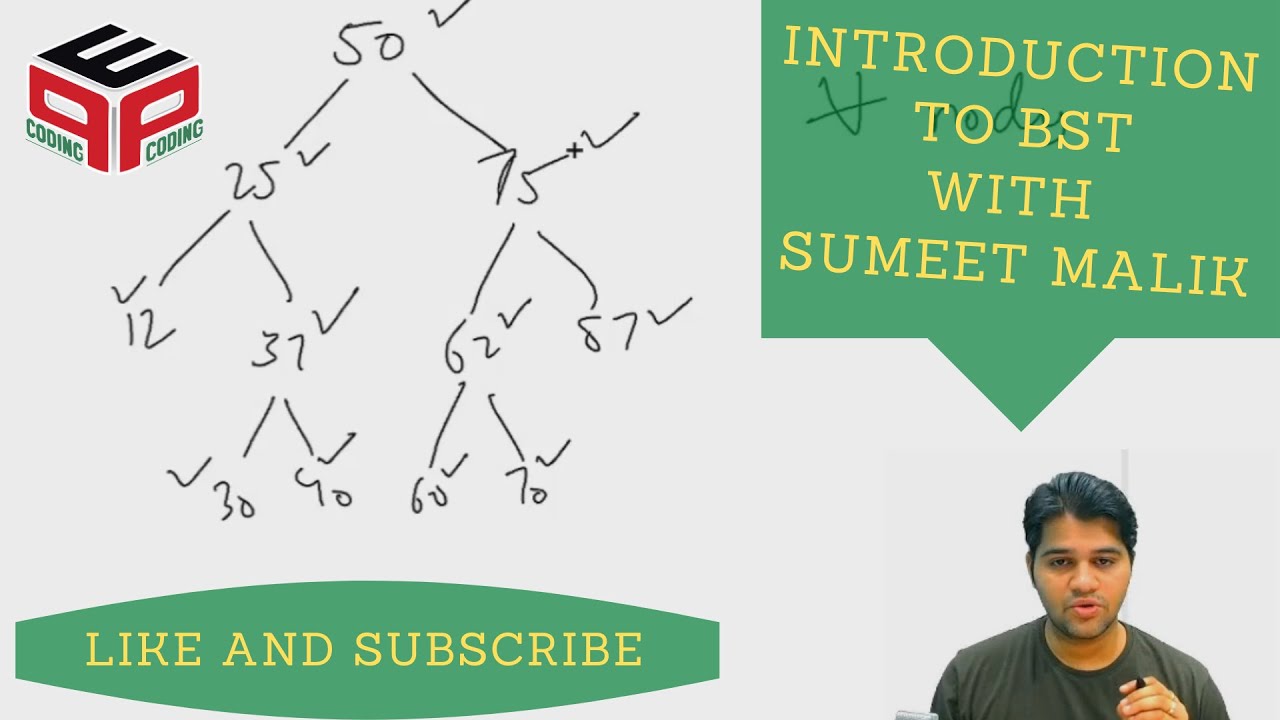
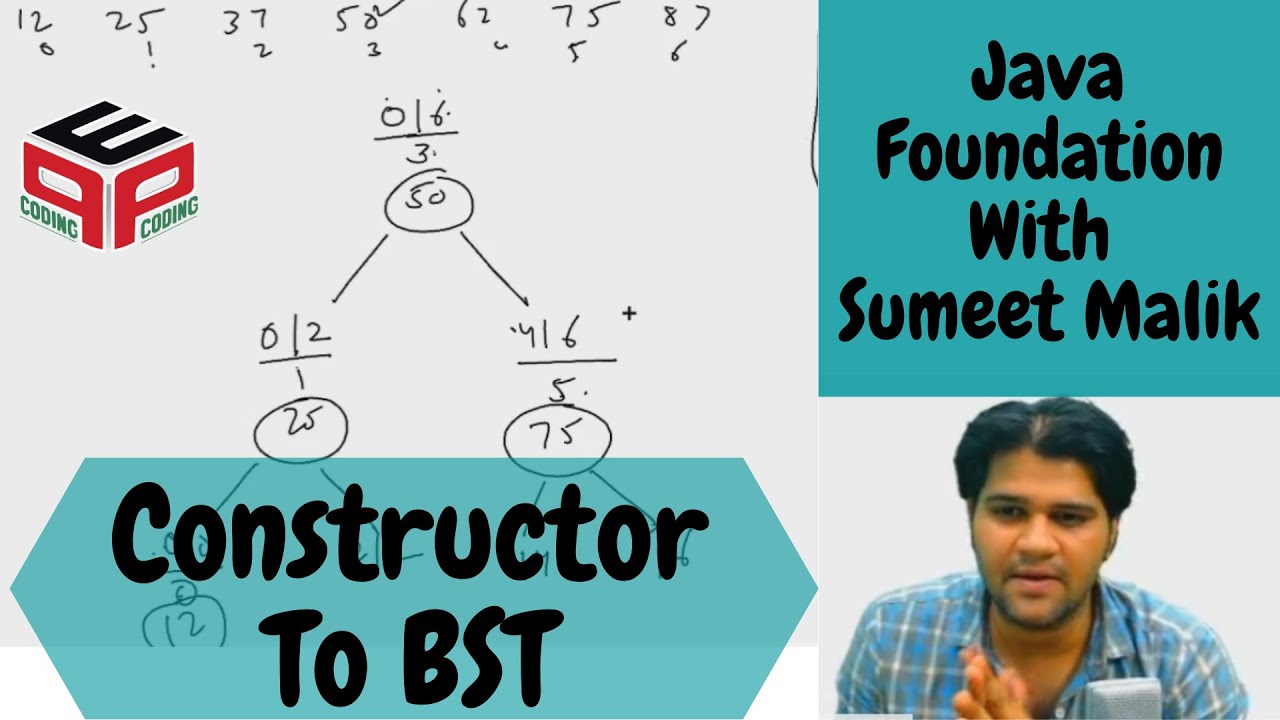
****

