Data Structure (II)

Luo Tsz Fung {`pepper1208`}

2025-06-26

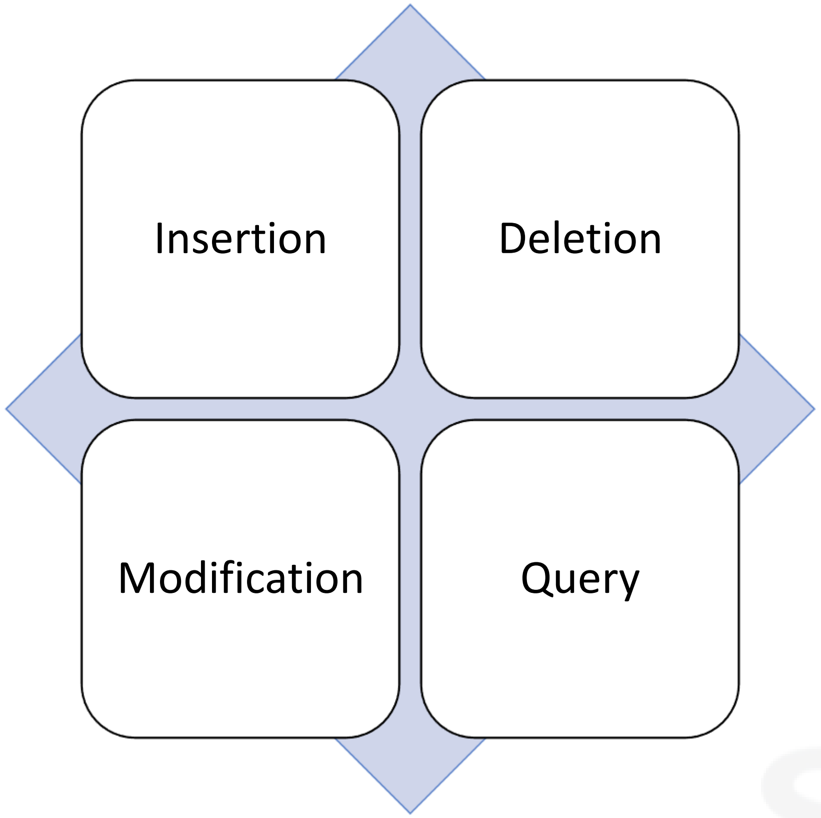
Data Structure

A data structure is a way to organize and store data so that we can perform operations on the data **efficiently**.

However, we cannot measure the efficiency unless a specific scenario is given.

Therefore, it is of vital importance for us to remember that **different data structures are required for different situations**.

Common Operations



Common Operations

What is the trivial upper bound of time complexity of insertion, deletion, modification and query for *N* data in a data structure?

O(N).

Therefore, we often aim for sublinear time like O(log N), O(1) or amortized time complexity in order to finish all O(Q) operations within time limit.

Content

Content

Binary Heap

Problem: Implement a data structure which supports:

- Insert an integer *x* inside the data structure.
- Output the minimum value of elements inside the data structure.
- Delete the element with the minimum value in the data structure.

Binary Heap

Approach 1: Insertion O(1), Deletion O(1), Query O(N)

Method: Implement two arrays storing the content and the valid flag of each content.

Approach 2: Insertion O(N), Deletion O(1), Query O(1)

Method: Implement an array and maintain its monotonicity. Insertion sort is made for each insertion.

Is there any better approach?

Approach 3: Use a binary heap.

Binary Heap

A binary heap is a **complete binary tree**. It is often implemented by an array.

- The root element will be at arr[1], which is the 1st node.
- The left child of the *x*th node is arr[2 * x].
- The right child of the *x*th node is arr[2 * x + 1].
- Obviously, the parent of the *x*th node is arr[x / 2].
- A common practice is that the size of the array is reserved to be 4*N* elements.

Binary Heap

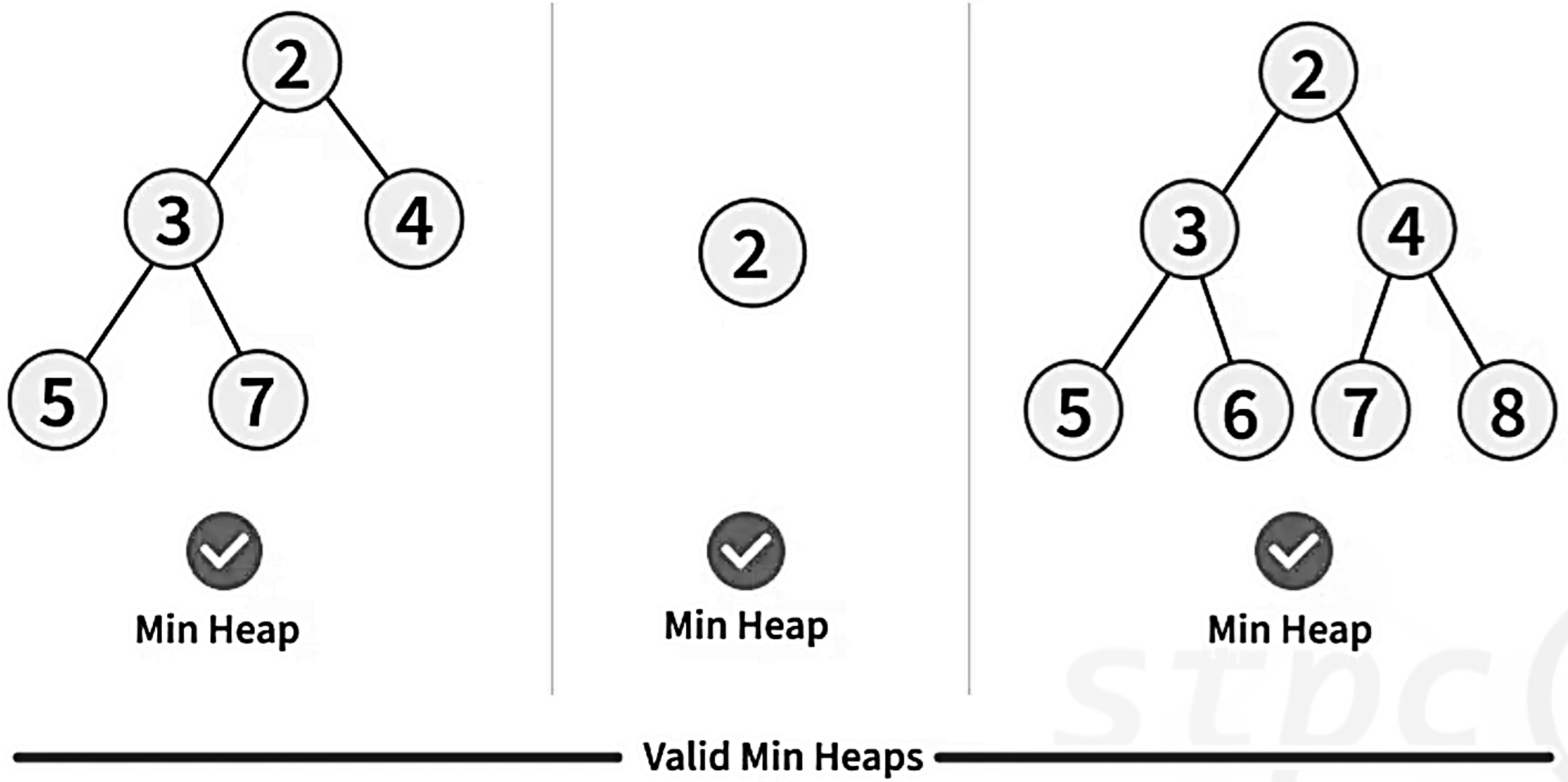
There is one additional rule on the binary heap tree: **the value of the parent node must not be larger than any value of its child node**.

As the value of the root node must be the maximum value among all nodes, the binary heap is called the **min heap**.

You can implement a **max heap** by some modification.

The depth of the binary heap tree must be log N.

Binary Heap



Binary Heap

Query: O(1)

Method: Return the value of the root node.

