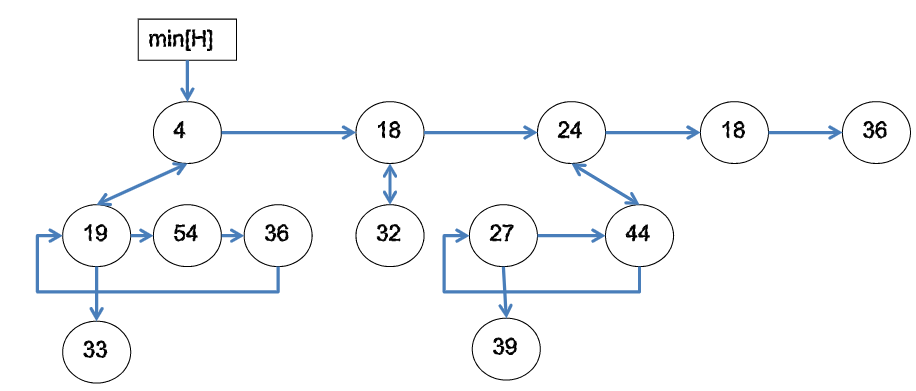
Fibonacci Heaps

**Concept**

* A Fibonacci heap is a collection of heap-ordered trees
* Properties of Fibonacci heaps
  + O(log n) time for extract-min/delete min operation
  + O(1) time for other operations
* Fibonacci heaps are desirable when the number of extract-min and delete operations are small relative to the number of other operations performed

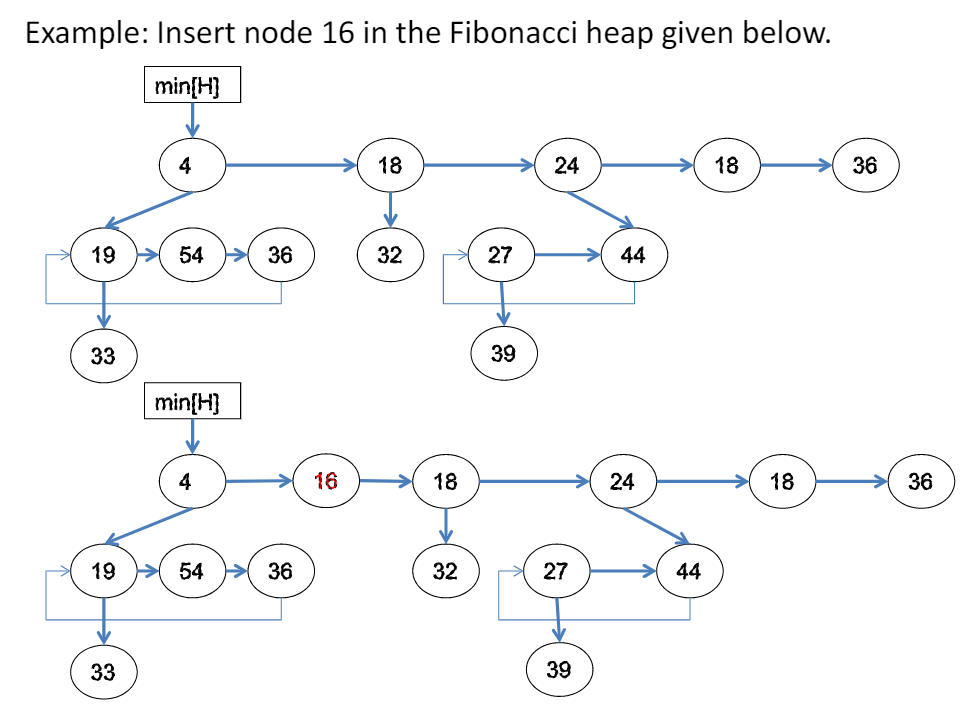
**Representation of Fibonacci Heaps**

1. Each heap ordered tree node is represented by a structure with the following fields
   1. Key – the key value
   2. Data – data carried by the node
   3. Parent – a pointer to its parent
   4. Child – a pointer to any one of its children
   5. Left – a pointer to the left sibling
   6. Right – a pointer to the right sibling
   7. Degree – the number of children
   8. Mark – a value indicates whether node x has lost a child
2. The children of a node are circularly linked list. If node x is the only child of its parent, then left[x] = right[x] = x
3. The root nodes of heap-ordered trees are linked list (root list). Start at the minimum key root.
4. Fibonacci heap H is generally accessed by a pointer called min[H] which points to the root that has a minimum value. If the Fibonacci heap H is empty, then min[H] = NULL.
5. The number of nodes in H are stored in n[H].