**Introduction To Binary Search Tree**

[](https://www.youtube.com/watch?v=wE5nFfdr7Ls)[](https://www.youtube.com/watch?v=UgOOA6azCVw)

**Constructor Of Binary Search Tree[](https://www.youtube.com/watch?v=yGrs-VuPCYg)**

Size, Sum, Max, Min, Find In Bst

Easy

1. You are given a partially written BST class.

2. You are required to complete the body of size, sum, max, min and find function. The functions are expected to:

2.1. size - return the number of nodes in BST

2.2. sum - return the sum of nodes in BST

2.3. max - return the value of node with greatest value in BST

2.4. min - return the value of node with smallest value in BST

2.5. find - return true if there is node in the tree equal to "data"

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

70

**Sample Output**

9

448

87

12

true

#include<bits/stdc++.h>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = nullptr;

right = nullptr;

}

};

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;

}

int min(Node \*node){

// Write your code here

if(node == NULL) {

return INT\_MAX;

}

if(node->left != NULL){

return min(node->left);

}else{

return node->data;

}

}

int max(Node \*node){

// Write your code here

if(node == NULL) {

return INT\_MIN;

}

if(node->right != NULL){

return max(node->right);

}else{

return node->data;

}

}

int sum(Node \* node){

// Write your code here

if(node == NULL ){

return 0;

}

int a = sum(node->left);

int b = sum(node->right);

return a+b+node->data;

}

int size(Node \* node){

// Write your code here

if(node == NULL ){

return 0;

}

int a = size(node->left);

int b = size(node->right);

return a+b+1;

}

bool find(Node \* node, int data){

// Write your code here

if(node == NULL){

return false;

}

if(data == node ->data){

return true;

}

if(data < node->data){

bool l = find (node->left , data);

return l;

}else{

bool r = find (node->right , data);

return r;

}

// if(l || r){

// return true;

// }

// return false;

}

int main() {

int n;

cin >> n;

vector<int> arr(n, 0);

for (int i = 0; i < n; i++) {

string x;

cin >> x;

if (x == "n") {

arr[i] = -1;

}

else {

arr[i] = stoi(x);

}

}

int data;

cin >> data;

Node\* root = construct(arr);

cout << size(root) << "\n" << sum (root) << "\n" << max(root) << "\n" << min(root) << "\n";

if (find(root, data)) {

cout << "true" << endl;

}

else {

cout << "false" << endl;

}

}

Add Node To Bst

Easy

1. You are given a partially written BST class.

2. You are required to complete the body of add function. "add" function is expected to add a new node with given data to the tree and return the new root.

