**LARGEST INDEPENDENT SET:**

#include <bits/stdc++.h>

using namespace std;

struct Node

{

    int data;

    struct Node \*left;

    struct Node \*right;

    Node(int x)

    {

        data = x;

        left = NULL;

        right = NULL;

    }

};

int LISS(struct Node \*root);

/\* Driver program to test size function\*/

int main()

{

    int t;

    struct Node \*child;

    scanf("%d\n", &t);

    while (t--)

    {

        map<int, Node \*> m;

        int n;

        scanf("%d\n", &n);

        struct Node \*root = NULL;

        while (n--)

        {

            Node \*parent;

            char lr;

            int n1, n2;

            scanf("%d %d %c", &n1, &n2, &lr);

            if (m.find(n1) == m.end())

            {

                parent = new Node(n1);

                m[n1] = parent;

                if (root == NULL)

                    root = parent;

            }

            else

                parent = m[n1];

            child = new Node(n2);

            if (lr == 'L')

                parent->left = child;

            else

                parent->right = child;

            m[n2] = child;

        }

        cout << LISS(root) << endl;

    }

    return 0;

}

int LISS(struct Node \*root)

{

    //Code here

    if (!root)

        return 0;

    int in = 0, ex = 0;

    ex = LISS(root->right) + LISS(root->left);

    in = 1;

    if (root->left)

        in += LISS(root->left->left) + LISS(root->left->right);

    if (root->right)

        in += LISS(root->right->left) + LISS(root->right->right);

    return max(in, ex);

}

**MAXIMUM SUM OF NON-ADJACENT NODES:**

#include <bits/stdc++.h>

using namespace std;

// Tree Node

struct Node

{

    int data;

    Node \*left;

    Node \*right;

};

Node \*newNode(int val)

{

    Node \*temp = new Node;

    temp->data = val;

    temp->left = NULL;

    temp->right = NULL;

    return temp;

}

Node \*buildTree(string str)

{

    if (str.length() == 0 || str[0] == 'N')

        return NULL;

    vector<string> ip;

    istringstream iss(str);

    for (string str; iss >> str;)

        ip.push\_back(str);

    Node \*root = newNode(stoi(ip[0]));

    queue<Node \*> queue;

    queue.push(root);

    int i = 1;

    while (!queue.empty() && i < ip.size())

    {

        Node \*currNode = queue.front();

        queue.pop();

        string currVal = ip[i];

        if (currVal != "N")

        {

            currNode->left = newNode(stoi(currVal));

            queue.push(currNode->left);

        }

        i++;

        if (i >= ip.size())

            break;

        currVal = ip[i];

        if (currVal != "N")

        {

            currNode->right = newNode(stoi(currVal));

            queue.push(currNode->right);

        }

        i++;

    }

    return root;

}

int getMaxSum(Node \*root);

int findMax(int a, int b)

{

    return a > b ? a : b;

}

int getSum(Node \*root)

{

    if (!root)

        return 0;

    int in = 0, ex = 0;

    ex = getSum(root->left) + getSum(root->right);

    in = root->data;

    if (root->left)

        in += getSum(root->left->left) + getSum(root->left->right);

    if (root->right)

        in += getSum(root->right->left) + getSum(root->right->right);

    return findMax(in, ex);

}

int getMaxSum(Node \*root)

{

    return getSum(root);

}

int main()

{

    int t;

    scanf("%d ", &t);

    while (t--)

    {

        string s;

        getline(cin, s);

        Node \*root = buildTree(s);

        cout << getMaxSum(root) << endl;

    }

    return 0;

} // } Driver Code Ends

**POST-ORDER TO PREORDER:**

#include <bits/stdc++.h>

using namespace std;

typedef struct Node

{

    int data;

    struct Node \*left, \*right;

} Node;

// A Stack has array of Nodes, capacity, and top

typedef struct Stack

{

    int top;

    int capacity;

    Node \*\*array;

} Stack;

// A utility function to create a new tree node

Node \*newNode(int data)

{

    Node \*temp = (Node \*)malloc(sizeof(Node));

    temp->data = data;

    temp->left = temp->right = NULL;

    return temp;

}

Stack \*createStack(int capacity)

{

    Stack \*stack = (Stack \*)malloc(sizeof(Stack));

    stack->top = -1;

    stack->capacity = capacity;

    stack->array = (Node \*\*)malloc(stack->capacity \* sizeof(Node \*));

    return stack;

}

int isFull(Stack \*stack)

{

    return stack->top == stack->capacity - 1;

}

int isEmpty(Stack \*stack)

{

    return stack->top == -1;

}

void push(Stack \*stack, Node \*item)

{

    if (isFull(stack))

        return;

    stack->array[++stack->top] = item;

}

Node \*pop(Stack \*stack)

{

    if (isEmpty(stack))

        return NULL;

    return stack->array[stack->top--];

}

Node \*peek(Stack \*stack)

{

    return stack->array[stack->top];

}

bool canRepresentBST(int pre[], int n)

{

    stack<int> s;

    // value

    int root = INT\_MIN;

    for (int i = 0; i < n; i++)

    {

        if (pre[i] < root)

            return false;

        while (!s.empty() && s.top() < pre[i])

        {

            root = s.top();

            s.pop();

        }

        s.push(pre[i]);

    }

    return true;

}

void printPostorder(Node \*node)

{

    if (node == NULL)

        return;

    printPostorder(node->left);

    printPostorder(node->right);

    printf("%d ", node->data);

}

Node \*constructTree(int \*, int);

int main()

{

    int arr[1000], x, t, n;

    cin >> t;

    while (t--)

    {

        cin >> n;

        for (int i = 0; i < n; i++)

            cin >> arr[i];

        printPostorder(constructTree(arr, n));

        cout << endl;

    }

    return 0;

}

Node \*insert(Node \*root, int data)

{

    if (!root)

        return newNode(data);

    if (data > root->data)

        root->right = insert(root->right, data);

    if (data < root->data)

        root->left = insert(root->left, data);

    return root;

}

Node \*constructTree(int pre[], int size)

{

    Node \*root = NULL;

    root = newNode(pre[0]);

    for (int i = 1; i < size; i++)

    {

        insert(root, pre[i]);

    }

    return root;

}

**TERNARY OPERATION TO BINARY TREE:**

#include <bits/stdc++.h>

using namespace std;

struct Node

{

    char data;

    struct Node \*left;

    struct Node \*right;

    Node(char x)

    {

        data = x;

        left = NULL;

        right = NULL;

    }

};

void preorder(Node \*root)

{

    if (root == NULL)

        return;

    cout << root->data << " ";

    preorder(root->left);

    preorder(root->right);

}

Node \*convertExpression(string str, int i);

int main()

{

    int t;

    cin >> t;

    while (t--)

    {

        string str;

        cin >> str;

        Node \*root = convertExpression(str, 0);

        preorder(root);

        cout << endl;

    }

}

Node \*convertExpression(string str, int i)

{

    if (i > str.length())

        return NULL;

    Node \*root = new Node(str[i++]);

    if (i < str.length() && str[i] == '?')

        root->left = convertExpression(str, i + 1);

    if (i < str.length() && str[i] == ':')

        root->right = convertExpression(str, i + 1);

    return root;

}