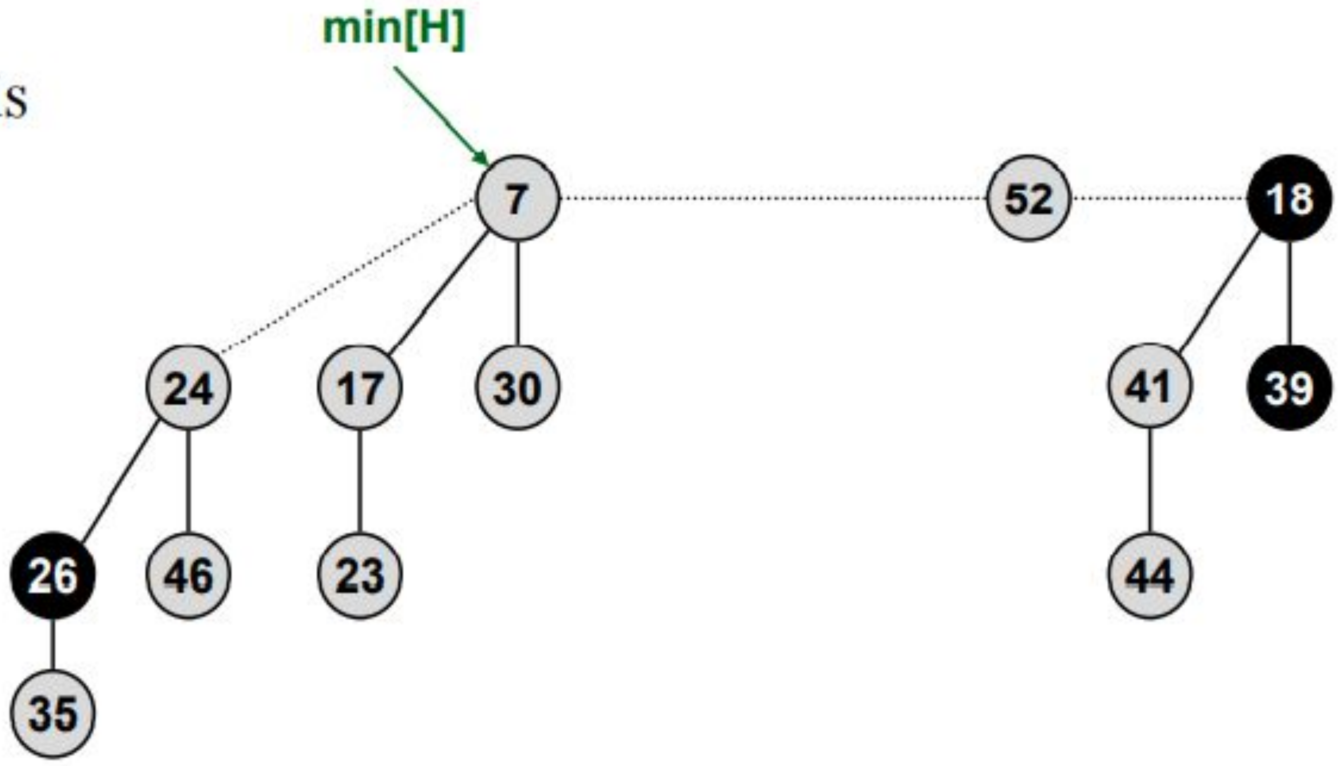


Extracting the Minimum (contd.)



