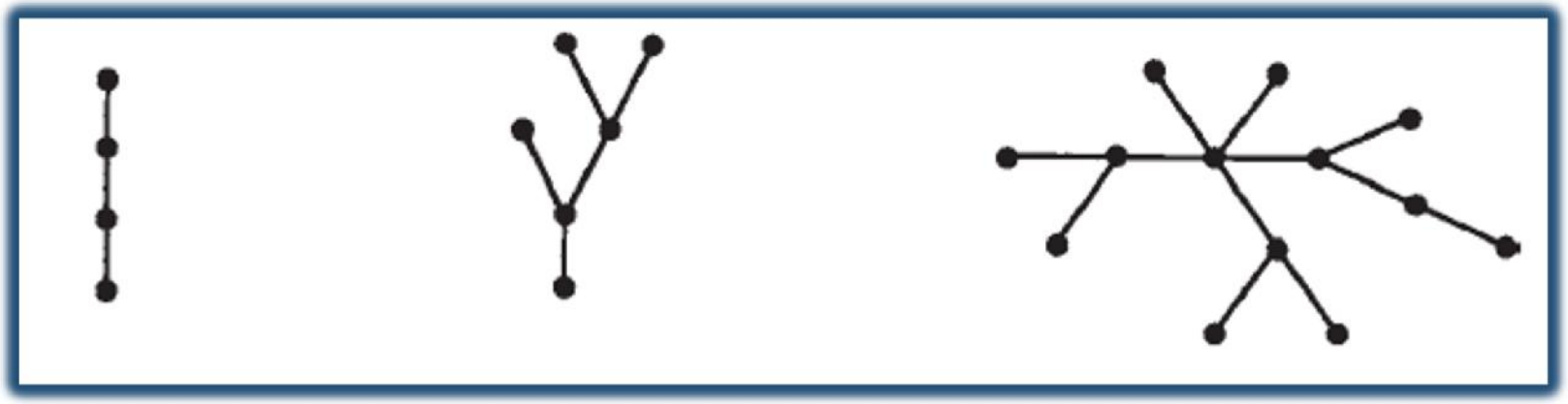
Introduction to trees

Tree is a connected graph with no cycles.



- An undirected graph with no cycles is a **forest**.
- A forest consists of one or more components, each of which is a tree.

Introduction to trees

- Three basic properties of trees are listed below.

1) For each pair of vertices u and v in a tree, there is exactly one path from u to v that does not repeat any vertices or edges.

2) Inserting an edge between any two vertices of a tree produces a graph containing a cycle.

3) Removing any edge from a tree produces a disconnected graph.

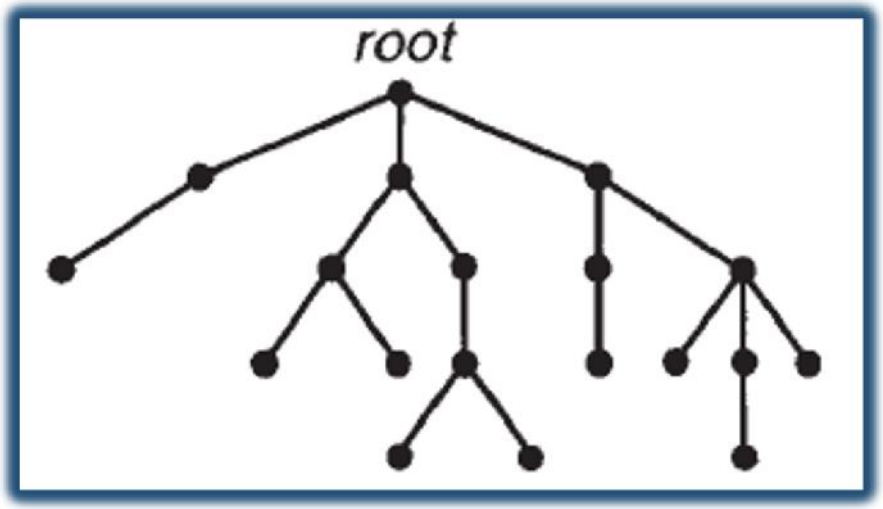
- Any tree with **n** vertices has **n-1** edges.
- A **weighted graph** is a simple graph in which each edge has a positive number attached to it. The number is called weight of the edge.

