Big-O Blue

LECTURE 13

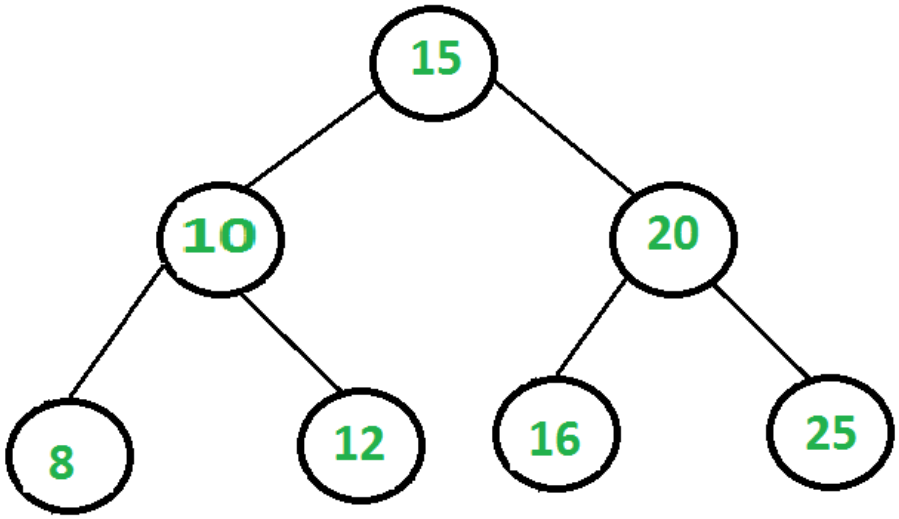
BINARY SEARCH TREE

Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

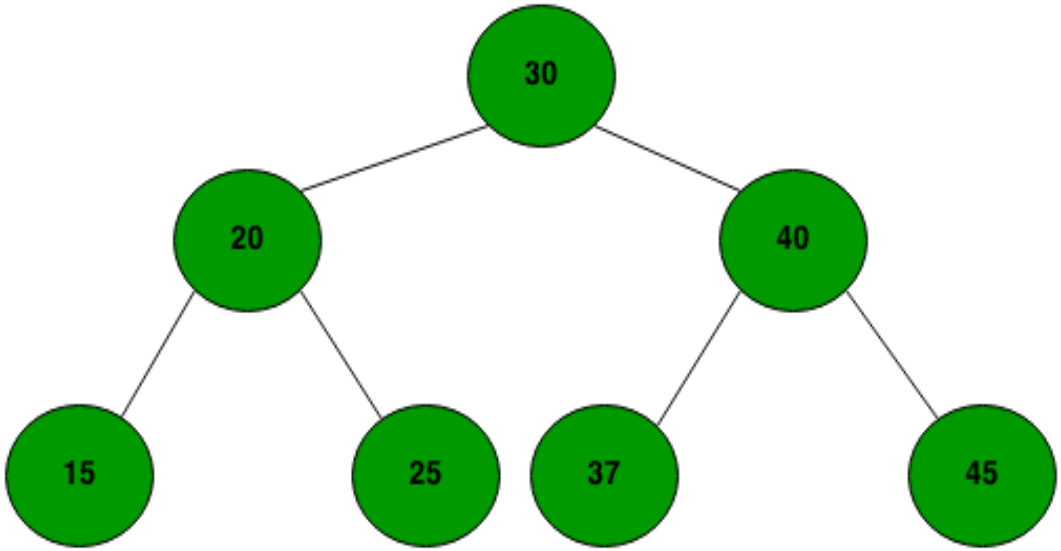
Binary Search Tree

Binary Search Tree (Cây tìm kiếm nhị phân) là cấu trúc dữ liệu rất mạnh dùng giải quyết các bài toán tìm kiếm. Các giá trị trên cây nhị phân tìm kiếm là phân biệt.



Các tính chất của Binary Search Tree

1. Mỗi node có **nhiều nhất** là **2 node con**.
2. **Bậc** của cây nhị phân **là 2** (do mỗi node tối đa 2 node con).
3. Xét một node bất kỳ, **giá trị** của mọi node trên **cây con trái** luôn **nhỏ hơn** node đang xét và **giá trị** của mọi node trên **cây con phải** luôn **lớn hơn** node đang xét.

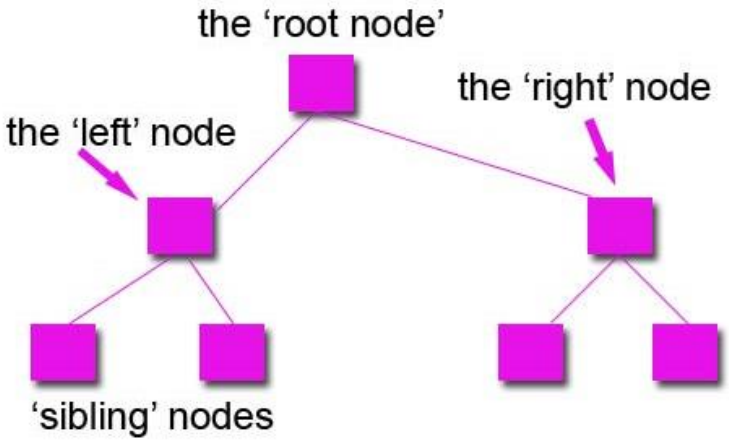


Các thao tác trên Binary Search Tree

0. Khai báo cấu trúc của một node.
1. Khởi tạo một node mới.
2. Chèn (insert) một giá trị mới vào BST.
3. Tạo một BST.
4. Tìm kiếm (search) một giá trị trong BST.
5. Xóa (delete) một giá trị trong BST.
6. Duyệt (Tree Traversals) các node của BST.
7. Tính kích thước của BST (đếm số lượng node).
8. Xóa BST.

0. Khai báo cấu trúc một node trong BST

Khai báo cấu trúc của một node trong Binary Search Tree bao gồm: giá trị cần lưu, địa chỉ cây con trái và địa chỉ cây con phải.



Độ phức tạp: O(1)

0. Khai báo cấu trúc một node trong BST

```cpp
struct Node
{
    int key;
    Node *left;
    Node *right;
};
```

```python
class Node:
    def __init__(self):
        self.key = 0
        self.left = None
        self.right = None
```

0. Khai báo cấu trúc một node trong BST

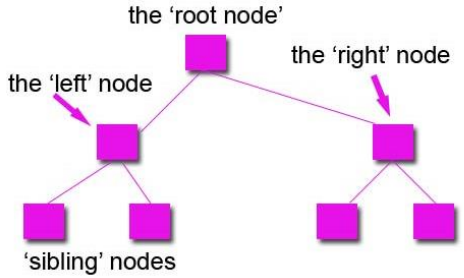
```java
class Node {
    int key;
    Node left, right;
    public Node(int item) {
        key = item;
        left = right = null;
    }
}
```


```java
class BinarySearchTree {
    private Node root = null;

    public BinarySearchTree() {
        root = null;
    }
```

1. Tạo một node mới

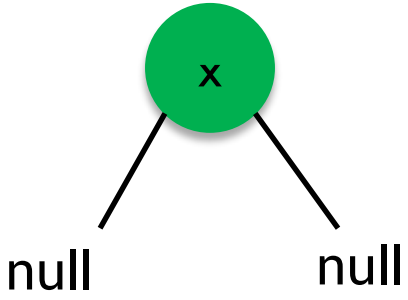
Bước 0: Khai báo cấu trúc của một node (**thao tác 0**)



Bước 1: Yêu cầu một vùng nhớ mới để chứa node.

Bước 2: Gán giá trị cần lưu cho node.

Bước 3: Gán địa chỉ nhánh phải, nhánh trái là rỗng.



