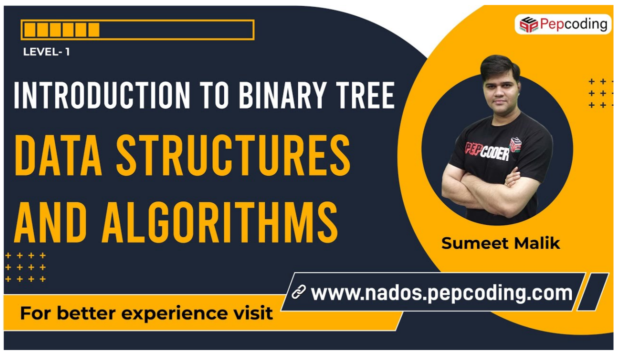
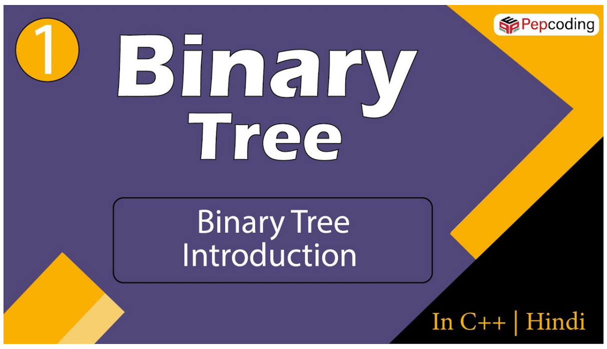
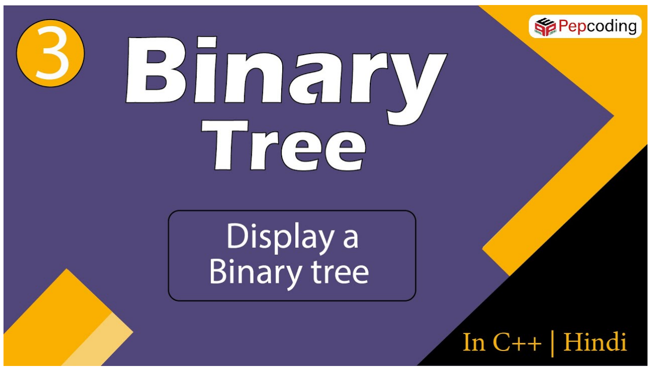
Binary Tree - Introduction And Data Members





Binary Tree - Constructor

Display A Binary Tree



Size, Sum, Maximum And Height Of A Binary Tree

Easy

1. You are given a partially written BinaryTree class.
2. You are required to complete the body of size, sum, max and height function. The functions are expected to
   2.1. size - return the number of nodes in BinaryTree
   2.2. sum - return the sum of nodes in BinaryTree
   2.3. max - return the highest node in BinaryTree
   2.4. height - return the height of tree in terms of edges
3. Input and Output is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

Output is managed for you

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
```

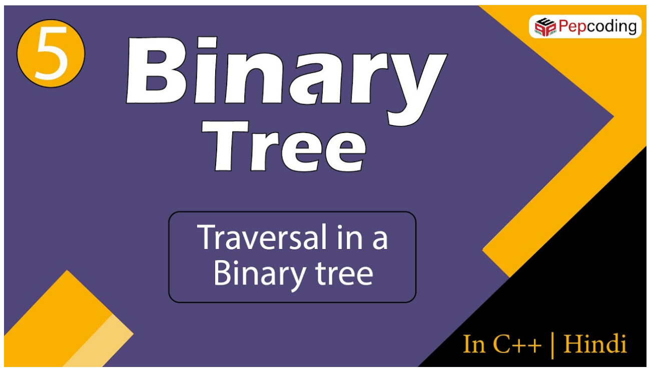
Sample Output

```
9
448
87
3
```

```cpp
#include<iostream>
#include<climits>
#include<string.h>
#include<vector>

using namespace std;

// structure of node
class Node
{
public:
    int data;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data)
    {
        this->data = data;
    }
};

class Pair {
    public:
    Node * node = nullptr;
    int state=0;

    Pair(Node *node, int state) {
      this->node = node;
      this->state = state;
    }
  };

int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++]);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

void display(Node *node)
{
    if (node == nullptr)
        return;
    string str = "";
    str += node->left != nullptr ? to_string(node->left->data) :
".";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right != nullptr ? to_string(node->right->data) :
".";
    cout << str << endl;

    display(node->left);
    display(node->right);
}


int size(Node *node)
{
   //write your code here
   if (node == NULL){
       return 0;
   }

   int s {};
   s += size(node->left);
   s += size(node->right);

   return s+1;
}

int height(Node *node)
{
    //write your code here
    //write your code here
//    if (node == NULL){
//        return 0;
//    }

//     if (node->left == NULL && node->right == NULL){
//        return 0;
//    }
   if (node == NULL){
       return -1;
   }
   int h1  = height(node->left);
   int h2  = height(node->right);

    int ch = h1 > h2 ? h1 : h2;
    // cout<<node->data<<" "<<ch+1<<endl;
   return ch+1;
}

int maximum(Node *node)
{
    //write your code here
    if (node == NULL){
```

```cpp
        return INT_MIN;
    }

    int a = maximum (node->left);
    int b = maximum (node->right);
     int c = a>b ? a : b;
     int ans = node->data > c ? node->data : c;

    return ans;
}

int sum(Node *root)
{
     //write your code here
     if (root == NULL){
        return 0;
    }

    int sm  {};
    sm += sum(root->left);
    sm += sum(root->right);

    return sm + root->data;

}

int main()
{

  int n;
  cin>>n;

  vector<int> arr(n,0);
  for(int i = 0; i < n; i++){
      string temp;
      cin>>temp;

      if(temp=="n")
      {
        arr[i] = -1;
      }
      else
      {
        arr[i] = stoi(temp);
      }
  }

   Node *root = constructTree(arr);

    int sz = size(root);
    int sm = sum(root);
    int max = maximum(root);
    int ht = height(root);
    cout<<sz<<endl;
    cout<<sm<<endl;
    cout<<max<<endl;
    cout<<ht<<endl;
  }
```

Traversal In A Binary Tree



Levelorder Traversal Of Binary Tree

Easy

1. You are given a partially written BinaryTree class.
2. You are required to complete the body of levelorder function. The function is expected to print tree level by level, left to right. Each level must be on a separate line and elements of same level should be separated by space
3. Input is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

Each level must be on a separate line and elements of same level should be separated by space

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
```

