Binary Search Tree Insert (không dùng đệ quy)

```
#include <bits/stdc++.h>
using namespace std;


class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:

        void preOrder(Node *root) {

        if( root == NULL )
                return;

        std::cout << root->data << " ";

        preOrder(root->left);
        preOrder(root->right);
        }

/* you only have to complete the function given below.
Node is defined as

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

*/

    Node * insert(Node * root, int data) {

        // no recursion
        Node* cur = root;
        Node* prev = NULL;
        Node* newNode = new Node(data);
        while(cur != NULL) {
            prev = cur;
            if(data <= cur->data) {
                cur = cur->left;
            } else {
```

```
                cur = cur->right;
            }
        }
        if(prev == NULL) {
            return newNode;
        } else if(data <= prev->data) {
            prev->left = newNode;
        } else {
            prev->right = newNode;
        }
        return root;
    }
};

int main() {

    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    myTree.preOrder(root);

    return 0;
}
```

Binary Search Tree Insert

```cpp

#include <bits/stdc++.h>
using namespace std;


class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:

        void preOrder(Node *root) {

        if( root == NULL )
                return;

        std::cout << root->data << " ";

        preOrder(root->left);
        preOrder(root->right);
        }

/* you only have to complete the function given below.
Node is defined as

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

*/

    Node * insert(Node * root, int data) {

        if(root == NULL) {
            return new Node(data);
        } else {
            Node* cur;
            if(data <= root->data) {
                cur = insert(root->left, data);
                root->left = cur;
            } else {
                cur = insert(root->right, data);
```

```
                root->right = cur;
            }

            return root;
        }
    }
};

int main() {

    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    myTree.preOrder(root);

    return 0;
}
```

Binary Search Tree Nút chung gần nhất

```
#include <bits/stdc++.h>

using namespace std;

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:
                Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
                }

                return root;
            }
        }

/*The tree node has data, left child and right child
class Node {
    int data;
    Node* left;
    Node* right;
};

*/

    Node *lca(Node *root, int v1,int v2) {
        if (root == NULL) {
            return NULL;
        }
        if (root->data == v1 || root->data == v2) {
            return root;
        }
        Node* left = lca(root->left, v1, v2);
        Node* right = lca(root->right, v1, v2);
        if (left != NULL && right != NULL) {
            return root;
        }
        return (left != NULL) ? left : right;
    }
```

```
60
61  }; //End of Solution
62
63  int main() {
64
65      Solution myTree;
66      Node* root = NULL;
67
68      int t;
69      int data;
70
71      std::cin >> t;
72
73      while(t-- > 0) {
74          std::cin >> data;
75          root = myTree.insert(root, data);
76      }
77
78          int v1, v2;
79          std::cin >> v1 >> v2;
80
81      Node *ans = myTree.lca(root, v1, v2);
82
83          std::cout << ans->data;
84
85      return 0;
86  }
87
```

khangtd.HoaNhac

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define fi first
#define se second
const int N = 1e6 + 9;
const int N2 = N * 10;
const int mod = 1e9 + 7;
const int inf = LLONG_MAX;



signed main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}

    int n;
    cin >> n;
    vector<int> v(n);
    for (int i = 0; i < n; ++i) {
        cin >> v[i];
    }
    stack<pair<int, int>> st;
    int ans = 0;

    for (int i = 0; i < n; ++i) {
        int c = 1;
        while (!st.empty() && st.top().fi <= v[i]) {
            if (st.top().fi < v[i]) {
                ans += st.top().se;
                st.pop();
            } else {
                ans += st.top().se;
                c += st.top().se;
                st.pop();
            }
        }
        if (!st.empty()) {
            ans += 1;
        }
        st.push(make_pair(v[i], c));
    }
    cout << ans;

