Binary Search Tree (BST)

chchiang
chyen
Penguin07

Lab 8: Integer ordered set

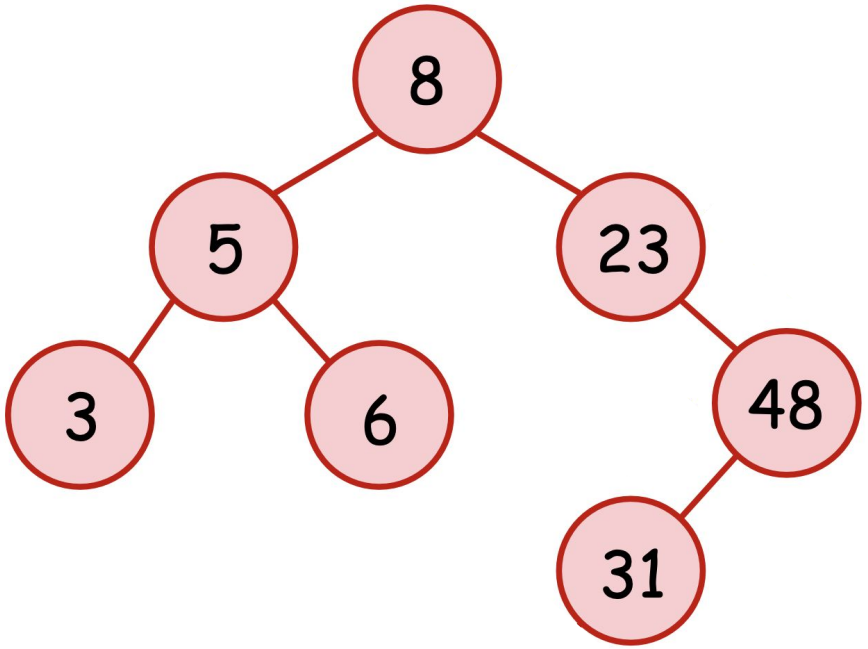
Maintain an integer set, which supports the following operations:

- `I` x: Insert x
- `D` x: Delete x
- `S` x: Search x
- `L` x: Lower bound of x
- `U` x: Upper bound of x

Insert x

- If x is **NOT** in the set, insert x into the set.
- Otherwise, do nothing.
- Print the number of integers in the set when the operation ends.