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**

23

50 25 12 n n 37 30 n n 40 n n 75 62 60 n n 70 n n 87 n n

61

**Sample Output**

25

#include<bits/stdc++.h>

using namespace std;

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = nullptr;

right = nullptr;

}

};

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 display(Node\* node) {

if (node == nullptr) {

return;

}

string str = " <- " + to\_string(node->data) + " -> ";

string left = (node->left == nullptr) ? "." : "" + to\_string(node->left->data);

string right = (node->right == nullptr) ? "." : "" + to\_string(node->right->data);

str = left + str + right;

cout << str << endl;

display(node->left);

display(node->right);

}

Node\* add(Node \* node, int val){

// Write your code here

if(node == NULL){

Node\* newnode = new Node(val);

return newnode;

}

if(node->data == val){

// cout<<"Can't add dupicates"<<endl;

//here in the case of duplicates

//we are not any node and returning the node

return node;

}

if(val < node ->data){

node ->left = add(node->left,val);

}else{

node->right = add(node->right,val);

}

return node ;

}

int main() {

int n;

cin >> n;

vector<int> a(n,0);

for (int i = 0; i < n; i++) {

string x;

cin >> x;

if (x == "n") {

a[i] = -1;

}

else {

a[i] = stoi(x);

}

}

int data;

cin >> data;

Node\* root = construct(a);

// display(root);

root = add(root, data);

display(root);

}

Remove Node From Bst

Medium

1. You are given a partially written BST class.

2. You are required to complete the body of remove function. "remove" function is expected to remove a new node with given data to the tree and return the new root.

3. Input and Output is managed for you.

Note -> Please watch the question video to figure out the algorithm required for deletion. It specifies some requirements of the question as well.

**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

62

**Sample Output**

25

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node\* left;

Node\* right;

Node(int val){

data=val;

left=nullptr;

right=nullptr;

}

};

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 display(Node\* node) {

if (node == nullptr) {

return;

}

string str = " <- " + to\_string(node->data) + " -> ";

string left = (node->left == nullptr) ? "." : "" + to\_string(node->left->data);

string right = (node->right == nullptr) ? "." : "" + to\_string(node->right->data);

str = left + str + right;

cout << str << endl;

display(node->left);

display(node->right);

}

int max(Node\* root){

if(root->right== nullptr){

return root->data;

}

return max(root->right);

}

Node\* remove(Node\* root,int data){

// Write your code here

if(root == NULL){

cout<<"couldn't found that element"<<endl;

return NULL;

}

if(root->data == data){

if(root -> left == NULL && root ->right == NULL ){

//no child

Node \*to\_delete = root ;

delete to\_delete;

return NULL;

}else if(root -> left != NULL && root ->right == NULL ){

//only left child

Node \* left\_of\_root = root ->left;

delete root;

return left\_of\_root;

}else if(root -> left == NULL && root ->right != NULL ){

//only right child

Node \* right\_of\_root = root ->right;

delete root;

return right\_of\_root;

}else{

//two children

int max\_from\_left = max(root->left);

root->data = max\_from\_left;

root ->left = remove(root->left, max\_from\_left);

return root;

}

}else if (data < root ->data){

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

}else{

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

}

return root;

}

int main(){

int n;

cin >> n;

vector<int> arr(n,0);

for(int i=0;i<n;i++){

string x;

cin >> x;

if(x=="n"){

arr[i]=-1;

}

else{

arr[i]=stoi(x);

}

}

int data;

cin >> data;

Node\* root= construct(arr);

root=remove(root,data);

display(root);

}

Replace With Sum Of Larger

Easy

1. You are given a partially written BST class.

2. You are required to complete the body of rwsol function. "rwsol" function is expected to replace a node's value with sum of all nodes greater than it.

3. Input and Output is managed for you.

Note -> Please watch the question video for clarity. Use the statis sum data member to complete your code.