Độ phức tạp: O(1)

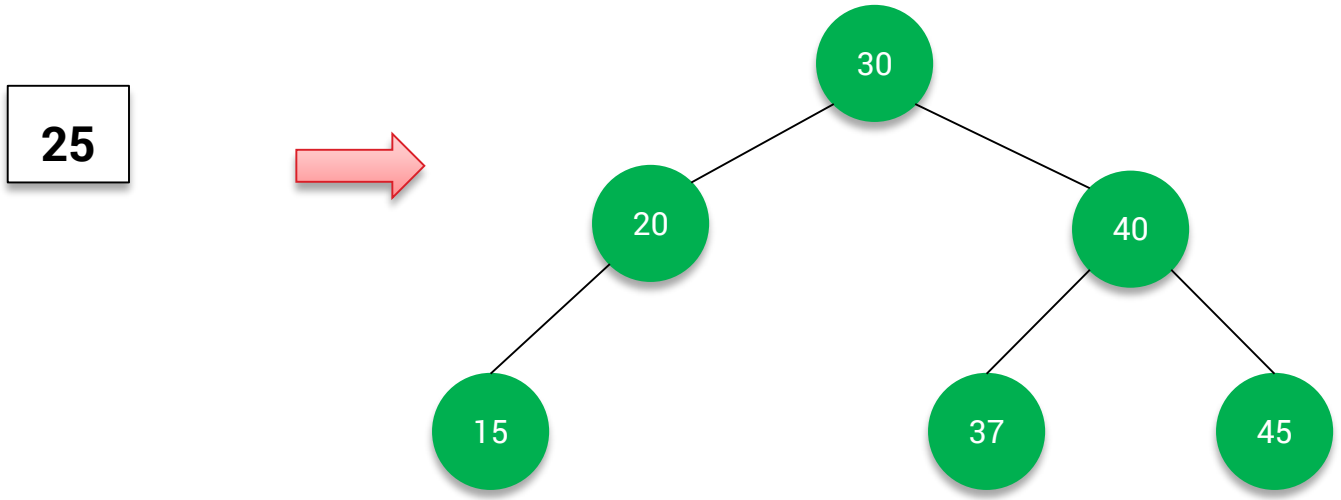
2. Tạo một node mới

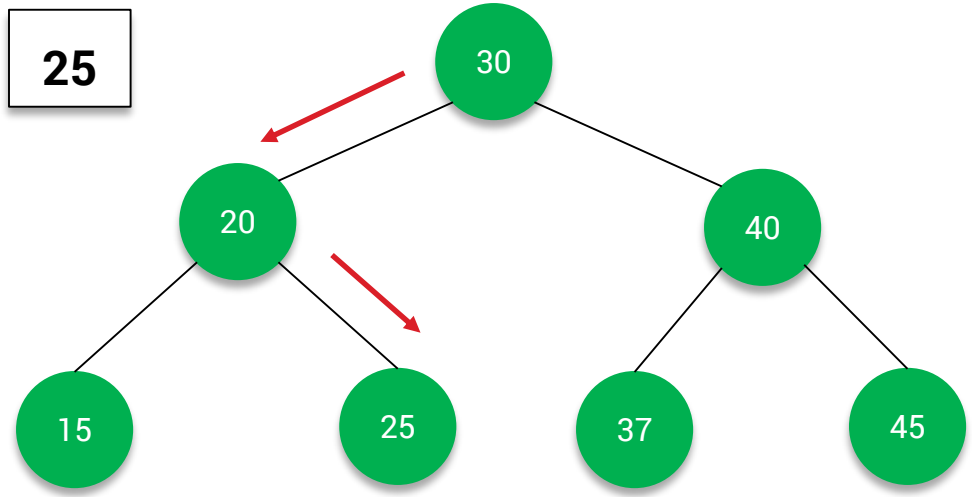
```cpp
Node* createNode(int x)
{
    Node *newNode = new Node;
    newNode->key = x;
    newNode->left = newNode->right = NULL;
    return newnode;
}
```


```python
def createNode(x):
    newNode = Node()
    newNode.key = x
    return newNode
```


```java
private Node createNode(int x) {
    Node newNode = new Node(x);
    return newNode;
}
```

2. Chèn một giá trị mới vào BST





Độ phức tạp: O(h) với h là chiều cao cây (Lưu ý: worst case O(n)).

2. Chèn một giá trị mới vào BST

```
Node* insertNode(Node* root, int x)
{
    if (root == NULL)
        return createNode(x);
    if (x < root->key)
        root->left = insertNode(root->left, x);
    else if (x > root->key)
        root->right = insertNode(root->right, x);
    return root;
}
```

2. Chèn một giá trị mới vào BST

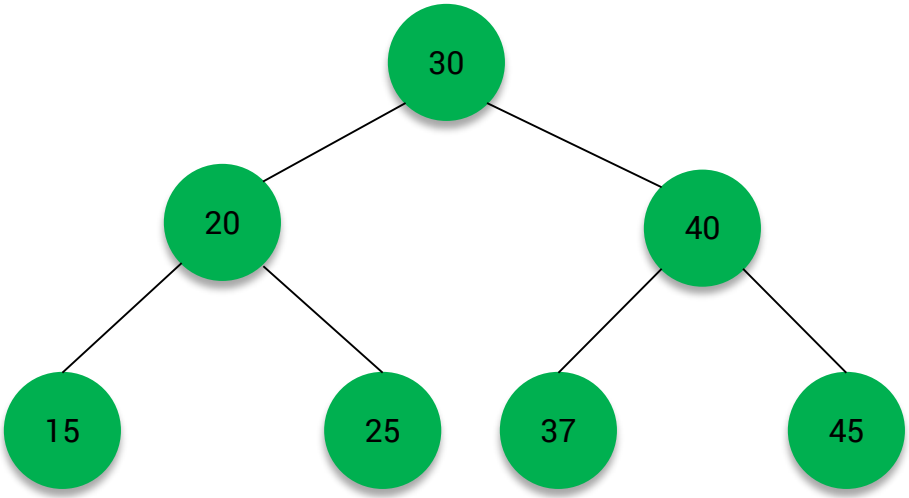
```
def insertNode(root, x):
    if root == None:
        return createNode(x)
    if x < root.key:
        root.left = insertNode(root.left, x)
    elif x > root.key:
        root.right = insertNode(root.right, x)
    return root
```

2. Chèn một giá trị mới vào BST

```java
public void insertNode(int x) {
    root = insertNode(root, x);
}
private Node insertNode(Node root, int x) {
    if (root == null)
        return createNode(x);
    if (x < root.key)
        root.left = insertNode(root.left, x);
    else if (x > root.key)
        root.right = insertNode(root.right, x);
    return root;
}
```

3. Tạo Binary Search Tree

30	20	40	15	25	37	45



Độ phức tạp: O(N*h) với h là chiều cao cây.