Introduction to trees

Definitions

- Let G be a weighted graph with vertices $v_1, v_2, \ldots, v_n$. The **weight matrix** of G is the **n x n** matrix for which the entry w_{ij} in the i-th row and j-th column is given by the rule:
- $$w_{ij} = \begin{cases} 0 \text{ if } i = j \\ \text{the weight of the edge from } i \text{ to } j \text{ if } v_i \text{ and } v_j \text{ are adjacent} \\ \infty \text{ if } i \neq j \text{ and } v_i \text{ and } v_j \text{ are not adjacent} \end{cases}$$
- Let G be a connected weighted graph, and let u and v be vertices in G.
- If P is a path from u and v, then the **length** of P is the sum of the weights of the edges in P.
- The **distance** from u to v is the smallest of the lengths of all the paths from u to v.

Introduction to trees



A vertex of degree 1 is called a **leaf**.
An edge incident to a leaf is a **leaf edge.**
A non-leaf vertex is an **internal vertex**.

- Sometimes, one vertex of the tree is distinguished, and called the **root**; in this case, the tree is called **rooted**.
- The vertices adjacent to the root (the 'first generation') are shown in a horizontal line below the root, the vertices that can be reached from the root by a path of length 2 (the 'second generation') are shown in a line below the first generation, and so on.

Introduction to trees

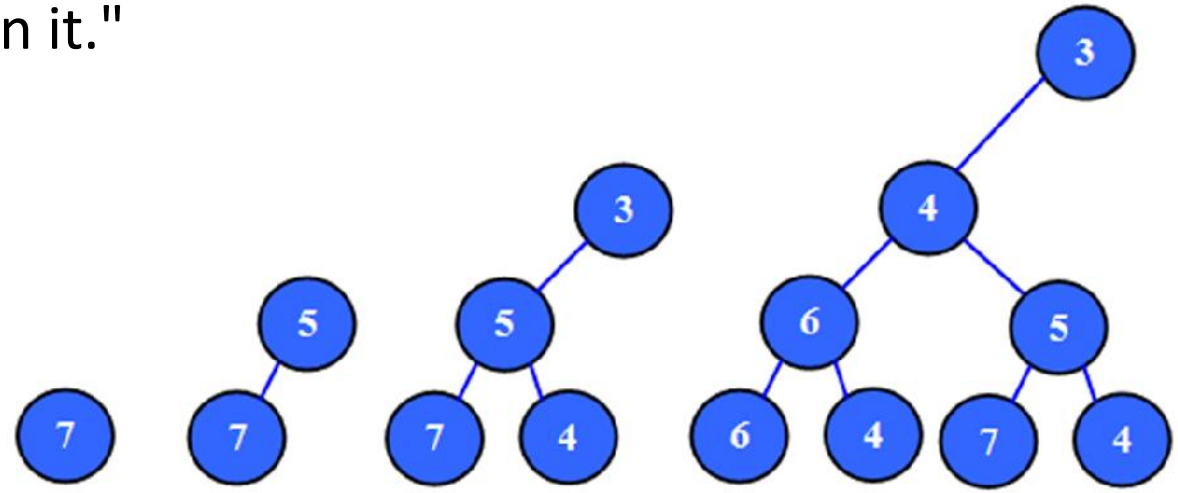
- Rooted trees are often treated as directed acyclic graphs with the edges pointing away from the root.
- We refer to the vertex immediately above a given vertex as the **parent** of the given vertex, while a vertex immediately below a given vertex is a **child** of that vertex.
- Every vertex in a rooted tree except the root has exactly one parent.
- A vertex with no children is called a **leaf**.
- Maximum length of chain from the root to its leaf is called the **height** of the tree.
- The rooted tree which has the property that every vertex that is not a leaf has exactly two children is called a **binary rooted tree**.