}
```

khangtd.HuffmanCoding

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define fi first
#define se second
const int N = 1e6 + 9;
const int N2 = N * 10;
const int mod = 1e9 + 7;
const int inf = LLONG_MAX;

int huffman(string s){
    int n = s.size();
    vector<int> cnt(256, 0);
    for (char c : s) cnt[c]++;
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int i = 0; i < 256; i++) if (cnt[i]) pq.push(cnt[i]);

    if (pq.size() == 1) return n;

    int ans = 0;
    while (pq.size() > 1){
        int a = pq.top(); pq.pop();
        int b = pq.top(); pq.pop();
        ans += a + b;
        pq.push(a + b);
    }
    return ans;
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}

    int n;
    cin >> n;
    string s;
    cin >> s;
    cout << huffman(s);

}
```

khangtd.KetBan

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define fi first
#define se second
const int N = 1e6 + 9;
const int N2 = N * 10;
const int mod = 1e9 + 7;
const int inf = LLONG_MAX;



signed main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}

    int n, m;
    cin >> n >> m;
    vector<int> prev(n + 1, 0);
    vector<int> next(n + 1, 0);

    for (int i = 1; i <= n; i++){
        prev[i] = i - 1;
        next[i] = i + 1;
    }
    prev[1] = n;
    next[n] = 1;

    while (m--){
        int x, y;
        cin >> x >> y;

        int a = prev[x];
        int b = next[x];

        next[a] = b;
        prev[b] = a;

        next[x] = next[y];
        prev[next[y]] = x;
        prev[x] = y;
        next[y] = x;

    }
    int tmp = 1;
    for (int i = 1; i <= n; i++){
        cout << tmp << " ";
        tmp = next[tmp];
    }
}
```

Tree Hieght of Tree

```
#include <bits/stdc++.h>

using namespace std;

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:
                Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
                }

                return root;
            }
        }
/*The tree node has data, left child and right child
class Node {
    int data;
    Node* left;
    Node* right;
};

*/
    int height(Node* root) {

        Node* cur = root;
        queue<pair<Node*, int>> q;
        q.push({cur, 1});
        int maxHeight = 0;
        while(!q.empty()) {
            pair<Node*, int> p = q.front();
            maxHeight = max(maxHeight, p.second);
            q.pop();
            Node* node = p.first;
            int height = p.second;
            if (node->left != NULL) {
                q.push({node->left, height + 1});
            }
            if (node->right != NULL) {
```

```
                q.push({node->right, height + 1});
            }
        }
        return maxHeight;
    }

}; //End of Solution

int main() {

    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    int height = myTree.height(root);

        std::cout << height;

    return 0;
}
```

Tree Inorder Traversal (LNR) - Duyệt cây BST theo LNR

```
#include <bits/stdc++.h>
using namespace std;


class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:
                Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
                }

               return root;
            }
        }

/* you only have to complete the function given below.
Node is defined as

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

*/

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

```
60          inOrder(root->right);
61      }
62
63  }; //End of Solution
64
65  int main() {
66
67      Solution myTree;
68      Node* root = NULL;
69
70      int t;
71      int data;
72
73      std::cin >> t;
74
75      while(t-- > 0) {
76          std::cin >> data;
77          root = myTree.insert(root, data);
78      }
79
80      myTree.inOrder(root);
81
82      return 0;
83  }
```

Tree Inorder Traversal (LNR) II - Duyệt cây BST theo LNR không đệ quy

```
#include <bits/stdc++.h>
using namespace std;


class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:
                Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
                }

               return root;
            }
        }

/* you only have to complete the function given below.
Node is defined as

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

*/

    void inOrder(Node *root) {
        stack<Node*> s;
        Node* curr = root;
        while (curr != NULL || !s.empty()) {
            while (curr != NULL) {
                s.push(curr);
```

```
                curr = curr->left;
            }
            curr = s.top();
            s.pop();
            cout << curr->data << " ";
            curr = curr->right;
        }
    }

}; //End of Solution

int main() {

    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    myTree.inOrder(root);

    return 0;
}
```