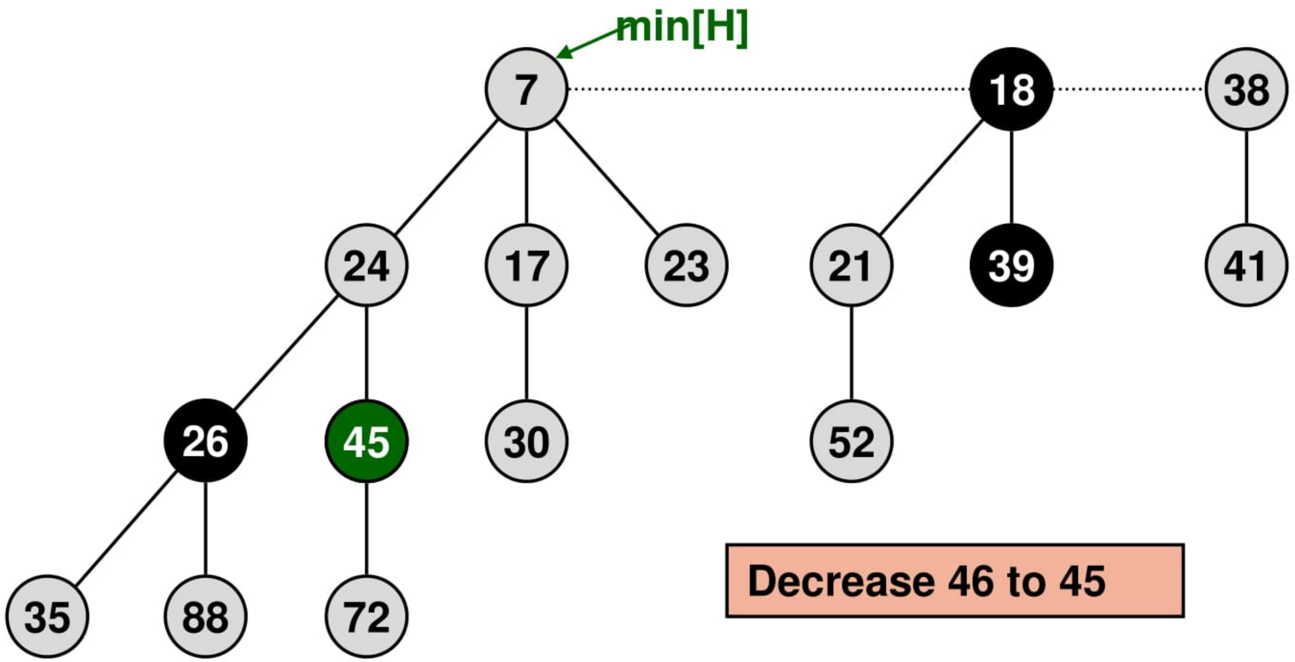
Extracting the Minimum (contd.)

- All roots covered by current pointer, so done
- Now find the minimum among the roots and make min[H] point to it (already pointing to minimum in this example)
- Final heap is