**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**

311

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node\* left;

Node\* right;

Node(int val){

data=val;

left=nullptr;

right=nullptr;

}

};

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 display(Node\* node) {

if (node == nullptr) {

return;

}

string str = " <- " + to\_string(node->data) + " -> ";

string left = (node->left == nullptr) ? "." : "" + to\_string(node->left->data);

string right = (node->right == nullptr) ? "." : "" + to\_string(node->right->data);

str = left + str + right;

cout << str << endl;

display(node->left);

display(node->right);

}

//my ans -----------------------------------------------

int curr\_sum = 0;

void replacewithsum\_worker(Node\* &root){

if (root == NULL){

// curr\_sum = 0;

return ;

}

replacewithsum\_worker(root->right);

int temp = curr\_sum;

curr\_sum += root->data;

root->data = temp;

replacewithsum\_worker(root->left);

}

void replacewithsum(Node\* &root){

// Write your code here

curr\_sum = 0;

replacewithsum\_worker(root);

}

int main(){

int n;

cin >> n;

vector<int> arr(n,0);

for(int i=0;i<n;i++){

string x;

cin >> x;

if(x=="n"){

arr[i]=-1;

}

else{

arr[i]=stoi(x);

}

}

Node\* root= construct(arr);

replacewithsum(root);

display(root);

}

Lca Of Bst

Easy

1. You are given a partially written BST class.

2. You are required to complete the body of lca function. "lca" function is expected to find the lowest commong ancestor of d1 and d2.

3. Input and Output is managed for you.

Note -> Please watch the question video for clarity.

**Constraints**

None

**Format**

**Input**

Input is managed for you

**Output**

Output is managed for you

**Example**

**Sample Input**

21

50 25 12 n n 37 30 n n n 75 62 60 n n 70 n n 87 n n

12

30

**Sample Output**

25

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node\* left;

Node\* right;

Node(int val){

data=val;

left=nullptr;

right=nullptr;

}

};

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;

}

// int ans;

int lca(Node\* root,int a,int b){

// Write your code here

// if(root == NULL){

// return 0;

// }

int ans;

if(a < root->data && b < root->data){

ans = lca(root->left ,a,b);

}else if(a > root->data && b > root->data){

ans = lca(root->right ,a,b);

}else {

ans = root->data;

}

return ans;

}

/\*

int lca(Node\* root,int a,int b){

// Write your code here

if(root == nullptr){

return 0;

}

int x{};

int y{};

if(root ->data == a){

x = 1;

}else if(root ->data == b){

y = 1;

}

if(a < root->data && b < root->data){

x += lca(root->left,a,b);

}else if(a > root->data && b > root->data){

y += lca(root->right,a,b);

}else{

x += lca(root->left,a,b);

y += lca(root->right,a,b);

}

if(x != 0 && y != 0){

ans = root->data;

return 0;

}else if (x != 0){

return x;

}else if(y != 0){

return y;

}else{

return 0;

}

}

\*/

int main(){

int n;

cin >> n;

vector<int> arr(n,0);

for(int i=0;i<n;i++){

string x;

cin >> x;

if(x=="n"){

arr[i]=-1;

}

else{

arr[i]=stoi(x);

}

}

int a,b;

cin >> a >> b;

Node\* root= construct(arr);

// ans=root->data;

// lca(root,a,b);

cout << lca(root,a,b) << endl;

}

Print In Range

Easy

1. You are given a partially written BST class.

2. You are required to complete the body of pir function. "pir" function is expected to print all nodes between d1 and d2 (inclusive and in increasing order).

3. Input and Output is managed for you.

Note -> Please watch the question video for clarity.

**Constraints**

None

**Format**

**Input**

Input is managed for you

**Output**

Output is managed for you

**Example**

**Sample Input**

21

50 25 12 n n 37 30 n n n 75 62 60 n n 70 n n 87 n n

12

65

**Sample Output**

12

25

30

37

50

60

62

#include <iostream>

#include <vector>

#include <stack>

#include <string>

using namespace std;

class Node{

public:

int data;

Node\* left = nullptr;

Node\* right = nullptr;

Node (int data)

{

this->data=data;

}

};

class Pair{

Node\* node = nullptr;

int state=0;

Pair(Node\* node, int state) {

this->node = node;

this->state = state;

}

};

void pir(Node\* node, int d1, int d2)

{

// write your code here

if(node == NULL){

return;

}

if(d1 < node->data && d2 < node->data){

pir(node->left ,d1,d2);

