Design and Analysis of Algorithms

Unit-2

Binomial Heap

Binomial Tree

Definition

The binomial tree B_k is an ordered tree defined recursively.

1. The binomial tree B_0 consists of a single node.
2. The binomial tree B_k consists of two binomial trees B_{k-1} i.e.

$$B_k = B_{k-1} + B_{k-1}$$

They are linked together in the following way:

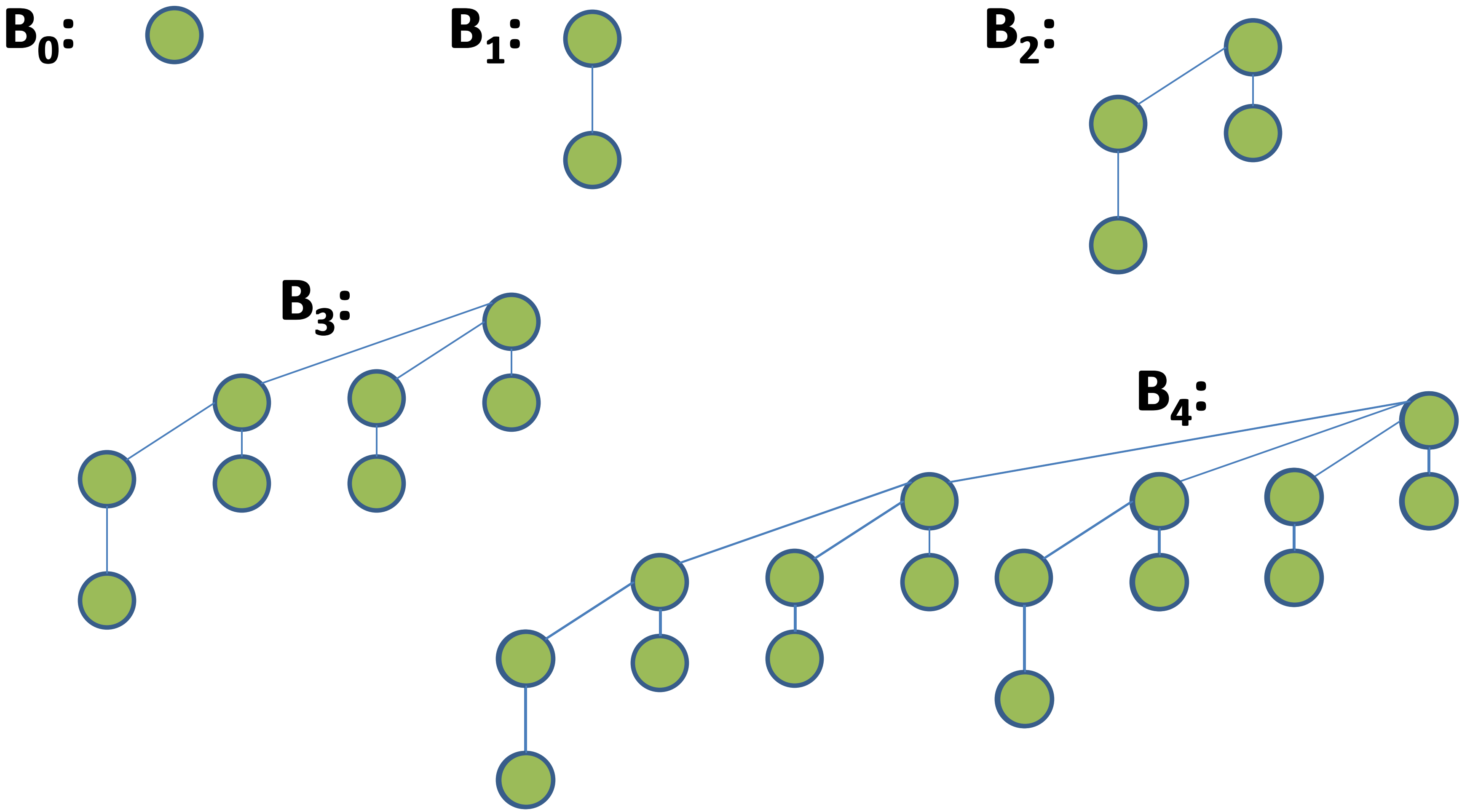
The root of one is the leftmost child of the root of the other.

Example: Some binomial trees are the following:-

B0:

Binomial Tree

Example: Some binomial trees are the following:-



Properties of binomial trees

For the binomial tree B_k,

1. There are 2^k nodes.
2. The height of the tree is k.
3. There are exactly $\binom{k}{i}$ nodes at depth i for i = 0, 1, ..., k.
4. The root has degree k, which is greater than that of any other node. Moreover if the children of the root are numbered from left to right by k−1, k−2, ..., 2, 1, 0, then child i is the root of a subtree B_i.

Properties of binomial trees(cont.)

Proof: The proof is by induction on k. For each property, the basis is the binomial tree B_0. Verifying that each property holds for B_0 is trivial. For the inductive step, we assume that all the properties holds for B_{k-1}.

1. Since binomial tree B_k consists of two copies of binomial tree B_{k-1}, therefore

   Number of nodes in $B_k = 2^{k-1} + 2^{k-1} = 2.2^{k-1} = 2^k$

2. Since in B_k, one B_{k-1} is child of the other B^{k-1}, therefore

   the height of B_k = the height of B_{k-1} +1

   $= (k-1) + 1$

   $= k$

Properties of binomial trees(cont.)

3. Let D(k, i) be the number of nodes at depth i of binomial tree B_k. Since B_k is composed of two copies of B_{k-1} linked together, a node at depth i in B_{k-1} appears in B_k once at depth i and i in B_k is the number of nodes at depth i in B_{k-1} plus the number of nodes at depth i−1 in B_{k-1}. Thus,

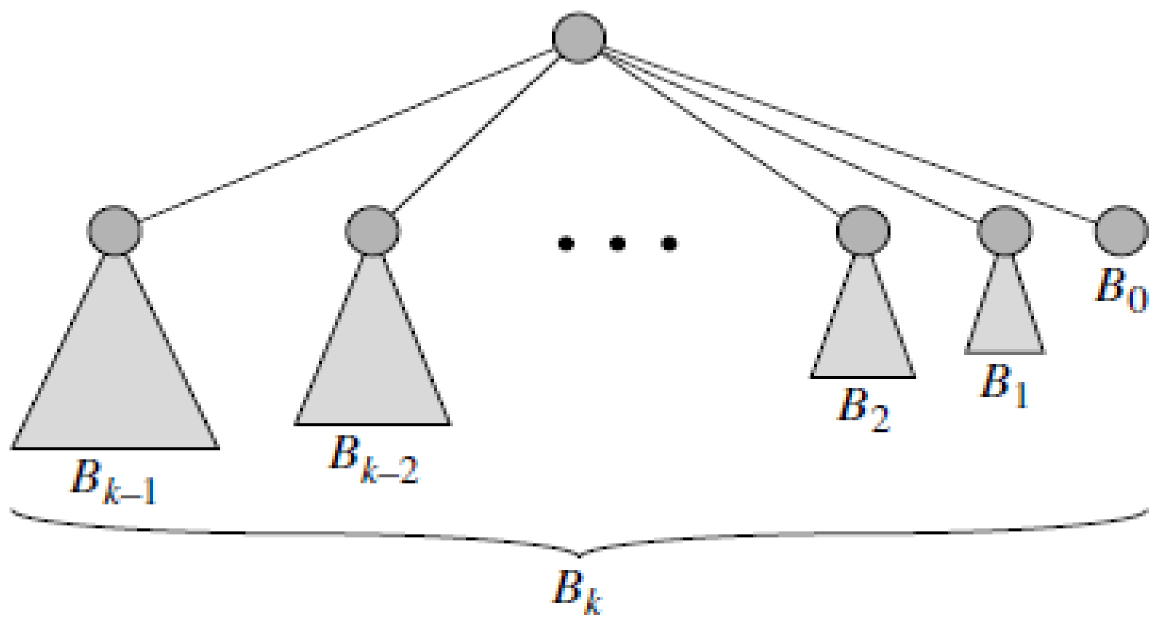
D(k,i) = D(k-1,i) + D(k-1, i-1)

$$= \binom{k-1}{i} + \binom{k-1}{i-1}$$

$$= \binom{k}{i}$$

Properties of binomial trees(cont.)