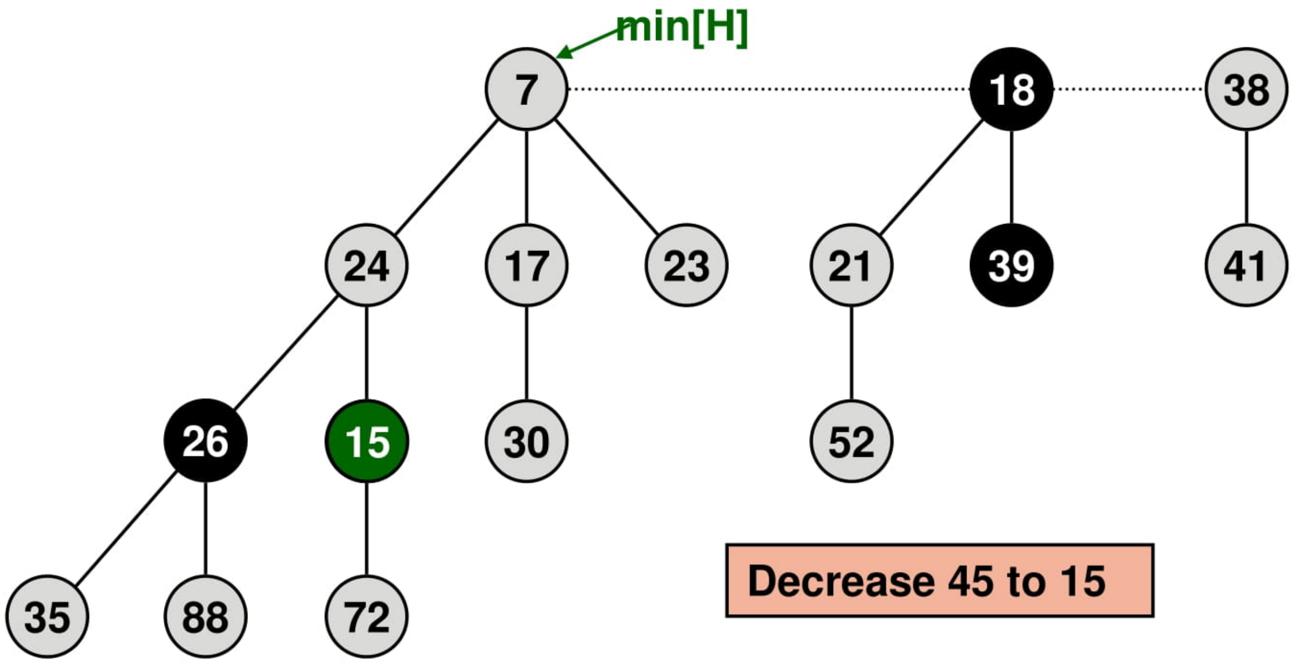


Decrease Key

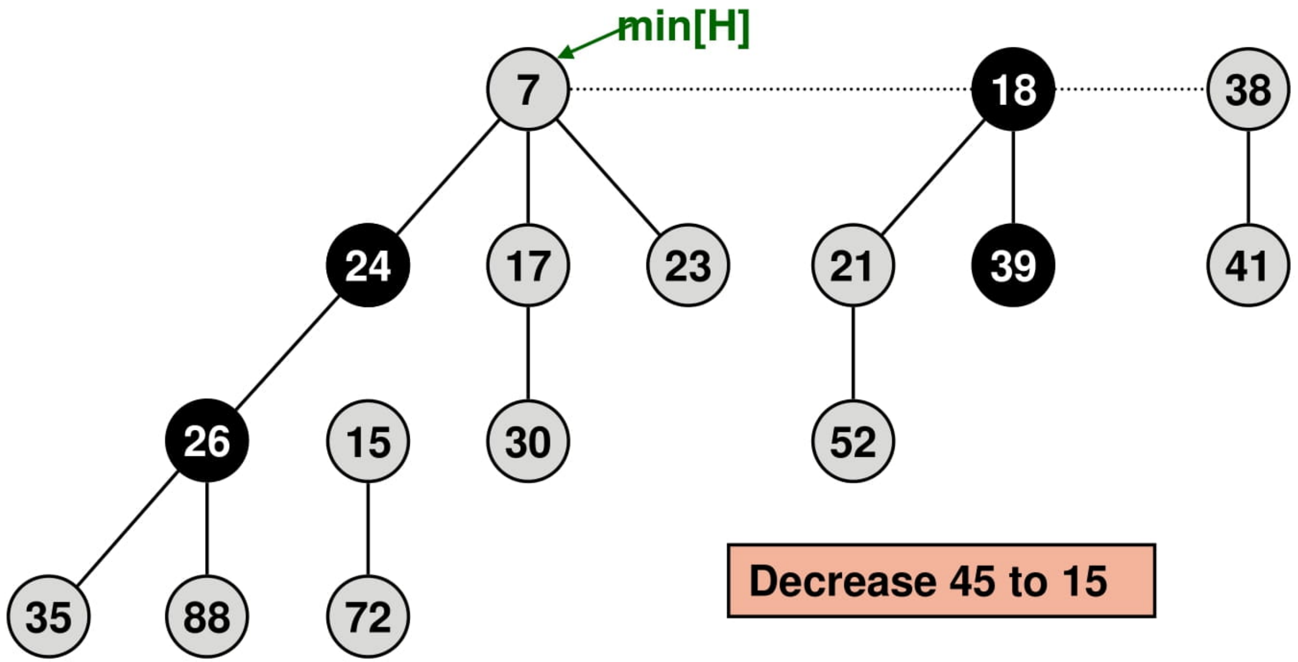
- Decrease key of element x to k
- Case 0: min-heap property not violated
  - decrease key of x to k
  - change heap min pointer if necessary