Sample Output

```
50
25 75
12 37 62 87
30 70
```

```
#include<iostream>
#include<string.h>
#include<vector>
#include<queue>

using namespace std;
class Node{
public:
  int data;
  Node* left = nullptr;
  Node* right = nullptr;
  Node(int data){
    this->data = data;
  }
};

class Pair {
public:
  Node* node = nullptr;
  int state = 0;
  Pair(Node* node, int state) {
    this->node = node;
    this->state = state;
  }
};

int idx = 0;
Node* constructTree(vector<int>& arr){
    if (idx == arr.size() || arr[idx] == -1){
    idx++;
    return nullptr;
  }Node* node = new Node(arr[idx++]);
  node->left = constructTree(arr);
  node->right = constructTree(arr);
  return node;
}

void display(Node* node){
  if (node == nullptr)
    return;
  string str = "";
  str += node->left != nullptr ? to_string(node->left->data) :
".";
  str += " <- " + to_string(node->data) + " -> ";
  str += node->right != nullptr ? to_string(node->right->data) :
".";
  cout << str << endl;
  display(node->left);
  display(node->right);
}
```

```
void levelOrder(Node* node) {
    // write your code here
    //see another solution in video
    if(node == NULL ){
        return ;
    }
    queue<Node*> q;
    q.push(node);
    queue<Node*> cq;
    while (q.empty() == false){
        /* code */
        Node* f = q.front();q.pop();
        cout<<f->data<<" ";

        if(f->left != NULL){
            cq.push(f->left);
        }
        if(f->right != NULL){
            cq.push(f->right);
        }
        if(q.empty()){
            q = cq;
            queue<Node*> temp;
            cq = temp;
            cout<<endl;
        }
    }
}
/*
 queue<Node*> q;
  q.push(node);
  queue<Node*> cq;
  while(q.size() > 0) {
    //REMOVE PRINT ADD
    Node * n = q.front();
    q.pop();
    cout<<n->data<<" ";
    for(Node* child:n->children) {
      cq.push(child);
    }
    if(q.empty()){
      q = cq;
      queue<Node*> temp;
      cq = temp;
      cout<<endl;
    }
  }*/
```

```cpp
int main(){
    int n;
  cin >> n;
  vector<int> arr(n, 0);
  for (int i = 0; i < n; i++) {
    string temp;
    cin >> temp;
    if (temp == "n"){
      arr[i] = -1;
    }else{
      arr[i] = stoi(temp);
    }
  }
  Node* root = constructTree(arr);
  levelOrder(root);
}
```

Iterative Pre, Post And Inorder Traversals Of Binary Tree

Easy

1. You are given a partially written BinaryTree class.
2. You are required to complete the body of iterativePrePostInTraversal function. The function is expected to print pre order, in order and post order of the tree in separate lines (first pre, then in and finally post order). All elements in an order must be separated by a space.
3. Input is managed for you.

Constraints

```
None
```

Format

Input

Input is managed for you

Output

pre order (elements separated by space)
in order (elements separated by space)
post order (elements separated by space)

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
```

Sample Output

```
50 25 12 37 30 75 62 70 87
12 25 30 37 50 62 70 75 87
12 30 37 25 70 62 87 75 50
```

```
#include<iostream>
#include<string.h>
#include<vector>
#include<queue>
#include<stack>
using namespace std;
class Node {
public:
  int data;
  Node* left = nullptr;
  Node* right = nullptr;
  Node(int data) {
    this->data = data;
  }
};

class Pair {
public:
  Node* node = nullptr;
```

```
    int state = 0;
    Pair(Node* node, int state) {
      this->node = node;
      this->state = state;
    }
  };

  Node* construct(vector<int>& arr) {
    Node* root = new Node(arr[0]);
    pair<Node*, int> p = {root, 1};

    stack<pair<Node*, int>> st;
    st.push(p);

    int idx = 1;
    while (!st.empty()) {
      if (st.top().second == 1) {
        st.top().second++;
        if (arr[idx] != -1) {
          st.top().first->left = new Node(arr[idx]);
          pair<Node*, int> lp = {st.top().first->left, 1};
          st.push(lp);
        }
        else {
          st.top().first->left = nullptr;
        }
        idx++;
      }
      else if (st.top().second == 2) {
        st.top().second++;
        if (arr[idx] != -1) {
          st.top().first->right = new Node(arr[idx]);
          pair<Node*, int> rp = {st.top().first->right, 1};
          st.push(rp);
        } else {
          st.top().first->right = nullptr;
        }
        idx++;
      }
      else {
        st.pop();
      }
    }
    return root;
  }


  void iterativePrePostInTraversal(Node* node) {
    // write your code here
    stack<Pair> st;
    Pair rp (node ,1);
    st.push(rp);
    string pre;
    string in;
    string post;

    while(st.size() > 0){
      // Pair t = st.top();st.pop();
      if(st.top().state == 1){//we are in preordre hence add it to
  pre, state++,go to left child
        pre += to_string(st.top().node->data) + " ";
        st.top().state += 1;
        if(st.top().node->left != NULL){
          Pair newp (st.top().node->left ,1);
          st.push(newp);
        }
      }else if(st.top().state == 2){//we are in inordre hence add it
  to in, state++,go to right child
        in += to_string(st.top().node->data) + " ";
        st.top().state += 1;
        if(st.top().node->right != NULL){
          Pair newp (st.top().node->right ,1);
          st.push(newp);
        }
      }else{//we are in postordre hence add it to post,pop from
  stack
        post += to_string(st.top().node->data) + " ";
        st.pop();
      }
    }
    cout<<pre<<endl;
    cout<<in<<endl;
    cout<<post<<endl;
  }

  int main() {
    int n;
    cin >> n;
    vector<int> arr(n, 0);
    for (int i = 0; i < n; i++) {
      string temp;
      cin >> temp;
      if (temp == "n") {
        arr[i] = -1;
      } else {
        arr[i] = stoi(temp);
      }
    }
    Node* root = construct(arr);
    iterativePrePostInTraversal(root);
  }
```

Find And Nodetorootpath In Binary Tree

Easy

1. You are given a partially written BinaryTree class. 2. You are given an element. 3. You are required to complete the body of find and nodeToRoot function. The functions are expected to 3.1. find -> return true or false depending on if the data is found in binary tree. 3.2. nodeToRoot -> returns the path from node (correspoding to data) to root in form of an arraylist (root being the last element) 4. Input iand Output is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

Output is managed for you

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
30
```