4. The only node with greater degree in B_k than in B_{k-1} is the root, which has one more child than in B_{k-1}. Since the root of B_{k-1} has degree k−1, therefore the root of B_k has degree k.



Now, by the inductive hypothesis, and as figure shows, from left to right, the children of the root of B_{k-1} are roots of B_{k-2}, B_{k-3}, ..., B_0. When B_{k-1} is linked to B_{k-1}, therefore, the children of the resulting root are roots of B_{k-1}, B_{k-2}, ..., B_0.

Binomial Tree

Lemma: The maximum degree of any node in an *n*-node binomial tree is lg *n*.

Proof: Let the maximum degree of any node is k.

According to property (4), the root node has a maximum degree . Therefore degree of root node is k. This imply that the binomial tree will be B_k.

According to property (1), the total number of nodes in binomial tree B_k is 2^k. Since the number of nodes in binomial tree is n, therefore

$$2^k = n \qquad \Rightarrow \qquad k = \lg n$$

It is proved.

Binomial heaps

A binomial heap H is a set of binomial trees that satisfies the following binomial heap properties.

1. Each binomial tree in H obeys the min-heap property: the key of a node is greater than or equal to the key of its parent. We say that each such tree is min-heap-ordered.

2. For any non-negative integer k, there is at most one binomial tree in H whose root has degree k.

Note: An n-node binomial heap H consists of at most $\lfloor \lg n \rfloor + 1$ binomial trees.

Example: Construct binomial heap for 27 nodes.

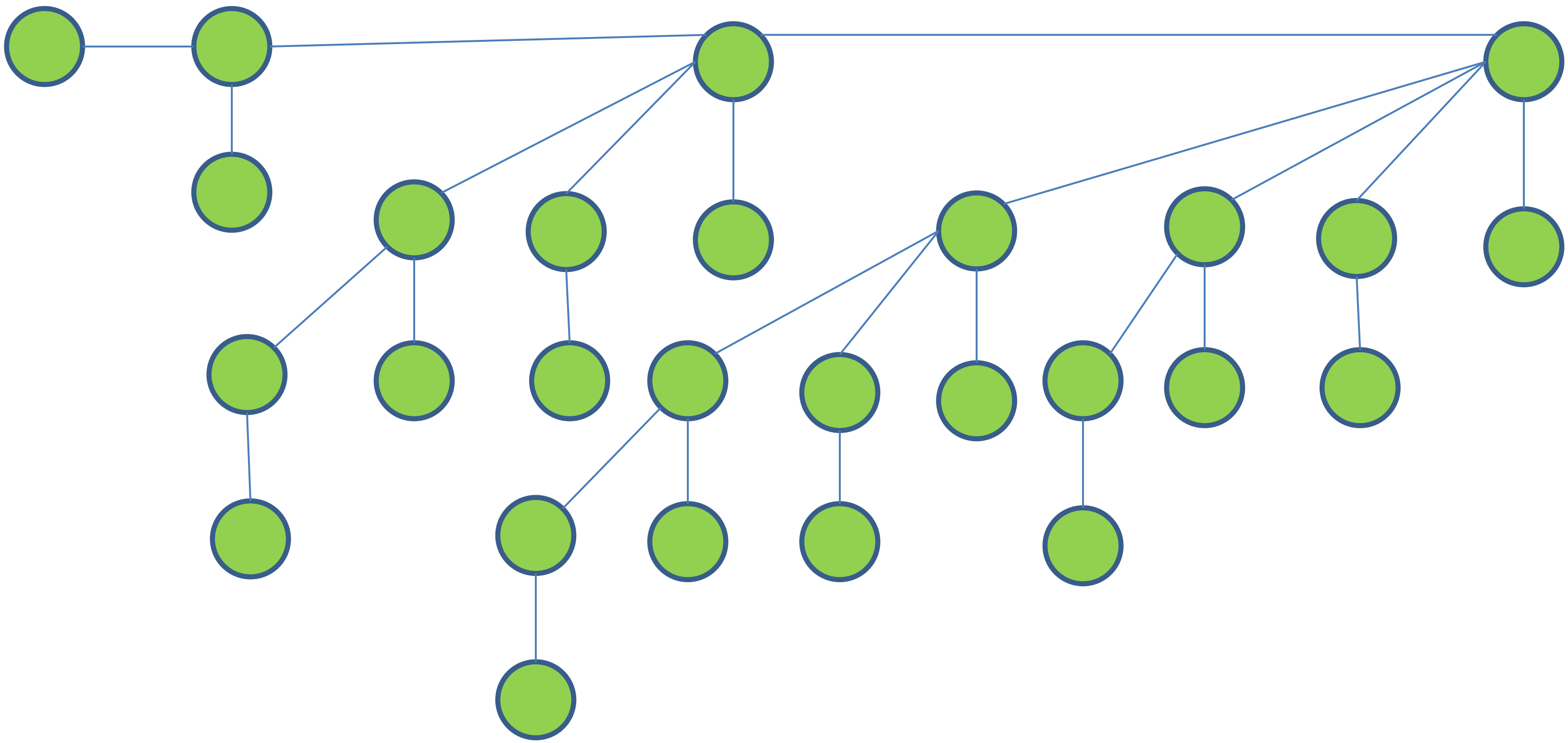
Solution: First we find binary number of 27. After it, we compare this number with $B_4B_3B_2B_1B_0$. If the corresponding binary number is 1, then we use the corresponding binomial tree in the Binomial heap.

Binary number of 27 = 11011

Therefore, binomial tree in binomial heap will be B_4, B_3, B_1, B_0.