**Fibonacci Heap Operations**

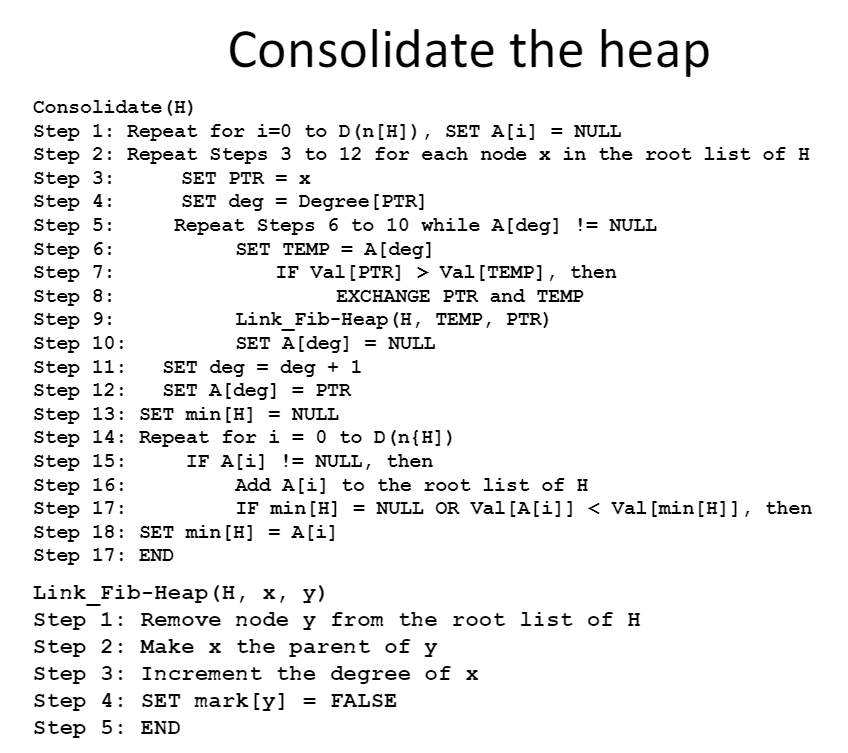
* Creating a new Fibonacci Heap
  + To create an empty Fibonacci heap, the create\_fib\_heap procedure is used to allocate and return the Fibonacci heap object H, where n[H] = 0 and min[H] = NULL.
    - Time: O(1)
* Finding the minimum node
  + Fibonacci heap maintains a pointer min[H], that points to the root having minimum value
    - Time: O(1)
* Insert operation
  + Algorithm:
  + Step 1: create a new singleton tree of the value
  + Add to the root list, update min pointer if necessary
    - Time: O(1)