Sample Output

```
true
[30, 37, 25, 50]
```

```
#include <iostream>
#include <string.h>
#include <vector>

using namespace std;

class Node{
public:
  int data;
  Node* left = nullptr;
  Node* right = nullptr;
  Node(int data)
  {
    this->data = data;
  }
};

class Pair{
public:
  Node* node = nullptr;
  int state = 0;
  Pair(Node* node, int state)
  {
    this->node = node;
    this->state = state;
  }
};

int idx = 0;
Node* constructTree(vector<int>& arr){

  if (idx == arr.size() || arr[idx] == -1){
    idx++;
    return nullptr;
  }
  Node* node = new Node(arr[idx++]);
  node->left = constructTree(arr);
  node->right = constructTree(arr);
  return node;
}

void display(Node* node){
  if (node == nullptr)
    return;
  string str = "";
  str += node->left != nullptr ? to_string(node->left->data) :
".";
  str += " <- " + to_string(node->data) + " -> ";
  str += node->right != nullptr ? to_string(node->right->data) :
".";
  cout << str << endl;

  display(node->left);
  display(node->right);
}
bool find(Node* node, int data){
  // write your code here
  if(node == NULL){
    return false;
  }
  if(node->data == data){
    return true;
  }

  bool a = find(node->left,data);
  if(a){
    return true;
  }

  bool b = find(node->right,data);
  if(b){
    return true;
  }
  return false;
}

vector<int> nodeToRootPath(Node* node, int data){
  // write your code here
  if(node == NULL){
    vector<int> vec;
    return vec;
  }
  if(node->data == data){
    vector<int> vec;
    vec.push_back(node->data);
    return vec;
  }
  vector<int> left = nodeToRootPath(node->left,data);
  vector<int> right = nodeToRootPath(node->right,data);
  if(left.size() > 0){
    left.push_back(node->data);
    return left;
  }
  if(right.size() > 0){
    right.push_back(node->data);
    return right;
  }

  vector<int> vec;
  return vec;

}

int main(){
  int n;
  cin >> n;
  vector<int> arr(n, 0);
  for (int i = 0; i < n; i++){
    string temp;
    cin >> temp;
    if (temp == "n"){
      arr[i] = -1;
    }else{
      arr[i] = stoi(temp);
    }
  }

  Node* root = constructTree(arr);
  int data;
  cin >> data;
  bool found = find(root, data);
  found == 1 ? cout << "true" << endl : cout << "false" << endl;
  vector<int> path = nodeToRootPath(root, data);
  cout << "[";
  for (int i = 0; i < path.size(); i++) {
    cout << path[i];
    if (i != path.size() - 1) {
      cout << ", ";
    }
  }
  cout << "]" << endl;
}
```

Print K Levels Down
Easy

1. You are given a partially written BinaryTree class.
2. You are given a value k.
3. You are required to complete the body of printKLevelsDown function. The function is expected to print in different lines all nodes which are k level deep. Use preorder for printing.
4. Input is managed for you.

Constraints

```
None
```

Format

Input

Input is managed for you

Output

All nodes k-level deep printed in separated lines and visited in preorder

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
3
```

Sample Output

```
30
70
```

```
#include<iostream>
#include<string.h>
#include<vector>
using namespace std;
class Node{
public:
  int data;
  Node* left = nullptr;
  Node* right = nullptr;
  Node(int data){
    this->data = data;
  }
};

class Pair {
public:
  Node* node = nullptr;
  int state = 0;

  Pair(Node* node, int state) {
    this->node = node;
    this->state = state;
  }
};

int idx = 0;
```

```
Node* constructTree(vector<int>& arr){
    if (idx == arr.size() || arr[idx] == -1){
    idx++;
    return nullptr;
  }
  Node* node = new Node(arr[idx++]);
  node->left = constructTree(arr);
  node->right = constructTree(arr);
  return node;
}

void display(Node* node)
{
  if (node == nullptr)
    return;
  string str = "";
  str += node->left != nullptr ? to_string(node->left->data) :
".";
  str += " <- " + to_string(node->data) + " -> ";
  str += node->right != nullptr ? to_string(node->right->data) :
".";
  cout << str << endl;

  display(node->left);
  display(node->right);
}

//recursion made easy
void kLevelsDown(Node* node, int k){
  // write your code here
  if(node == NULL || k < 0){
    return;
  }

  if(k == 0){
    cout<<node->data<<endl;
  }

  kLevelsDown(node->left, k-1);
  kLevelsDown(node->right, k-1);
  return ;
}



//without recursion long ans
// void kLevelsDown(Node* node, int k){
//   // write your code here
//   if(k == 0){
//     cout<<node->data<<endl;
//   }
//   queue <Node*> q;
//   queue <Node*> cq;
//   q.push(node);

//   int l{};

//   while(q.size() > 0){
//     Node * front = q.front();q.pop();
//     if(front->left != NULL){
//       cq.push(front->left);
//     }
//     if(front->right != NULL){
//       cq.push(front->right);
//     }
//     if(q.empty()){
//       l++;
//       if(l == k){
//         while(cq.size() > 0){
//           cout<< cq.front()->data<<endl;
//           cq.pop();
//         }
//         cout<<endl;
//         break;
//       }
//       q = cq;
//       queue <Node*> temp;
//       cq = temp;


//     }
//   }
// }

int main(){
  vector<int> arr;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    string inp;
    cin >> inp;
    if (inp != "n") {
      arr.push_back(stoi(inp));
    }
    else {
      arr.push_back(-1);
    }
  }
  Node* root = constructTree(arr);
  int k;
  cin >> k;
  kLevelsDown(root, k);
  return 0;
}
```

Print Nodes K Distance Away

Easy

1. You are given a partially written BinaryTree class.
2. You are given a value data and a value k.
3. You are required to complete the body of printKNodesFar function. The function is expected to print all nodes which are k distance away in any direction from node with value equal to data.
4. Input is managed for you.

Constraints

```
None
```

Format

Input

Input is managed for you

Output

All nodes which are k distance away in any direction from node with value equal to data, each in a separate line

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
37
2
```