Binomial heaps

Therefore, binomial heap for 27 nodes will be



Representation of binomial heaps

- Each binomial tree within a binomial heap is stored in the left-child, right-sibling representation.
- Each node x in binomial heap consists of following fields:-

Key[x] → value stored in the node x

p[x] → pointer representing parent of node x

child[x] → pointer representing left most child of node x

sibling[x] → pointer representing immediate right sibling of node x

degree[x] → the number of children of node *x*

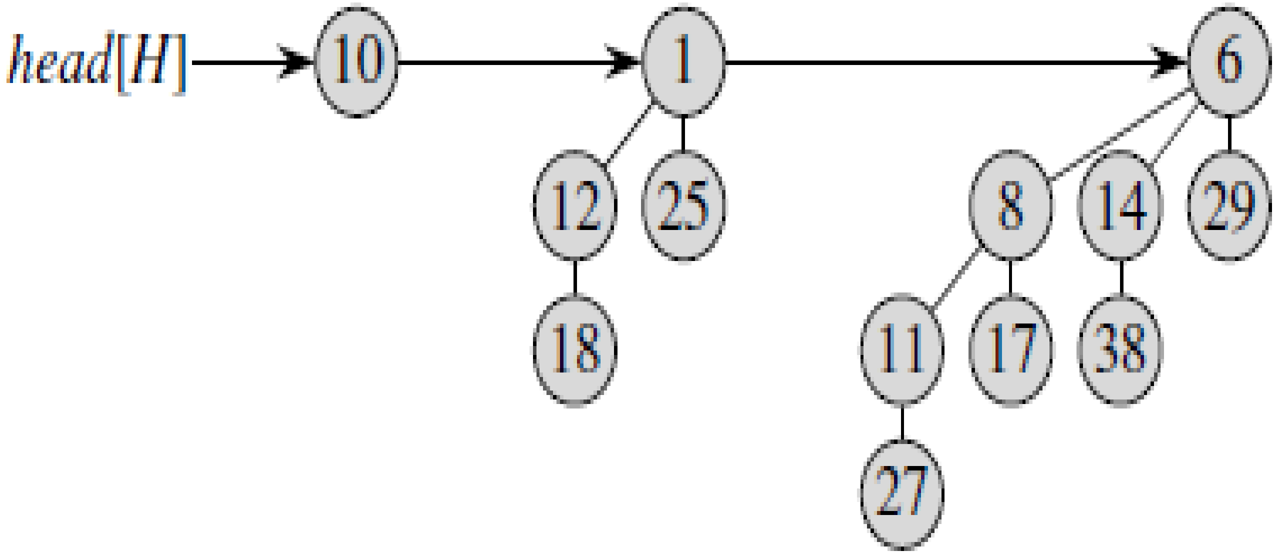
<table>
<tr><td colspan="2">p[x]</td></tr>
<tr><td colspan="2">Key[x]</td></tr>
<tr><td colspan="2">Degree[x]</td></tr>
<tr><td>Child[x]</td><td>Sibling[x]</td></tr>
</table>

Representation of binomial heaps

- The roots of the binomial trees within a binomial heap are organized in a linked list, which we refer to as the root list. The degrees of the roots strictly increase as we traverse the root list.
- The sibling field has a different meaning for roots than for non roots. If x is a root, then sibling[x] points to the next root in the root list.
- A given binomial heap H is accessed by the field head[H], which is simply a pointer to the first root in the root list of H. If binomial heap H has no elements, then head[H] = NIL.

Representation of binomial heaps

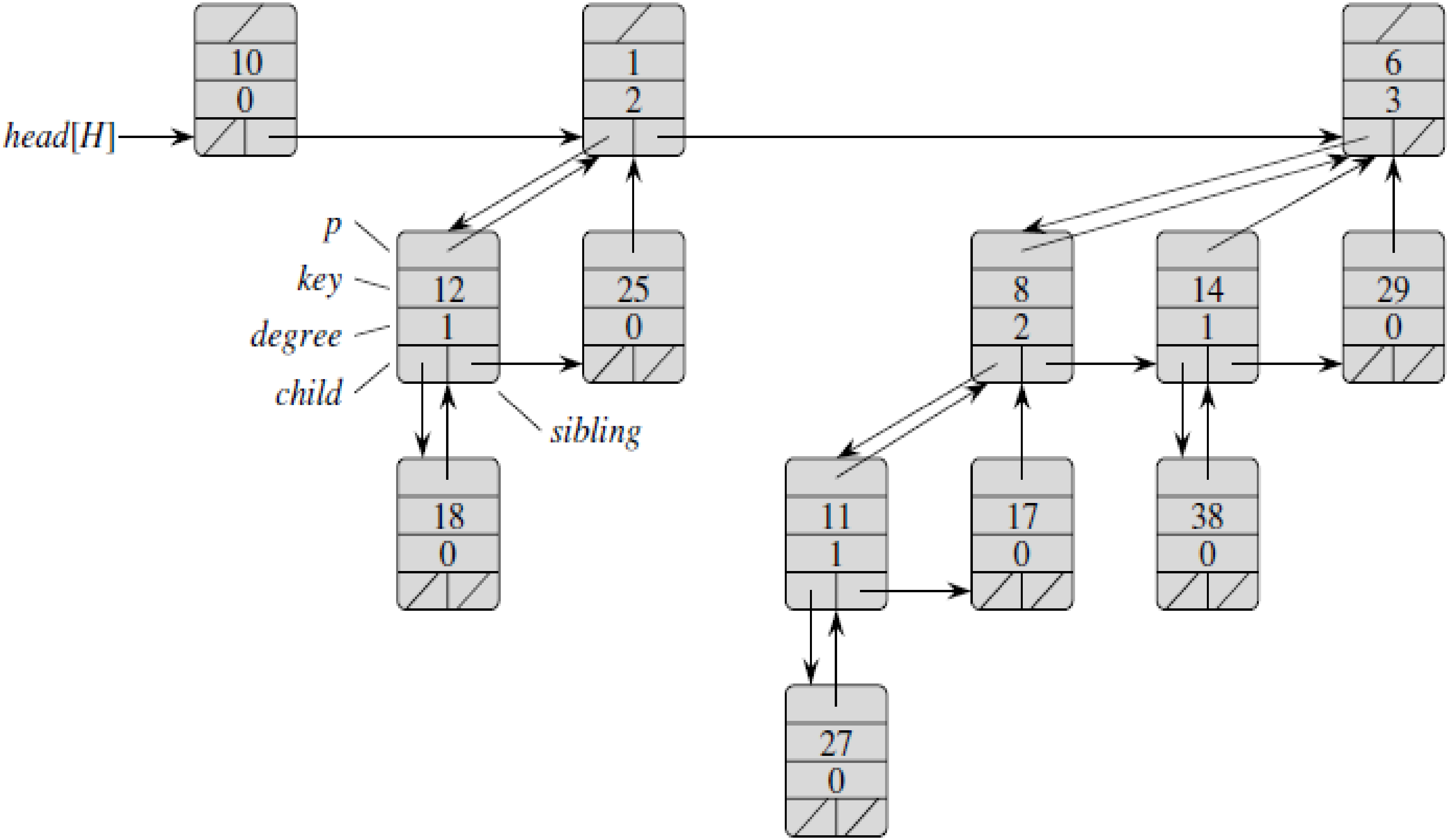
Example: Consider the following binomial heap:-



Find the representation of this binomial heap.