}else if(d1 > node->data && d2 > node->data){

pir(node-> right,d1,d2);

}else{

pir(node->left ,d1,d2);

cout<<node ->data<<endl;

pir(node-> right,d1,d2);

}

}

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;

}

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);

}

}

int k1;

cin>>k1;

int k2;

cin>>k2;

Node\* root = construct(arr);

pir(root,k1,k2);

}

Target Sum Pair In Bst

Easy

1. You are given a partially written BST class. 2. You are given a value. You are required to print all pair of nodes which add up to the given value. Make sure all pairs print the smaller value first and avoid duplicacies. Make sure to print the pairs in increasing order. Use the question video to gain clarity. 3. Input and Output is managed for you.

**Constraints**

None

**Format**

**Input**

Input is managed for you

**Output**

"smaller node" "larger node"

.. all pairs that add to target on separate lines

**Example**

**Sample Input**

21

50 25 12 n n 37 30 n n n 75 62 60 n n 70 n n 87 n n

100

**Sample Output**

25 75

30 70

#include <iostream>

#include <vector>

#include <stack>

#include <string>

using namespace std;

class Node{

public:

int data;

Node \* left = nullptr;

Node \* right = nullptr;

Node (int data)

{

this->data=data;

}

};

class Pair{

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;

}

// //SOLUTION 1

//time complexity = n\*height

//space complexity = height

// bool find(Node \* node, int data){

// // Write your code here

// if(node == NULL){

// return false;

// }

// if(data == node ->data){

// return true;

// }

// if(data < node->data){

// bool l = find (node->left , data);

// return l;

// }else{

// bool r = find (node->right , data);

// return r;

// }

// }

// void travelAndPrint(Node \* root , Node \* node , int tar){

// //write your code here

//

// if(root == NULL || node == NULL){

// return ;

// }

// travelAndPrint(root, node->left, tar);

// int compliment = tar - node ->data;

// if(compliment > node->data && find(root,compliment)){

// cout<<node-> data<<" "<<compliment<<endl;

// }

// travelAndPrint(root, node->right, tar);

// }

//SOLUTION 2

//time complexity = n

//space complexity = n

// void inorderfiller(vector<int> &vec , Node \* node ) {

// if(node == NULL){

// return ;

// }

// inorderfiller(vec ,node->left);

// vec.push\_back(node->data);

// inorderfiller(vec ,node->right);

// }

// void travelAndPrint(Node \* node , int tar){

// //write your code here

// vector<int> vec;

// inorderfiller(vec,node);

// // for(auto a :vec){

// // cout<<a;

// // }

// int s = 0;

// int b = vec.size() -1;

// while(s < b){

// if(vec[s] + vec[b] == tar){

// cout<<vec[s]<<" "<<vec[b]<<endl;

// s++;

// b--;

// }else if(vec[s] + vec[b] < tar) {

// s++;

// }else {

// b--;

// }

// }

// }

//SOLUTION 3

//TIME COMPLEXITY =

//SPACE COMPLEXITY =

class ItPair{

public:

Node \* node;

int state;

ItPair(Node\* node ,int state){

this->node = node ;

this->state = state;

}

};

Node \*nextNodeFromInorder (stack<ItPair> & ls){

while(ls.empty() == false){

// ItPair top = ls.top();ls.pop();

if(ls.top().state == 0){

// cout<<ls.top().node->data<<" 1----------"<<endl;

ls.top().state++;

if(ls.top().node->left != NULL){

// ls.push(top);

ls.push(ItPair(ls.top().node->left,0));

}

}else if(ls.top().state == 1){

ls.top().state++;

Node \* a = ls.top().node;

if(ls.top().node->right != NULL){

// ls.push(top);

ls.push(ItPair(ls.top().node->right,0));

}

return a;

}else{

ls.pop();

}

}

return NULL;

}

Node \*nextNodeFromReverseInorder (stack<ItPair> & rs){

while(rs.empty() == false){

if(rs.top().state == 0){

rs.top().state++;

if(rs.top().node->right != NULL){

rs.push(ItPair(rs.top().node->right,0));

}

}else if(rs.top().state == 1){

rs.top().state++;

Node \*a = rs.top().node;

if(rs.top().node->left != NULL){

rs.push(ItPair(rs.top().node->left,0));

