Big-O Blue

LECTURE 14

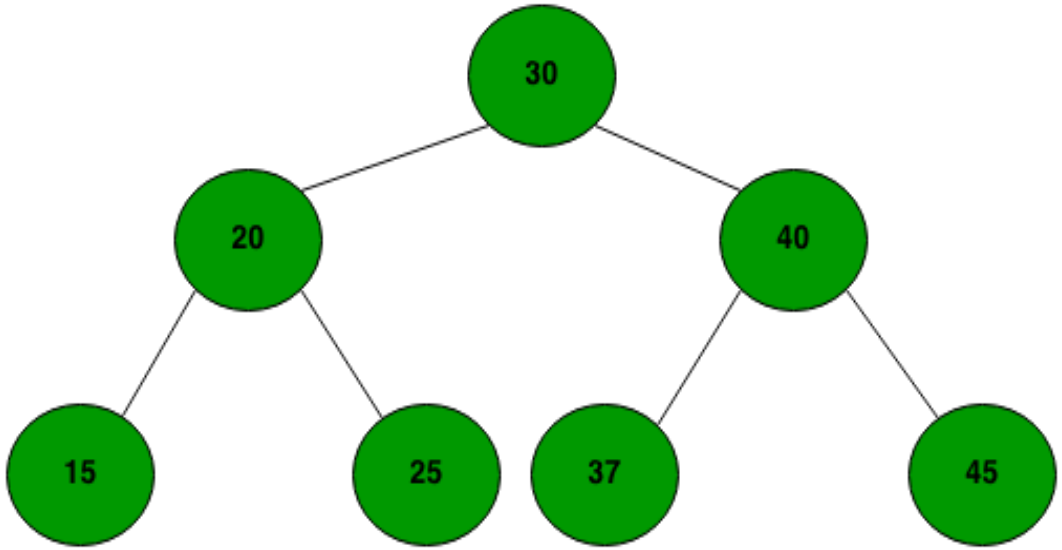
BINARY SEARCH TREE & SET

Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

Định nghĩa 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 không giống nhau.



Các tính chất của Binary Search Tree

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

Khai báo cấu trúc BST

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

Tạo một node mới trong BST

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

Tìm kiếm một node trong BST

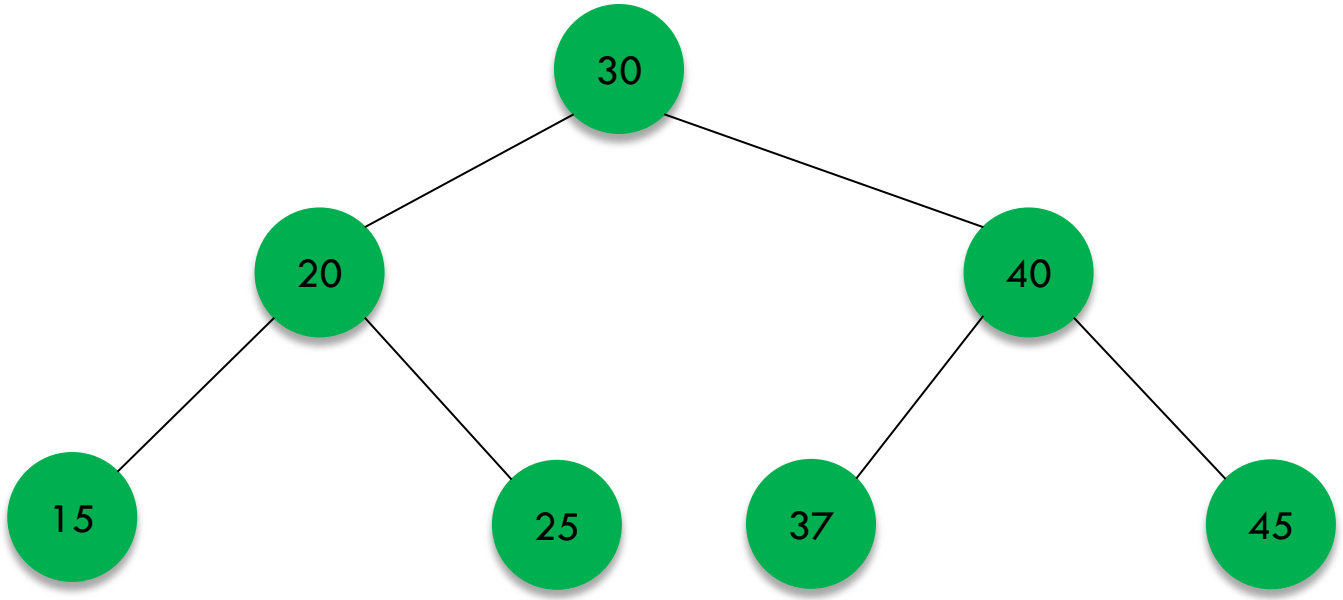
```
Node* searchNode(Node *root, int x)
{
    if (root == NULL || root->key == x)
        return root;

    if (root->key < x)
        return searchNode(root->right, x);

    return searchNode(root->left, x);
}
```