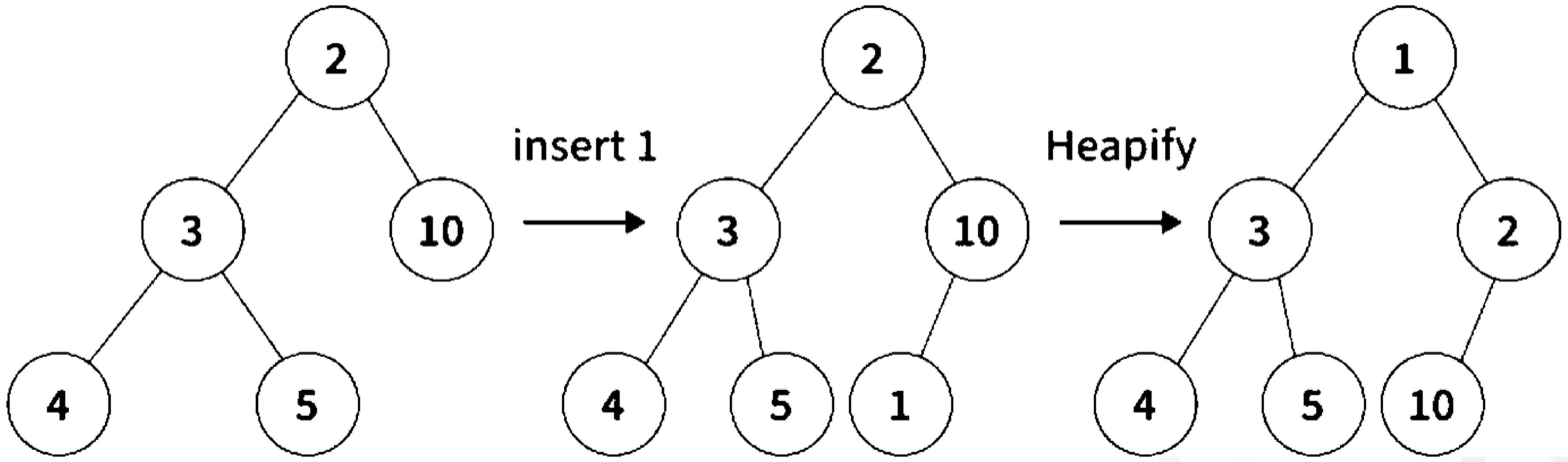
Binary Heap

Insertion: O(log N)

Method: Insert the element to the “back” of the heap tree. Then, **heapify** the tree.

- If there is *N* items in the heap, then insert the element in arr[N + 1].
- Then, “sift-up” the element to its parent whenever possible, which swap two nodes if the parent node is larger than the child node.

Binary Heap



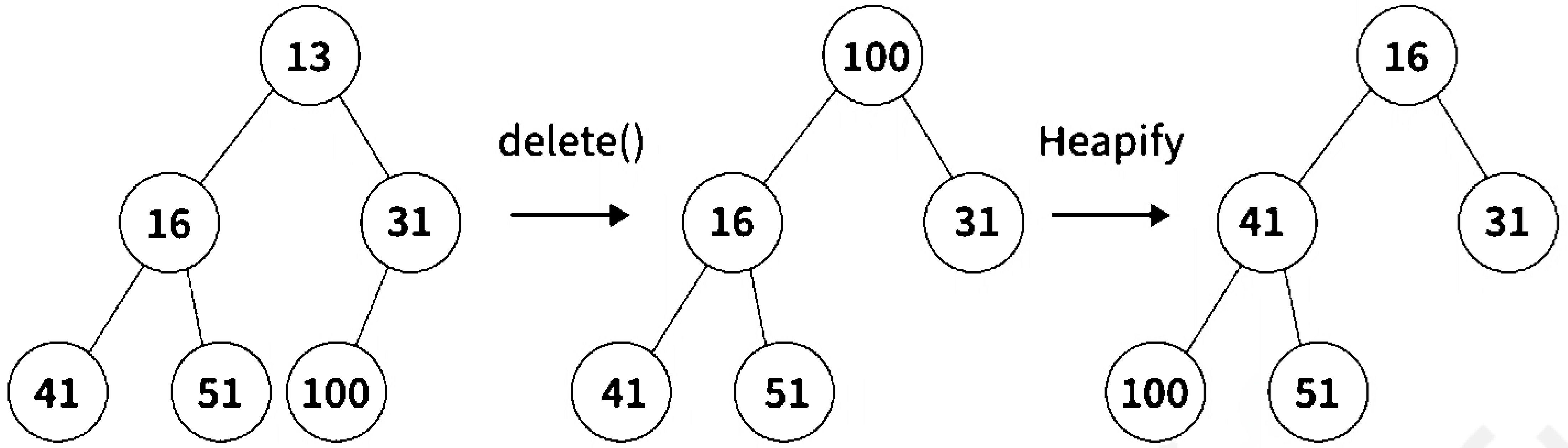
Binary Heap

Deletion: O(log N)

Method: Modify the element of the root node to be the "back" node. Then, delete the back node. After that, heapify the tree.

- If there is *N* items in the heap, then update the value of arr[1] as arr[N]. Then, delete arr[N].
- Then, "sift-down" the root whenever possible, which swap two nodes if the parent node is larger than the child node.

Binary Heap



Binary Heap in C++

Nevertheless, C++ STL supports the binary heap container. Please refer to the C++ implementation of the binary heap.

Practice Problems

- Binary Heap
- [NOIP-S 2004] 合并果子
- Merge The Array
- [NOIP-S 2016] 蚯蚓
- [JSOI 2007] 建築搶修

Binary Heap Trick: K-th Minimum

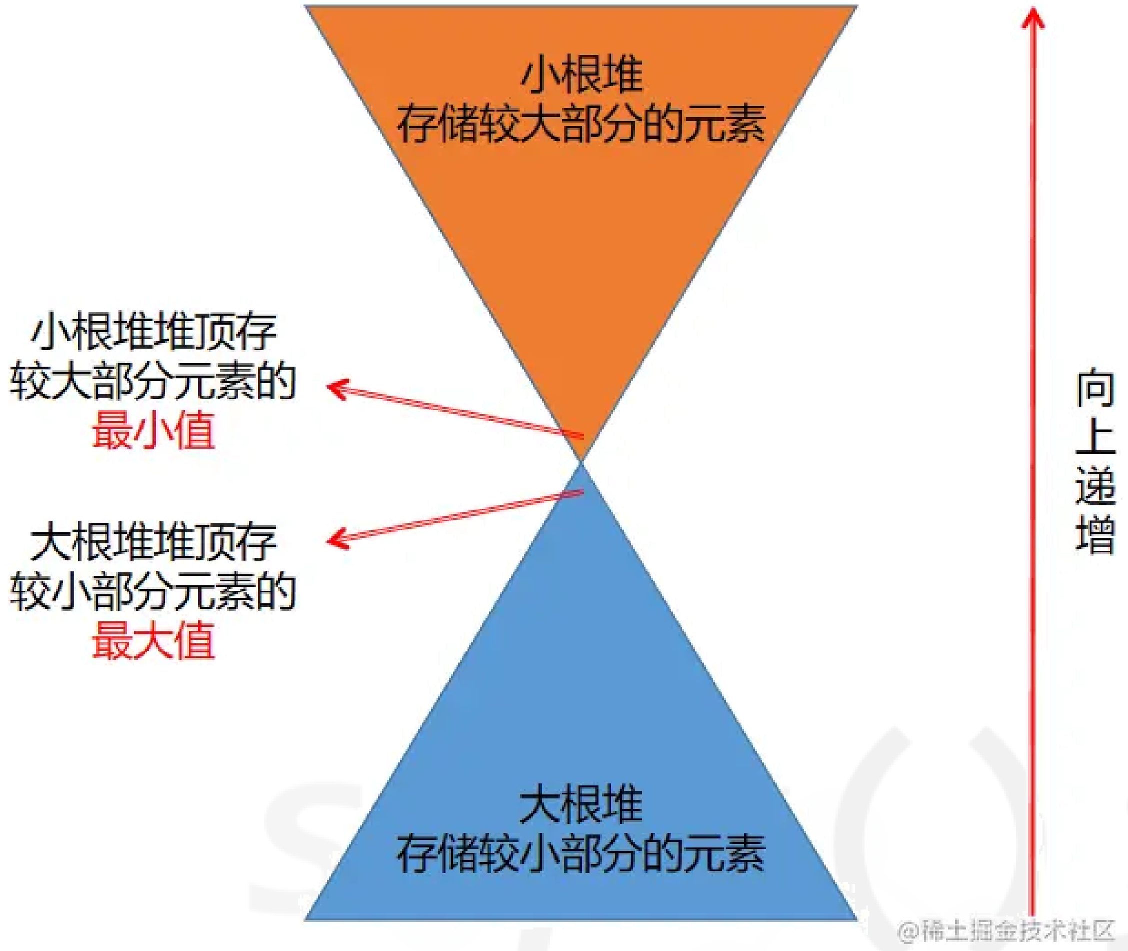
Problem: Implement a data structure which supports:

- Insert an integer *x* inside the data structure.
- Output the **K-th** minimum value of elements inside the data structure.
  - K may be updated dynamically.

Binary Heap Trick: K-th Minimum

We can use **double heaps trick** to tackle this problem, consists of a max heap and a min heap.