Sample Output

```
12
50
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data,Node *left,Node *right)
    {
        this->data = data;
        this->left = left;
        this->right = right;
    }
};

int idx = 0;
Node *constructTree(vector<int> &arr)
{
    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++],nullptr,nullptr);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

vector<Node*> nodeToRootPath(Node *node, int data) {
    vector<Node*> temp;
    if (node == nullptr){
      return temp;
    }

    vector<Node*> ans;
    if(node->data == data) {
      ans.push_back(node);
      return ans;
    }
    vector<Node*> left = nodeToRootPath(node->left, data);
    if(left.size() > 0) {
      left.push_back(node);
      return left;
    }
    vector<Node*> right = nodeToRootPath(node->right, data);
    if(right.size() > 0) {
      right.push_back(node);
      return right;
    }
    return temp;
  }


void printKLevelsDown(Node *node, int k, Node *block)
{
    if (node == nullptr || node == block)
        return;

    if (k == 0)
    {
        cout << node->data <<endl;
        return;
    }

    printKLevelsDown(node->left, k - 1, block);
    printKLevelsDown(node->right, k - 1, block);
}
```

```
void printKNodesFar(Node *node, int data,int k)
{
    // write your code here
    //using nodeToRootPath done earlier
    //also printKLevelsDown with slight midification
    vector <Node *> vec= nodeToRootPath( node, data );
    for(int i{} ;i < vec.size();i++){
        printKLevelsDown(vec[i],k-i,i == 0 ? NULL :vec[i-1]);
    }
}

int main()
{
    vector<int> arr;
    int n;
    cin>>n;

    for(int i = 0; i<n; i++){
        string inp;
        cin>>inp;
        if(inp != "n"){
            arr.push_back(stoi(inp));
        }
        else{
            arr.push_back(-1);
        }
    }

    int data;
    cin>>data;
    int k;
    cin>>k;
    Node *root = constructTree(arr);
    printKNodesFar(root, data, k);

    return 0;
}
//my solution aaa... can see, but nto easy to understand
// #include <iostream>
// #include <vector>
// #include <algorithm>
// #include <queue>

// using namespace std;

// class Node
// {
// public:
//     int data;
//     Node *left = nullptr;
//     Node *right = nullptr;
//     Node(int data,Node *left,Node *right)
//     {
//         this->data = data;
//         this->left = left;
//         this->right = right;
//     }
// };

// int idx = 0;
// Node *constructTree(vector<int> &arr)
// {

//     if (idx == arr.size() || arr[idx] == -1)
//     {
//         idx++;
//         return nullptr;
//     }
//     Node *node = new Node(arr[idx++],nullptr,nullptr);
//     node->left = constructTree(arr);
//     node->right = constructTree(arr);
//     return node;
// }

// vector<Node*> nodeToRootPath(Node *node, int data) {
//     vector<Node*> temp;
//     if (node == nullptr){
//       return temp;
//     }

//     vector<Node*> ans;
//     if(node->data == data) {
//       ans.push_back(node);
//       return ans;
//     }
//     vector<Node*> left = nodeToRootPath(node->left, data);
//     if(left.size() > 0) {
//       left.push_back(node);
//       return left;
//     }
//     vector<Node*> right = nodeToRootPath(node->right, data);
//     if(right.size() > 0) {
//       right.push_back(node);
//       return right;
//     }
//     return temp;
//   }


// void printKLevelsDown(Node *node, int k, Node *block)
// {
//     if (node == nullptr || node == block)
//         return;
//     if (k == 0)
//     {
//         cout << node->data <<endl;
//         return;
//     }

//     printKLevelsDown(node->left, k - 1, block);
//     printKLevelsDown(node->right, k - 1, block);
// }

// int how_deep_is_data(Node * node , int data){
//   if (node == NULL ){
//     return -1;
//   }

//   if(node->data == data){
//     return 0;
//   }

//   int l = how_deep_is_data(node->left, data);
//   if(l >= 0){
//     return l+1;
//   }

//   int r = how_deep_is_data(node->right, data);
//   if(r >= 0){
//     return r+1;
//   }

//   return -1;
// }
// char which_side(Node * node , int data){
//   if (node == NULL ){
//     return 'n';// n = not found
//   }

//   if(node->data == data){
//     return 'a'; //a = at node
//   }

//   char l = which_side(node->left, data);
//   if(l != 'n'){
//     return 'l';
//   }

//   char r = which_side(node->right, data);
//   if(r != 'n'){
//     return 'r';
//   }

//   return 'n';
// }
// void print_required_nodes(Node* node ,int data, int d,char c){
//   if(node == NULL || d < 0 ){
//     return ;
//   }
//   if (d == 0){
//     cout<<node->data<<endl;
//   }
//   if(node->left != NULL && c == 'l'){
//       print_required_nodes(node->left, data,d-1,'a');
//   }else if(node->right != NULL && c == 'r'){
//       print_required_nodes(node->right, data,d-1,'a');
//   }else {
//       if(node->left != NULL ){
//           print_required_nodes(node->left, data,d-1,'a');
//       }
//       if(node->right != NULL ){
//           print_required_nodes(node->right, data,d-1,'a');
//       }
//   }

//   return ;
// }
// void printKNodesFar(Node *node, int data,int k)
// {
//     // write your code here
//     if(node == NULL ){
//       return ;
//     }

//     printKNodesFar(node->left,data,k);
//     printKNodesFar(node->right,data,k);
//     int h = how_deep_is_data(node, data);

//     if( h <= k && h >= 0){
//       char x = which_side(node,data);
//       // cout<<node->data<<" "<<h<<" ";
//       // cout<<x<<endl;
//       if(x == 'a'){
//         print_required_nodes(node,data,k-h,'a');
//       }else if(x == 'r'){
//         print_required_nodes(node,data,k-h, 'l');
//       }else if(x == 'l'){
//         print_required_nodes(node,data,k-h,'r');
//       }
//     }


// }

// int main()
// {
//     vector<int> arr;
//     int n;
//     cin>>n;

//     for(int i = 0; i<n; i++){
//         string inp;
//         cin>>inp;
//         if(inp != "n"){
//             arr.push_back(stoi(inp));
//         }
//         else{
//             arr.push_back(-1);
//         }
//     }

//     int data;
//     cin>>data;
//     int k;
//     cin>>k;
//     Node *root = constructTree(arr);
//     // cout<<"ans ============= b==================="<<endl;
//     printKNodesFar(root, data, k);

//     return 0;
// }
```

Path To Leaf From Root In Range

Easy

1. You are given a partially written BinaryTree class.
2. You are given a value lo and a value hi
3. You are required to complete the body of pathToLeafFromRoot function. The function is expected to print all paths from root to leaves which have sum of nodes in range from lo to hi (both inclusive). The elements in path should be separated by spaces. Each path should be in a separate line.
4. Input is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

The elements in path should be separated by spaces. Each path should be in a separate line.