Chèn một node vào trong 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;
}
```

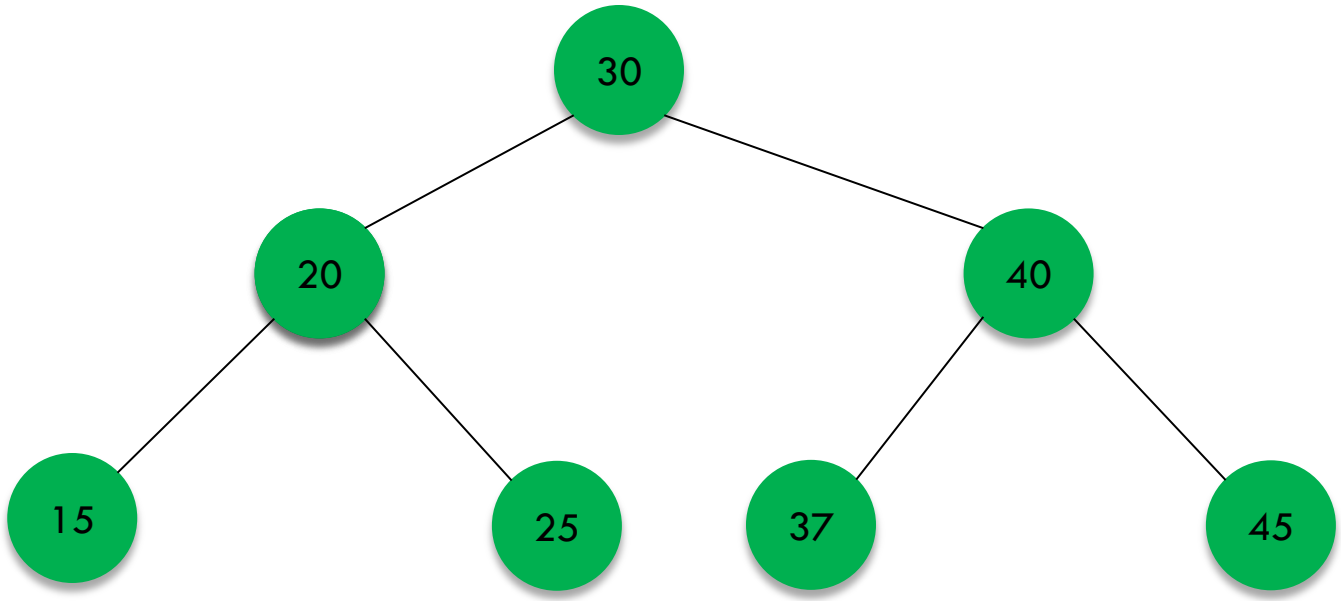
Xóa một node trong BST

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



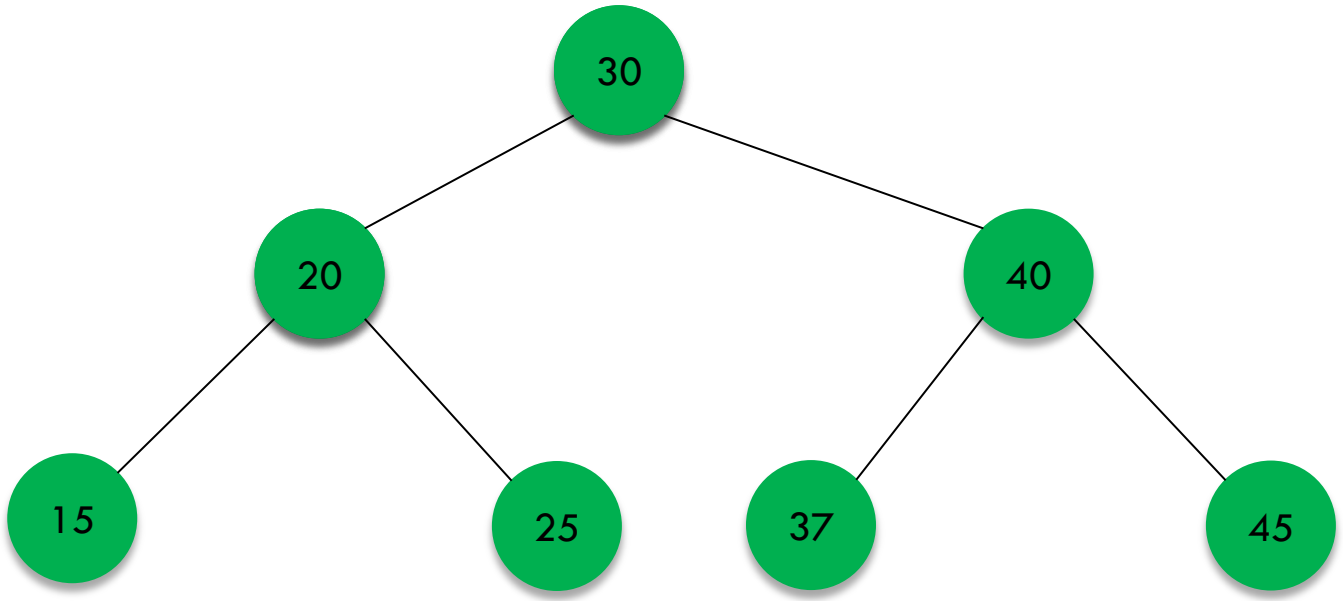
Xóa một node trong BST

Trường hợp 2: Xóa một node có một node lá hoặc 2 node lá. Đưa node lá có giá trị "phù hợp" lên thay, rồi xóa node cũ.



Xóa một node 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.



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;
        }
```