Introduction to trees

- In a binary rooted tree, the two children of each parent are identified as the **left child** and the **right child**, according to where they are placed when the tree is drawn.
- For any given parent vertex, the left child is the root of the left subtree, the binary rooted tree consisting of the left child and its descendants. Similarly, the right child is the root of the right subtree.
- A single vertex is a binary rooted tree.

Binomial heap

- Pyramidal tree is the tree of size 2^N with all elements, except the root and leaves, with exactly two child elements.
- The root element always has the left son, except for the degenerate case when the tree consists of a single element.
- For each element of the tree the rule is performed: "All the elements of the left subtree of the current element are not less than it."



Examples of pyramidal trees

Binomial heap

- **Binomial heap** is a data structure that contains a set of pyramidal trees with pairs of various sizes.
- We can make a direct analogy between the size of pyramidal trees in the binomial heap and the binary representation of the number.
- If the binomial heap contains **11** elements, then it means that it is composed of pyramidal trees with dimensions of **8, 2** and **1**.

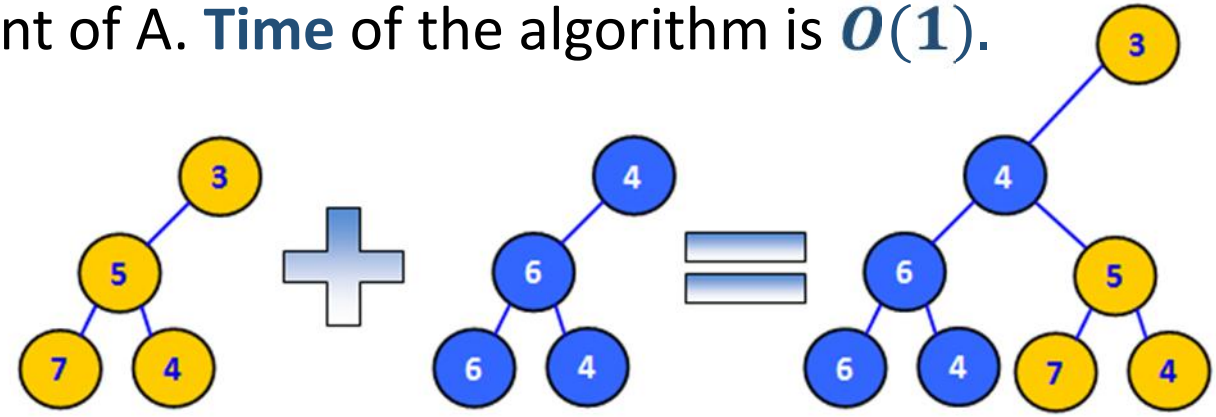
The search of minimal element

- The **minimum element** of the whole binomial heap must be searched for among the roots of pyramidal trees contained in this pile.
- Search of all the roots will be executed for $O\ (\log(N))$

Binomial heap

Combining the two pyramidal trees

- Two pyramidal trees **A** and **B** can be combined only if they have the **same size**.
- Consider that the **root** of the tree **A** is **less** than the **root** of the tree **B**.
- The **root** of the **merged tree** will be the **root** of the **tree A**, its left subtree will be the right subtree of the root element of the tree B, and the tree B itself will be the left subtree of the root element of A. **Time** of the algorithm is $O(1)$.



Binomial heap

Adding a new element

- This operation repeats the logic of a single increment.
- Let the original binomial heap consist of 11 elements.

$11_{10} + 1_{10} = 1011_2 + 0001_2 = 1100_2 = 12_{10.}$

1011

+ 0001

1100

- The operation of adding two single bits is equivalent to the operation of combining two pyramidal trees of the same size.
- The complexity of this procedure is $O\ (\log N)$.