Insert x

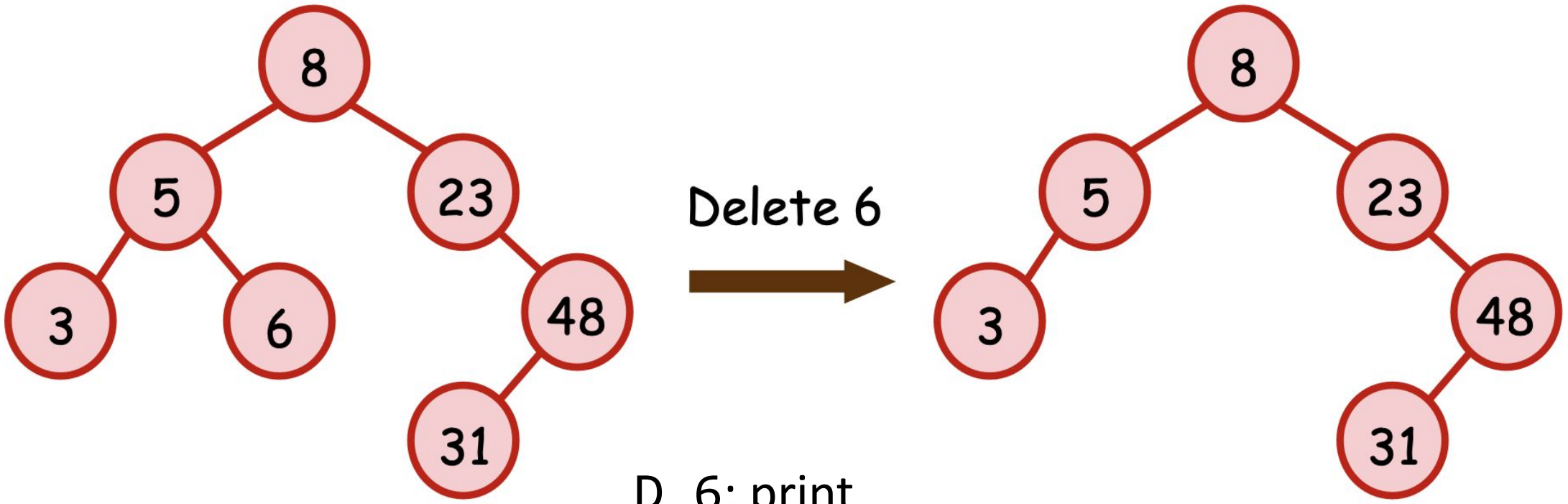


- I 31: print “7”
- I 30: print “8”

Delete x

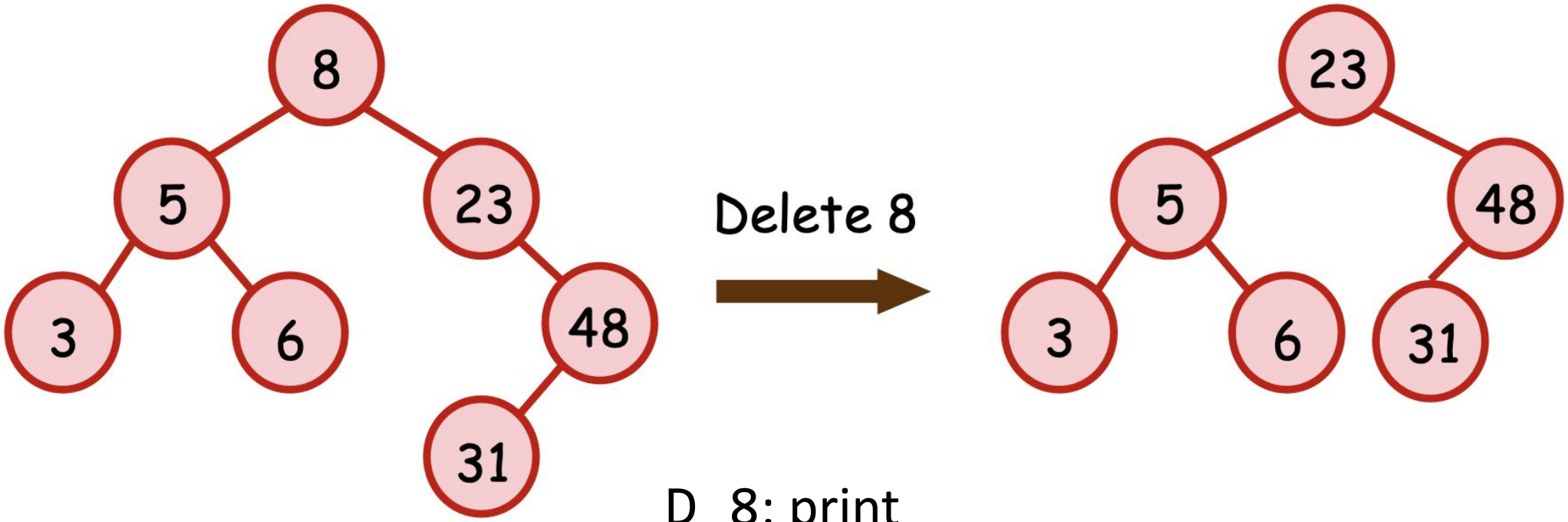
- If x is in the set, delete x from the integer set.
- Otherwise, don't do anything.
- Print the number of integers in the set when the operation ends.