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;
}
```

Xóa một node trong BST (part 3)

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

CÁC HÀM KHÁC LIÊN QUAN

BINARY SEARCH TREE

Tạo và duyệt cây BST

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

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

Tính kích thước và xóa cây BST

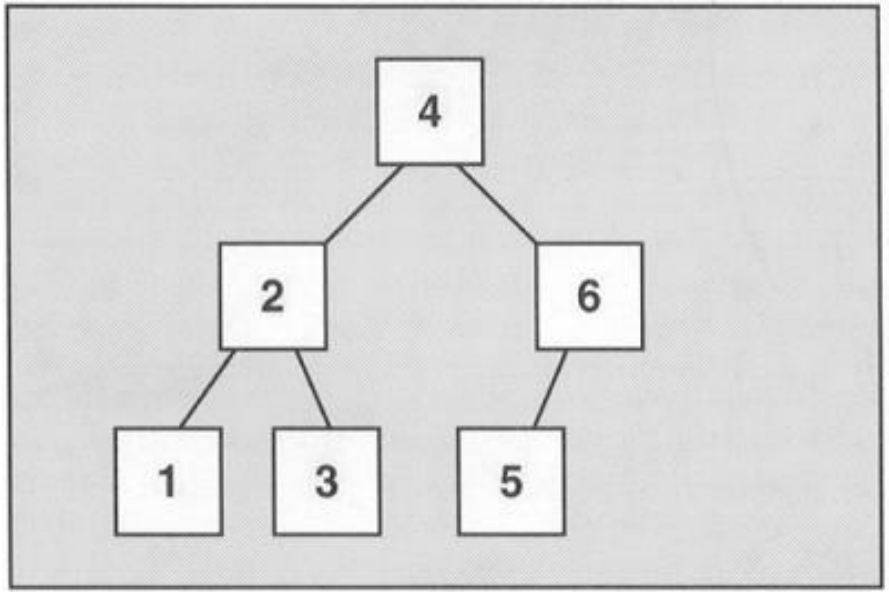
```
int size(Node* node)
{
    if (node == NULL)
        return 0;
    else
        return(size(node->left) + 1 + size(node->right));
}
```

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

SỬ DỤNG BINARY SEARCH TREE BẰNG THƯ VIỆN STL

Định nghĩa Set

Set là một loại associative containers, dùng để lưu trữ các phần tử không bị trùng lặp (unique elements).



Khai báo và sử dụng (1)

Thư viện:

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

Khai báo: khai báo dạng mặc định.