Binomial heap

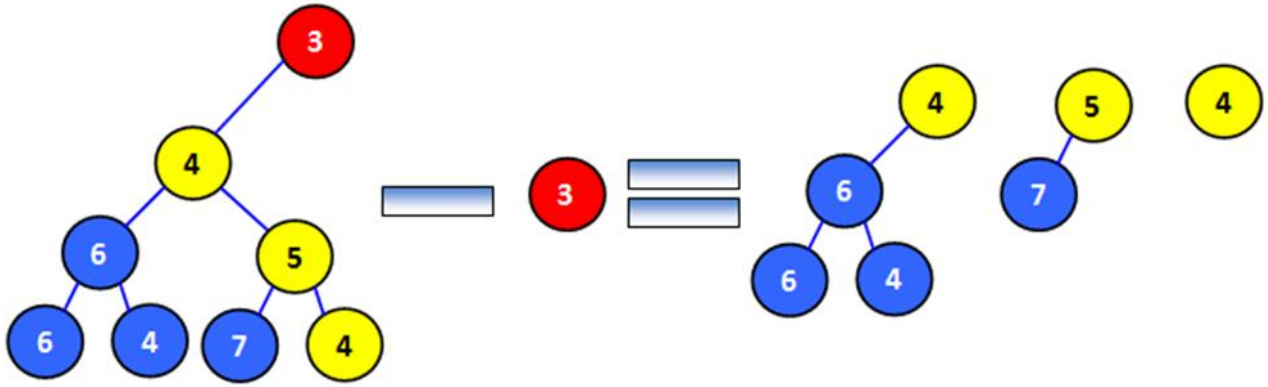
Combining the two binomial heaps

- Operation of combining two binomial heaps is similar to the addition of a single element in the binomial heap.
- This is similar to the addition of two binary numbers.
- The complexity of this procedure is $O(\log N)$.

Binomial heap

Removing the minimum element

- When you remove the minimum element of the heap, you must first find a pyramidal tree, for which this element is a root, and to exclude this tree from the general list of trees of the binomial heap.
- After that it is necessary to divide the original tree into $log_2(count)$ pyramidal trees (count - the number of elements of the tree), which are subtrees of the original tree.



Binomial heap

- These trees will have sizes equal to degrees of two, and also will be pairwise distinct.
- These trees can be combined into a separate binomial heap T.
- After that it is necessary just to combine original binomial heap with the heap T.
- This operation requires applying two sequential operations to the asymptotic complexity of $O(logN)$, so the overall complexity is also $O(logN)$.

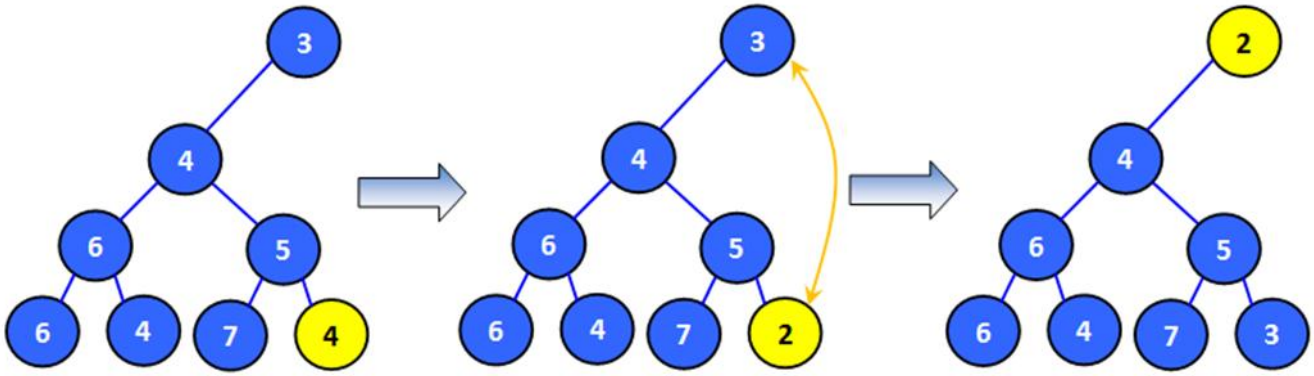
Binomial heap

Changing the element

- Change of the element may impair the integrity of the pyramidal tree.
- To restore the integrity it is necessary to apply the operation of sifting up or down.
- Sifting down is carried down by the following rule: "If the current element is less than the minimum of his sons, they are swapped."