Representation of binomial heaps

Representation of binomial heap is



Operations defined on binomial heaps

Finding the minimum key

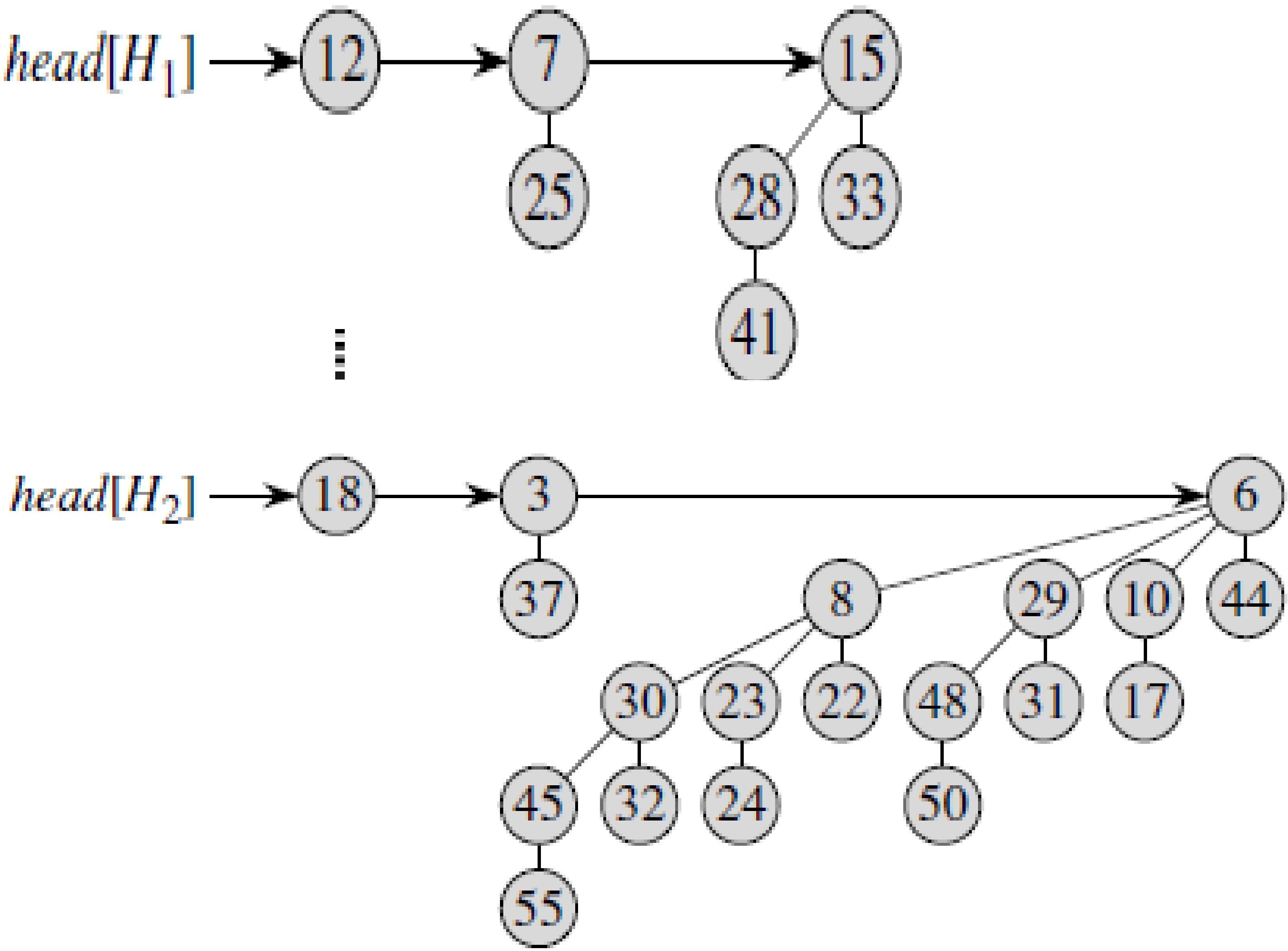
The procedure BINOMIAL-HEAP-MINIMUM returns a pointer to the node with the minimum key in an n-node binomial heap H.

```
BINOMIAL-HEAP-MINIMUM(H)
y ← NIL
x ← head[H]
min ← ∞
while x ≠ NIL
      do if key[x] < min
            then min ← key[x]
                 y ← x
         x ← sibling[x]
return y
```

Note: The running time of BINOMIAL-HEAP-MINIMUM is O(lg n).

Union of two binomial heaps

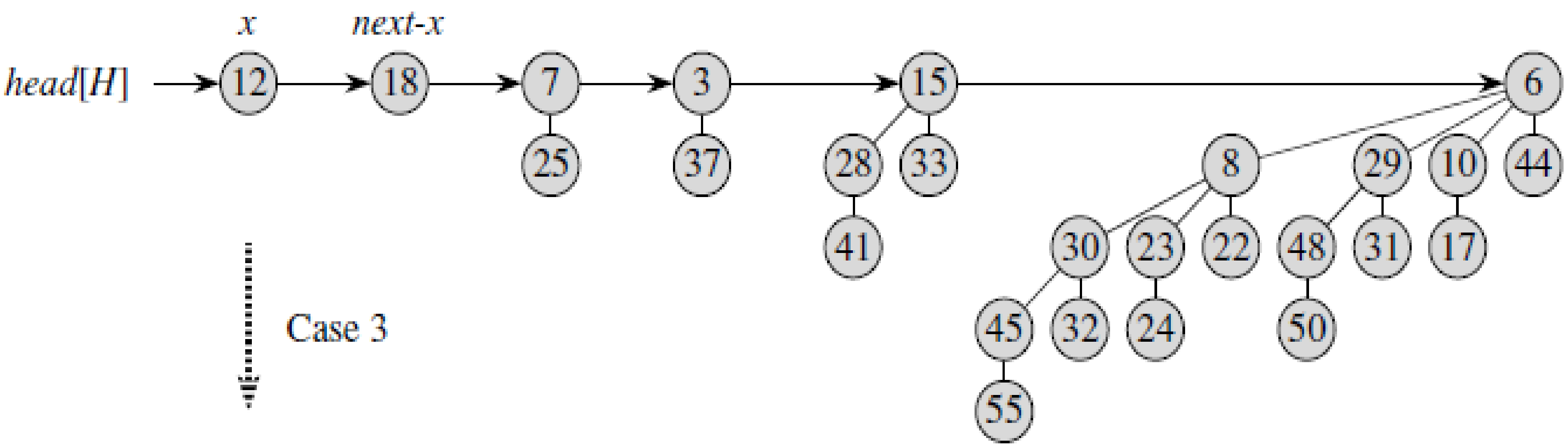
Example: Consider following two binomial heaps H_1 and H_2. Find the union of these binomial heaps.



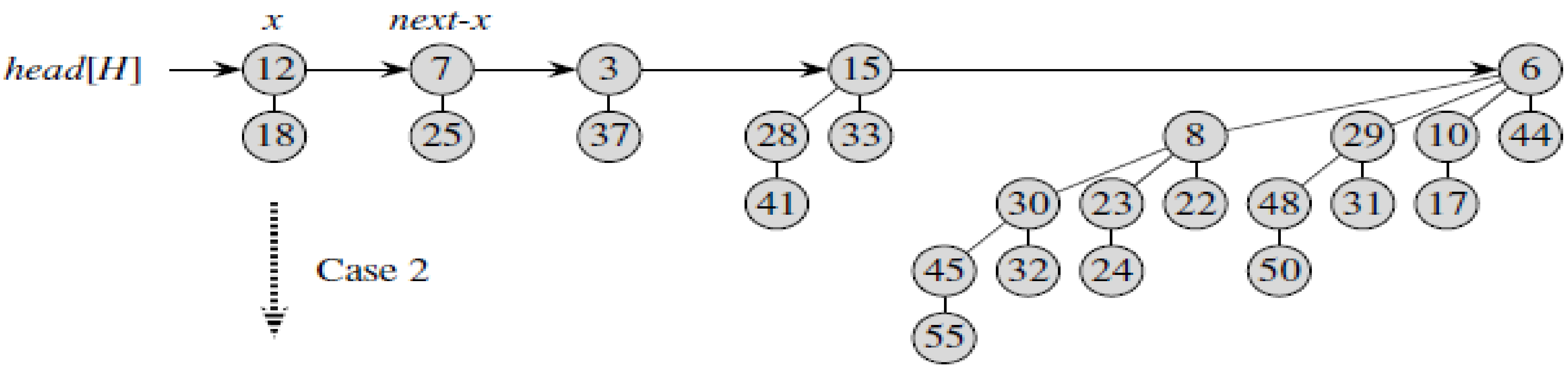
Union of two binomial heaps

Solution:

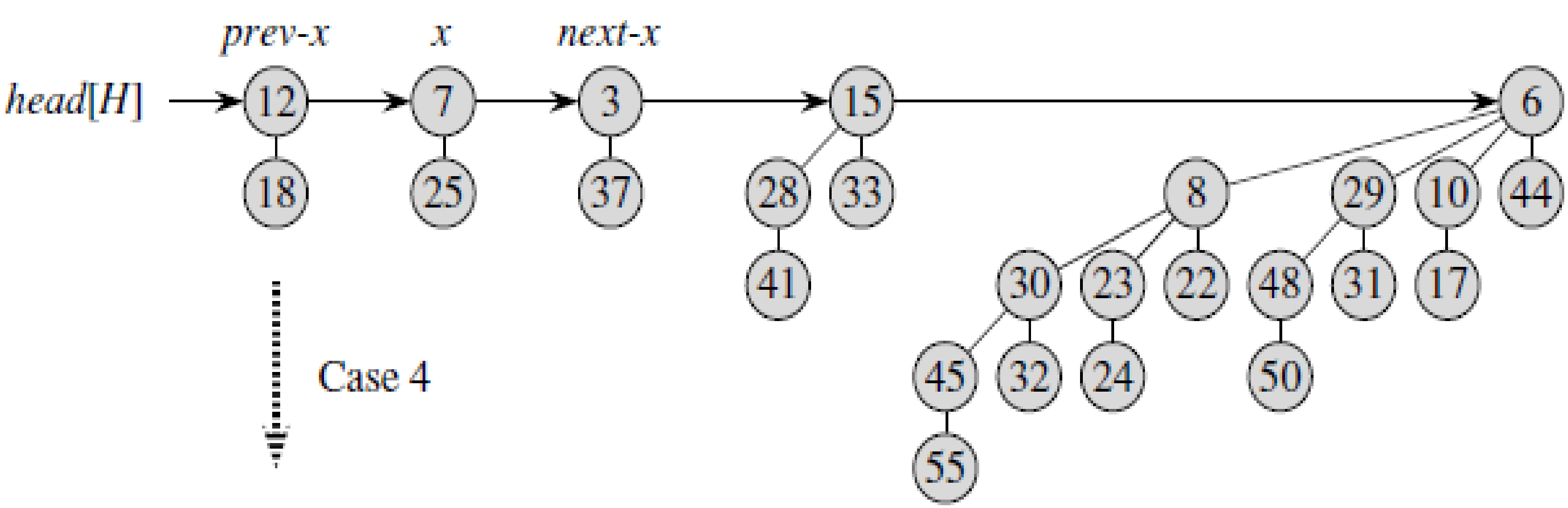
Step-1: Merge both binomial heaps.

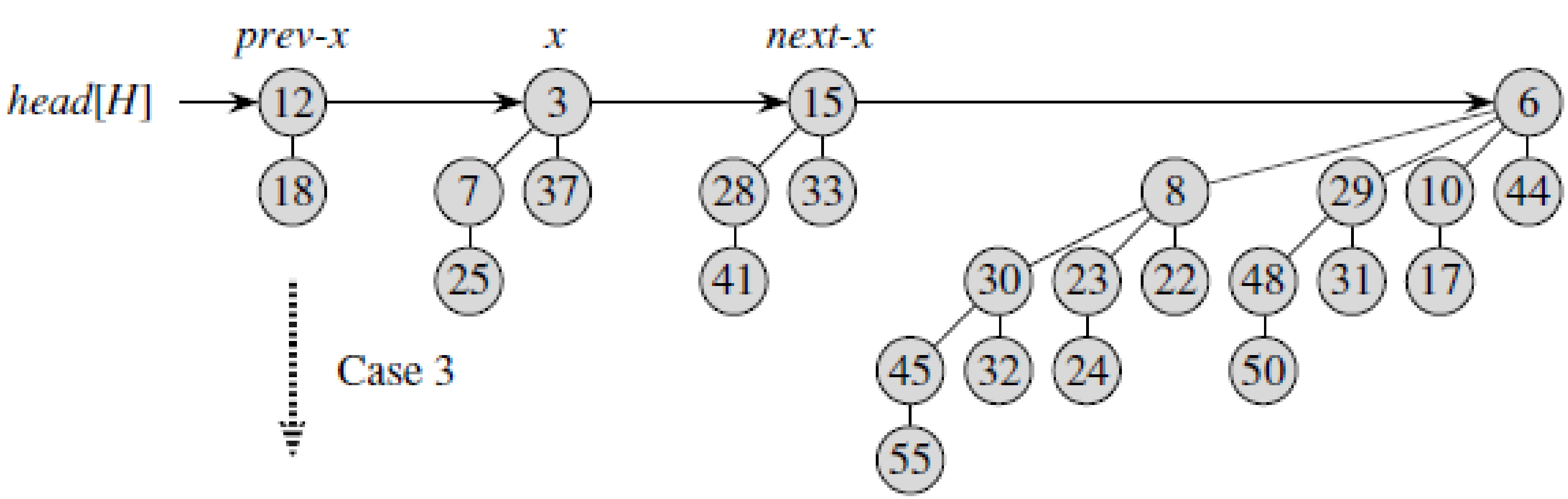


Step-2: Apply the linking process of equal degree root nodes.