```
set<data_type> variable_name;
```

Ví dụ:

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

Khai báo và sử dụng (2)

Khai báo: khởi tạo từ mảng cho trước.

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

Ví dụ:

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

Kết quả

Khai báo và sử dụng (3)

Khai báo: khởi tạo sáo chép từ set này sang set kia.

```
set<data_type> variable_name;
set<data_type> variable_name1(variable_name);
```

Ví dụ:

```
set<int> s;
set<int> s1(s);
```

Hoặc:

```
set<int> s1(s.begin(), s.end());
```

Các hàm thành viên của Set (1)

insert: Thêm một phần tử vào set.

0	1	2	3	4
10	20	50	70	90

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

Kết quả

0	1	2	3	4	5
10	20	50	60	70	90

Các hàm thành viên của Set (2)

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

10	20	50	70	90

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

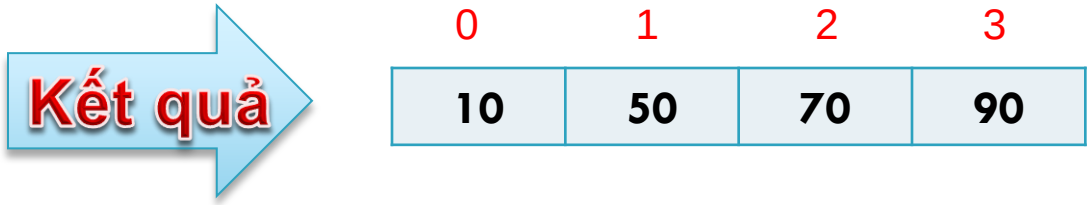
Kết quả

```
50
```

Các hàm thành viên của Set (3)

erase: Xóa giá trị trong set.


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



Các hàm thành viên của Set (4)

lower_bound: Trả về phần tử **đầu tiên không bé hơn** giá trị khóa tìm kiếm **[first**, last).

13	29	42	65	75

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

Kết quả

29

Các hàm thành viên của Set (5)

upper_bound: Trả về phần tử **đầu tiên** lớn hơn giá trị **khóa tìm kiếm** [first, **last).**

13	29	42	65	75

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

Kết quả

42

Một số hàm thành viên khác

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: Hóa đổi 2 set với nhau.

CÁC LƯU Ý KHI SỬ DỤNG SET

Truy cập vào thành phần Set

KHÔNG thể truy cập vào thành phần set.

0	1	2	3	4
13	29	42	65	75

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

KHÔNG thể duyệt set theo dạng vector hoặc mảng tĩnh.

Duyệt các phần tử của Set

Duyệt bằng cách sử dụng iterator (duyệt xuôi).

0	1	2	3	4
13	29	42	65	75

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

Kết quả

13, 29, 42, 65, 75

Duyệt các phần tử của Set

Duyệt bằng cách sử dụng iterator (duyệt ngược).


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

Kết quả

75, 65, 42, 29, 13

MULTISET

Định nghĩa: Tương tự như set, multiset có cách khai báo và sử dụng các hàm tương tự như set nhưng mutilset các phần tử có thể có giá trị giống nhau.

Multiset cũng sử dụng thư viện #include <set>. Cách khai báo và sử dụng multiset tương tự như cách sử dụng set.

Hỏi đáp