Binomial heap

- When sifting up we are searching for such parent element, for which the updated element is in the left subtree, and if the updated value is less than the parent, they are swapped.
- This action repeated for the new position of the updated element.



Sifting up in a pyramidal tree

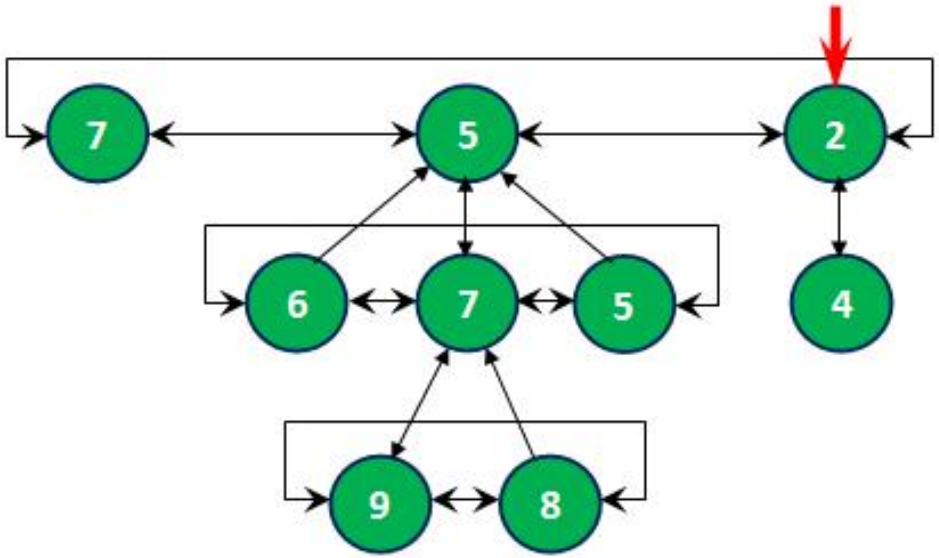
Binomial heap

Deleting an element

- To remove an item, you must first make it the root node of the pyramidal tree, in which it is located, and then remove the root node from this tree.
- The easiest way to do it is to set this element the value less than the minimum element of the whole binomial heap.

Fibonacci heap

- **Fibonacci heap** is a set of **Fibonacci trees**.



Example of Fibonacci tree

- **Fibonacci tree** is a ***k***-ary tree if for each item holds the rule "child element does not exceed its parent.“
- Roots of Fibonacci trees are stored in an annular list.

Fibonacci heap

- All child elements that have a common ancestor, are stored in an annular list, so the parent element does not need to store a reference to all their descendants.
- It is enough to know the information about only one of them.
- Each node of the Fibonacci tree, in addition to pointers to left and right brothers, parents and one of his sons, contains information about the number of child nodes.
- This information will be needed to delete the minimum element in the process of sealing the heap.

Fibonacci heap

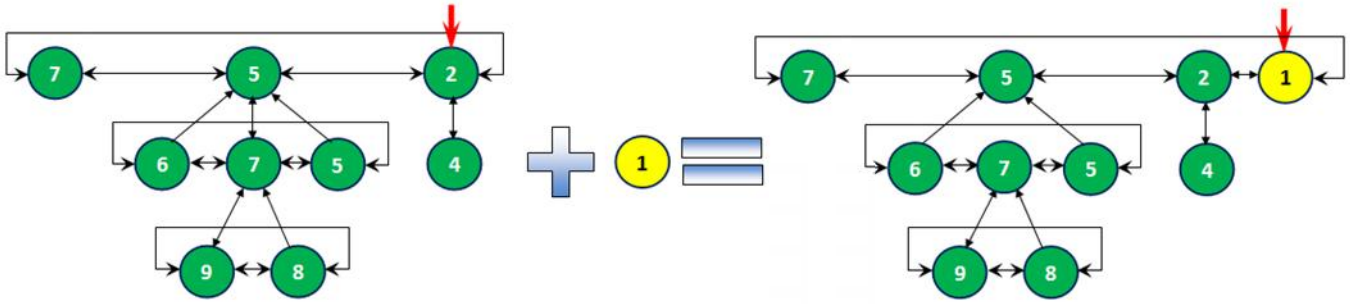
Search of the minimal element

- As in the case of a binomial heap minimum element of the Fibonacci heap would be one of the root nodes of the Fibonacci trees.
- Since the length of the annular list of root elements can be sufficiently large, to achieve the desired asymptotics it is easier to store a reference to the minimum of them.
- The complexity of this operation is $O(1)$.