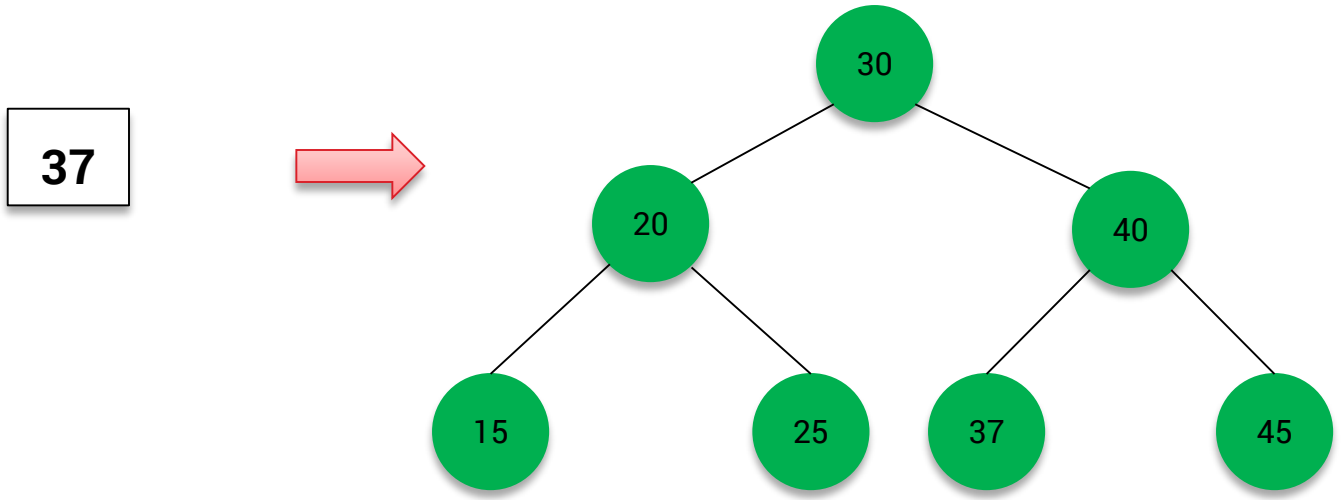
3. Tạo Binary Search Tree

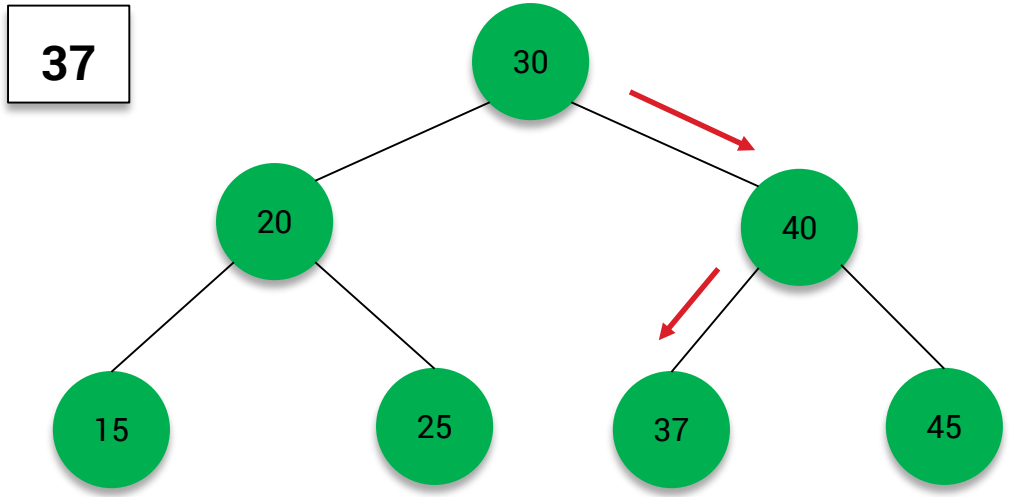
```cpp
void createTree(Node* &root, int a[], int n)
{
    for (int i = 0; i < n; i++)
        root = insertNode(root, a[i]);
}
```


```python
def createTree(a, n):
    root = None
    for i in range(n):
        root = insertNode(root, a[i])
    return root
```


```java
public void createTree(int a[], int n) {
    root = null;
    for (int i = 0; i < n; i++) {
        root = insertNode(root, a[i]);
    }
}
```

4. Tìm kiếm một giá trị trong BST





Độ phức tạp: O(h) với h là chiều cao cây (Lưu ý: worst case O(n))

4. Tìm kiếm một giá trị trong BST

```cpp
Node* searchNode(Node *root, int x)
{
    if (root == NULL || root->key == x)
        return root;
    if (root->key < x)
        return searchNode(root->right, x);
    return searchNode(root->left, x);
}
```

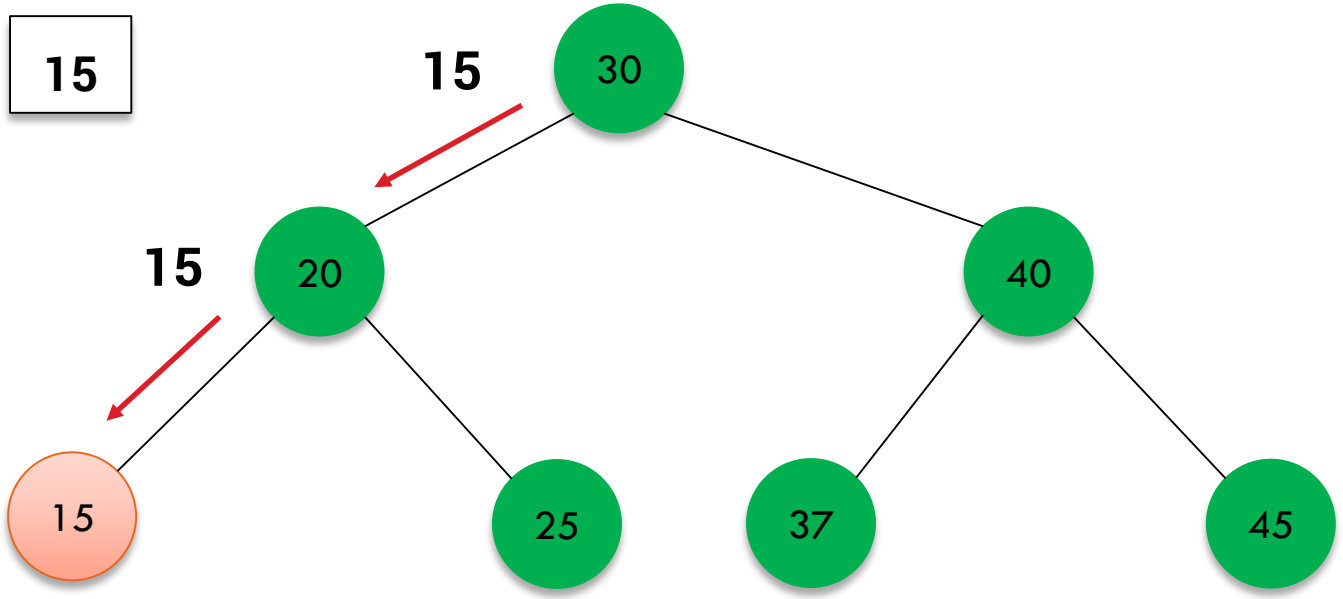
```python
def searchNode(root, x):
    if root == None or root.key == x:
        return root
    if root.key < x:
        return searchNode(root.right, x)
    return searchNode(root.left, x)
```

4. Tìm kiếm một giá trị trong BST

```java
public Node searchNode(int x) {
    return searchNode(root, x);
}
private Node searchNode(Node root, int x) {
    if (root == null || root.key == x) {
        return root;
    }
    if (root.key < x)
        return searchNode(root.right, x);
    return searchNode(root.left, x);
}
```

5. Xóa một giá trị trong BST

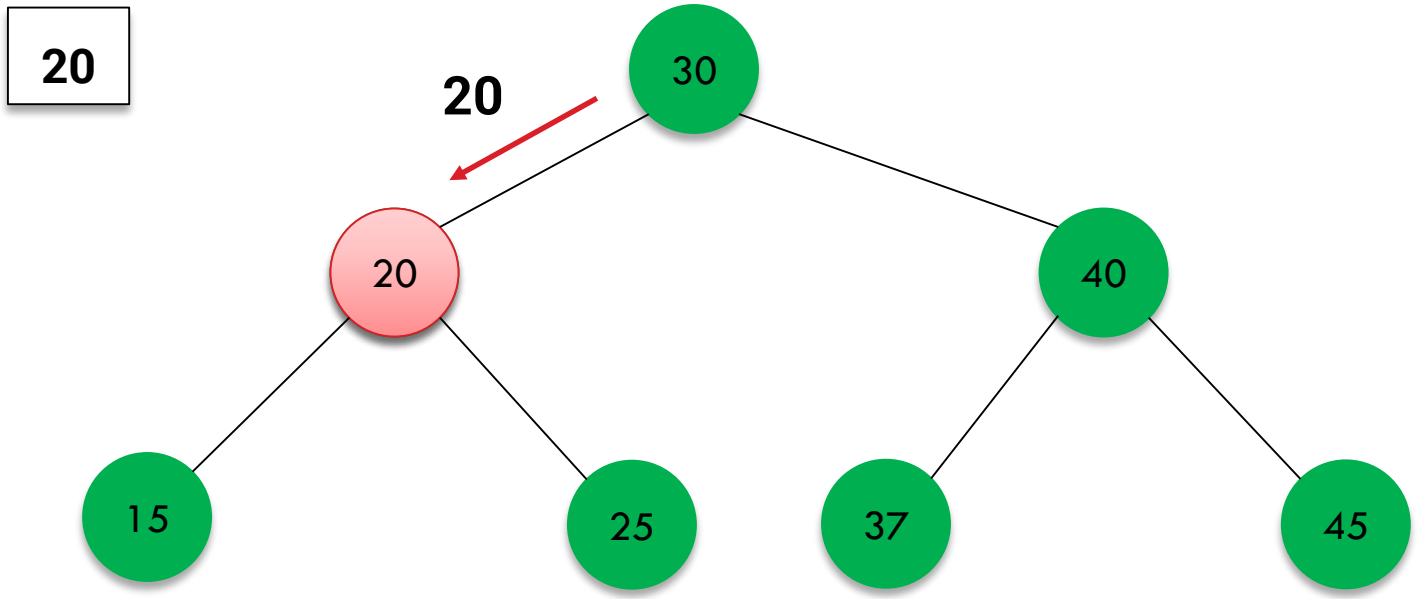
Trường hợp 1: Xóa một node là **node lá** của cây. Duyệt từ node gốc đến node lá cần xóa. Xóa node lá đó.