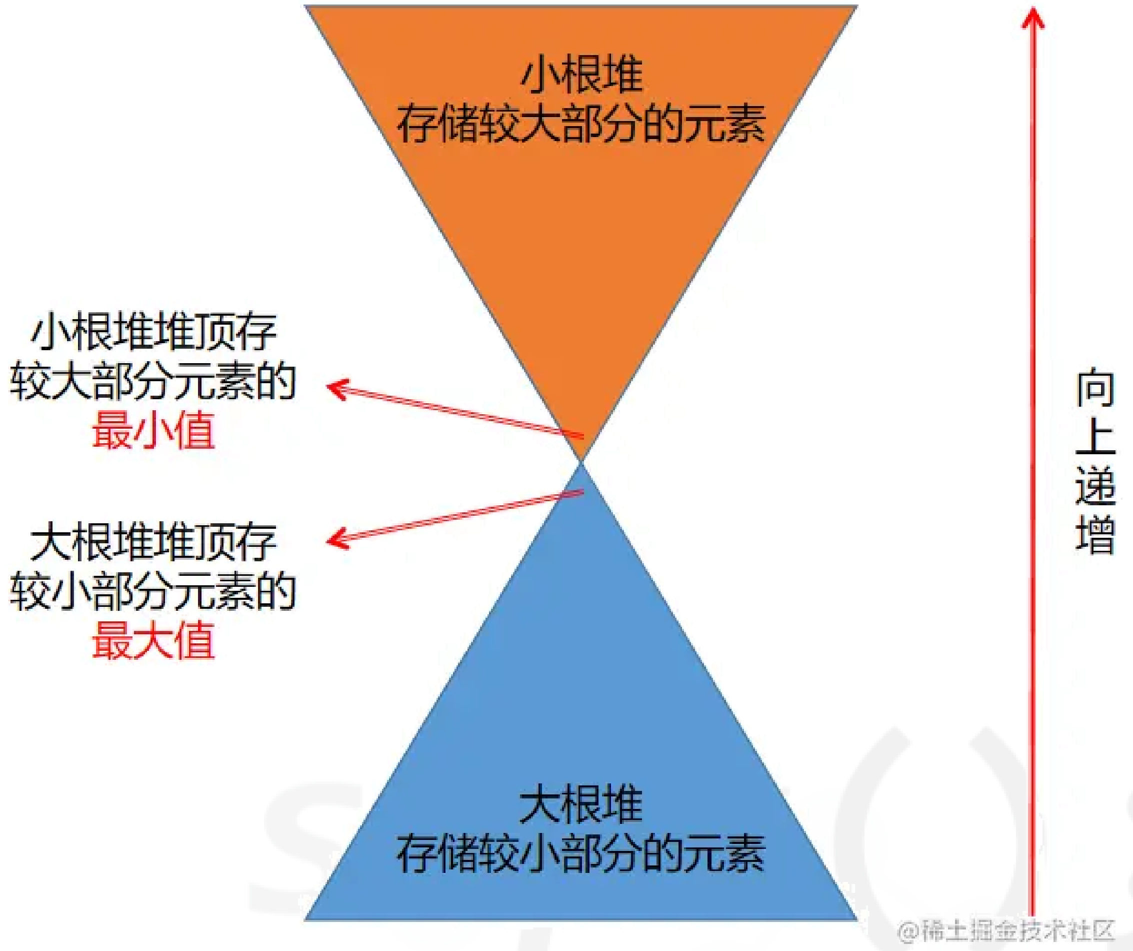
The double heaps trick can effectively maintain a sequence with additional monotonicity



Binary Heap Trick: K-th Minimum

Double heap trick:

- If the size of the max heap exceeds K, pop the top of the max heap and push the element into the min heap.
- If the size of the max heap is under K, pop the top of the min heap and push the element into the max heap.

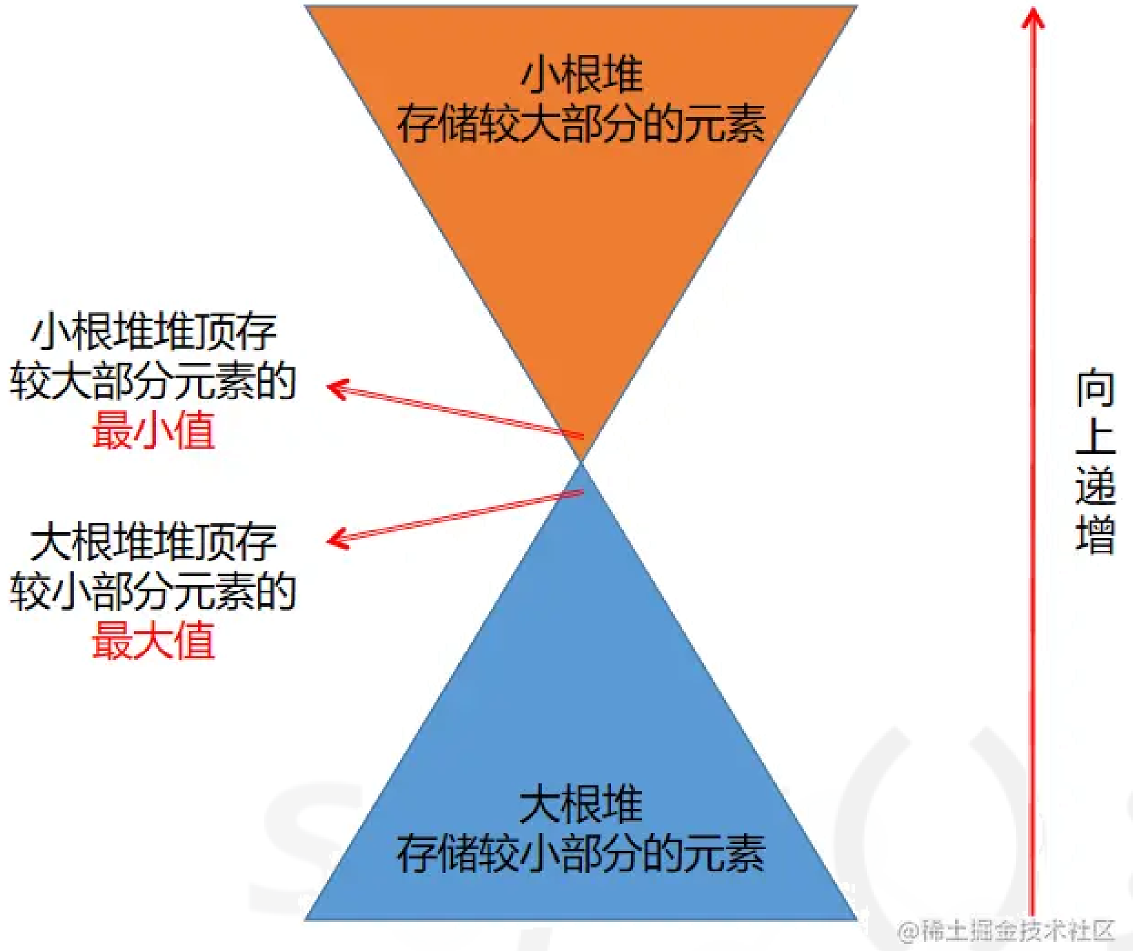


Binary Heap Trick: K-th Minimum

How can we use the trick to find the K-th minimum elements?

We can insert all the elements into the max heap. After each insertion, perform the double heap trick.