Delete x



D 6: print "6"

Delete x



D 8: print “6”

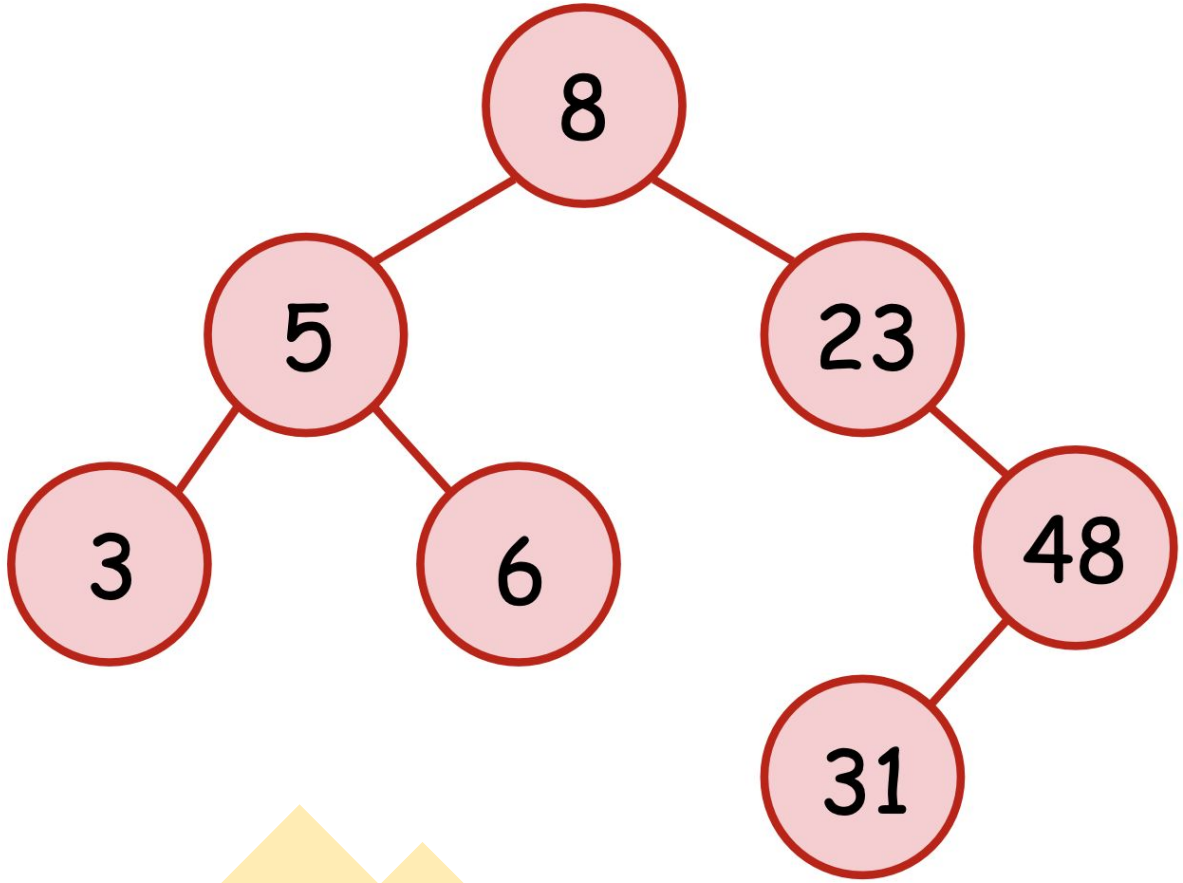
Search x

- If x is in the set, print "YES".
- Otherwise, print "NO".

Lower bound of x

- Print the smallest element **greater than or equal to** x in the set.
- If the element doesn't exist, print "-1".