Fibonacci heap

Adding a new element

- The new element is always added to the root list of items.
- In this case, it is convenient to make it a left or a right brother of the minimum element.
- It may happen that the new element is less than the minimum element.
- In this case, you need to update the pointer to the minimal element of the entire heap.



- The complexity of this operation is $O(1)$.

Fibonacci heap

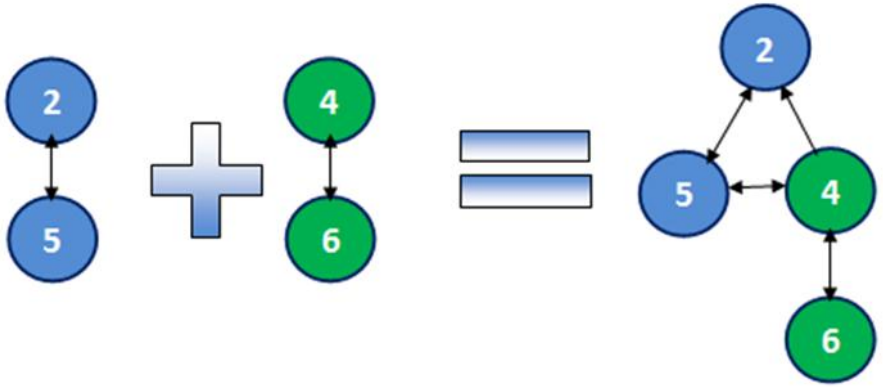
Combining the two Fibonacci heap

- To join the two Fibonacci heaps it is necessary to unite the annular lists of their root elements.
- Pointer to a minimal element of the combined heap should be chosen among the minimal elements of the original heaps.
- This can be done in constant time.

Fibonacci heap

Combining the two Fibonacci trees

- You can combine any two Fibonacci trees, but to achieve a logarithmic complexity of operations using the operation as auxiliary, you should combine the two Fibonacci trees of the same size.

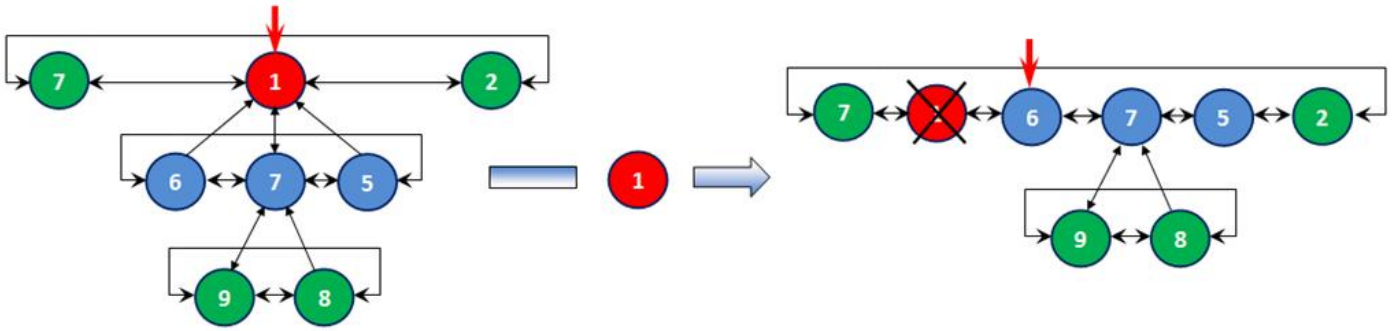


- The minimum of the roots of the united trees will be the root of the united tree.
- The second root will be the son of the minimum root.