5. Xóa một giá trị trong BST

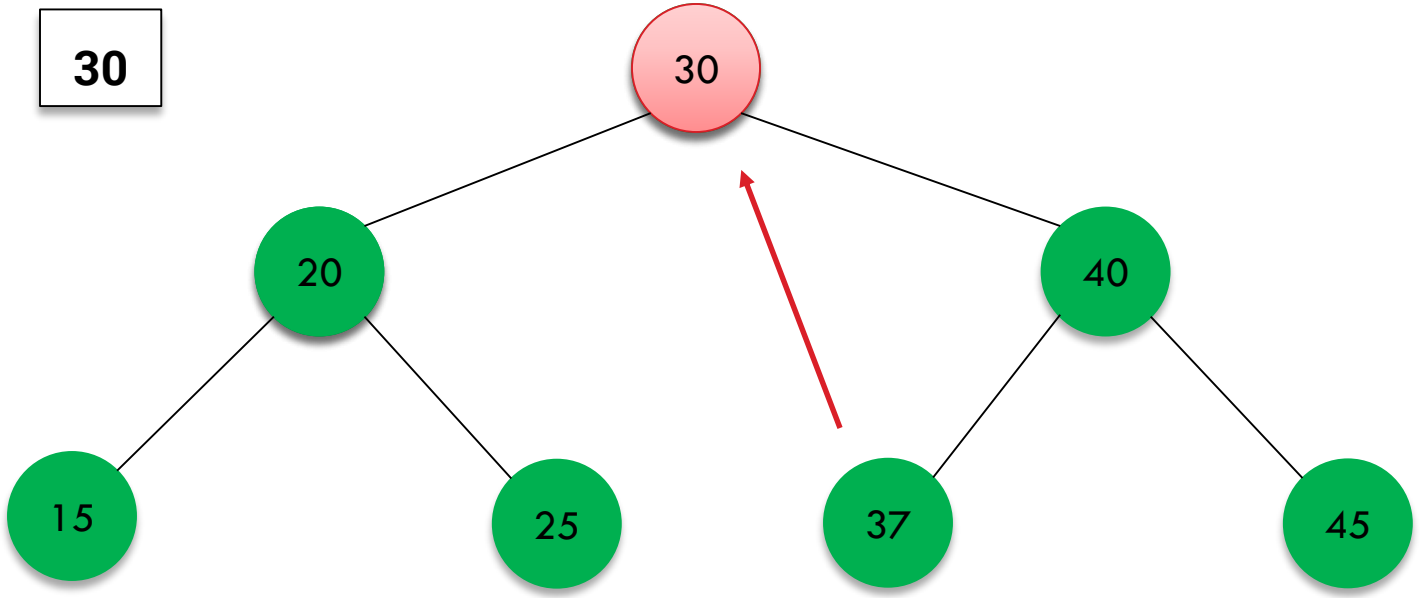
Trường hợp 2: Xóa một node **có 1** node lá **hoặc 2** node lá.

Đưa node lá có giá trị "phù hợp" lên thay, rồi xóa node cũ.



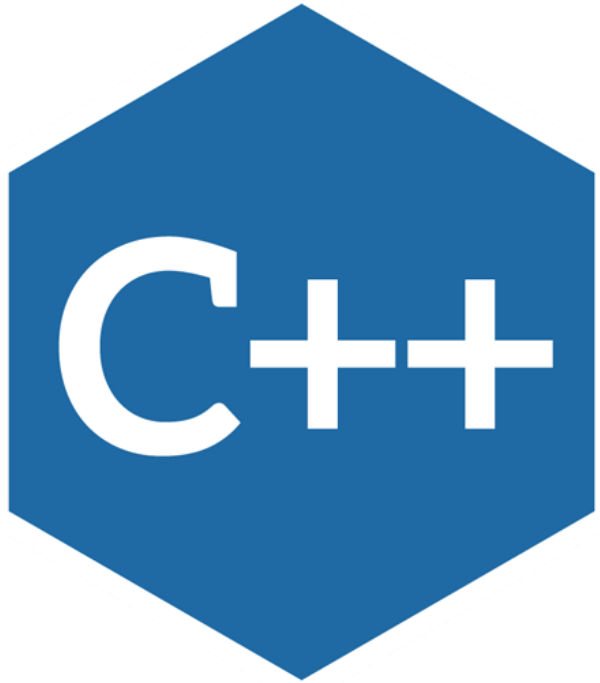
5. Xóa một giá trị trong BST

Trường hợp 3: Xóa node gốc (hoặc xóa node có nhiều node con). Tìm node có giá trị **nhỏ nhất** bên **cây con phải** (hoặc lớn nhất bên cây con trái) lên thay, rồi xóa node đó đi.



Độ phức tạp: O(h) với h là chiều cao cây (Lưu ý: worst case O(n))

MÃ NGUỒN MINH HỌA BẰNG C++



5. Xóa một node trong BST (part 1)

```
Node* deleteNode(Node* &root, int x)
{
    if (root == NULL)
        return root;
    if (x < root->key)
        root->left = deleteNode(root->left, x);
    else if (x > root->key)
        root->right = deleteNode(root->right, x);
    else
    {
        if (root->left == NULL)
        {
            Node *temp = root->right;
            delete root;
            return temp;
        }
//to be continued
```

5. Xóa một node trong BST (part 2)

```
        else if (root->right == NULL)
        {
            Node *temp = root->left;
            delete root;
            return temp;
        }
        Node* temp = minValueNode(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}
```

5. Xóa một node trong BST (part 3)

Hàm hỗ trợ tìm node nhỏ nhất bên cây con phải. Bắt đầu từ node con phải của node gốc, đi tất cả nhánh trái sẽ tìm được node nhỏ nhất bên cây con phải.

```
Node* minValueNode(Node* node)
{
    Node* current = node;
    while (current->left != NULL)
    {
        current = current->left;
    }
    return current;
}
```

MÃ NGUỒN MINH HỌA BẰNG PYTHON

5. Xóa một node trong BST (part 1)

```
def deleteNode(root, x):
    if root == None:
        return root
    if x < root.key:
        root.left = deleteNode(root.left, x)
    elif x > root.key:
        root.right = deleteNode(root.right, x)
    else:
        if root.left == None:
            temp = root.right
            del root
            return temp
        elif root.right == None:
            temp = root.left
            del root
            return temp
        temp = minValueNode(root.right)
        root.key = temp.key
        root.right = deleteNode(root.right, temp.key)
    return root
```

5. Xóa một node trong BST (part 2)

Hàm hỗ trợ tìm node nhỏ nhất bên cây con phải. Bắt đầu từ node con phải của node gốc, đi tất cả nhánh trái sẽ tìm được node nhỏ nhất bên cây con phải.

```python
def minValueNode(node):
    current = node
    while current.left != None:
        current = current.left
    return current
```

MÃ NGUỒN MINH HỌA BẰNG JAVA

5. Xóa một node trong BST (part 1)

```java
public void deleteNode(int x) {
    root = deleteNode(root, x);
}
private Node deleteNode(Node root, int x) {
    if (root == null) {
        return root;
    if (x < root.key) {
        root.left = deleteNode(root.left, x);
    }
    else if (x > root.key) {
        root.right = deleteNode(root.right, x);
    }
    else {

//to be continued
```

