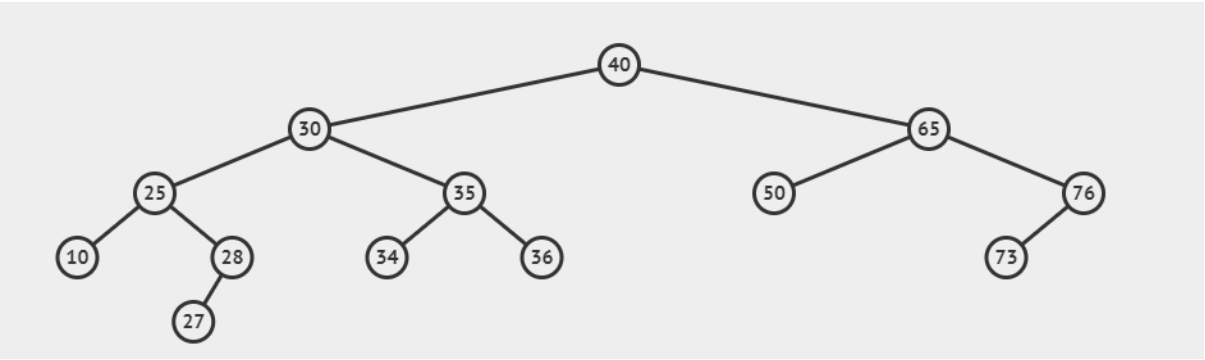
Bài 1.



Duyệt theo thứ tự trước - Preorder

40 – 30 – 25 – 10 – 28 – 27 – 35 – 34 – 36 – 65 – 50 – 76 – 73

Duyệt theo thứ tự giữa - Inorder

10 – 25 – 27 – 28 – 30 – 34 – 35 – 36 – 40 – 50 – 65 – 73 – 76

Duyệt theo thứ tự sau – Postorder

10 – 27 – 28 – 25 – 34 – 36 – 35 – 30 – 50 – 73 – 75 – 65 – 40

Xóa 1 node

Phần A: Tìm kiếm

Bước 1. Gán 1 node temp là node hiện tại. Bắt đầu giải thuật temp là root

Bước 2. Kiểm tra giá trị của node hiện tại

2.1 Nếu là Node cần tìm. Thực hiện sang phần B

2.2 Nếu giá trị cần tìm < node hiện tại. Tìm kiếm ở subtree trái

2.3 Nếu giá trị cần tìm > node hiện tại. Tìm kiếm ở subtree phải

Lặp lại cho đến khi tìm được node cần xóa

Phần B: Xóa

Trường hợp 1: Xóa node lá

- Xóa bình thường

Trường hợp 2: Xóa Node 1 con (one children)

- Xóa node
- Gắn node con vào vị trí của node đã xóa

Trường hợp 3: Xóa Node 2 con (two children)

- Thay thế node cần xóa với node successor của nó. Node successor là node con của node cần xóa mà có giá trị lớn hơn node cần xóa nhưng lại nhỏ nhất trong tất cả các node con còn lại
- Tìm Node successor bằng cách thăm node bên phải của node cần xóa. Rồi tiếp tục thăm các node trái đến khi không thăm được nữa

Ví dụ: Xóa node 65

- Node successor là 73
- Thay thế 65 = 73

Trường hợp 4: Xóa Node 2 con nhưng node successor có 1 node con phải

- Tương tự trường hợp 3. Nhưng sau khi thay thế node cần xóa = node successor. Gán node phải của node successor vào node trái của node parent của node successor.

Bài 2.

```cpp
#include<iostream>
using namespace std;

//Creating a node
struct Node {
    int value;
    Node* left;
    Node* right;
};

//init a node
Node* create_node(int value) {
    Node* temp = new Node;
    temp->value = value;
    temp->left = temp->right = nullptr;
    return temp;
}

//insert a key to BST
Node* insert(Node* root, int value){
    if(root == nullptr) {
        return create_node(value);
    }
    if(value < root->value) {
        root->left = insert(root->left, value);
    }
    else {
        root->right = insert(root->right, value);
    }
    return root;
}

//find node with the maximum key value
Node* find_maximum(Node* root) {
    Node* max;
    if(root == nullptr) {
        return nullptr;
    }
    max = root;
    while(max->right != nullptr)
    {
        max = max->right;
    }
    return max;
}

Node* find_minimum(Node* root) {
```

```
    Node* min;
    if(root == nullptr) {
        return nullptr;
    }
    min = root;
    while(min->left != nullptr)
    {
        min = min->left;
    }
    return min;
}
int find_min(Node* root){
    Node* temp = find_minimum(root);
    return temp->value;
}
int find_max(Node* root) {
    Node* temp = find_maximum(root);
    return temp->value;
}

void inorder(Node* root) {
    if(root == nullptr) {
        return;
    }
    inorder(root->left);
    cout << root->value << " ";
    inorder(root->right);
}

void pre_order(Node* root) {
    if(root == nullptr) {
        return;
    }
    cout << root->value << " ";
    pre_order(root->left);
    pre_order(root->right);
}

void post_order(Node* root) {
    if(root == nullptr) {
        return;
    }
    post_order(root->left);
    post_order(root->right);
    cout << root->value << " ";
}
int count_even(Node* root) {
    if(root == nullptr) {
```

```
        return 0;
    }
    int n = count_even(root->left) + count_even(root->right);
    if(root->value % 2 == 0){
        return n + 1;
    }
    else {
        return n;
    }
}


int main() {
    Node* root = nullptr;
    root = insert(root, 40);
    root = insert(root, 30);
    root = insert(root, 65);
    root = insert(root, 25);
    root = insert(root, 50);
    root = insert(root, 76);
    root = insert(root, 35);
    root = insert(root, 10);
    root = insert(root, 28);
    root = insert(root, 27);
    root = insert(root, 73);
    root = insert(root, 36);
    root = insert(root, 34);
    //pre_order(root);
    post_order(root);
    cout << "\nMinimum value:" << find_min(root) << "\n";
    cout << "Maximum Value: " << find_max(root) << "\n";
    cout << "Number of even number: " << count_even(root) << "\n";
    return 0;
}
```