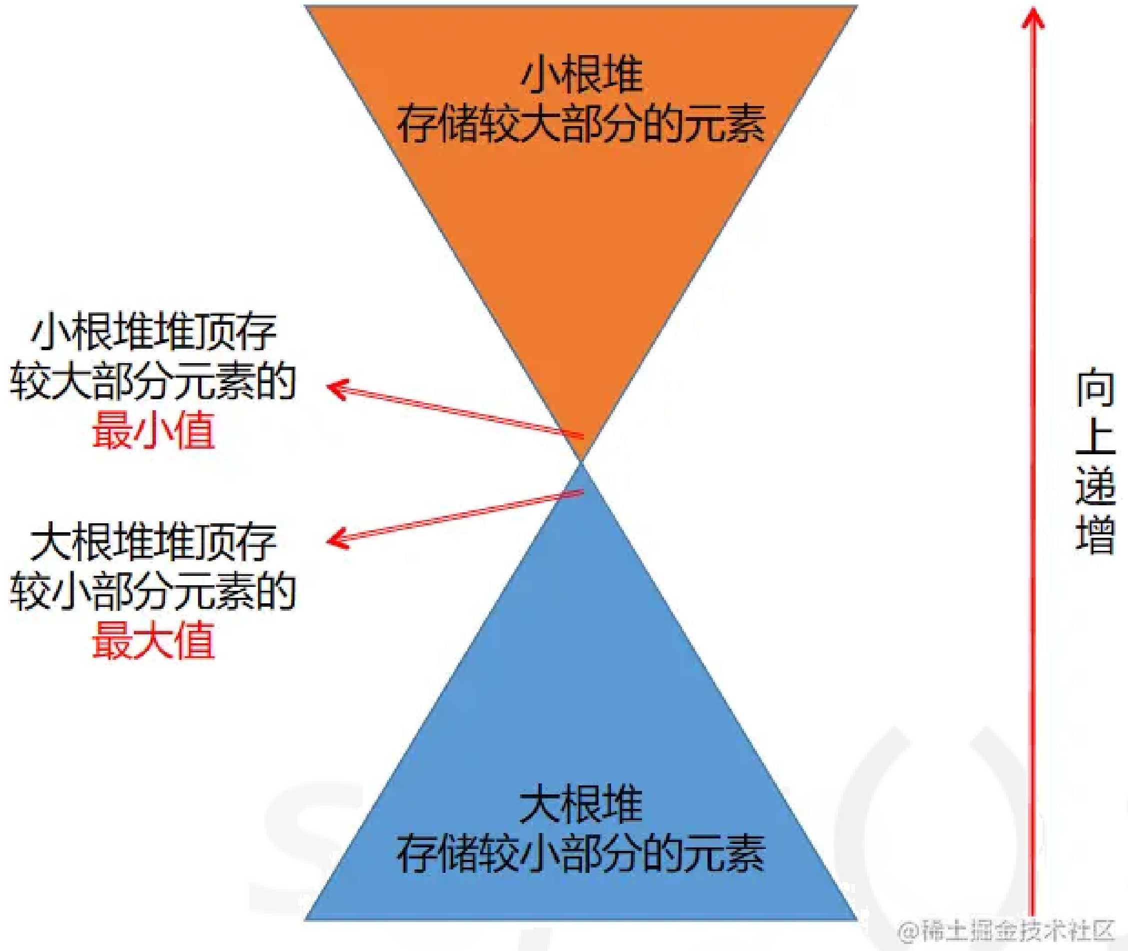
The top of the max heap is the answer desired.



Binary Heap Trick: K-th Minimum

Actually, the idea can be extended flexibly as our main idea is “retrieving a critical element”.

The idea can be extended to solve the Running Median problem, which is an alternative form of the K-th Minimum problem.



Practice Problems

- Running Median
- Black Box
- Smallest Values of Functions

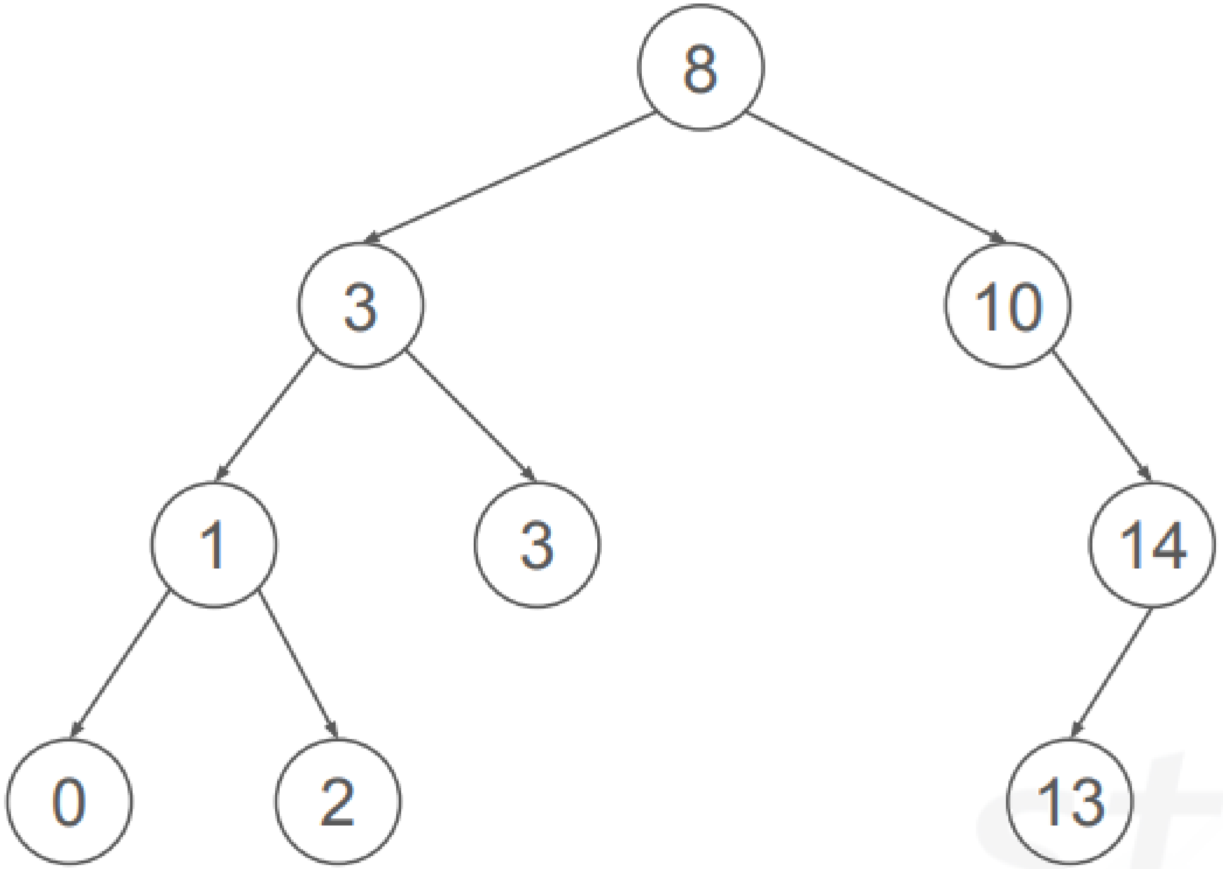
Content

Binary Search Tree

Binary search tree is a binary tree where

- Value of all nodes in the **left subtree** of a node K must be **smaller than** the value of node K
- Value of all nodes in the **right subtree** of a node K must be **bigger than or equal to** the value of node K

Binary Search Tree



Binary Search Tree

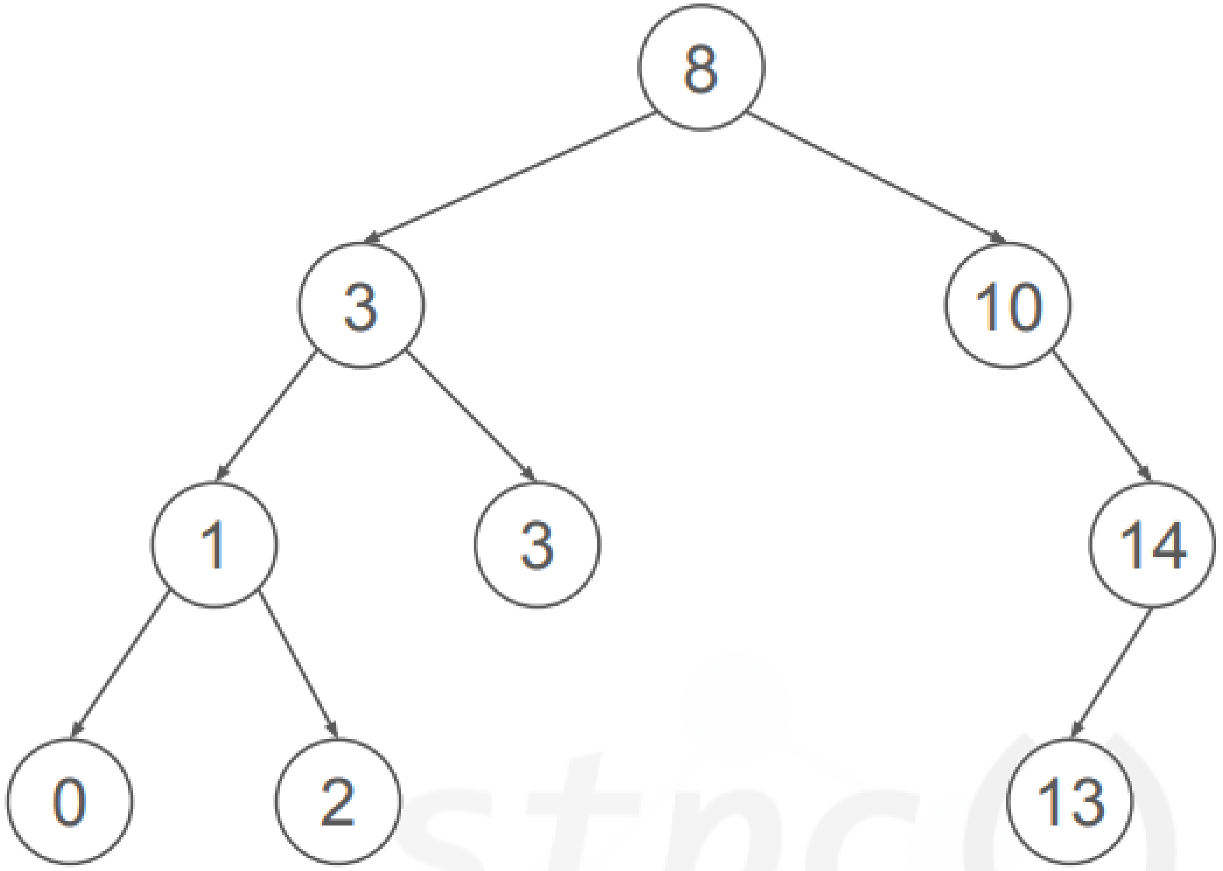
Compare to heap:

- Heap only maintains larger / smaller relationship between parent and child.
- BST maintain the **complete order** of elements.
- This mean we know how to locate a value, somehow putting a sorted array in a tree structure.
- Operations supported: insertion, deletion, query minimum, query maximum, find specific elements, binary search

Binary Search Tree

Insertion

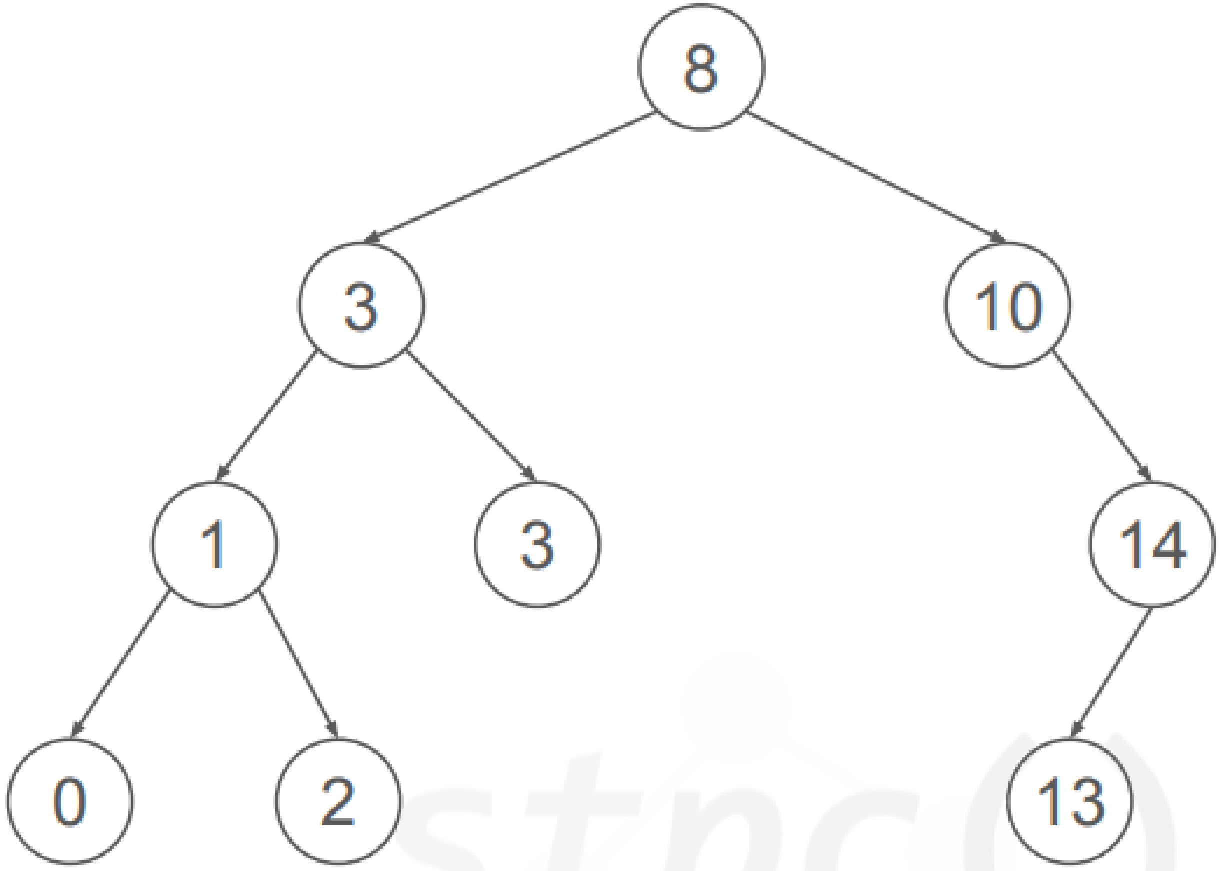
- DFS from the root
- Repeatedly travel down the tree: If the inserted value is smaller than the current node’s value, go left; go right otherwise
- Insert when we find an empty space



Binary Search Tree

Find

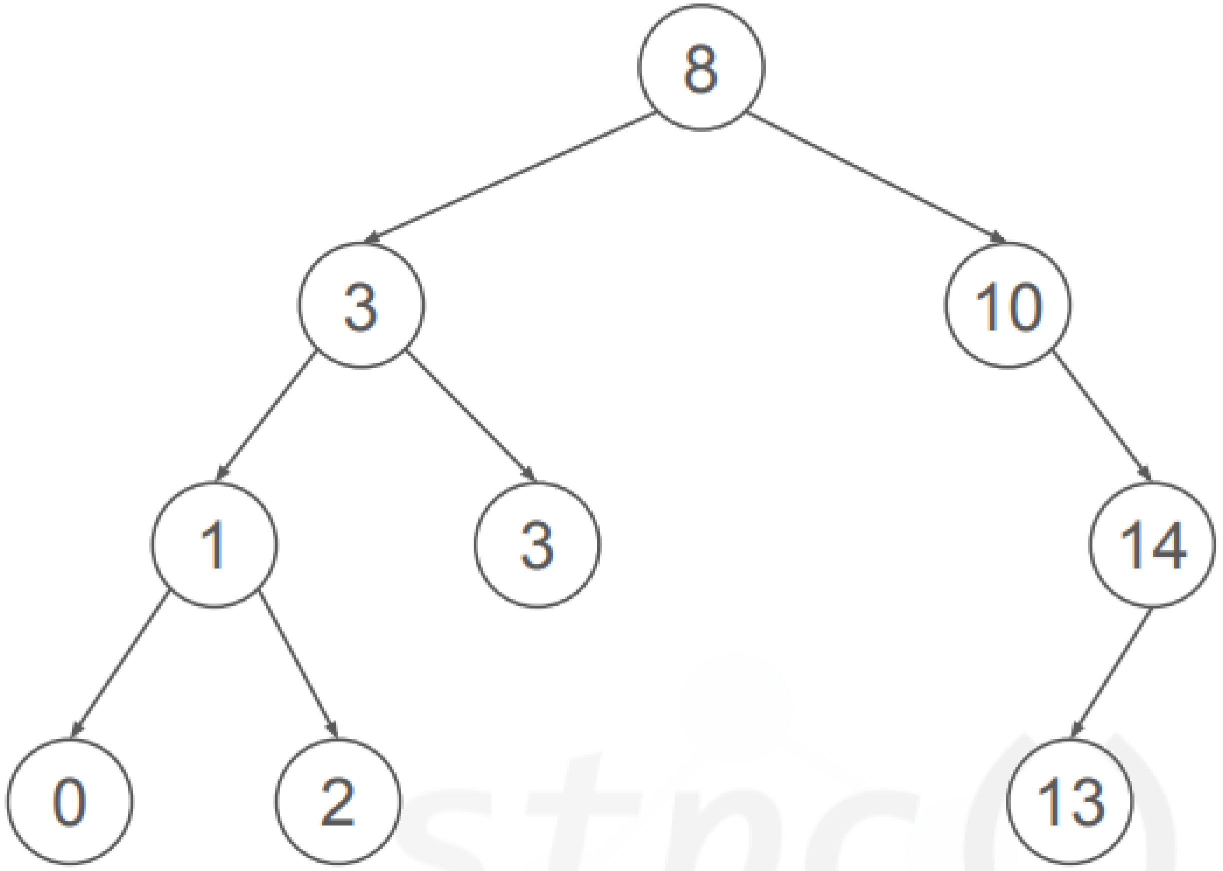
- DFS from the root
- Repeatedly travel down the tree: If the inserted value is smaller than the current node's value, go left; go right otherwise
- Until the value is found