Lower bound of x

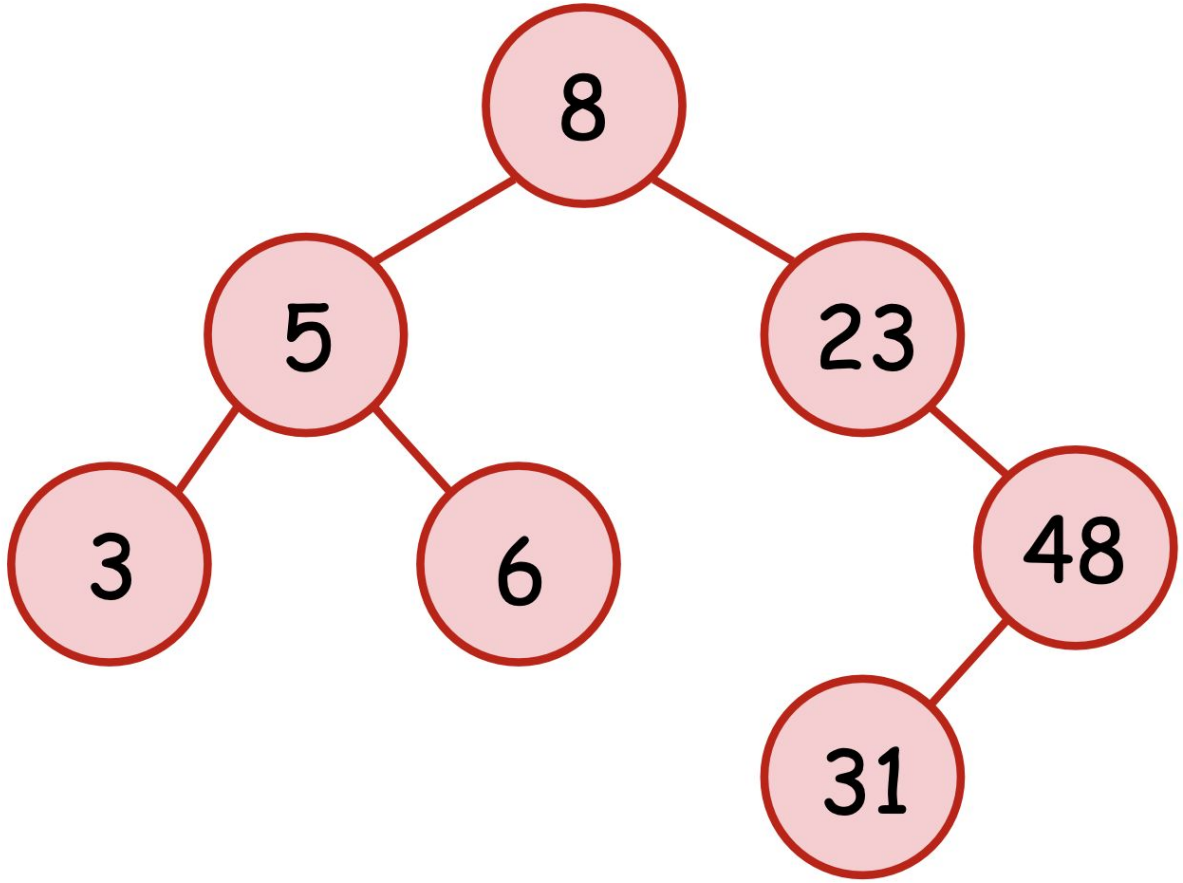


- L 31: print “31”
- L 47: print “48”
- L 50: print “-1”

Upper bound of x

- Print the smallest element **greater than** x in the set.
- If the element doesn't exist, print "-1".

Upper bound of x



- U 31: print “48”
- U 48: print “-1”
- U 20: print “23”

Set in C++

- Declare a Set : `set<T>s`
- Insert : `s.insert(x)`
- Delete : `s.erase(x)` (x can be a element, or an iterator)
- Size : `s.size()`

Set in C++

- lower_bound : `s.lower_bound(x)` (Return a iterator)
- find an element in set : `s.find(x)` (Return a iterator)
- Traverse all the element in set :
  1. `for(auto i : s){ ... }`
  2. `for(auto it = s.begin(); it != s.end(); it++){ }`

Multiset

- Declare a multiset : `multiset<T>ms`

- Others : Same as set

Note:

`ms.erase(x)` will delete all the elements with value `x` in `ms`.

You should write `ms.erase(ms.find(x))` to delete exactly one element (But make sure `x` is in the multiset)

Multiple integer ordered set

Maintain a multiset, supporting the following operations :

- I x : Insert x to the multiset
- D x : Delete one of x in the multiset
- C x : Print the occurrence of x in the multiset
- L x : Lower bound
- U x : Upper bound

Multiple integer ordered set

We can use a set with data type pair<int, int> the store the element’s value and it’s occurrence.