Fibonacci heap

Removing the minimum element

- This operation is divided into several stages:
- If the deleted element has children, their annular list is combined with an annular list of root elements.



Joining the element that is being removed to the root annular list of child elements.

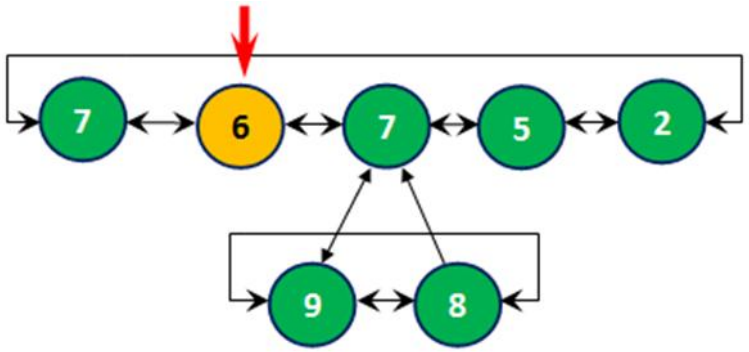
- Removing the minimum element from the root list.
- The link to the minimal element of the heap will point to the right sibling of the removed item, but not the fact that this element is minimal.

Fibonacci heap

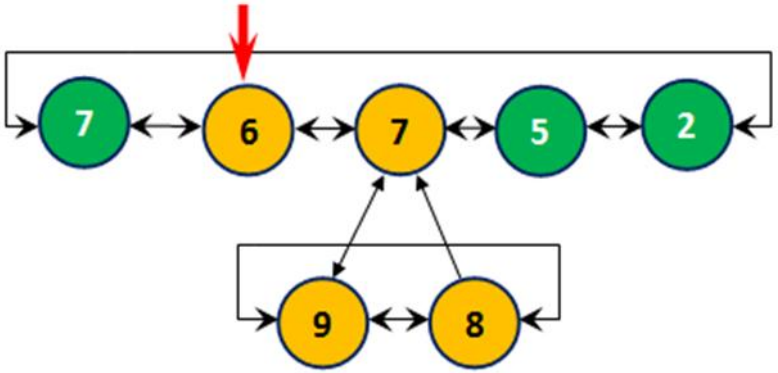
- Sealing the pile and finding the minimum element.
- Sequentially we iterate through all the Fibonacci trees, starting with the tree containing the minimum element of the entire heap.
- If at some stage there are two trees of the same size, they need to be combined.
- For the merged tree again it is necessary to check whether among the previously considered trees there is a tree with the same size, and if there is - then repeat the operation.

Fibonacci heap

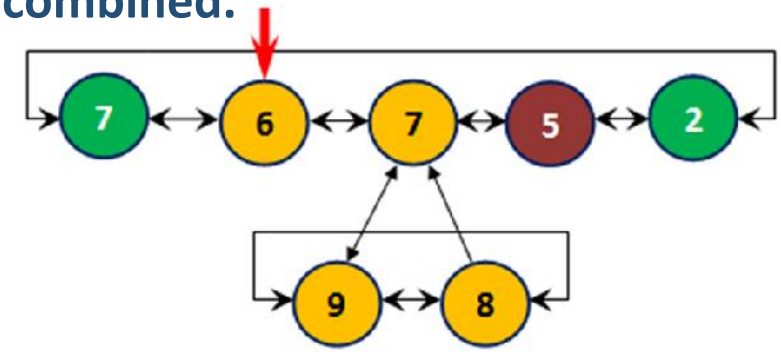
1. Begin viewing the trees from the tree "6"



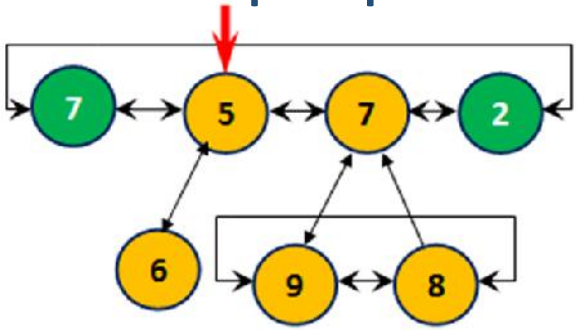
2. Tree "7-9-8" has a unique size among the previously considered.