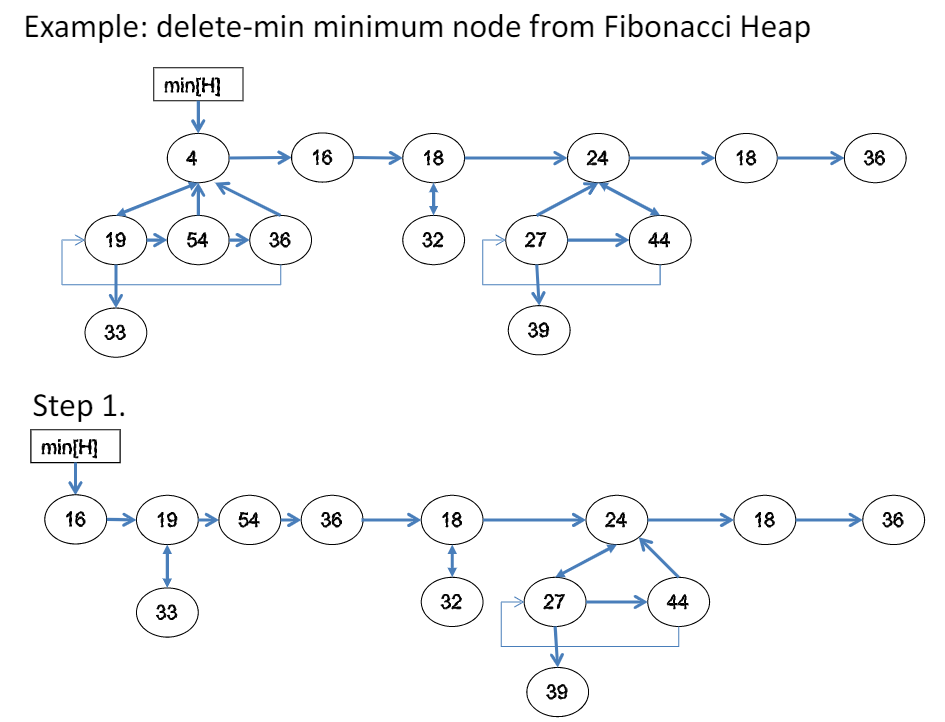


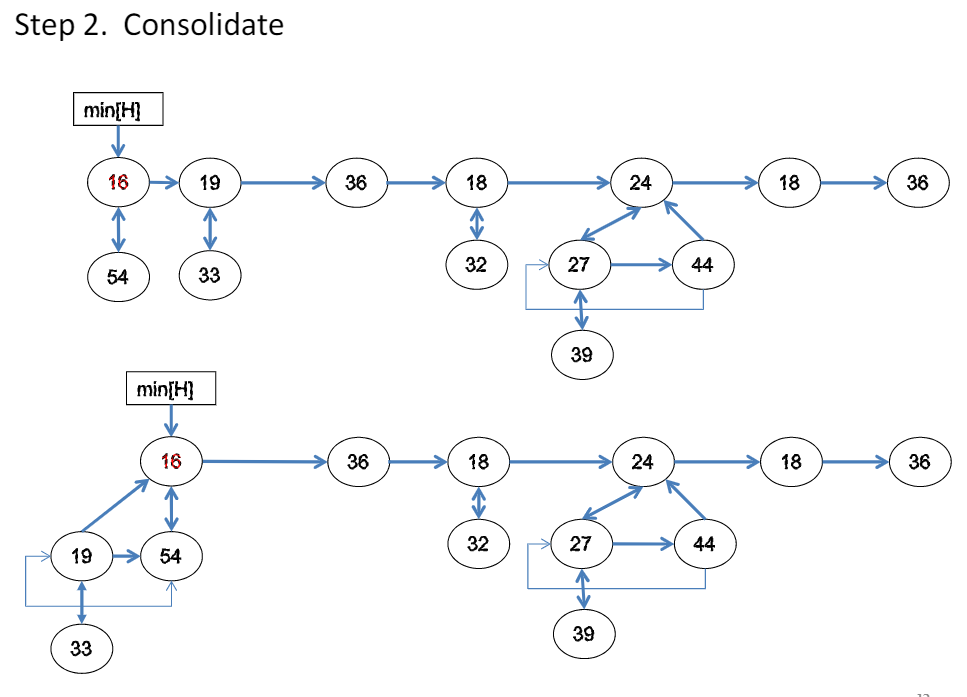
* Delete-min / extract-min
  + Algorithm:
  + Step 1: Delete the min root, put its children into root list, update min
  + Step 2: Consolidate trees so that no two roots have the same degree (eagerly consolidate)
  + Amortized time complexity: O(log n) the degree of root node is bounded by log n