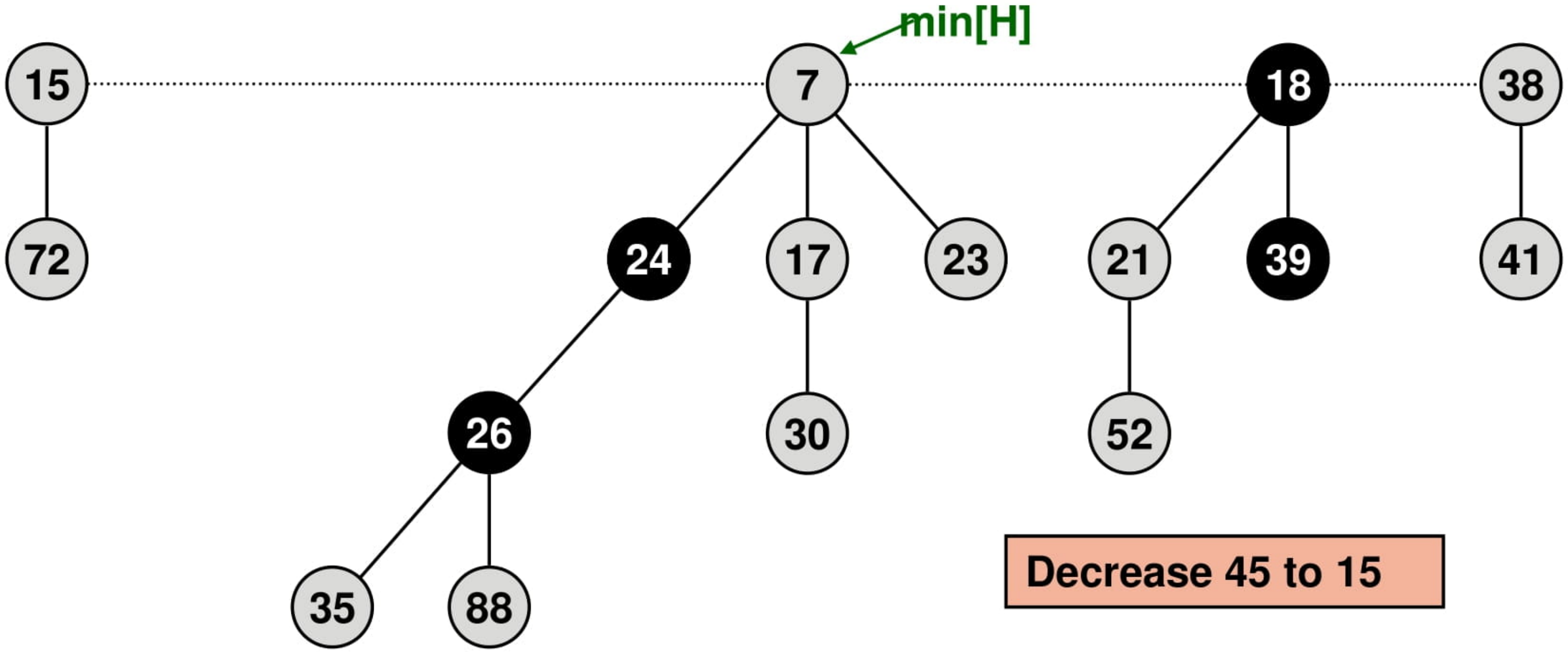
- Case 1: parent of x is unmarked
  - decrease key of x to k
  - cut off link between x and its parent, unmark x if marked
  - mark parent
  - add tree rooted at x to root list, updating heap min pointer