Map

Every element is a pair, the first element in pair is called “key”, the second is called “value”

- Declare a map : `map<T1, T2>mp;`
- Insert a element : `mp[key] = value`
- Erase a element : `mp.erase(key)`

Map

- Access the value of a key : `mp[key]`
  note: If key is not in the map, it will construct a pair automatically

- Another way : `mp.find(key)` (Same as set)

- The iterator of map :
  If you dereference the iterator, the type will become pair<T1, T2>

Implementation

- Write a BST
- http://codepad.org/yQD4UJA7

- P1 using `set<int>`
- http://codepad.org/dO9VJ82R

- P2 using map<int, int>
- https://pastebin.com/1eGTrrwN

Think about it (recommended)

- `V x`: Print the **second** smallest element greater than x in the set.
- `W x`: Print the **third** smallest element greater than x in the set.

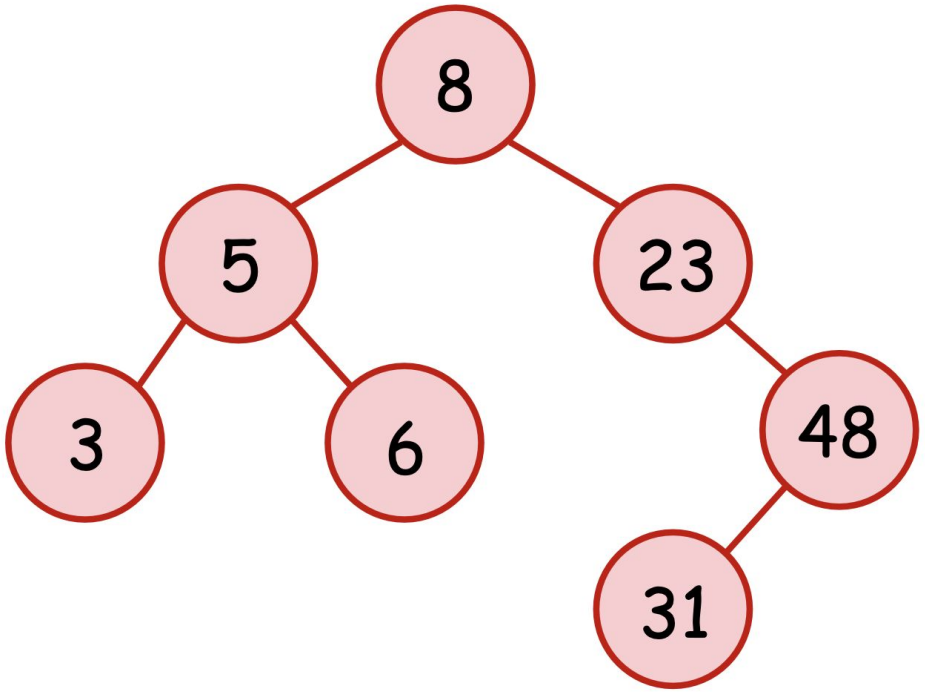
- Hint: If y = U x, then U y = V x

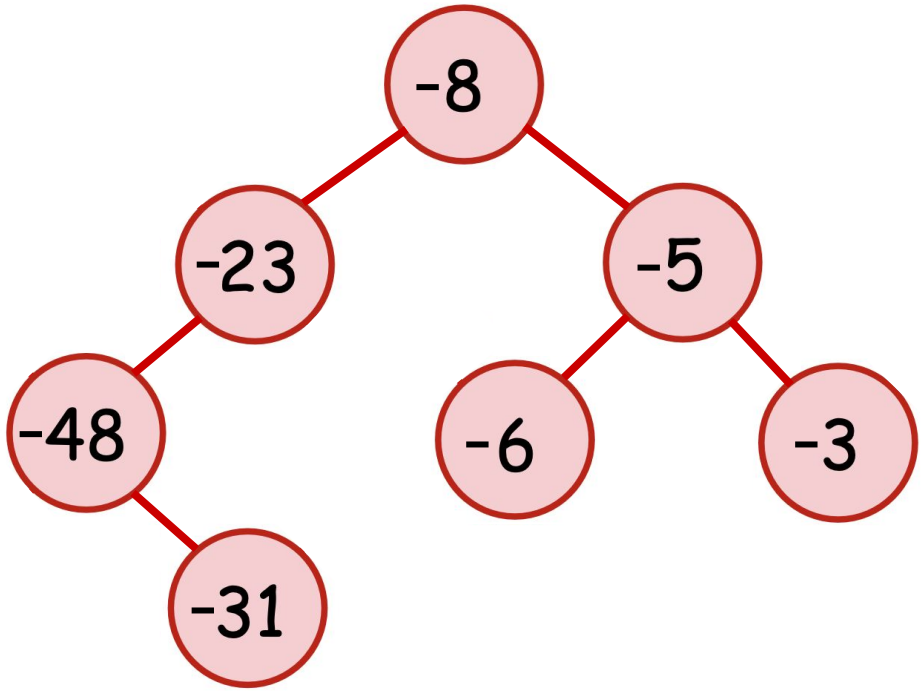
Think about it

- Y x: Print the **largest element** smaller than x in the set.
- Z x: Print the **second largest element** smaller than x in the set.

- Hint: Build a negative-value BST. Then Y x = -(U (-x))

Build two trees at the same time





Problem with BST

The maximum depth of BST may be O(n) in worst case

Example:

Insert the element in ascending / decreasing order

Solution

Make the tree more “balance”

- AVL tree
- RB tree
- Splay tree
- Treap

Find kth smallest

```
s.begin() + k; // Illegal operator!

auto it = s.begin();
for(int i = 0; i < k; i++) {
   it++;
}
// O(log n + k) -> TLE
```

Policy-Based Data Structure

Not a standard library, some compiler doesn’t support this header.

We will talk about __gnu_pbds::tree.

Policy-Based Data Structure

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