5. Xóa một node trong BST (part 2)


```
        if (root.left == null) {
            Node temp = root.right;
            root = null;
            return temp;
        }
        else if (root.right == null) {
            Node temp = root.left;
            root = null;
            return temp;
        }
        Node temp = minValueNode(root.right);
        root.key = temp.key;
        root.right = deleteNode(root.right, temp.key);
    }
    return root;
}
```

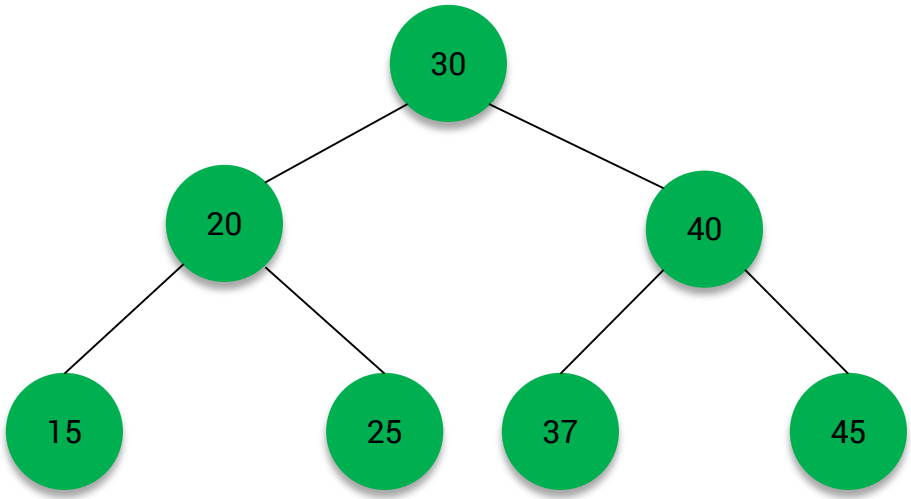
5. Xóa một node trong BST (part 3)

Hàm hỗ trợ tìm node nhỏ nhất bên cây con phải. Bắt đầu từ node con phải của node gốc, đi tất cả nhánh trái sẽ tìm được node nhỏ nhất bên cây con phải.

```java
private Node minValueNode(Node node) {
    Node current = node;
    while (current.left != null) {
        current = current.left;
    }
    return current;
}
```

CÁC HÀM KHÁC LIÊN QUAN BINARY SEARCH TREE

6. Duyệt Binary Search Tree



(a) Inorder Traversal (Left → Root → Right): 15 20 25 30 37 40 45

(b) Preorder Traversal (Root → Left → Right): 30 20 15 25 40 37 45

(c) Postorder Traversal (Left → Right → Root): 15 25 20 37 45 40 30

Độ phức tạp: O(N)

6. Duyệt Binary Search Tree

```cpp
void traversalTree(Node *root)
{
    if (root != NULL)
    {
        traversalTree(root->left);
        cout << root->key << " ";
        traversalTree(root->right);
    }
}
```

```python
def traversalTree(root):
    if root != None:
        traversalTree(root.left)
        print(root.key, end=' ')
        traversalTree(root.right)
```

6. Duyệt Binary Search Tree

```java
public void traversalTree() {
    traversalTree(root);
}
private void traversalTree(Node root) {
    if (root != null) {
        traversalTree(root.left);
        System.out.print(root.key + " ");
        traversalTree(root.right);
    }
}
```

7. Tính kích thước của BST

```cpp
int size(Node* node)
{
    if (node == NULL)
        return 0;
    else
        return (size(node->left) + 1 + size(node->right));
}
```

```python
def size(root):
    if root == None:
        return 0
    return size(root.left) + 1 + size(root.right)
```

Độ phức tạp: O(N)

7. Tính kích thước của BST

```java
public int size() {
    return size(root);
}

private int size(Node root) {
    if (root == null)
        return 0;
    return size(root.left) + size(root.right) + 1;
}
```

8. Xóa Binary Search Tree

```cpp
void deleteTree(Node* root)
{
    if (root == NULL)
        return;
    deleteTree(root->left);
    deleteTree(root->right);
    delete(root);
}
```

```python
def deleteTree(root):
    if root == None:
        return
    deleteTree(root.left)
    deleteTree(root.right)
    del root
```

Độ phức tạp: O(N)

8. Xóa Binary Search Tree

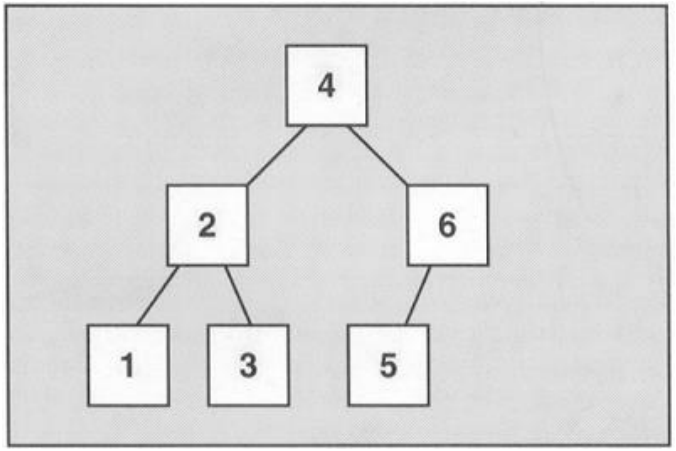
```java
public void deleteTree() {
    deleteTree(root);
}
private void deleteTree(Node root) {
    if (root == null)
        return;
    deleteTree(root.left);
    deleteTree(root.right);
    root = null;
}
```

SỬ DỤNG BINARY SEARCH TREE BẰNG THƯ VIỆN C++/JAVA

PYTHON KHÔNG CÓ CLASS BST DỰNG SẴN NÊN DÙNG SET CÀI BẰNG HASH

Cách sử dụng Set và Map

Set là một loại cấu trúc dữ liệu dùng để lưu trữ các phần tử phân biệt (unique elements) .



C++: Set

Java: TreeSet

Map là một cấu trúc dữ liệu chứa **danh sách** các phần tử mà mỗi phần tử là một cặp khóa-giá trị (key-value), 2 giá trị này có thể có kiểu dữ liệu khác nhau. Mỗi phần tử là 1 cấu trúc pair/Map.Entry

1120217	Nikhilesh
1120236	Navneet
1120250	Vikas
1120255	Doodrah

Keys

values

C++: Map

Java: TreeMap

Khai báo và sử dụng (1)

set

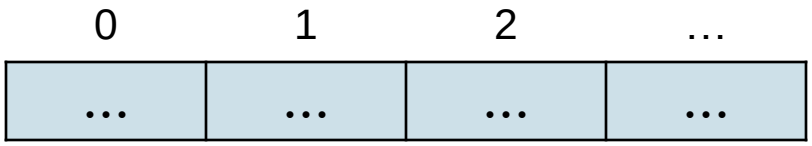
Thư viện:

```
#include <set>
using namespace std;
```

Khai báo:

```
set<data_type> variable;
```

```
set<int> s;
```



map

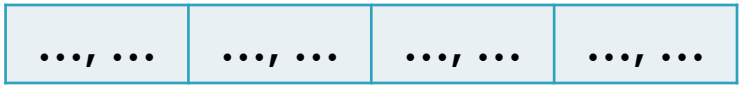
Thư viện:

```
#include <map>
using namespace std;
```

Khai báo:

```
map<data_type1, data_type2>
variable;
```

```
map<int, string> m;
```



Khai báo và sử dụng (1)

TreeSet

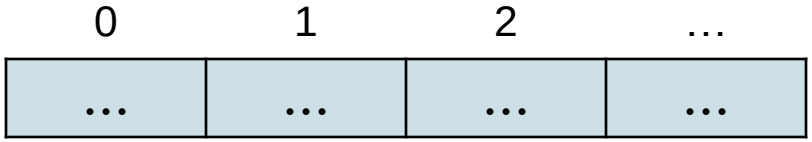
Thư viện:

```
import java.util.TreeSet;
```

Khai báo:

```
TreeSet variable = new TreeSet();
```

```
TreeSet s = new TreeSet();
```



