REPORT BIG ASSIGNMENT
Chương 3

Họ tên: Nguyễn Anh Tuấn
MSSV: 20200400

Requirement:

Cài đặt trực quan mô phỏng 1 cây nhị phân tìm kiếm gồm:

- Tạo cây
- Duyệt cây thêm node
- Xoá node
- Tìm kiếm phần tử trong cây

Bài làm:

Code:

Khởi tạo cấu trúc cho Node và cho cây:

```
#include <queue>
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;
};

struct Tree {
    Node* root;
};
```

Hàm tạo node với data đi kèm:

```
Node* getNode(int value){
    Node* node = new Node;
    node->data = value;
    node->left = NULL;
    node->right = NULL;
    return node;
}
```

Hàm thêm một node vào cây:

```
void addNode(Tree& tree, int value) {
    Node* newNode = getNode(value);
    if (tree.root == NULL) {
        tree.root = newNode;
        return;
    }
    Node* tmp = tree.root;
    while (true) {
        if (value < tmp->data) {
            if (tmp->left == NULL) {
                tmp->left = newNode;
                break;
            } else {
                tmp = tmp->left;
            }
        } else {
            if (tmp->right == NULL) {
                tmp->right = newNode;
                break;
            } else {
                tmp = tmp->right;
            }
        }
    }
}
```

Hàm duyệt và in cây ra màn hình lần lượt từng node theo dạng Left Node Right:

```
void traverse(Node* root){
    if ( root == NULL ) return;
    traverse(root->left);
    cout << root->data << " ";
    traverse(root->right);
}
```

Hàm tạo cây với n phần tử nhập vào từ bàn phím

```
void createTree(Tree* tree, int n){
    cout << "Nhap gia tri tung phan tu: \n";
    for ( int i = 0 ; i < n ; i++){
        int tmp;
        cin >> tmp;
        addNode(*tree, tmp);
    }
}
```

Hàm khởi tạo cây rỗng

```
void init(Node* root){
    root = NULL;
}
```

Hàm tìm kiếm Node với giá trị đi kèm và giá trị trả về là một Node tại giá trị tìm đc đi kèm với tất cả các Node con của nó: (chi phí $O(nlog_2 n)$)

```
Node* search(Node* root, int x) {
    Node* tmp = root;
    while (tmp != NULL) {
        if (x == tmp->data) {
            return tmp;
        } else if (x < tmp->data) {
            tmp = tmp->left;
        } else {
            tmp = tmp->right;
        }
    }
    return NULL;
}
```

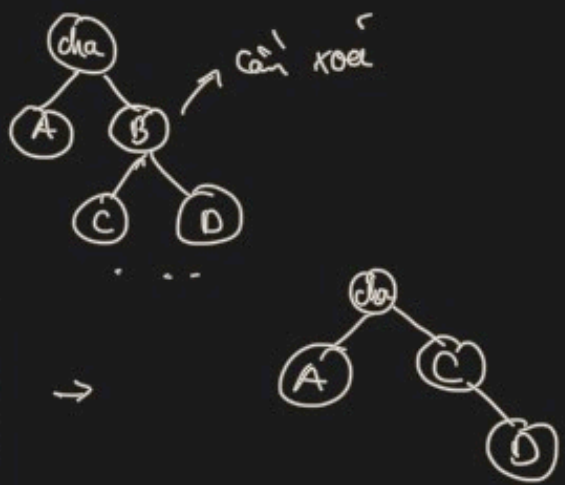
Hàm xoá Node với giá trị đi kèm + giữ lại các Node con của nó và sắp xếp lại vị trí cây:

```cpp
// Delete Node and re-arrange the tree
void deleteNode(Node* root, int x) {
    Node* tmp = root;
    Node* parent = NULL;

    while (tmp != NULL && tmp->data != x) {
        parent = tmp;
        if (x < tmp->data) {
            tmp = tmp->left;
        } else {
            tmp = tmp->right;
        }
    }
    if (tmp == NULL) {
        return;
    }
    if (tmp->left == NULL) {
        Node* temp = tmp->right;
        if (parent == NULL) {
            root = temp;
        } else if (tmp == parent->left) {
            parent->left = temp;
        } else {
            parent->right = temp;
        }
        delete tmp;
    } else if (tmp->right == NULL) {
        Node* temp = tmp->left;
        if (parent == NULL) {
            root = temp;
        } else if (tmp == parent->left) {
            parent->left = temp;
        } else {
            parent->right = temp;
        }
        delete tmp;
    } else {
        Node* min = tmp->right;
        parent = tmp;
        while (min->left != NULL) {
            parent = min;
            min = min->left;
        }
        tmp->data = min->data;
        if (parent->left == min) {
            parent->left = min->right;
        } else {
            parent->right = min->right;
        }
        delete min;
    }
}
```