Tree levelOrder Traversal - Duyệt cây BST theo chiều rộng

```cpp

#include <bits/stdc++.h>
using namespace std;


class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:
                Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
                }

               return root;
           }
        }

/* you only have to complete the function given below.
Node is defined as

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

*/

    void levelOrder(Node *root) {
        if (root == NULL) {
            return;
        }
        queue<Node*> q;
        q.push(root);
```

```
        while (!q.empty()) {
            Node* node = q.front();
            cout << node->data << " ";
            q.pop();
            if (node->left != NULL) {
                q.push(node->left);
            }
            if (node->right != NULL) {
                q.push(node->right);
            }
        }
    }

}; //End of Solution

int main() {

    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    myTree.levelOrder(root);

    return 0;
}
```

Tree Postorder Traversal (LRN) - Duyệt cây BST theo LRN

```
#include <bits/stdc++.h>
using namespace std;


class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:
                Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
                }

               return root;
            }
        }

/* you only have to complete the function given below.
Node is defined as

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

*/

    void postOrder(Node *root) {

        if (root == NULL) {
            return;
        }
        postOrder(root->left);
```

```
        postOrder(root->right);
        cout << root->data << " ";

    }

}; //End of Solution

int main() {

    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    myTree.postOrder(root);

    return 0;
}
```

Tree Postorder Traversal (LRN) II - Duyệt cây BST theo LRN không đệ quy

```
#include <bits/stdc++.h>
using namespace std;


class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:
                Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
                }

               return root;
            }
        }

/* you only have to complete the function given below.
Node is defined as

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

*/

    void postOrder(Node *root) {

        stack<Node*> s;
        Node* curr = root;
        Node* prev = NULL;
        while (curr != NULL || !s.empty()) {
```

```
            while (curr != NULL) {
                s.push(curr);
                curr = curr->left;
            }
            curr = s.top();
            if (curr->right == NULL || curr->right == prev) {
                cout << curr->data << " ";
                s.pop();
                prev = curr;
                curr = NULL;
            } else {
                curr = curr->right;
            }
        }

    }

}; //End of Solution

int main() {

    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    myTree.postOrder(root);

    return 0;
}
```

Tree Top view

```cpp
#include <bits/stdc++.h>
#include <regex>
using namespace std;
#define int long long
#define fi first
#define se second
const int N = 1e6 + 9;
const int N2 = N * 10;
const int mod = 1e9 + 7;
const int inf = LLONG_MAX;

map<int, int> check;

struct Node{
    int data;
    Node *left, *right;
    Node(int x) : data(x), left(NULL), right(NULL) {}
};

struct BST{
    Node *root;

    BST() : root(NULL) {}

    void insert(int x){
        if (root == NULL){
            root = new Node(x);
        } else {
            insertHelper(root, x);
        }
    }

    void insertHelper(Node *node, int x){
        if (x <= node->data){
            if (node->left == NULL){
                node->left = new Node(x);
            } else {
                insertHelper(node->left, x);
            }
        } else {
            if (node->right == NULL){
                node->right = new Node(x);
            } else {
                insertHelper(node->right, x);
            }
        }
    }

    void printRoot(){
        if (root != NULL){
            cout << root->data << endl;
        } else {
            cout << "Tree is empty" << endl;
        }
    }

    void traverseFromRoot(){
        if (root == NULL) return;
        queue<pair<Node*, int>> q;
```

```
        q.push({root, 0});
        while (!q.empty()){
            pair<Node*, int> p = q.front();
            Node *current = p.first;
            int level = p.second;
            if (check[level] == 0){
                // cout << current->data << " " << level << "\n";
                check[level] = current -> data; // Mark this level as visite
            }
            q.pop();
            if (current->right != NULL) q.push({current->right, level + 1});
            if (current->left != NULL) q.push({current->left, level - 1});
        }
    }

};

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}

    int n;
    cin >> n;

    BST tree;
    for (int i = 0; i < n; i++){
        int x;
        cin >> x;
        tree.insert(x);
    }
    // tree.printRoot();
    tree.traverseFromRoot();
    vector<int> levels;
    for (auto it : check){
        levels.push_back(it.second);
    }
    sort(levels.begin(), levels.end());
    for (int i = 0; i < levels.size(); i++){
        cout << levels[i] << " ";
    }
}
```