TreeMap

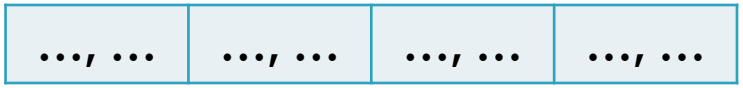
Thư viện:

```
import java.util.TreeMap;
```

Khai báo:

```
TreeMap variable = new
TreeMap<object_type1, object_type2>();
```

```
TreeMap m = new
TreeMap<Integer, String>();
```



Khai báo và sử dụng (2)

set

```
<Array> a;
set<data_type> variable(a, a + n);
```

```
int a[]= {10, 70, 20, 90, 50};
set<int> s(a, a + 5);
```

0	1	2	3	4
10	20	50	70	90

map

```
map<data_type1, data_type2> variable;
variable[key] = value;
```

```
map<int, string> m;
m[10] = "abc";
m[20] = "def";
m[10] = "mpk";
```

10, “mpk”	20, “def”

Khai báo và sử dụng (2)

TreeSet

```
TreeSet s = new TreeSet(Collection<?
    extends E> c)
```

```
int[] arr = new int[]{10, 70, 20,
90, 50};
ArrayList a = new ArrayList();
for (int x : arr) {
    a.add(x);
}
TreeSet ts = new TreeSet(a);
```

0	1	2	3	4
10	20	50	70	90

TreeMap

```
variable.put(object1, object2);
```

```
TreeMap<Integer, String> m
= new TreeMap<Integer, String> ();
m.put(10, "abc");
m.put(20, "def");
m.put(10, "mpk");
```

10, “mpk”	20, “def”

Thêm một phần tử

set

0	1	2	3
10	20	50	70

insert(value)

```
s.insert(60);
```

0	1	2	3	4
10	20	50	60	70

map

10, “mpk”	20, “def”

insert(pair)

```
pair<int, string> p(14, "abc");
m.insert(p);
```

10, “mpk”	14, “abc”	20, “def”

Thêm một phần tử

TreeSet

0	1	2	3
10	20	50	70

add(obj)

```
s.add(60);
```

0	1	2	3	4
10	20	50	60	70

TreeMap

10, “mpk”	20, “def”

put(obj)

```
m.put(14, "abc");
```

10, “mpk”	14, “abc”	20, “def”

Tìm phần tử

set

0	1	2	3
10	20	50	70

find(value): Tìm và trả về một iterator nếu tìm thấy, ngược lại sẽ trả về set::end.

```
set<int>::iterator it;
it = s.find(50);
if(it != s.end())
    cout << "found";
else
    cout << "not found";
```

map

10, “mpk”	14, “abc”	20, “def”

find(key): Tìm và trả về iterator của đối tượng nếu tìm thấy, ngược lại trả về map::end.

```
map<int, string>::iterator it;
it = m.find(20);
if(it != m.end())
    cout << "found";
else
    cout << "not found";
```

found

Tìm phần tử

TreeSet

0	1	2	3
10	20	50	70

contains(obj): trả về true nếu đối tượng tìm kiếm tồn tại trong cây, ngược lại false.

```
if (s.contains(50))
    System.out.print("found");
else
    System.out.print("not found");
```

TreeMap

10, “mpk”	14, “abc”	20, “def”

containsKey(Object key)

```
if (m.containsKey(10))
    System.out.print("found");
else
    System.out.print("not found");
```

containsValue(Object value)

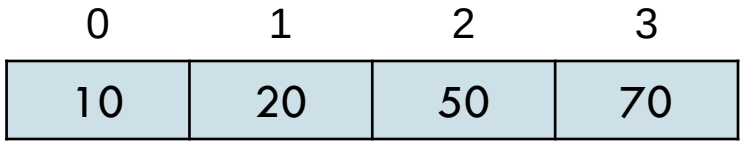
```
if (m.containsValue("mpk"))
    System.out.print("found");
else
    System.out.print("not found");
```

NOTE: *Hàm containsValue có độ phức tạp O(N)*

found

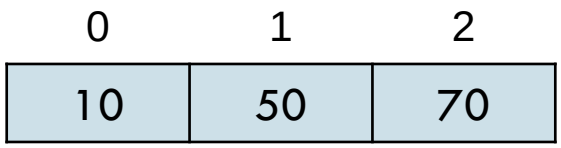
Xóa một phần tử

set

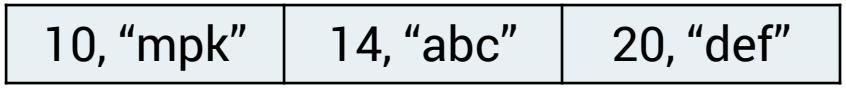


erase(value)

```
s.erase(20);
```



map



erase(key): xóa mỗi lần 1 phần tử.

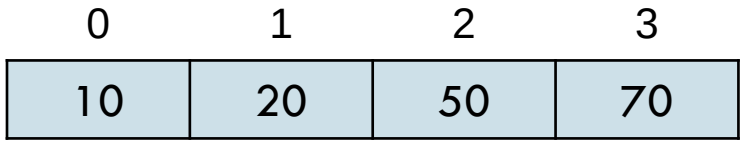
```
m.erase(14);
```

Hoặc:

erase(iterator, iterator): dựa vào iterator xóa một loạt phần tử.

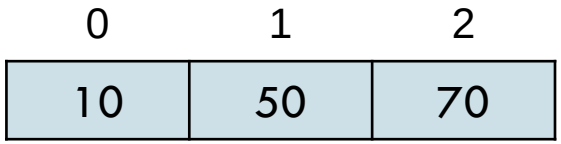
Xóa một phần tử

TreeSet



remove(obj)

```
s.remove(20);
```



TreeMap

10, “mpk”	14, “abc”	20, “def”

remove(obj): Xóa giá trị trong **TreeMap** dựa vào khóa. Trả về object value của khóa tương ứng, nếu khóa không tồn tại thì trả về **null.**

```
m.remove(14);
```

10, “mpk”	20, “def”

Hàm tìm cận dưới ≥

set

0	1	2	3
13	29	42	65

lower_bound(value): Trả về iterator chỉ đến phần tử đầu tiên **lớn hơn hoặc bằng** giá trị tìm kiếm **[first**, last).

Nếu không tìm thấy trả về set::end

```
set<int>::iterator it;
it = s.lower_bound(29);
cout << *it;
```

29

map

10, “mpk”	14, “abc”	20, “def”

lower_bound(value): Trả về con trỏ chỉ đến phần tử đầu tiên **lớn hơn hoặc bằng** giá trị tìm kiếm **[first**, last).

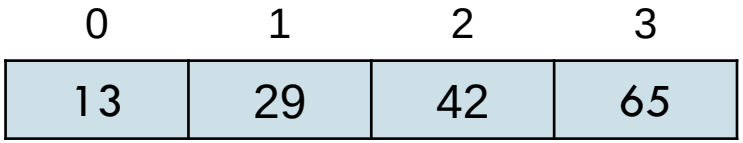
Nếu không tìm thấy trả về map::end

```
map<int,string>::iterator it;
it = s.lower_bound(14);
cout << it->first << " " << it->second;
```

14 abc

Hàm tìm cận dưới ≥

TreeSet



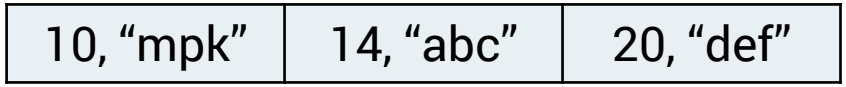
ceiling(obj): Trả về giá trị phần tử đầu tiên **lớn hơn hoặc bằng** giá trị tìm kiếm. Nếu không tìm thấy trả về null.

```
int x = (int)s.ceiling(29);
System.out.println(x);
```

29

*(**NOTE**: Java còn 1 hàm đối xứng với ceiling là floor trả về phần tử lớn nhất, nhỏ hơn hoặc bằng giá trị khóa tìm kiếm) ≤*

TreeMap



ceilingEntry/ceilingKey(obj): Trả về giá trị phần tử đầu tiên **lớn hơn hoặc bằng** giá trị tìm kiếm. Nếu không tìm thấy trả về null.