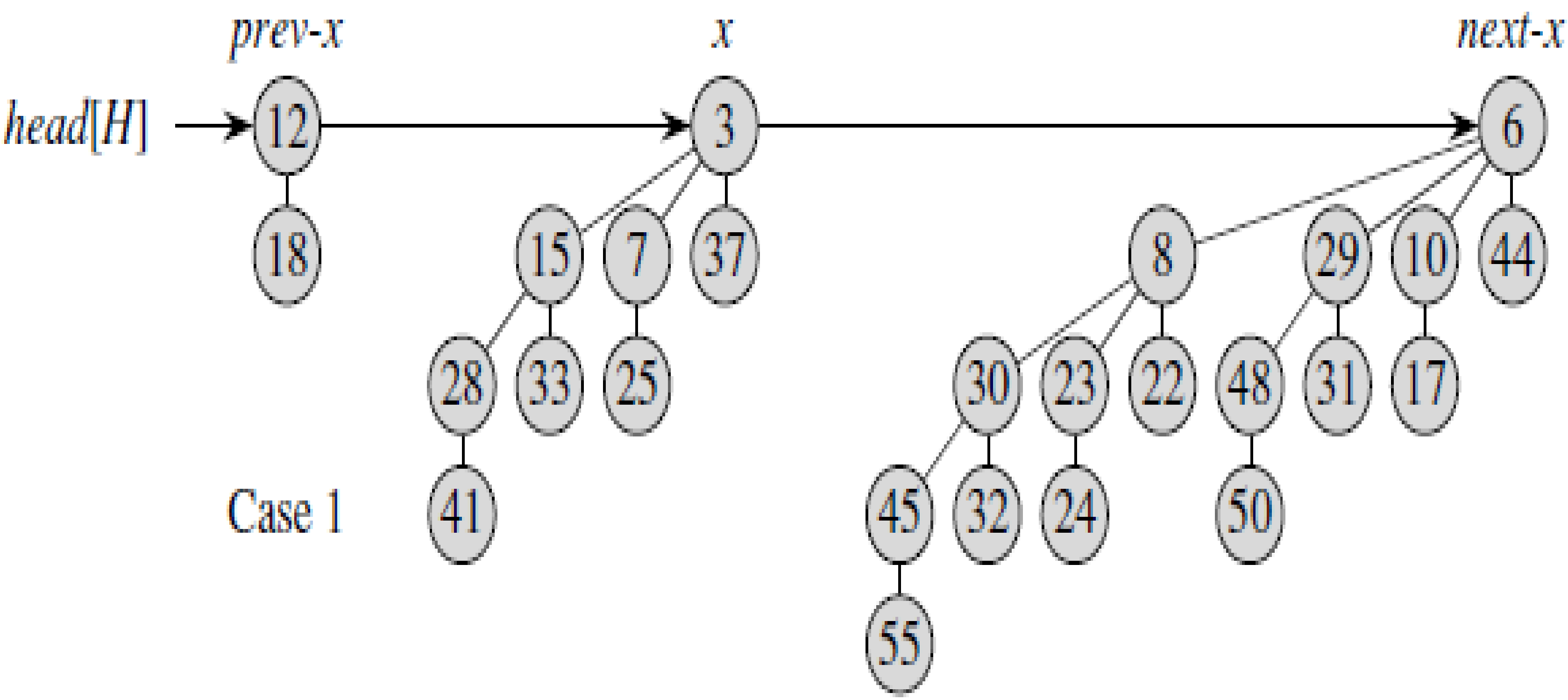


Union of two binomial heaps





Union of two binomial heaps



Final binomial heap

Union of two binomial heaps

```
BINOMIAL-HEAP-UNION(H1, H2)
H ← MAKE-BINOMIAL-HEAP()
head[H] ← BINOMIAL-HEAP-MERGE(H1, H2)
free the objects H1 and H2 but not the lists they point to
if head[H] = NIL
   then return H
prev-x ← NIL
x ← head[H]
next-x ← sibling[x]
while next-x ≠ NIL
      do if (degree[x] ≠ degree[next-x]) or
            (sibling[next-x] ≠ NIL and degree[sibling[next-x]] = degree[x])
            then prev-x ← x                                  ▷ Cases 1 and 2
                 x ← next-x                                  ▷ Cases 1 and 2
            else if key[x] ≤ key[next-x]
                    then sibling[x] ← sibling[next-x]        ▷ Case 3
                         BINOMIAL-LINK(next-x, x)            ▷ Case 3
                    else if prev-x = NIL                     ▷ Case 4
                            then head[H] ← next-x            ▷ Case 4
                            else sibling[prev-x] ← next-x    ▷ Case 4
                         BINOMIAL-LINK(x, next-x)            ▷ Case 4
                         x ← next-x                          ▷ Case 4
         next-x ← sibling[x]
return H
```

Union of two binomial heaps

The BINOMIAL-HEAP-UNION procedure has two phases.

- The first phase, performed by the call of BINOMIAL-HEAP-MERGE, merges the root lists of binomial heaps H1 and H2 into a single linked list H that is sorted by degree into monotonically increasing order.
- In the second phase, we link roots of equal degree until at most one root remains of each degree.

BINOMIAL-LINK(y, z)

1 $p[y] \leftarrow z$
2 $sibling[y] \leftarrow child[z]$
3 $child[z] \leftarrow y$
4 $degree[z] \leftarrow degree[z] + 1$

Time Complexity:

Time complexity of BINOMIAL-HEAP-UNION is O(lg n).

Inserting a node

The following procedure inserts node x into binomial heap H, assuming that x has already been allocated and key[x] has already been filled in.

BINOMIAL-HEAP-INSERT(H, x)

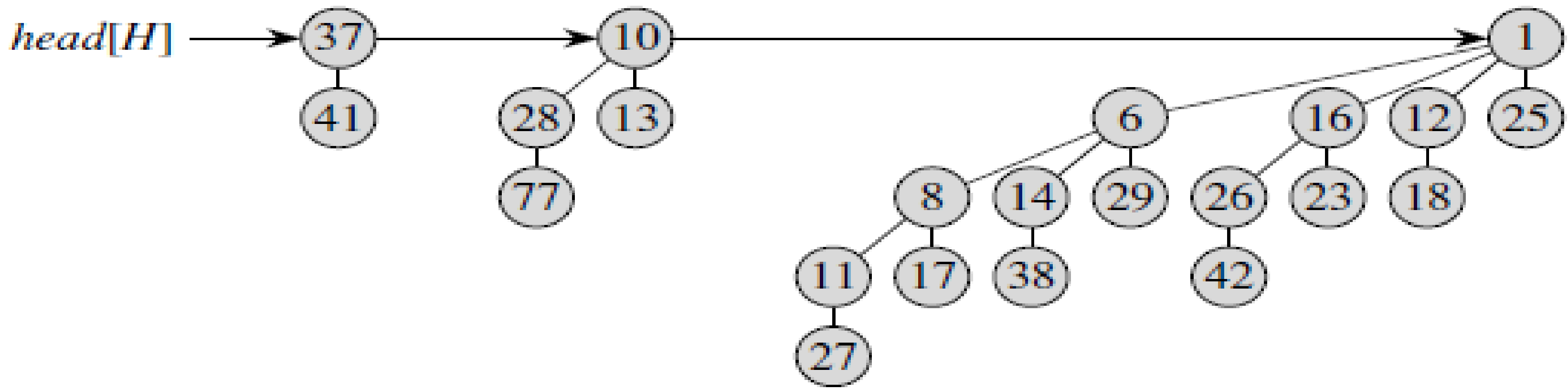
$H' \leftarrow$ MAKE-BINOMIAL-HEAP()
$p[x] \leftarrow$ NIL
$child[x] \leftarrow$ NIL
$sibling[x] \leftarrow$ NIL
$degree[x] \leftarrow 0$
$head[H'] \leftarrow x$
$H \leftarrow$ BINOMIAL-HEAP-UNION(H, H')

Time Complexity:

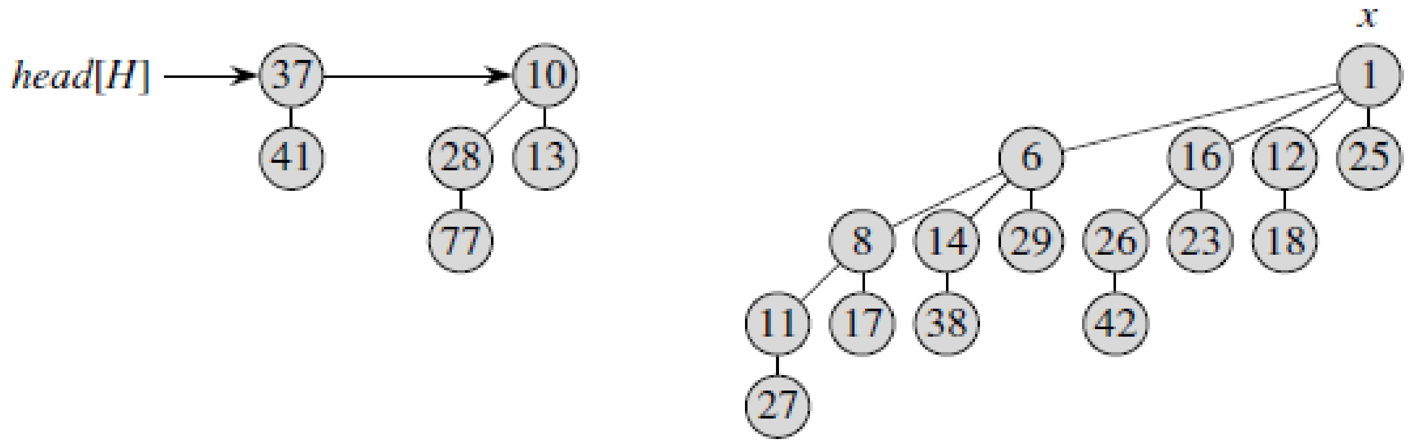
Time complexity of this algorithm is O(lg n).