Example

Sample Input

```
23
50 25 12 n n 37 30 n n 40 n n 75 62 60 n n 70 n n 87 n n
150
250
```

Sample Output

```
50 25 37 40
50 75 62 60
50 75 87
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data)
    {
        this->data = data;
    }
};
```

```
  int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++]);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}


void display(Node *node) {
    if (node == nullptr) {
      return;
    }

    string str = "";
    str += node->left == nullptr ? "." : to_string(node->left-
>data) + "";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right == nullptr ? "." : to_string(node->right-
>data) + "";
    cout<<str<<endl;

    display(node->left);
    display(node->right);
}

  void pathToLeafFromRoot(Node *node, string path,int sum, int lo,
int hi){
    // write your code here
    if(node == NULL ){
      return ;
    }
    if(node->left == NULL && node->right == NULL){
        path += to_string(node->data) ;
    }else{
        path += to_string(node->data) + " ";
    }
    sum += node->data;
    if(node->left == NULL && node->right == NULL && lo <= sum &&
hi >= sum){
      cout<<path<<endl;
      return ;
    }
    pathToLeafFromRoot(node->left,path,sum,lo,hi);
    pathToLeafFromRoot(node->right,path,sum ,lo,hi);
  }
  //my solution without using sum
//   void pathToLeafFromRoot(Node *node, string path,int sum, int
lo, int hi){
//     // write your code here
//     if(node == NULL ){
//       return ;
//     }
//     if(node->left == NULL && node->right == NULL){
//         path += to_string(node->data) ;
//     }else{
//         path += to_string(node->data) + " ";
//     }
//     lo -= node->data;
//     hi -= node->data;
//     if(node->left == NULL && node->right == NULL && lo <= 0 &&
hi >= 0){
//       cout<<path<<endl;
//       return ;
//     }
//     pathToLeafFromRoot(node->left,path,0,lo,hi);
//     pathToLeafFromRoot(node->right,path,0,lo,hi);
//   }

  int main()
    {
        vector<int> arr;
        int n;
        cin>>n;

        for(int i = 0; i<n; i++){
            string inp;
            cin>>inp;
            if(inp != "n"){
                arr.push_back(stoi(inp));
            }
            else{
                arr.push_back(-1);
            }
        }

        int lo;
        cin>>lo;
        int hi;
        cin>>hi;

        Node *root = constructTree(arr);
        pathToLeafFromRoot(root, "", 0, lo, hi);
        return 0;
    }
```

Transform To Left-cloned Tree

Easy

1. You are given a partially written BinaryTree class.
2. You are required to complete the body of createLeftCloneTree function. The function is expected to create a new node for every node equal in value to it and inserted between itself and it's left child. Check question video for clarity.
3. Input and Output is managed for you.

Constraints

None

Format

Input

Input is managed for you.

Output

Output is managed for you.

Example

Sample Input

```
15
50 25 12 n n 37 n n 75 62 n n 87 n n
```

Sample Output

```
50
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data,Node *left,Node *right)
    {
        this->data = data;
        this->left = left;
        this->right = right;
    }
};

  int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
```

```cpp
        return nullptr;
    }
    Node *node = new Node(arr[idx++],nullptr,nullptr);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

void display(Node *node) {
    if (node == nullptr) {
      return;
    }

    string str = "";
    str += node->left == nullptr ? "." : to_string(node->left-
>data) + "";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right == nullptr ? "." : to_string(node->right-
>data) + "";
    cout<<str<<endl;

    display(node->left);
    display(node->right);
}


  Node *createLeftCloneTree(Node *node){
    // write your code here
    if(node == NULL){
        return NULL;
    }
    Node* cn = new Node(node->data,node->left,NULL);
    node -> left = cn;
    cn->left = createLeftCloneTree(cn->left);
    node->right = createLeftCloneTree(node->right);

    return node;

  }

  int main()
    {
        vector<int> arr;
        int n;
        cin>>n;

        for(int i = 0; i<n; i++){
            string inp;
            cin>>inp;
            if(inp != "n"){
                arr.push_back(stoi(inp));
            }
            else{
                arr.push_back(-1);
            }
        }


        Node *root = constructTree(arr);
        root = createLeftCloneTree(root);
        display(root);
    }
```

Transform To Normal From Left-cloned Tree

Easy

1. You are given a partially written BinaryTree class.
2. You are required to complete the body of transBackFromLeftClonedTree function. The function is expected to convert a left-cloned tree back to it's original form. The left cloned tree is dicussed in previous question. In a left-clone tree a new node for every node equal in value to it is inserted between itself and it's left child. For clarity check out the question video.
3. Input and Output is managed for you.

Constraints

None

Format

Input

Input is managed for you.

Output

Output is managed for you.

Example

Sample Input

```
37
50 50 25 25 12 12 n n n n 37 37 30 30 n n n n n n 75 75 62 62 n n 70 70 n n
n n 87 87 n n n
```

Sample Output

```
25
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data,Node *left,Node *right)
    {
        this->data = data;
        this->left = left;
        this->right = right;
    }
};

  int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++],nullptr,nullptr);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

void display(Node *node) {
    if (node == nullptr) {
      return;
    }

    string str = "";
    str += node->left == nullptr ? "." : to_string(node->left-
>data) + "";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right == nullptr ? "." : to_string(node->right-
>data) + "";
    cout<<str<<endl;

    display(node->left);
    display(node->right);
}


  Node *transBackFromLeftClonedTree(Node *node){
    // write your code here
    if(node == NULL){
        return NULL;
    }
    node ->left ->left = transBackFromLeftClonedTree( node ->left
->left);
    node ->right = transBackFromLeftClonedTree(node ->right);

    Node *extra = node ->left;
    node ->left = node ->left -> left ;
    delete extra;
    return node;

  }

  int main()
    {
        vector<int> arr;
        int n;
        cin>>n;

        for(int i = 0; i<n; i++){
            string inp;
            cin>>inp;
            if(inp != "n"){
                arr.push_back(stoi(inp));
            }
            else{
                arr.push_back(-1);
            }
        }


        Node *root = constructTree(arr);
        root = transBackFromLeftClonedTree(root);
        display(root);
    }
```

Print Single Child Nodes

Easy

1. You are given a partially written BinaryTree class.
2. You are required to complete the body of printSingleChildNodes function. The function is expected to print in separate lines, all such nodes which are only child of their parent. Use preorder for traversal.
3. Input and Output is managed for you.

Constraints

None

Format

Input

Input is managed for you.

Output

Output is managed for you.