Binary Search Tree

Query Min

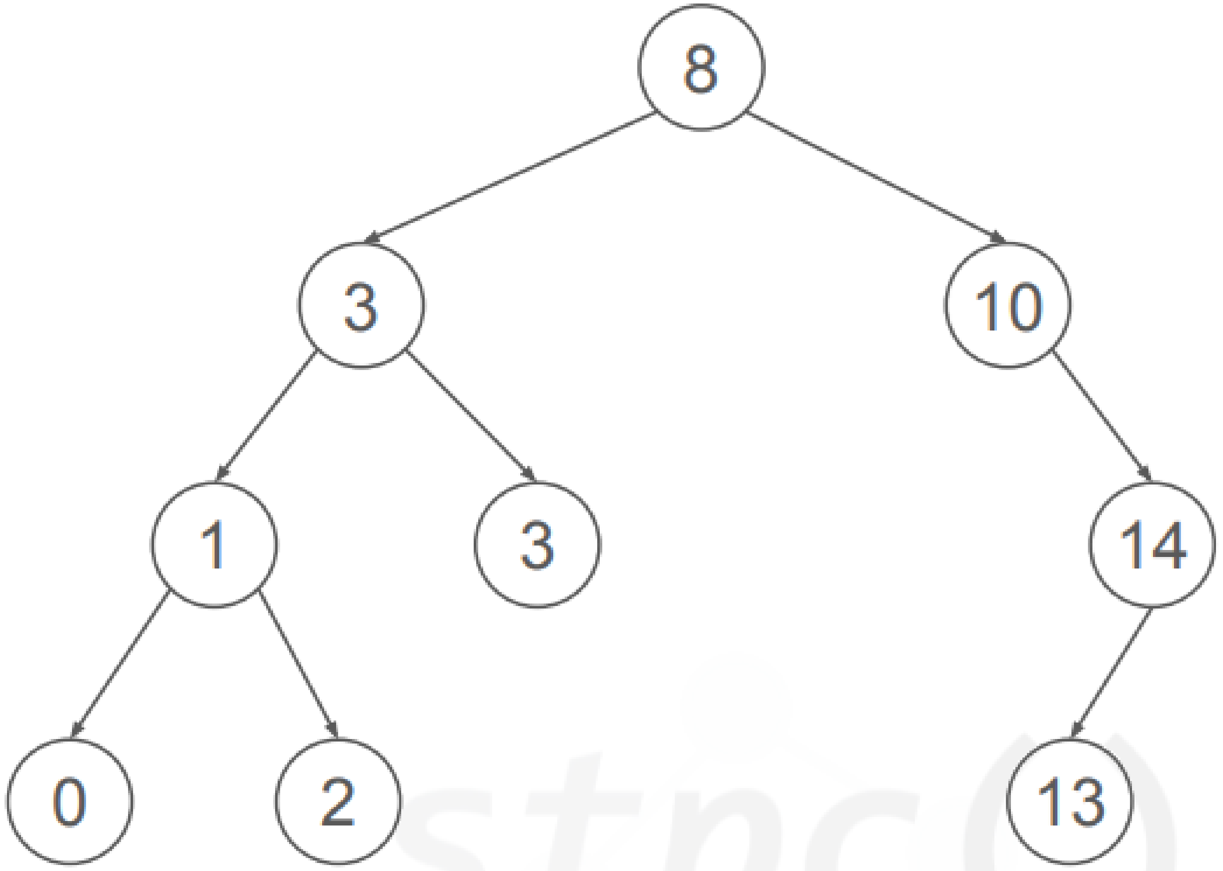
- Find the leftmost node



Binary Search Tree

Query Max

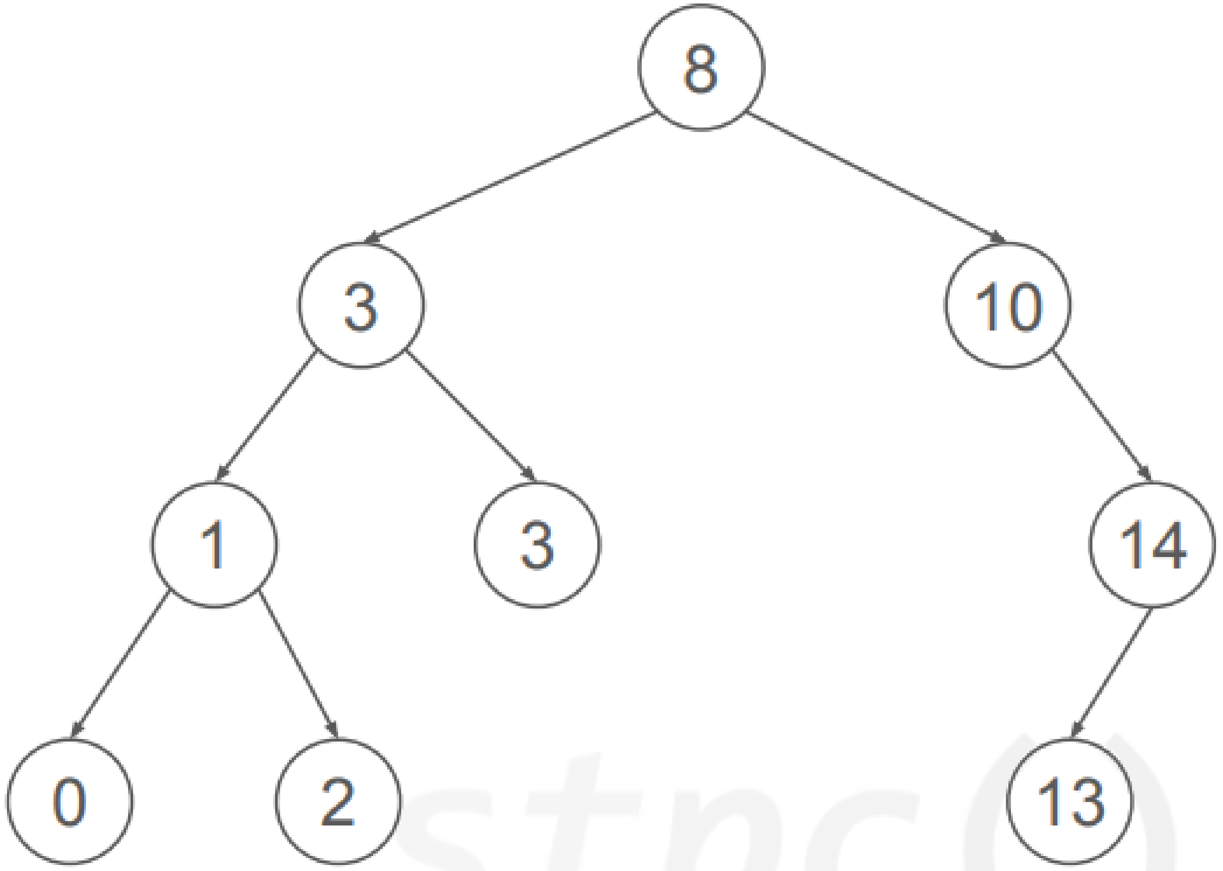
- Find the rightmost node



Binary Search Tree

Find element >= lower_bound

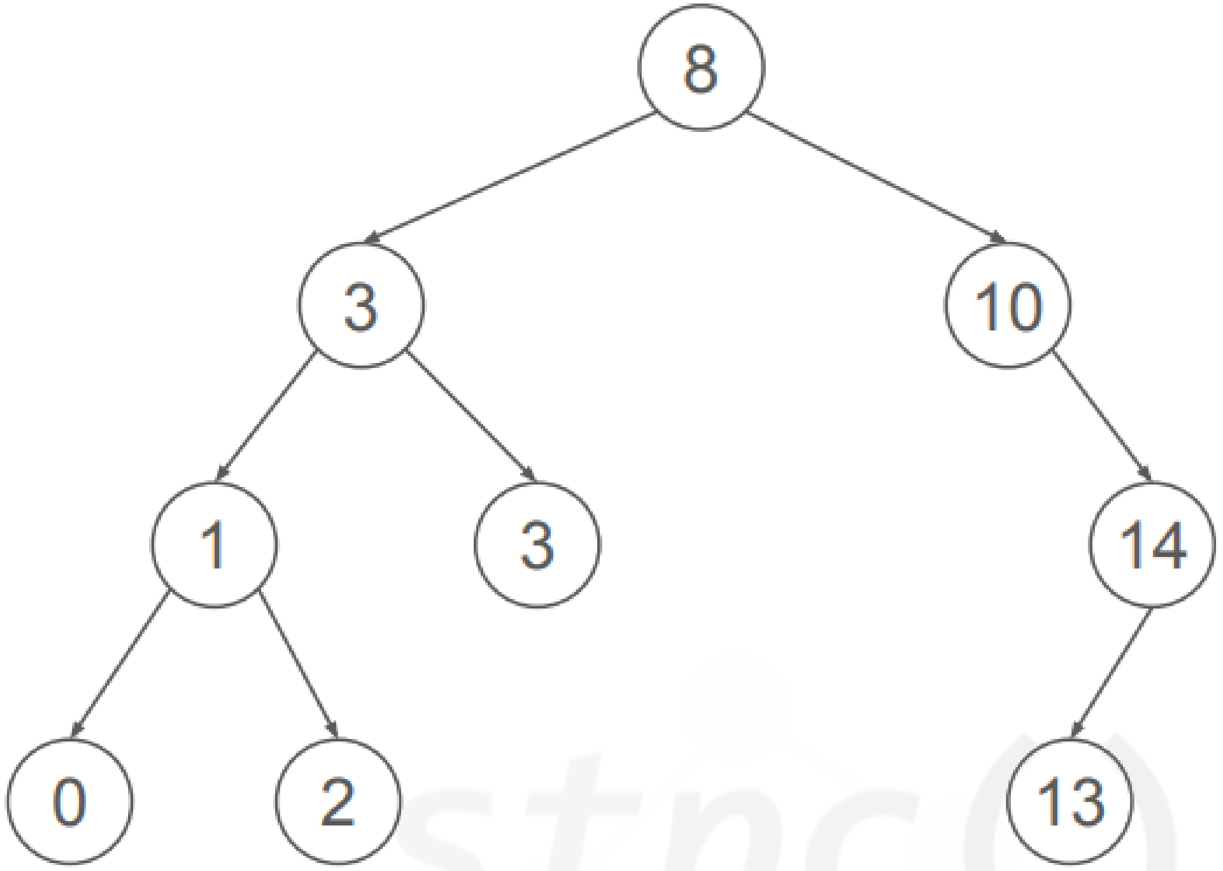
- DFS from the root
- If current_value >= lower_bound
  - Res = current_value
  - gotoLeftSubtree;
- Else
  - gotoRightSubtree;