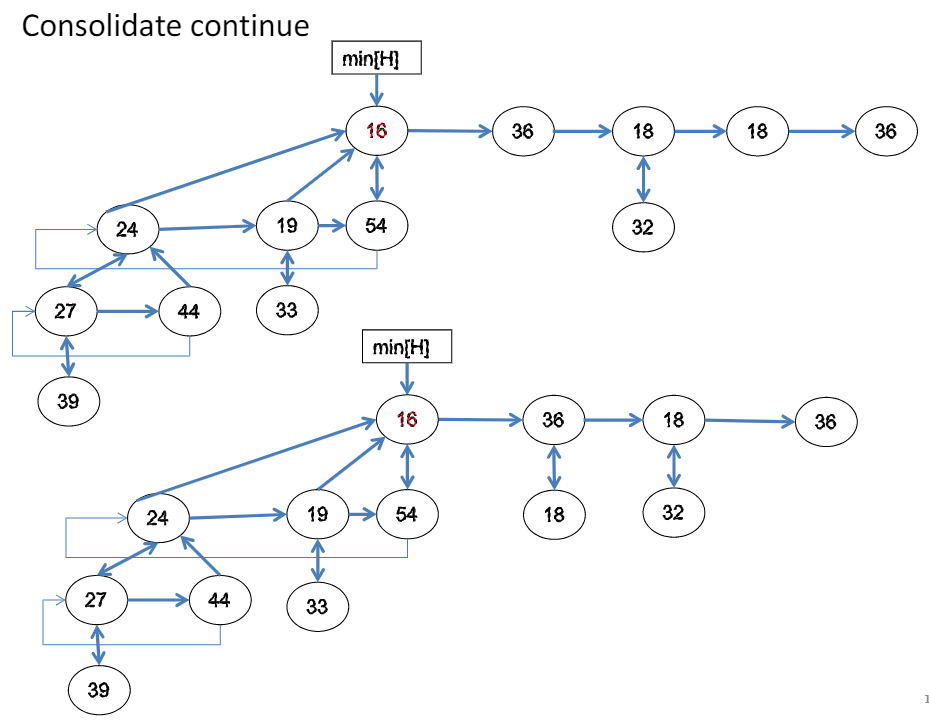
**Consolidate operation**

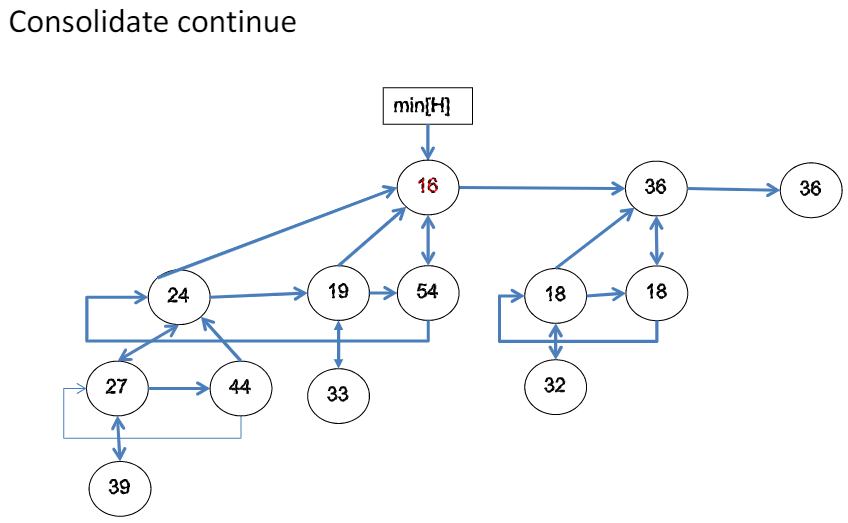
* The Fibonacci heap is consolidated to reduce the number of heap ordered trees. When consolidating the root list of H, the following steps are repeatedly executed until every root in the root list has a distinct *degree* value.
* Find two roots x and y in the root list that has the same degree, and where Val[x] <= Val[y]. Link y to x. That is, remove y from the root list of H and make it a child of x. This operation is done by the Link\_Fib\_Heap procedure. Finally, the degree[x] is incremented, and the mark on y, if any, is cleared.











**Decrease-Key operation**

Algorithm: decreasing the key of node x

Step 1: decrease the key value of x to the given value

Step 2: If heap-order is not violated, exit

Step 3: c =x, p = parent[c]

Cut tree rooted at c and meld into root list

If mark[p] = false, mark[p] = true

Step 4: else go to step 3

Step 5: update min

Time: O(1) in amortized time

**Deleting a node**

* A node from the Fibonacci heap can be very easily deleted in o(D(n)) amortized time. The procedure to delete a node is given below.
* The amortized time of the delete procedure is the sum of O(1) amortized time of DECREASE\_VAL\_FIB\_HEAP and the O(D(n)) amortized time of EXTRACT\_MIN\_FIB\_HEAP

DEL\_FIB\_HEAP

1. DECREASE\_VAL\_FIB\_HEAP (H, X, -INFINITY)
2. EXTARCT\_MIN\_FIB\_HEAP(H)
3. END