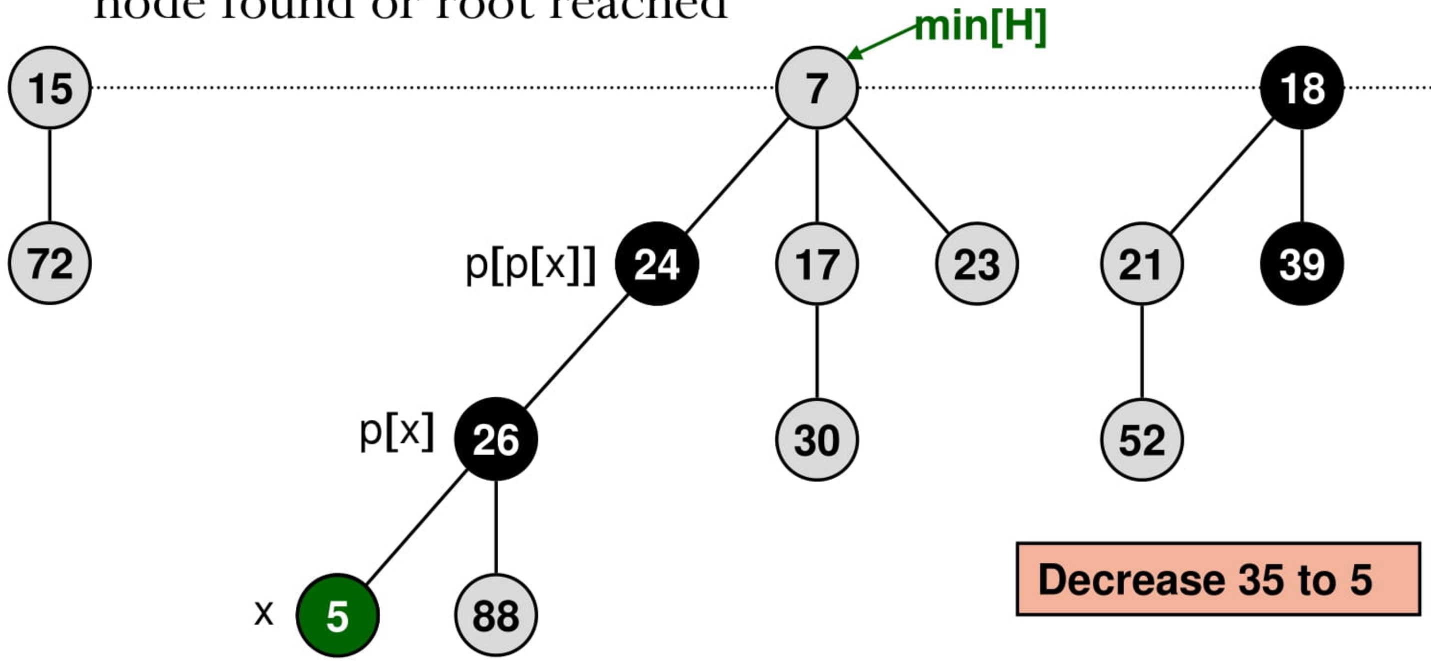
- Case 1: parent of x is unmarked
  - decrease key of x to k
  - cut off link between x and its parent, unmark x if marked
  - mark parent
  - add tree rooted at x to root list, updating heap min pointer



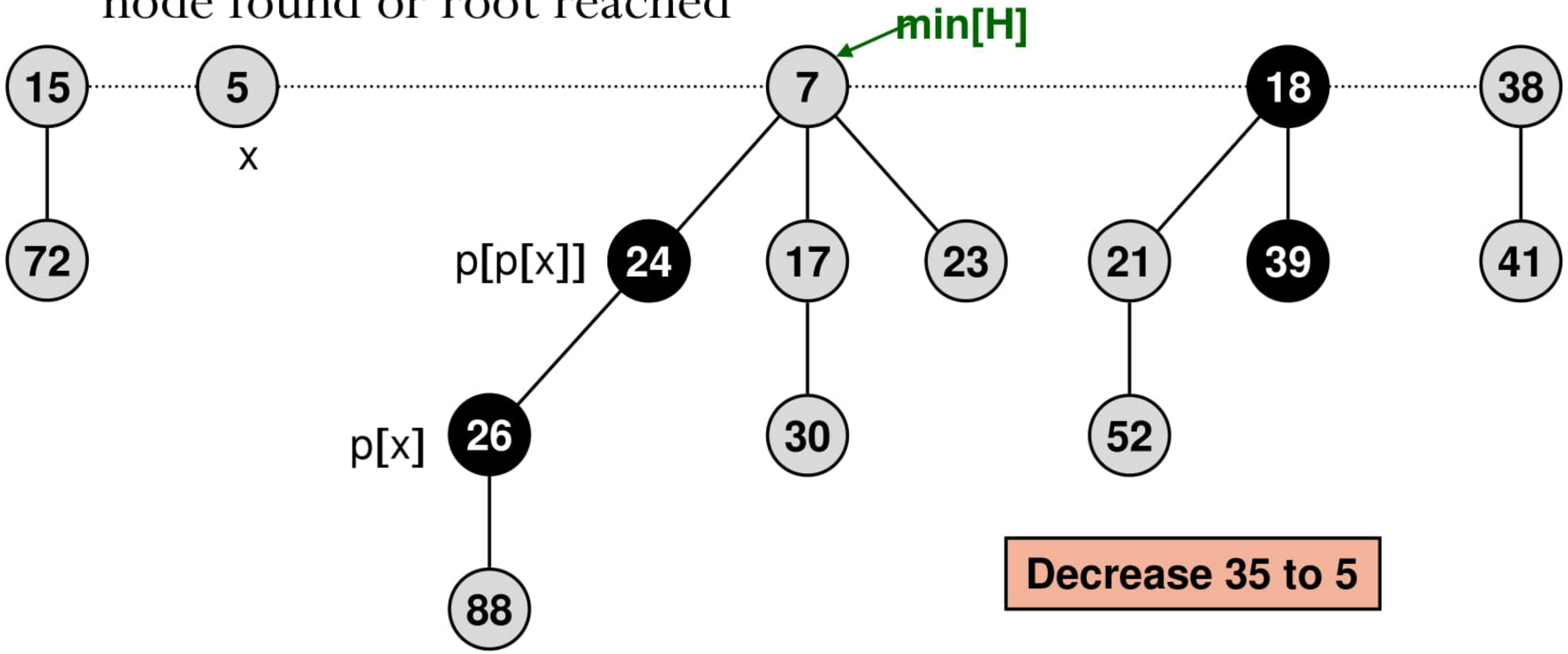
- Case 1: parent of x is unmarked
  - decrease key of x to k
  - cut off link between x and its parent, unmark x if marked
  - mark parent
  - add tree rooted at x to root list, updating heap min pointer