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
```

Sample Output

```
30
70
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data,Node *left,Node *right)
    {
        this->data = data;
        this->left = left;
        this->right = right;
    }
};

  int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++],nullptr,nullptr);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

void display(Node *node) {
    if (node == nullptr) {
      return;
    }

    string str = "";
    str += node->left == nullptr ? "." : to_string(node->left-
>data) + "";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right == nullptr ? "." : to_string(node->right-
>data) + "";
    cout<<str<<endl;

    display(node->left);
    display(node->right);
}
```

```
////my solution without using parent
// void printSingleChildNodes(Node *node, Node *parent){
//     // write your code here
//     if(node == NULL){
//         return ;
//     }
//     if (node->left != NULL && node ->right == NULL){
//         cout<<node->left->data<<endl;
//         return;
//     }
//     if (node->right != NULL && node ->left == NULL){
//         cout<<node->right->data<<endl;
//         return;
//     }
//     printSingleChildNodes(node ->left,NULL);
//     printSingleChildNodes(node ->right,NULL);

//   }

  void printSingleChildNodes(Node *node, Node *parent){
    // write your code here
    if(node == NULL){
        return ;
    }
    if (parent != NULL && (parent->left == NULL || parent ->right
== NULL)){
        cout<<node->data<<endl;
        return;
    }
    printSingleChildNodes(node ->left,node);
    printSingleChildNodes(node ->right,node);

  }

  int main()
    {
        vector<int> arr;
        int n;
        cin>>n;

        for(int i = 0; i<n; i++){
            string inp;
            cin>>inp;
            if(inp != "n"){
                arr.push_back(stoi(inp));
            }
            else{
                arr.push_back(-1);
            }
        }
        Node *root = constructTree(arr);
        printSingleChildNodes(root, nullptr);
    }
```

Remove Leaves In Binary Tree

Easy

1. You are given a partially written BinaryTree class.
2. You are required to complete the body of removeLeaves function. The function is expected to remove all leaf nodes from the tree.
3. Input and Output is managed for you.

Constraints

None

Format

Input

Input is managed for you.

Output

Output is managed for you.

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
```

Sample Output

```
25
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data)
    {
        this->data = data;
    }
};
```

```
  int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++]);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

void display(Node *node) {
    if (node == nullptr) {
      return;
    }

    string str = "";
    str += node->left == nullptr ? "." : to_string(node->left-
>data) + "";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right == nullptr ? "." : to_string(node->right-
>data) + "";
    cout<<str<<endl;

    display(node->left);
    display(node->right);
}

  Node *removeLeaves(Node *node){
    // write your code here
    if(node == NULL ){
        return NULL;
    }
    if (node -> left == NULL && node ->right == NULL){
        Node * leaf = node ;
        delete leaf;
        return NULL;
    }
    node -> left = removeLeaves(node ->left);
    node -> right = removeLeaves(node ->right);

    return node;

  }

   int main()
    {
        vector<int> arr;
        int n;
        cin>>n;

        for(int i = 0; i<n; i++){
            string inp;
            cin>>inp;
            if(inp != "n"){
                arr.push_back(stoi(inp));
            }
            else{
                arr.push_back(-1);
            }
        }


        Node *root = constructTree(arr);
        root = removeLeaves(root);
        display(root);
    }
```

Diameter Of A Binary Tree

Easy

1. You are given a partially written BinaryTree class.
2. You are required to complete the body of diameter1 function. The function is expected to return the number of edges between two nodes which are farthest from each other in terms of edges.
3. Input and Output is managed for you.

Constraints

```
None
```

Format

Input

Input is managed for you.

Output

Output is managed for you.

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
```

Sample Output

```
6
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class Node
{
public:
    int data=0;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data)
    {
        this->data = data;
    }
};

 class Pair {
    public:
    Node *node=nullptr;
    int state=0;

    Pair(Node *node, int state) {
      this->node = node;
      this->state = state;
    }
  };

int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++]);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

//Display function
void display(Node *node)
{
    if (node == nullptr)
        return;
    string str = "";
    str += node->left != nullptr ? to_string(node->left->data) :
".";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right != nullptr ? to_string(node->right->data) :
".";
    cout << str << endl;

    display(node->left);
    display(node->right);
}


//Height function
int height(Node *node)
{
    return node == nullptr ? -1 : max(height(node->left),
height(node->right)) + 1; // for no of edges: -1, and in terms of
no of nodes return 0;
}

/*
    for efficient solution making a class
    which has pair of diameter and height
    because earliear because of height complexity was n-square
    me -> when you want to return two things make class of them
*/
class diaPair{
  public:
  int dia;
  int height;
};

diaPair* diameter_eff(Node* node ){
  if(node == NULL){
    diaPair* base = new diaPair();
    base->dia = 0;//base case of diameter
    base->height = -1;//base case of height
    return base;
  }
  diaPair *lf = diameter_eff(node->left);
  diaPair *ri = diameter_eff(node->right);

  diaPair* here = new diaPair();
  here->height = max(lf->height ,ri->height) + 1;

  int dia_from_this_node = lf->height + ri->height + 2;
  here->dia = max(max(lf->dia,ri->dia),dia_from_this_node);

  return here;
}

//another solution -> using a global varible
```

```
// int dia{};
// int diameter(Node *root)
// {
//     // write your code here
//     if ( root == NULL){
//       return -1;
//     }
//     int lh = diameter(root->left);
//     int rh = diameter(root->right);

//     int child_height = lh > rh ? lh :rh;
//     int d{};
//     d += diameter(root->left);
//     d += diameter(root->right);
//     d += 2;
//     if(d > dia ){
//       dia = d;
//     }

//     return child_height +1;


// }
//not efficient n-square complexity
// int diameter(Node *root)
// {
//     // write your code here
//     if ( root == NULL){
//       return 0;
//     }
//     int ld = diameter(root->left);
//     int rd = diameter(root->right);

//     int child_dia = ld > rd ? ld :rd;
//     int dia{};
//     dia += height(root->left); //always make these two call
//     dia += height(root->right);//coz NULL hone pare -1 bhi
chahiye hume
//     dia += 2;

//     return dia > child_dia ? dia :child_dia;



// }
int diameter(Node *root)
{
   // write your code here
   diaPair*  d = diameter_eff(root);
   return d->dia;
}
```

```
int main(){
    int n;
    cin>>n;

    vector<int> arr(n,0);
    for(int i = 0; i < n; i++) {
        string tmp;
        cin>>tmp;
      if (tmp=="n") {
        arr[i] = -1;
      } else {
        arr[i] = stoi(tmp);
      }
    }


    Node * root = constructTree(arr);

    int dia = 0;
    dia = diameter(root);
    cout<<dia;
}
```

Tilt Of Binary Tree

Easy

1. You are given a partially written BinaryTree class. 2. You are required to complete the body of tilt function. The function is expected to set the value of data member "tilt". "tilt" of a node is the absolute value of difference between sum of nodes in it's left subtree and right subtree. "tilt" of the whole tree is represented as the sum of "tilt"s of all it's nodes. 3. Input and Output is managed for you. Note -> Please refer the video for clarity.

Constraints

None

Format

Input

Input is managed for you.

Output

Output is managed for you.

Example

Sample Input

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
```