Binary Search Tree

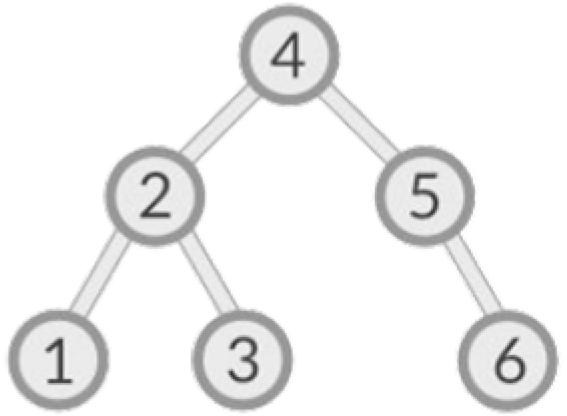
Delete

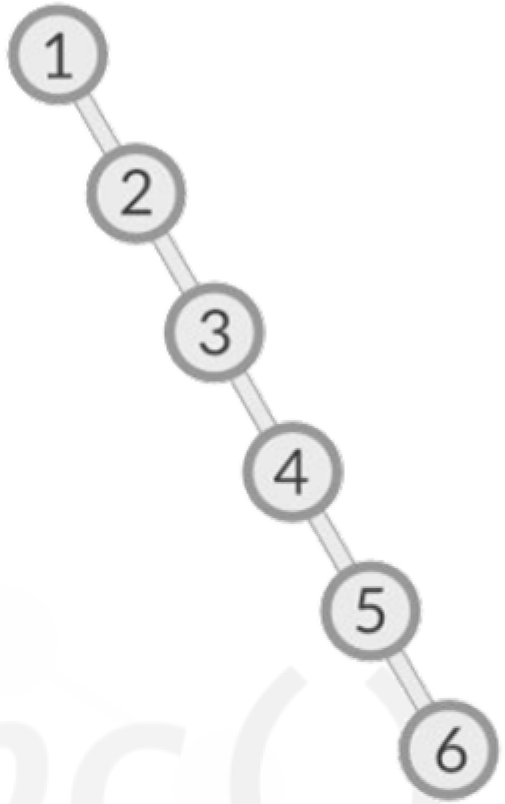
- Locate the to-be-deleted element
- If it is a leaf, delete directly
- If it has a left subtree, swap it with the largest element in its left subtree
- If it has a right subtree, swap it with the smallest element in its right subtree
- If it has both left and right subtrees, perform either one operations above
- Do it recursively until the to-be-deleted element is deleted as a leaf.



Binary Search Tree

- Trivially, the time complexity of all operations cost O(height of BST).
- What is the worse time complexity of all operations?
- O(N), as the tree may be imbalanced and form a chain.





Binary Search Tree

How can we avoid such case?

- Shuffle the element before insertion may help.
- Use self-balancing BST (Red-black tree, AVL tree, Treap, Splay tree, etc.).
  - BST with self-rotation, very hard to code.
- Use other search tree wherever suitable (Trie, Segment tree).

Binary Search Tree in C++

Nevertheless, C++ STL supports the binary search tree container. Please refer to the C++ implementation of the binary search tree.

In C++ STL, binary search tree container is implemented as a red-black tree, with all operations cost O(log N) time complexity.

However, be aware of the general usage of `std::set` and `std::map`. It can be sometimes convenience like the index of map can be customized!

Practice Problems

- Warehouse
- Student Management
- A-B Problem

Content

Disjoint sets

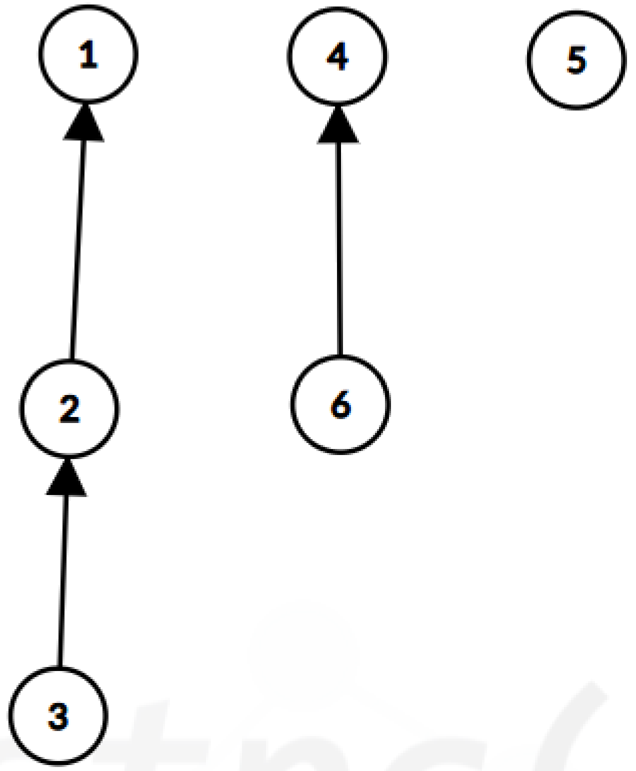
What is disjoint sets?

- One element belongs to exactly one group
- One group may consist of any number of elements
- For example, given six numbers 1, 2, 3, 4, 5, 6.
- {1, 2, 3}, {4, 6}, {5} are disjoint subsets.
- {1, 2, 3}, {2, 4, 6}, {5} are not disjoint subsets.

Disjoint sets

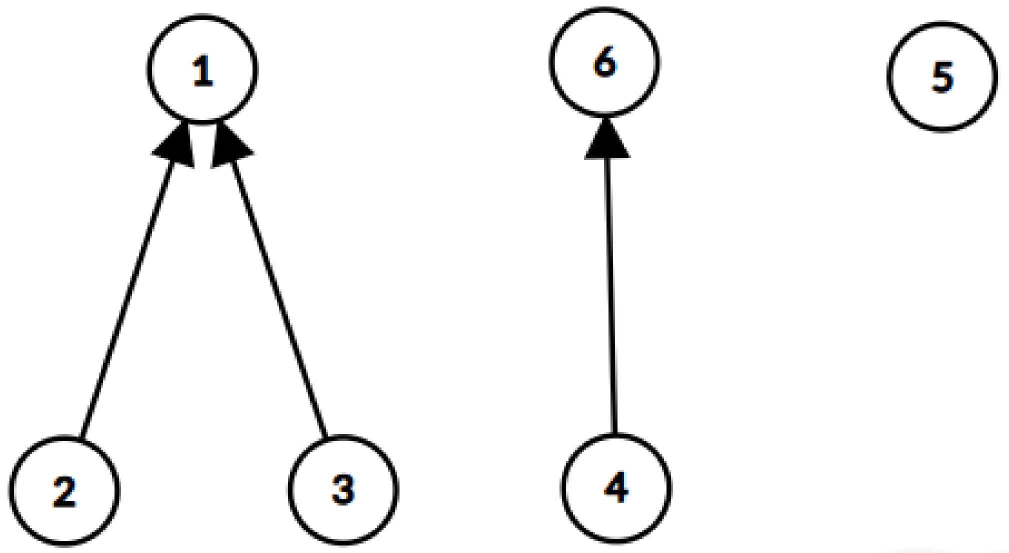
We can represent disjoint groups by graphs. It is not necessary for us to assign an “index” for each group, as all groups are disjoint.