Tìm node cần xoá trong cây
mỗi lần so sánh với data của left & right
để node cha được lưu lại
thuỷ method nếu ko có node cần xoá trong cây

Sắp xếp lại Node cha + Node con
của Node con về trái hoặc phải
của Node cha



Hàm tìm và xoá node đi kèm với giá trị đầu vào tương ứng sau đó xoá hết tất cả con của nó:

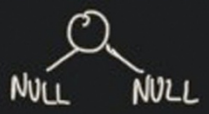
```
// Delete node and remove all child
void deleteNodeWithChild(Tree tree, int x){
    Node* tmp = tree.root;
    Node* parent = NULL;
    if ( tmp->data == x ){
        tree.root = NULL;
        return;
    }
    while (tmp != NULL) {
        if (x == tmp->data) {
            if (tmp->left != NULL) {
                tmp->left = NULL;
            }
            if (tmp->right != NULL) {
                tmp->right = NULL;
            }
            if (parent == NULL) {
                tree.root = NULL;
            } else if (parent->left == tmp) {
                parent->left = NULL;
            } else {
                parent->right = NULL;
            }
            delete tmp;
            return;
        } else if (x < tmp->data) {
            parent = tmp;
            tmp = tmp->left;
        } else {
            parent = tmp;
            tmp = tmp->right;
        }
    }
}
```

→ Khai báo con trỏ đến Node root của cây

→ Node trỏ đến vị trí cha của node cần xoá

nếu root mang data cần xoá
thì trả về root null = cây rỗng

Tại vị trí tìm kiếm ra node cho các con = NULL



Tại parent nếu node con phải hay trái là node cần tìm kiếm xoá thì gán cả 2 là null

Xoá node root tạm giải phóng bộ nhớ và return

Check và tìm kiếm xuống cây nhớ lưu sang node con left hoặc right thì node cha nhận giá trị trước đó của node con.

Hàm main:

```
int main() {
    Tree* tree = new Tree;
    init(tree->root);

    int n;
    cout << "Nhap so luong phan tu: \n";
    cin >> n;
    createTree(tree, n);

    cout << "\nCay nhi phan tim kiem vua tao: \n";
    traverse(tree->root);

    int node;
    cout << "\nNhap gia tri muon tim kiem: ";
    cin >> node;
    Node* searchNode = search(tree->root, node);
    cout << "Node vua tim kiem duoc:\n";
    traverse(searchNode);

    int val;
    cout << "\nNhap gia tri data muon xoa trong cay: ";
    cin >> val;
    cout << "Cay nhi phan tim kiem sau khi xoa node " << val << " va cac con cua no:\n";
    deleteNodeWithChild(*tree,val);
    traverse(tree->root);

    int del;
    cout << "\nNhap gia tri data muon xoa trong cay va sap xep lai cay: ";
    cin >> del;
    cout << "Cay nhi phan tim kiem sau khi xoa node " << del << " va sap xep lai vi tri cay:\n";
    deleteNode(tree->root,del);
    traverse(tree->root);

    delete tree;
}
```

Kết quả chụp màn hình :

```
(base) macad@nganhhtuann-3 output % ./"binarySearchTree"
Nhap so luong phan tu:
5
Nhap gia tri tung phan tu:
5
7
2
8
3

Cay nhi phan tim kiem vua tao:
2 3 5 7 8
Nhap gia tri muon tim kiem: 2
Node vua tim kiem duoc:
2 3
Nhap gia tri data muon xoa trong cay: 7
Cay nhi phan tim kiem sau khi xoa node 7 va cac con cua no:
2 3 5
Nhap gia tri data muon xoa trong cay va sap xep lai cay: 2
Cay nhi phan tim kiem sau khi xoa node 2 va sap xep lai vi tri cay:
3 5 %
```

Nhận xét:

- Cài đặt thuật toán xoá Node và sắp xếp lại cây tốn chi phí khá lớn và phức tạp.