template<class T>
using ordered_set = tree<T,
                         null_type,
                         less<T>,
                         rb_tree_tag,
                         tree_order_statistics_node_update>;

ordered_set<int> s; // insert erase lower_bound upper_bound find
```

Policy-Based Data Structure

ordered_set<int> s;
s.insert(3); s.insert(5); s.insert(7); s.insert(10); // Remaining [3, 5, 7, 10]
s.lower_bound(3); // Returns the iterator of 3
s.find(4); // Returns s.end()
s.upper_bound(7); // Returns the iterator of 10
s.erase(5); // Remaining: [3, 7, 10]

s.find_by_order(1); // Return the kth iterator from s.begin() in O(log n)
s.order_of_key(10); // 2 ([3, 7, 10]), returns the rank of the key (0-based)

Policy-Based Data Structure

__gnu_pbds::tree with multiset is buggy.

Use pair instead (if you want to use multiset of ints, use pair<int, int> and give unique keys to second).

Example: Insert [5, 1, 3, 5, 6]

```
ordered_set<pair<int, int>> s;
s.insert({5, 1});
s.insert({1, 2});
s.insert({3, 3});
s.insert({5, 4});
s.insert({6, 5});
// (1, 2), (3, 3), (5, 1), (5, 4), (6, 5)
```

Number of Inversions

Given a[0], …, a[n-1], find number of i < j pairs such that a[i] > a[j].

- n <= 10^5

Hint: Try to use pb_ds::tree!

“Merge sort? Why not pb_ds::tree?” – 鲁迅

Other Policy-Based Data Structure

Persistent Data Structure

- __gnu_cxx::rope<char>; // persistent balanced binary search tree
  - Perform most operations in O(log n), (copying the whole string is O(log n)!)
  - Large constant factor, and has some bugs, don’t use it.
  - Implement your own persistent bbst (such as treap).

References

Binary Search Tree, Kai’s teaching material

- http://www.cs.nthu.edu.tw/~wkhon/ds/ds12/lecture/lecture13.pdf