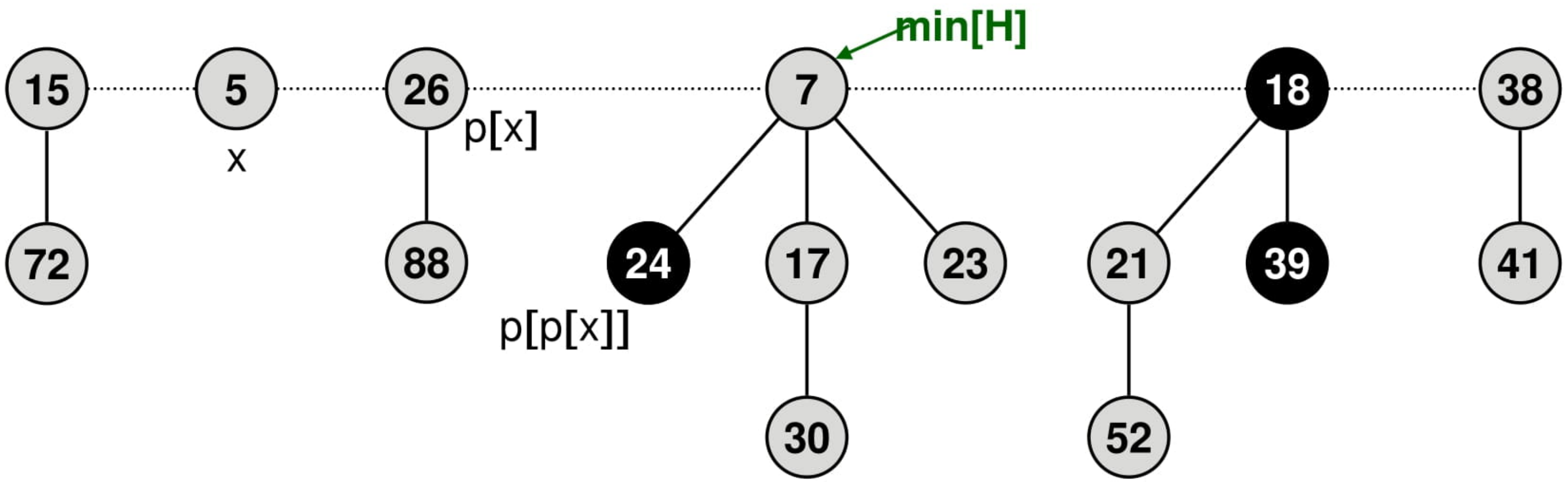
- Case 2: parent of x is marked
  - decrease key of x to k
  - cut off link between x and its parent p[x], add x to root list, unmark x if marked
  - cut off link between p[x] and p[p[x]], add p[x] to root list, unmark p[x] if marked
    - If p[p[x]] unmarked, then mark it and stop
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat until unmarked node found or root reached