Therefore, by stating one of the elements in each group is distinguishable enough. They act like the “leader” of the group.


$$\begin{tikzcd} 1 & 4 & 5 \\ 2 \arrow[u] & 6 \arrow[u] & \\ 3 \arrow[u] & & \end{tikzcd}$$

Disjoint sets

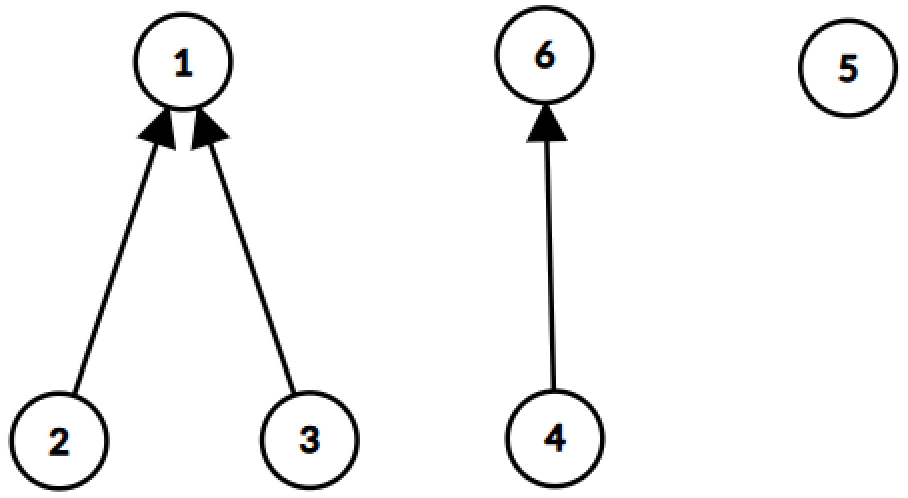
- This graph shows the same disjoint structure as the previous one. We only care about the “groups” only.

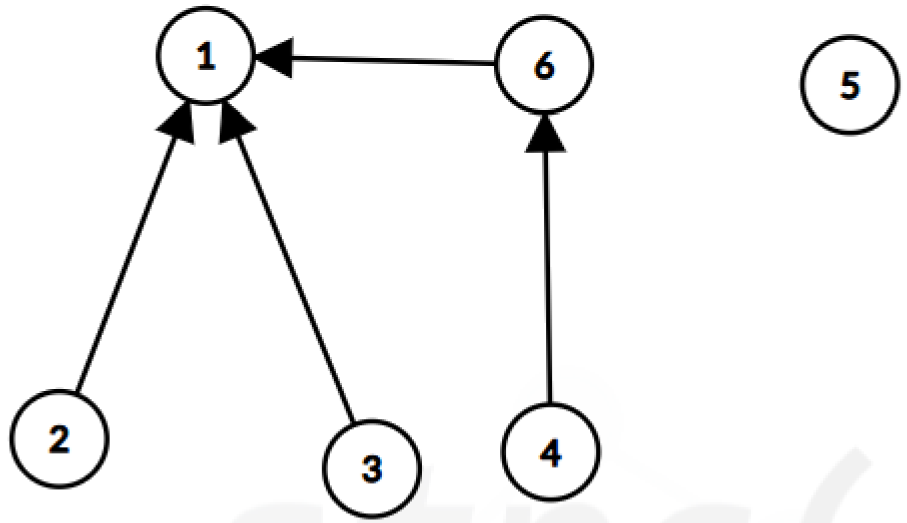


Disjoint sets

How can we merge two groups?

- Merge a “leader” from one group to another group.

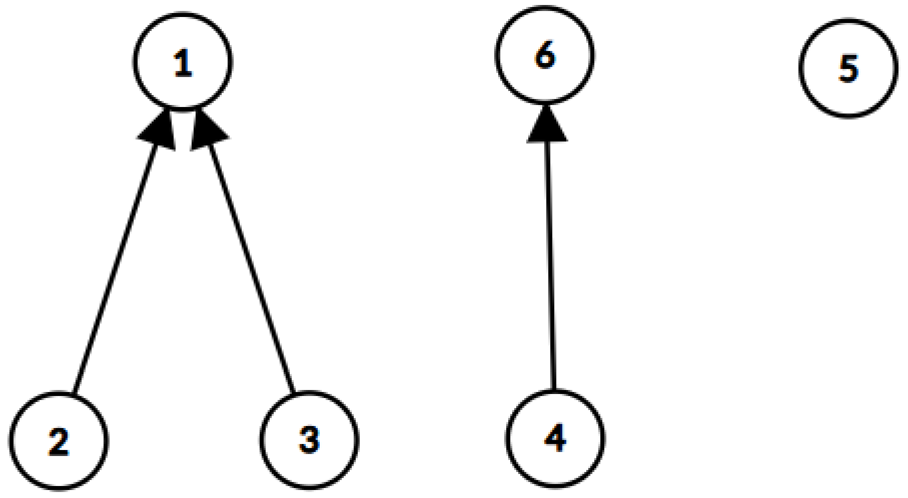


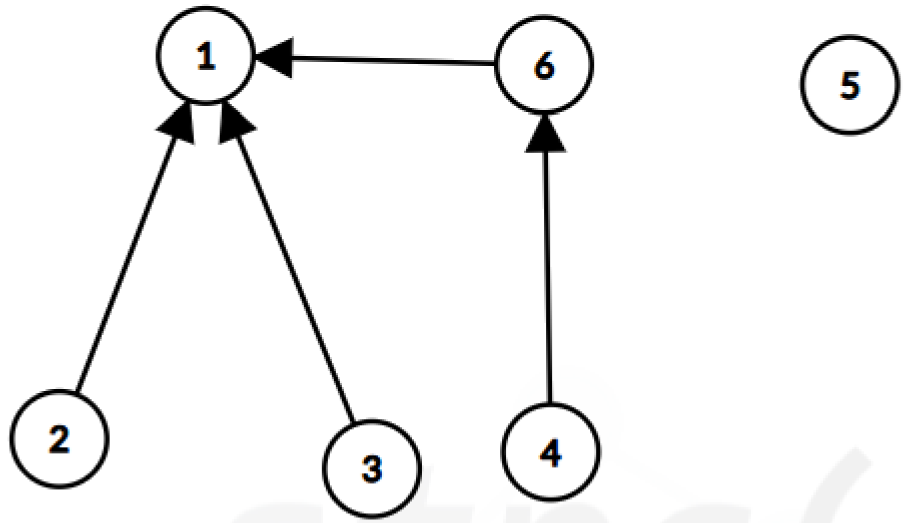


Disjoint sets

How can we find the leader of the group where element *x* belongs to?

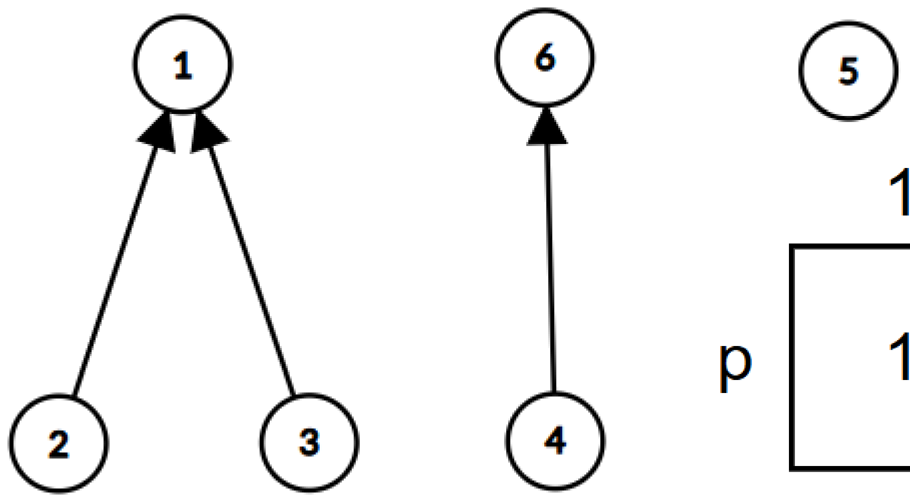
- Simply search.





Disjoint sets union-find

Maintain an array p[i] which represents the group leader of element i.



	1	2	3	4	5	6
p	1	1	1	6	5	6

Disjoint sets union-find

Merge operation: **union(u, v)** Time complexity: O(1)

Simply merge a leader to a group.

Find operation: **find(u)** Time complexity: O(N)

We need to recursively find the parent of **u** until **p[u] = u**.

The worst case appears if the “tree” is a chain.

Path compression

We can optimize the find(u) operation such that its time complexity of find operation will have amortized O(log N) time complexity.

During finding root of element u, simultaneously update the parents of visited elements.

The proof of the time complexity is quite tedious. If you are interested, search yourself. ☺

Disjoint sets union-find in C++

Unfortunately, there is no C++ STL implementation for DSU.

However, DSU is not hard to code. Therefore, familiar with the template. ☺

Practice Problems

- Relatives
- 村村通
- 保齡球
- [NOIP-S 2010] 關押罪犯 [拓展域並查集]
- Some Sets
- [BalticOI 2003] 團伙 [反集]
- [NOI 2015] 程序自動分析 [離散化]