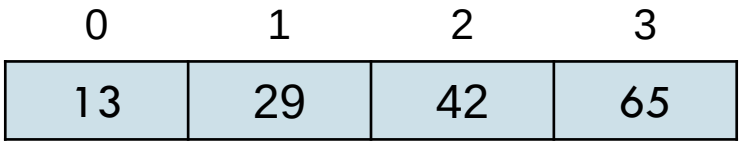
```
Map.Entry<Integer, String> x =
m.ceilingEntry(14);
System.out.println(x.getKey() + " " +
x.getValue());
```

14 abc

*(**NOTE**: Java còn 2 hàm đối xứng với ceilingEntry/ceilingKey là floorEntry/floorKey trả về phần tử/khóa lớn nhất, nhỏ hơn hoặc bằng giá trị khóa tìm kiếm) ≤*

Hàm tìm cận trên >

set



upper_bound(value): Trả về iterator chỉ đến phần tử đầu tiên **lớn hơn** giá trị tìm kiếm [first, **last)**.

Nếu không tìm thấy trả về set::end.

```
set<int>::iterator it;
it = s.upper_bound(29);
cout << *it;
```

42

map

10, “mpk”	14, “abc”	20, “def”

upper_bound(value): Trả về con trỏ chỉ đến phần tử **đầu tiên lớn hơn** giá trị tìm kiếm [first, **last)**.

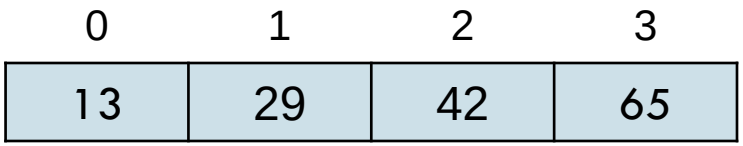
Nếu không tìm thấy trả về map::end.

```
map<int,string>::iterator it;
it = s.upper_bound(14);
cout << it->first << " " << it->second;
```

20 def

Hàm tìm cận trên >

TreeSet



higher(obj): Trả về phần tử **đầu tiên lớn hơn** giá trị tìm kiếm. Nếu không tìm thấy trả về null.

```
int x = (int)s.higher(29);
System.out.println(x);
```

42

*(**NOTE**: Java còn 1 hàm đối xứng với higher là lower trả về phần tử lớn nhất, nhỏ hơn giá trị khóa tìm kiếm) <*

TreeMap

10, “mpk”	14, “abc”	20, “def”

higherEntry/higherKey(obj): Trả về phần tử **đầu tiên lớn hơn** tìm kiếm. Nếu không tìm thấy trả về null.

```
Map.Entry<Integer, String> x =
m.higherEntry(14);
System.out.println(x.getKey() + " " +
x.getValue());
```

20 def

*(**NOTE**: Java còn 2 hàm đối xứng với higherEntry/higherKey là lowerEntry/lowerKey trả về phần tử/khóa lớn nhất, nhỏ hơn giá trị khóa tìm kiếm) <*

Một số hàm thành viên khác

set

size(): Trả về số lượng phần tử hiện tại có trong set.

empty(): Kiểm tra set có rỗng hay không.

clear(): Xóa hết tất cả các giá trị trong set.

swap(other set): Hoán đổi 2 set với nhau.

map

size(): Trả về số lượng phần tử hiện tại có trong map.

empty(): Kiểm tra map có rỗng hay không.

clear(): Xóa hết tất cả các phần tử trong map.

swap(other map): Hoán đổi 2 map với nhau.

Một số hàm thành viên khác

TreeSet

size(): Trả về số lượng phần tử hiện tại có trong TreeSet.

isEmpty(): Kiểm tra TreeSet có rỗng hay không.

clear(): Xóa hết tất cả các giá trị trong TreeSet.

TreeMap

size(): Trả về số lượng phần tử hiện tại có trong TreeMap.

isEmpty(): Kiểm tra TreeMap có rỗng hay không.

clear(): Xóa hết tất cả các phần tử trong TreeMap.

MỘT SỐ LƯU Ý KHI SỬ DỤNG SET & MAP

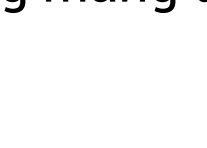
Truy cập vào thành phần set & map

KHÔNG thể truy cập ngẫu nhiên vào thành phần set và map và không thể duyệt theo dạng mảng tĩnh.

set

0	1	2	3
13	29	42	65

```
s[2] = 9;
int a = s[2];
cout << a;
```



map

10, “mpk”	14, “abc”	20, “def”

```
m[2] = make_pair(14, "kmp");
pair<int, string> p = m[2];
cout << p.first;
```

CÓ THỂ truy cập vào **key** để lấy giá trị của **value**. Nếu đầu vào là một key không có trong map thì nó sẽ tự tạo ra phần tử mới.

```
string s = m[14];
cout << s;
```

abc

Truy cập vào thành phần TreeSet & TreeMap

KHÔNG thể truy cập vào thành phần TreeSet và TreeMap và không thể duyệt cây theo dạng mảng tĩnh.

TreeSet

0	1	2	3
13	29	42	65

```
String value = s[2];
```

TreeMap

10, “mpk”	14, “abc”	20, “def”

```
String value = m[20];
```

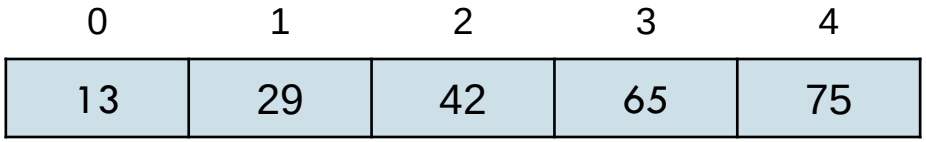
CÓ THỂ dùng hàm **get** đã hướng dẫn ở phía trên để lấy dữ liệu.

```
String s = m.get(14);
System.out.print(s);
```

abc

Duyệt các phần tử của set và map

Duyệt bằng cách sử dụng iterator (duyệt xuôi).



set

```
set<int>::iterator it;
for(it=s.begin(); it!=s.end(); it++)
    cout<< *it << ", ";
```

13, 29, 42, 65, 75,

map

```
map<int, string>::iterator it;
for(it=m.begin(); it!=m.end(); it++)
    cout<<it->first<<" "<<it->second<<", ";
```

10 mpk, 14 abc, 20 def, 25 po,

Duyệt các phần tử của TreeSet và TreeMap

Duyệt xuôi TreeSet và TreeMap.

```
import API java.util.Iterator
```

TreeSet

0	1	2	3	4
13	29	42	65	75

```
for (Iterator it = s.iterator(); it.hasNext();)
    System.out.print(it.next() + ", ");
```

13, 29, 42, 65, 75,

TreeMap

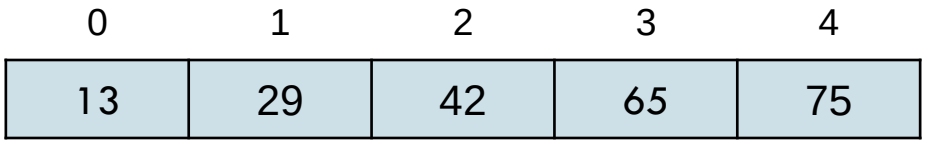
10, “mpk”	14, “abc”	20, “def”	25, “po”

```
for (Map.Entry<Integer, String> kvp : m.entrySet())
  System.out.print(kvp.getKey() + " " + kvp.getValue() + ", ");
```

10 mpk, 14 abc, 20 def, 25 po,

Duyệt các phần tử của set và map

Duyệt bằng cách sử dụng iterator (duyệt ngược).



set

```
set<int>::reverse_iterator it;
for(it=s.rbegin(); it!=s.rend(); it++)
    cout << *it << ", ";
```

75, 65, 42, 29, 13,

map

```
map<int, string>::reverse_iterator it;
for(it=m.rbegin(); it!=m.rend(); it++)
    cout << it->first << " " << it->second << ", ";
```

25 po, 20 def, 14 abc, 10 mpk,

Duyệt các phần tử của TreeSet và TreeMap

Duyệt ngược TreeSet và TreeMap.

```
import API java.util.Map
```

TreeSet

0	1	2	3	4
13	29	42	65	75

```
Iterator it = s.descendingIterator();
while (it.hasNext())
    System.out.print(it.next() + ", ");
```

75, 65, 42, 29, 13,

TreeMap

10, “mpk”	14, “abc”	20, “def”	25, “po”

```
for (Map.Entry<Integer, String> entry: m.descendingMap().entrySet())
    System.out.print(entry.getKey() + " " + entry.getValue() + ", ");
```

25 po, 20 def, 14 abc, 10 mpk,

Cấu trúc tương tự trong Python

set

Định nghĩa: là cấu trúc dữ liệu dùng Hash Table để quản lý (không dùng cấu trúc Binary Search Tree giống C++ và Java để quản lý), nhưng tính năng và cách sử dụng để giải quyết bài toán gần như tương tự Set (C++) và TreeSet (Java).