- Case 2: parent of x is marked
  - decrease key of x to k
  - cut off link between x and its parent p[x], add x to root list, unmark x if marked
  - cut off link between p[x] and p[p[x]], add p[x] to root list, unmark p[x] if marked
    - If p[p[x]] unmarked, then mark it and stop
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat until unmarked node found or root reached

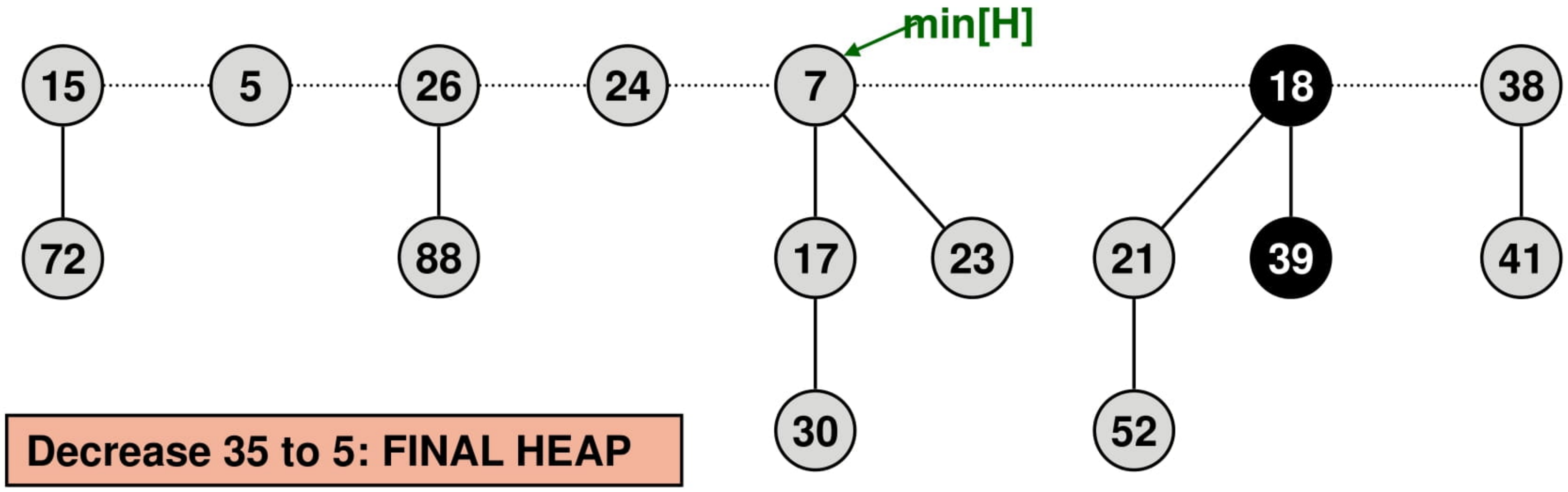


- Case 2: parent of x is marked
  - decrease key of x to k
  - cut off link between x and its parent p[x], add x to root list, unmark x if marked
  - cut off link between p[x] and p[p[x]], add p[x] to root list, unmark p[x] if marked
    - If p[p[x]] unmarked, then mark it and stop
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat until unmarked node found or root reached



Decrease 35 to 5

- Case 2: parent of x is marked
  - decrease key of x to k
  - cut off link between x and its parent p[x], add x to root list, unmark x if marked
  - cut off link between p[x] and p[p[x]], add p[x] to root list, unmark p[x] if marked
    - If p[p[x]] unmarked, then mark it and stop
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat until unmarked node found or root reached (cascading cut)



Operations	Binomial Heap	Fibonacci Heap
Procedure	Worst-case	Amortized
Making Heap	$\Theta(1)$	$\Theta(1)$
Inserting a node	$O(\log(n))$	$\Theta(1)$
Finding Minimum key	$O(\log(n))$	$O(1)$
Extract-Minimum key	$\Theta(\log(n))$	$O(\log(n))$
Union or merging	$O(\log(n))$	$\Theta(1)$
Decreasing a Key	$\Theta(\log(n))$	$\Theta(1)$
Deleting a node	$\Theta(\log(n))$	$O(\log(n))$