Sample Output

```
390
```

```cpp
#include <iostream>
#include <vector>
#include <cstdlib>

using namespace std;

class Node
{
public:
    int data=0;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data)
    {
        this->data = data;
    }
};

 class Pair {
    public:
    Node *node=nullptr;
    int state=0;

    Pair(Node *node, int state) {
      this->node = node;
      this->state = state;
    }
  };

int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++]);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

//Display function
void display(Node *node)
{
    if (node == nullptr)
        return;
    string str = "";
    str += node->left != nullptr ? to_string(node->left->data) :
".";
```

```
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right != nullptr ? to_string(node->right->data) :
".";
    cout << str << endl;

    display(node->left);
    display(node->right);
}


//Height function
int height(Node *node)
{
    return node == nullptr ? -1 : max(height(node->left),
height(node->right)) + 1; // for no of edges: -1, and in terms of
no of nodes return 0;
}

static int til = 0;
int tilt(Node *node)
{
    // write your code here
    /*
    using travel and change
    here we are returing the sum and changing the tilt
    */
    if(node == NULL){
      return 0;
    }
    int a = tilt (node->left );
    int b = tilt (node->right);
    int t = abs(a-b);
    til += t;
    return a+b+node->data;

}

int main(){
    int n;
    cin>>n;

    vector<int> arr(n,0);
    for(int i = 0; i < n; i++) {
        string tmp;
        cin>>tmp;
      if (tmp=="n") {
        arr[i] = -1;
      } else {
        arr[i] = stoi(tmp);
      }
    }
    Node * root = constructTree(arr);
    int r = tilt(root);
    cout<<til;
}
```

Is A Binary Search Tree

Easy

1. You are given a partially written BinaryTree class.
2. You are required to check if the tree is a Binary Search Tree (BST) as well. In a BST every node has a value greater than all nodes on it's left side and smaller value than all node on it's right side.
3. Input is managed for you.

Note -> Please refer the question video for clarity.

Constraints

```
Time complexity must be O(n)
Space should not be more than required for recursion
(call-stack)
```

Format

Input

Input is managed for you.

Output

true if the tree is a BST, false otherwise

Example

Sample Input

```
15
50 25 12 n n 37 n n 75 62 n n 87 n n
```

Sample Output

```
true
```

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

using namespace std;

class Node
{
public:
    int data=0;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data)
    {
        this->data = data;
    }
};

 class Pair {
    public:
    Node *node=nullptr;
    int state=0;

    Pair(Node *node, int state) {
      this->node = node;
      this->state = state;
    }
  };

int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++]);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

//Display function
void display(Node *node)
{
    if (node == nullptr)
        return;
    string str = "";
    str += node->left != nullptr ? to_string(node->left->data) :
".";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right != nullptr ? to_string(node->right->data) :
".";
    cout << str << endl;

    display(node->left);
    display(node->right);
}


//Height function
int height(Node *node)
{
    return node == nullptr ? -1 : max(height(node->left),
height(node->right)) + 1;
```

```
}

  class bst{
    public:
      int isbst;
      int max;
      int min;
  };
  bst Bst(Node *node){
      // write your code here
      if(node == NULL ){
        bst base ;
        base.isbst = 1 ;
        base.max = INT_MIN;
        base.min = INT_MAX;
        return base;
      }
        bst l = Bst(node->left);
        bst r = Bst(node->right);
        bst now ;
        now.isbst = l.isbst && r.isbst && node->data > l.max &&
node->data < r.min;
        // int a = l.min<r.min? l.min:r.min;
        now.min = l.min < node ->data ? l.min:node->data;
        // int b = r.max>l.max?r.max:l.max;
        now.max = r.max > node ->data ? r.max :node->data;
        return now;

    }
  // bst Bst(Node *node){
  //     // write your code here
  //     if(node == NULL ){
  //       bst base ;
  //       base.isbst = 1 ;
  //       base.max = INT_MIN;
  //       base.min = INT_MAX;
  //       return base;
  //     }
  //     if(node->left == NULL && node ->right == NULL){
  //       bst base ;
  //       base.isbst = 1 ;
  //       base.max = node->data;
  //       base.min = node->data;
  //       return base;
  //     }
  //     bst l = Bst(node->left);
  //     bst r = Bst(node->right);

  //     bst now ;
  //     if(l.isbst && r.isbst && node->data > l.max && node->data
< r.min) {
  //       now.isbst = 1;
  //       now.min = l.min < node ->data ? l.min:node->data; //why
this comparison ? ->this is because NULL return min as INT_MAX and
if we assgin this to now min that may create problem on the upper
levels(this problem is for one child )
  //       now.max = r.max > node ->data ? r.max :node->data;
  //     }else{
  //       now.isbst = 0;
  //     }
  //     // cout<<node ->data<<" "<<now.isbst<<endl;
  //     return now ;


  // }

//another solution one child case handled separatly
//thodha adjust kr leve  leet code par submit kiya tha
// /**
//  * Definition for a binary tree node.
//  * struct TreeNode {
//  *     int val;
//  *     TreeNode *left;
//  *     TreeNode *right;
//  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
//  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr)
{}
//  *     TreeNode(int x, TreeNode *left, TreeNode *right) :
val(x), left(left), right(right) {}
//  * };
//  */
// class Solution {
// public:

//     class bst{
//         public:
//         int isbst;
//         long int max;
//         long int min;
//     };
//   bst Bst(TreeNode *node){
//       // write your code here
//       if(node == NULL ){
//         bst base ;
//         base.isbst = 1 ;
//         return base;
//       }
//       if(node->left == NULL && node ->right == NULL){
//         bst base ;
//         base.isbst = 1 ;
//         base.max = node->val;
//         base.min = node->val;
//         return base;
//       }
//       if(node->left != NULL && node ->right == NULL){
//         cout<<"l "<<node->val<<endl;
//         bst now ;
//         now.isbst = 1 ;
//         now.max = node->val;
//         bst l = Bst(node->left);
//         if(l.isbst && node->val > l.max){
//           now.isbst = 1;
//           now.min = l.min;
//         }else{
//           now.isbst = 0;
//         }
//         return now;
//       }

//       if(node->left == NULL && node ->right != NULL){
//        cout<<"r "<<node->val<<endl;
//         bst now ;
//         now.isbst = 1 ;
//         now.min = node->val;
//         bst r = Bst(node->right);
//         if(r.isbst && node->val < r.min){
//           now.isbst = 1;
//           now.max = r.max;
//         }else{
//           now.isbst = 0;
//         }
//         return now;
//       }

//     //   cout<<node ->val<<" ";
//       bst l = Bst(node->left);
//       bst r = Bst(node->right);