}

return a;

}else{

rs.pop();

}

}

return NULL;

}

void bestApproach(Node \* node , int tar){

//write your code here

stack<ItPair> ls;//for inorder traversal (iterative)

stack<ItPair> rs;//for reverse inorder traversal (iterative)

ls.push(ItPair(node,0));

rs.push(ItPair(node,0));

Node \* left = nextNodeFromInorder(ls);

Node \* right = nextNodeFromReverseInorder(rs);

while(left->data < right ->data) {

if(left->data + right ->data < tar){

left = nextNodeFromInorder(ls);

}else if(left->data + right ->data > tar){

right = nextNodeFromReverseInorder(rs);

}else{

cout<<left->data <<" "<< right ->data << endl;

left = nextNodeFromInorder(ls);

right = nextNodeFromReverseInorder(rs);

}

}

}

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);

}

}

int k1;

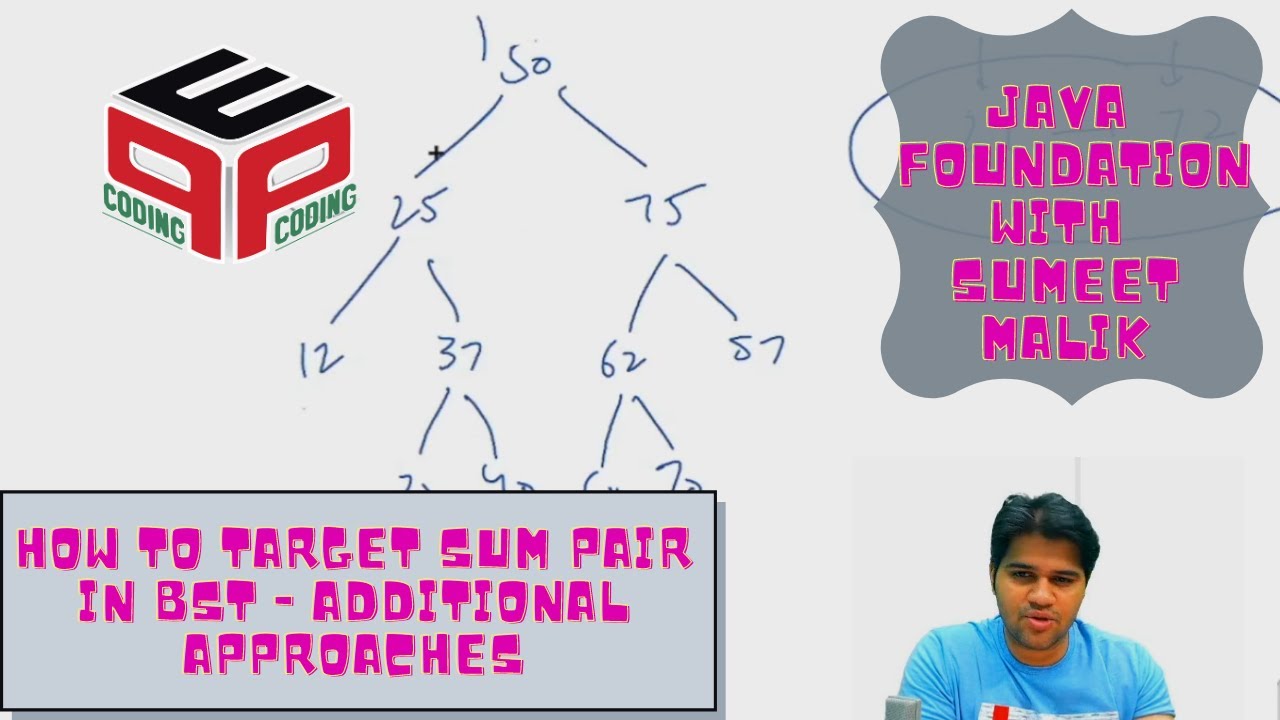
cin>>k1;

Node\* root = construct(arr);

bestApproach(root,k1);

}

**Target Sum Pair - BST - Alternate Approaches**

[](https://www.youtube.com/watch?v=iK2VFYxFC4o)[](https://www.youtube.com/watch?v=WE5ahU5YUWY)