3. Tree size of "5" coincides with the size of tree "6". They should be combined.

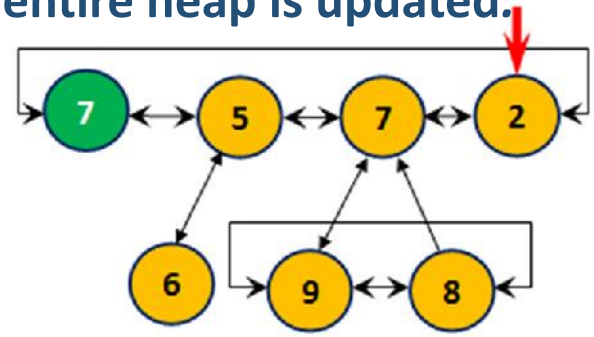


4. Combining trees «6» and «5». As a result, the minimum element of the whole heap is updated.

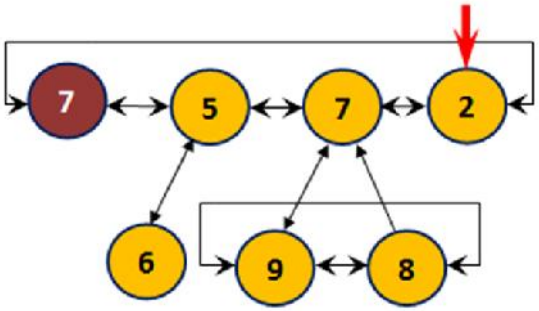


Fibonacci heap

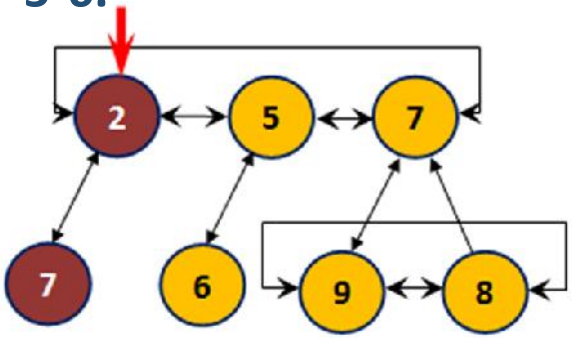
5. New tree "2" has a unique size. The minimum element of the entire heap is updated.



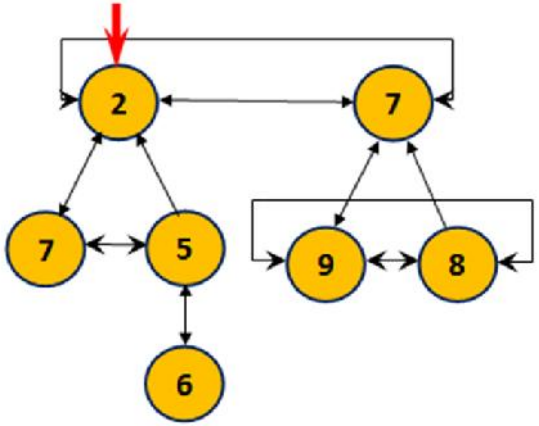
6. New tree "7" should be combined with the tree "2".



7. The combined tree "2-7" should be combined with the tree "5-6."



8. All of the trees were considered. Sealing the heap is over.



Complexity of the operation is **O(logN).**

Fibonacci heap

Changing the element

- If the value of the new element has increased it is necessary to make sifting down, i.e., "If the current element is less than the minimum of his sons, they are swapped."
- If the value of the element has decreased so much that it has become less than its parent, it is necessary to remove the element from the annular list of brothers, break the link with its current father and put it in the root element of the annular list.

Deleting an element

- The element is easier to remove by making it the minimum element of the whole heap, and then to perform an operation of removing the minimum element.