//       bst now ;
//       if(l.isbst && r.isbst && node->val > l.max && node->val <
r.min) {
//         now.isbst = 1;
//         now.min = l.min;
//         now.max = r.max;
//       }else{
//         now.isbst = 0;
//       }
//       cout<<"b "<<node ->val<<" "<<now.isbst<<now.max<<"
"<<now.min<<endl;
//       return now ;

//   }
//     bool isValidBST(TreeNode* root) {
//         bst ans = Bst(root);
//         if(ans.isbst){
```

```
//              return true;
//          }
//          return false;
//      }
// };


int main(){
    int n;
    cin>>n;

    vector<int> arr(n,0);
    for(int i = 0; i < n; i++) {
        string tmp;
        cin>>tmp;
      if (tmp=="n") {
        arr[i] = -1;
      } else {
        arr[i] = stoi(tmp);
      }
    }


    Node *root = constructTree(arr);

    bst r = Bst(root);
    if(r.isbst == 1)
    cout << "true";
    else
    cout<<"false";
}
```

Largest Bst Subtree

Medium

1. You are given a partially written BinaryTree class.
2. You are required to find the root of largest sub-tree which is a BST. Also, find the number of nodes in that sub-tree.
3. Input is managed for you.

Note -> Please refer the question video for clarity.

Constraints

```
Time complexity must be O(n)
Space should not be more than required for recursion
(call-stack)
```

Format

Input

Input is managed for you.

Output

@

Example

Sample Input

```
15
50 25 12 n n 37 n n 75 62 n n 87 n n
```

Sample Output

```
50@7
```

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

using namespace std;

class Node
{
public:
    int data=0;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data)
    {
        this->data = data;
    }
};

 class Pair {
    public:
    Node *node=nullptr;
    int state=0;

    Pair(Node *node, int state) {
      this->node = node;
      this->state = state;
    }
  };

int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++]);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}


void display(Node *node)
{
    if (node == nullptr)
        return;
    string str = "";
    str += node->left != nullptr ? to_string(node->left->data) :
".";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right != nullptr ? to_string(node->right->data) :
".";
    cout << str << endl;

    display(node->left);
    display(node->right);
}



int height(Node *node)
{
    return node == nullptr ? -1 : max(height(node->left),
height(node->right)) + 1;
}

 class bst{
     public:
    bool isbst = false;
    int max =0;
    int min=0;
    Node *root=nullptr;
    int size=0;
};

 bst Bst(Node *node){

     if(node == nullptr)
     {
        bst bres;
        bres.isbst = true;
        bres.max = INT_MIN;
        bres.min = INT_MAX;
        return bres;
     }
     bst l = Bst(node->left);
     bst r = Bst(node->right);

     bst ans;
     ans.max = max(node->data, max(l.max,r.max));
     ans.min = min(node->data,min(l.min,r.min));

     if(l.isbst==true && r.isbst==true && (l.max < node->data &&
r.min > node->data)){
         ans.isbst=true;
     }

     // write your code here
     if(ans.isbst){
         ans.size = l.size + r.size + 1;
         ans.root = node;
     }else{
         if(l.size>r.size){
             ans.size = l.size;
             ans.root = l.root;
         }else{
             ans.size = r.size;
             ans.root = r.root;
         }
     }

    return ans;
 }

int main(){
    int n;
    cin>>n;

    vector<int> arr(n,0);
    for(int i = 0; i < n; i++) {
        string tmp;
        cin>>tmp;
      if (tmp=="n") {
        arr[i] = -1;
      } else {
        arr[i] = stoi(tmp);
      }
    }

    Node *root = constructTree(arr);

    bst r = Bst(root);
    cout<<r.root->data<<"@"<<r.size;



}
```

Is Balanced Tree

Easy

1. You are given a partially written BinaryTree class.
2. You are required to check if the tree is balanced. A binary tree is balanced if for every node the gap between height's of it's left and right subtree is not more than 1.
3. Input is managed for you.

Note -> Please refer the question video for clarity.

Constraints

```
Time complexity must be O(n)
Space should not be more than required for recursion
(call-stack)
```

Format

Input

Input is managed for you.

Output

true if the tree is balanced, false otherwise

Example

Sample Input

```
21
50 25 12 n n 37 30 n n 51 n n 75 62 60 n n 70 n n n
```

Sample Output

```
false
```

```cpp
#include <iostream>
#include <vector>
#include <bits/stdc++.h>

using namespace std;

class Node
{
public:
    int data=0;
    Node *left = nullptr;
    Node *right = nullptr;
    Node(int data)
    {
        this->data = data;
    }
};

 class Pair {
    public:
    Node *node=nullptr;
    int state=0;

    Pair(Node *node, int state) {
        this->node = node;
        this->state = state;
      }
    };

int idx = 0;
Node *constructTree(vector<int> &arr)
{

    if (idx == arr.size() || arr[idx] == -1)
    {
        idx++;
        return nullptr;
    }
    Node *node = new Node(arr[idx++]);
    node->left = constructTree(arr);
    node->right = constructTree(arr);
    return node;
}

//Display function
void display(Node *node)
{
    if (node == nullptr)
        return;
    string str = "";
    str += node->left != nullptr ? to_string(node->left->data) :
".";
    str += " <- " + to_string(node->data) + " -> ";
    str += node->right != nullptr ? to_string(node->right->data) :
".";
    cout << str << endl;

    display(node->left);
    display(node->right);
}


//Height function
int height(Node *node)
{
    return node == nullptr ? -1 : max(height(node->left),
height(node->right)) + 1;
}
// class bpair{
//   public:
//   isb= false;
//   height ;
// }
// bpair isbal(Node * node){
//   if(node == NULL){
```

```
//    }
// }
int isbal {};
int isbalance(Node *node)
{
  // write your code here
  /*
  a (binary) tree is balanced tree
  if difference of the height of left and right
  sub is less than or equal to 1
  (in terms of node)
  */

  if(node == NULL){
    isbal= 1;
    return 0;
  }
  int l = isbalance(node ->left);
  int r = isbalance(node->right);
  if(isbal && abs(l-r) <= 1){
    return 1 + (l > r ? l : r) ;
  }else{
    isbal = 0;
    return 1 + (l > r ? l : r) ;
  }

}

int main(){
    int n;
    cin>>n;

    vector<int> arr(n,0);
    for(int i = 0; i < n; i++) {
        string tmp;
        cin>>tmp;
      if (tmp=="n") {
        arr[i] = -1;
      } else {
        arr[i] = stoi(tmp);
      }
    }


    Node *root = constructTree(arr);

    int r = isbalance(root);
    if(isbal == 1)
    cout << "true";
    else
    cout<<"false";
}
```