Những phần tử được đưa vào sẽ nằm **ngẫu nhiên** mà không được sắp xếp sẵn.

dict

Định nghĩa: là cấu trúc dữ liệu dùng Hash Table để quản lý (không dùng cấu trúc Binary Search Tree giống C++ và Java để quản lý), nhưng tính năng và cách sử dụng để giải quyết bài toán gần như tương tự Map (C++) và TreeMap (Java).

Những phần tử được đưa vào sẽ nằm **ngẫu nhiên** mà không sắp xếp sẵn.

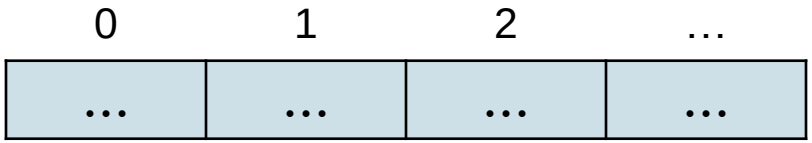
Khai báo và sử dụng set - dict

set

Khai báo: Khai báo dạng mặc định.

```
variable = set()
```

```
s = set()
```



dict

Khai báo: Khai báo dạng mặc định.

```
variable = dict()
```

```
d = dict()
```

Khai báo và sử dụng set - dict

set

Khai báo: khởi tạo với giá trị cho trước.

```
s = set(value)
```

```
a = [10, 20, 50, 70, 90]
s = set(a)
```

0	1	2	3	4
10	20	50	60	90

dict

Khai báo: khởi tạo xong sau đó gán giá trị.

```
d = dict()
variable[key] = value
```

```
d = dict()
d[10] = 'abc'
d[20] = 'def'
d[10] = 'mpk'
```

10, “mpk”	20, “def”

Các hàm thành viên của set - dict

set

0	1	2	3
10	20	50	70

add(obj): thêm một phần tử vào Set.

```
variable.add(value)
```

```
s.add(60)
```

0	1	2	3	4
10	20	50	60	70

dict

10, “mpk”	20, “def”

Thêm/cập nhật một phần tử vào Dict thông qua toán tử [ ] với cú pháp:

```
variable[key] = value
```

```
d[14] = 'abc'
```

10, “mpk”	14, “abc”	20, “def”

Các hàm thành viên của set - dict

set

0	1	2	3	4
10	20	50	60	70

remove(obj): xóa mỗi lần 1 phần tử trong set.

```
s.remove(60)
```

0	1	2	3
10	20	50	70

Lưu ý: *hàm này sẽ raise một KeyError nếu như key không tồn tại trong set, do đó cần kiểm tra trước key có tồn tại trong set hay không.*

dict

10, “mpk”	14, “abc”	20, “def”

pop(key[, default]): Xóa phần tử có key trong dict và trả về value của nó. Nếu key không tồn tại thì trả về default.

```
print(d.pop(14))
```

10, “mpk”	20, “def”

abc

Lưu ý: *hàm này sẽ raise một KeyError nếu như key không tồn tại trong dict, do đó cần kiểm tra trước key có tồn tại trong dict hay không.*

Các hàm thành viên của set - dict

KeyError là gì?

set

0	1	2	3	4
10	20	50	60	70

```
if s.remove(30, None) == None:
    print('Object not found')
else:
    print('deleted')
```

Object not found

dict

10, “mpk”	14, “abc”	20, “def”

```
if d.pop(30, None) == None:
    print('Key not found')
else:
    print('deleted')
```

Key not found

Lưu ý: *tham số thứ 2 của hàm pop chúng ta có thể thay thế bất kỳ chữ gì nếu muốn. Nhưng không nên trùng giá trị trong dict vì sẽ không xác định được pop thành công hay không.*

Các hàm thành viên của set - dict

set

0	1	2	3	4
10	20	50	60	70

Kiểm tra giá trị có tồn tại trong set hay không:

<key> in <set>

```
if 20 in s:
    print("exists")
else:
    print("not exists")
```

exists

dict

10, “mpk”	14, “abc”	20, “def”

Kiểm tra key có tồn tại trong dict hay không:

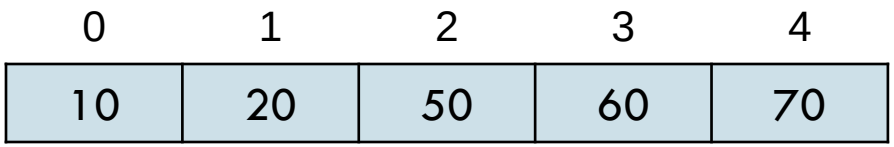
<key> in <dict>

```
if 20 in d:
    print("exists")
else:
    print("not exists")
```

exists

Các hàm thành viên của set - dict

set

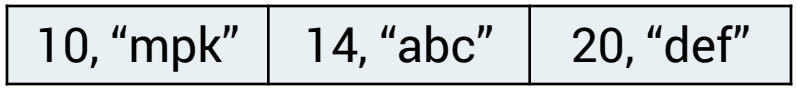


Không có hàm get, dùng <key> in <set> kiểm tra.

```
a = [10, 20, 50, 70, 90]
s = set(a)
print(50 in s)
```

True

dict



get(key[, default]): Lấy giá trị của dict theo key.

```
print(d.get(14))
print(d.get(1))
```

abc
None

Lưu ý: *nếu key không có trong dict thì khi dùng toán tử [ ] sẽ raise KeyError.*

Hàm get() sẽ return default (mặc định là None), tức không bao giờ raise KeyError.

Các hàm thành viên của set - dict

set

0	1	2	3	4
10	20	50	60	70

```
a = [10, 20, 50, 60, 70]
s = set(a)
for value in s:
   print(value)
```

dict

10, “mpk”	14, “abc”	20, “def”

items: Trả về danh sách các phần tử (item) trong dict.

```
for item in d.items():
    print(item)
```

keys: Trả về danh sách các khóa tìm kiếm (key) trong dict.

```
for key in d.keys():
    print(key, end = ', ')
```

values: Trả về danh sách các giá trị (value) trong dict.

```
for value in d.values():
    print(value, end = ', ')
```

Các hàm thành viên của set - dict

set

	0	1	2	3
s1	10	50	60	70

	0	1
s2	10	20

set.update(other set): Thêm các phần tử của set này vào set khác, nếu như có key trùng thì sẽ bỏ qua key trùng.

```
s1.update(s2)
```

s1

0	1	2	3	4
10	20	50	60	70

dict

d1	10, “mpk”	14, “abc”	20, “def”

d2	10, “bigo”	22, “coding”

dict.update(other dict): Thêm các phần tử của dict này vào dict khác, nếu như có key trùng thì value của key sẽ được lấy theo value trong dict thứ 2.

```
d1.update(d2)
```

d1

10, “bigo”	**20, “def”**	**22, “coding”**	**14, “abc”**

Các hàm thành viên của set - dict

set

len(<set>): Trả về số lượng phần tử hiện tại có trong set sử dụng tương tự như với list.

clear(): Xóa hết tất cả các giá trị trong set.

copy(): Tạo một shallow copy của set.

```
s1 = set()
s1.add(1)
s1.add(2)
s1.add(3)
s2 = s1.copy()
```

dict

len(<dict>) Trả về số lượng phần tử hiện tại có trong dict.

clear(): Xóa hết tất cả các phần tử trong dict.

copy(): Tạo một shallow copy của dict.

```
other = m.copy()
```

Hỏi đáp