Extracting the node with minimum key

Example: Extract the node with the minimum key from the following binomial heap H:-

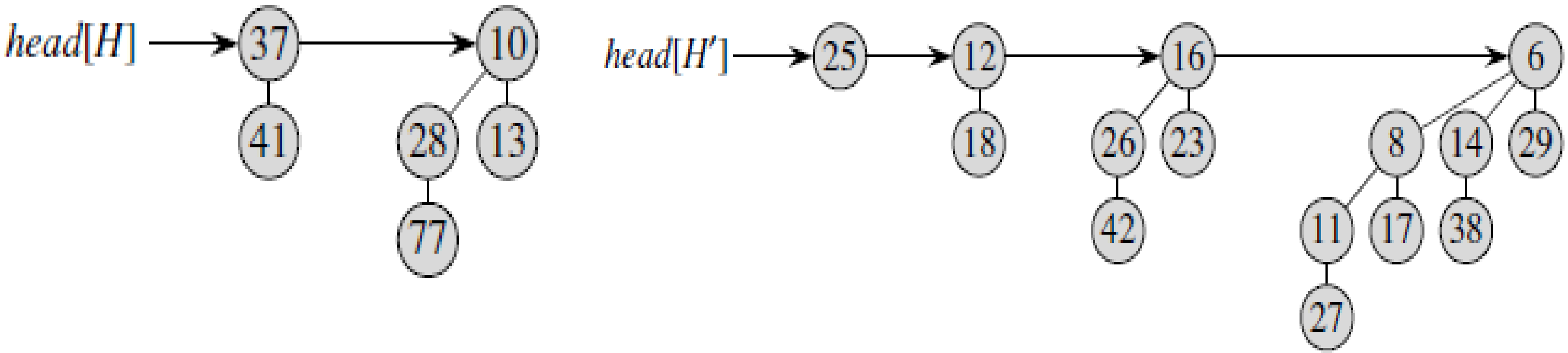


Solution: Step-1: Find the node with minimum key and remove that node.

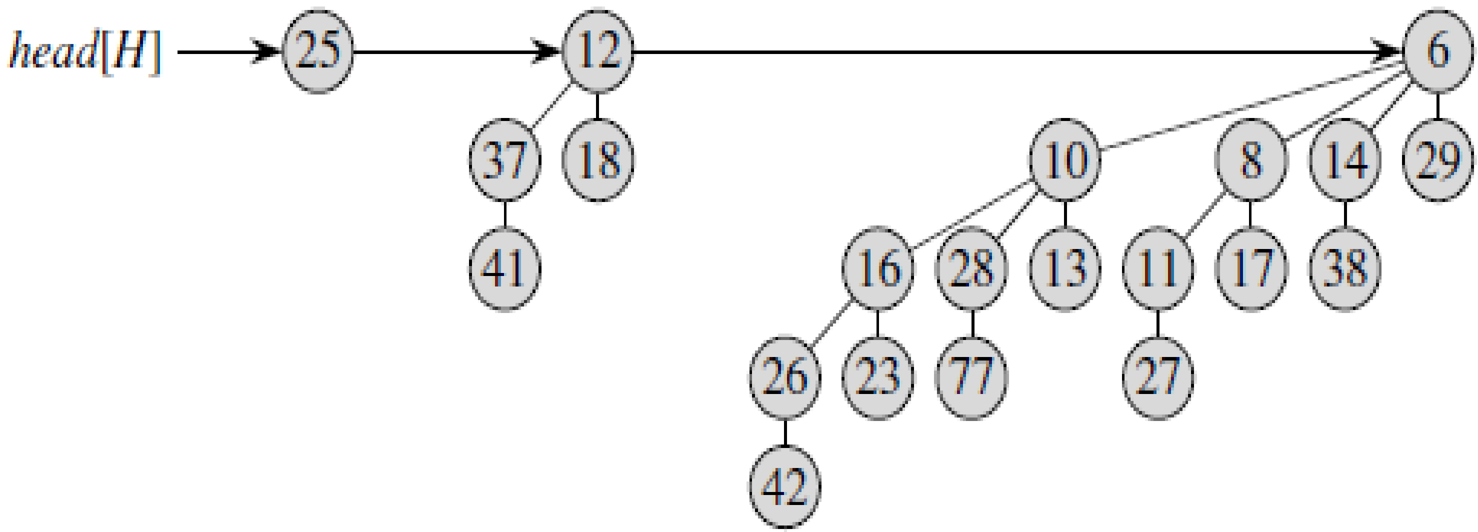


Extracting the node with minimum key

Step-2: Make the binomial heap H' from children of minimum node.



Step-3: Find the union of H and H'.



Extracting the node with minimum key

The following procedure extracts the node with the minimum key from binomial heap H and returns a pointer to the extracted node.

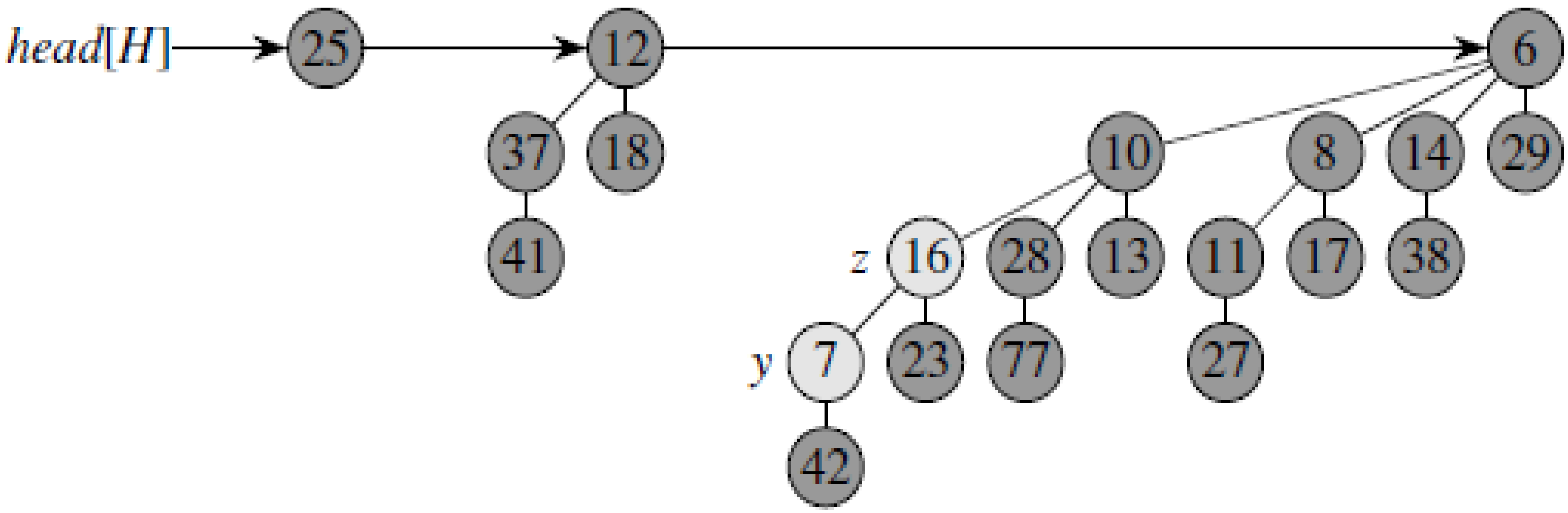
```
BINOMIAL-HEAP-EXTRACT-MIN(H)
    find the root x with the minimum key in the root list of H,
            and remove x from the root list of H
    H' ← MAKE-BINOMIAL-HEAP()
    reverse the order of the linked list of x's children,
            and set head[H'] to point to the head of the resulting list
    H ← BINOMIAL-HEAP-UNION(H, H')
    return x
```

Time Complexity:

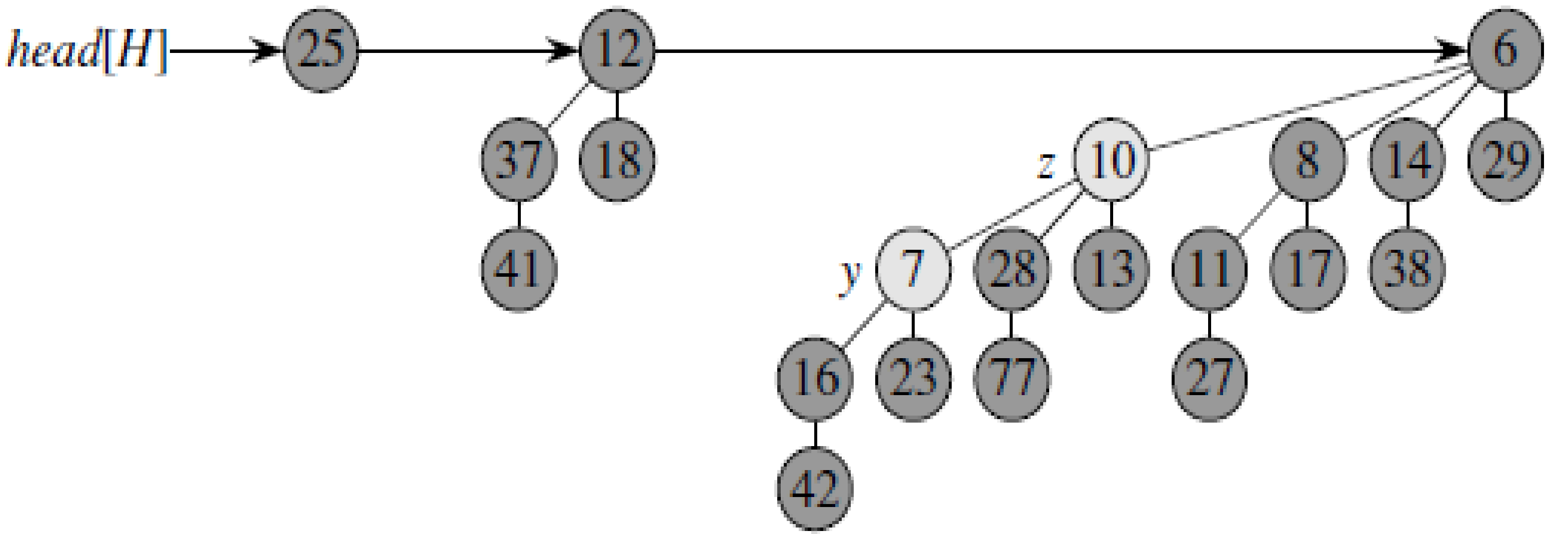
Time complexity of this algorithm is O(lg n).

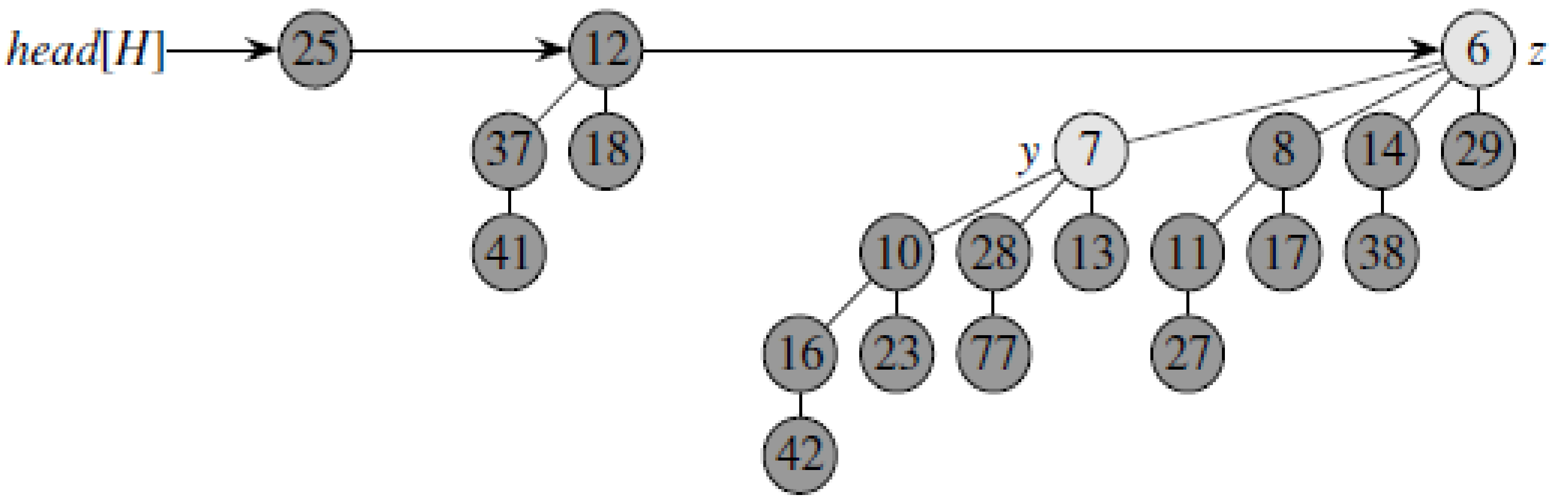
Decreasing a key

Example: Decrease the value of a node y to be 7.



Decreasing a key





Decreasing a key

The following procedure decreases the key of a node x in a binomial heap H to a new value k. It signals an error if k is greater than x's current key.

```
BINOMIAL-HEAP-DECREASE-KEY(H, x, k)
if k > key[x]
   then error "new key is greater than current key"
key[x] ← k
y ← x
z ← p[y]
while z ≠ NIL and key[y] < key[z]
     do exchange key[y] ↔ key[z]
        ▷ If y and z have satellite fields, exchange them, too.
        y ← z
        z ← p[y]
```

<u>Time Complexity:</u>

Time complexity of this algorithm is O(lg n).

Deleting a key

Following procedure is used to delete the key value of a node. This implementation assumes that no node currently in the binomial heap has a key of $-\infty$.

BINOMIAL-HEAP-DELETE(H, x)

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

Time Complexity:

Time complexity of this algorithm is O(lg n).

AKTU examination questions

1. Explain various properties of Binomial Tree.
2. Prove that maximum degree of any node in an n node binomial tree is log n.
3. Explain the algorithm to extract the minimum elements in a binomial Heap. Give an example for the same.
4. Define Binomial Heap with example.
5. Explain the algorithm to delete a given element in a binomial Heap. Give an example for the same.
6. Explain properties of Binomial Heap in .Write an algorithm to perform uniting two Binomial Heaps. And also to find Minimum Key.
7. Explain the different conditions of getting union of two existing binomial Heaps. Also write algorithm for union of two Binomial Heaps. What is its complexity?

Thank You.