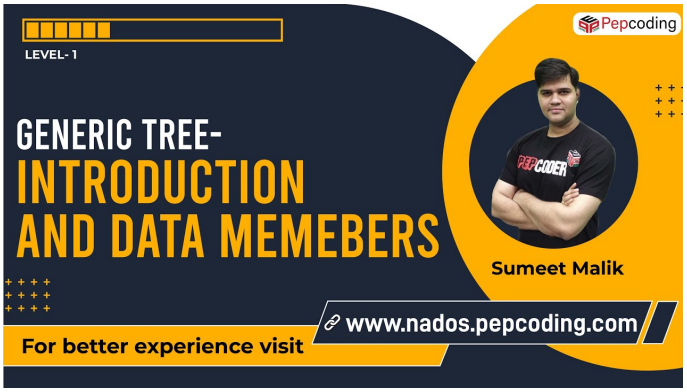
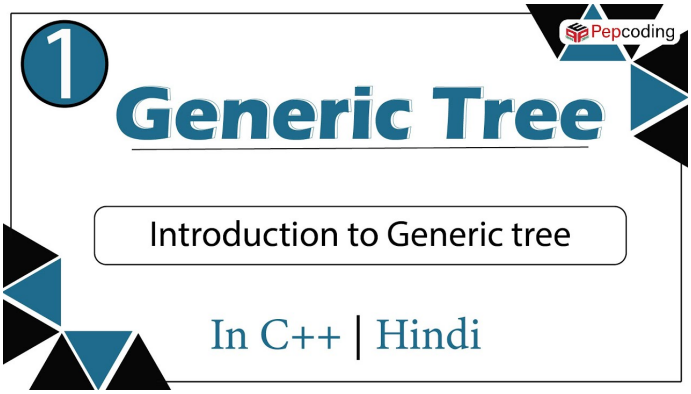
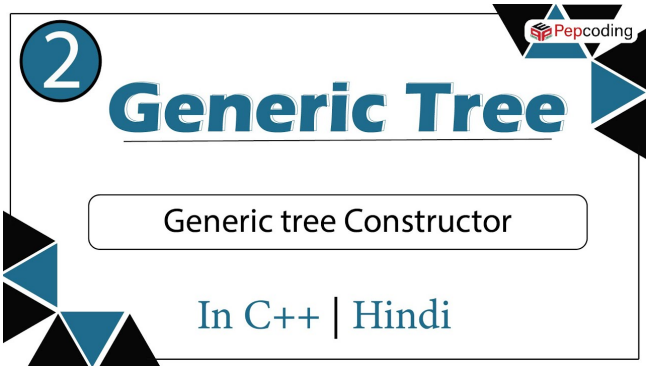
Generic Tree - Introduction And Data Members

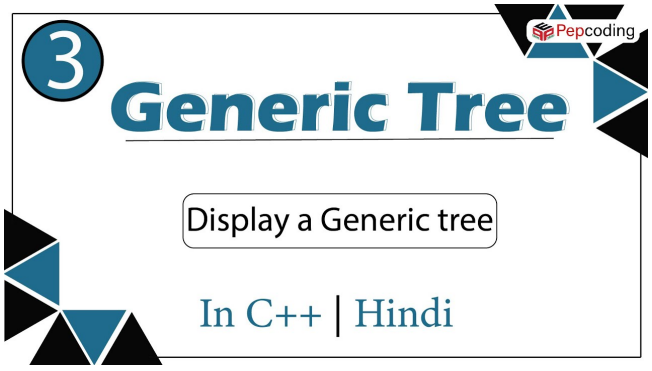




Generic Tree - Constructor



Generic Tree - Display



Size Of Generic Tree

Medium

1. You are given a partially written GenericTree class.
2. You are required to complete the body of size function. The function is expected to count the number of nodes in the tree and return it.
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

```
12
10 20 -1 30 50 -1 60 -1 -1 40 -1 -1
```

Sample Output

```
6
```

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
    int data;
    vector<Node*>children;
};

Node *newNode(int key){
    Node *temp=new Node;
    temp->data=key;
    return temp;

}
```

```
Node *construct(int arr[],int n ){
    Node *root=NULL;
    stack<Node*>st;
    for(int i=0;i<n;i++){
        if(arr[i]==-1){
            st.pop();
        }else{
            Node *t=newNode(arr[i]);
            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}
void display(Node *node)
{
    string str= to_string(node->data) + "-> ";
    for(Node *child: node->children)
    {
        str+=to_string(child->data)+", ";
    }
    str= str+".";
    cout << str <<endl;
    for(Node *child : node->children)
    {
        display(child);
    }
}

int size(Node *node){
  //write your code here
  if(node == NULL){
      return 0;
  }
    int s{};
    s++;
    for(Node* child: node->children){
        s += size(child);
    }
    return s;
}
```

```
int main(){

    int n;
    cin>>n;

    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    Node *root=construct(arr,n);
    int sz=size(root);
    cout<<sz<<endl;
   //display(root);
}
```

Maximum In A Generic Tree

Easy

1. You are given a partially written GenericTree class. 2. You are required to complete the body of max function. The function is expected to find the node with maximum value and return it. 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

```
12
10 20 -1 30 50 -1 60 -1 -1 40 -1 -1
```

Sample Output

```
60
```

```cpp
#include <iostream>
#include<vector>
#include<string>
#include<stack>
#include<limits.h>
using namespace std;
struct Node{
    int data;
    vector<Node*>children;
};
Node *newNode(int key){
    Node *temp=new Node;
    temp->data=key;
    return temp;

}

Node *construct(int arr[],int n ){
    Node *root=NULL;
    stack<Node*>st;
    for(int i=0;i<n;i++){
        if(arr[i]==-1){
            st.pop();
        }else{
            Node *t=newNode(arr[i]);

            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}
int max1(Node *node)
{
   //write your code here
   int max = INT_MIN;
   if(node->data > max){
       max = node->data;
   }
   for(Node* child:node->children){
       int mm = max1(child);
       if(mm > max){
           max = mm;
       }
   }
    return max;

}
```

```
int main(){

    int n;
    cin>>n;

    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }

    Node *root=construct(arr,n);
    int maximum=max1(root);

    cout << maximum << endl;
}
```

Height Of A Generic Tree

Easy

1. You are given a partially written GenericTree class.
2. You are required to complete the body of height function. The function is expected to find the height of tree. Depth of a node is defined as the number of edges it is away from the root (depth of root is 0). Height of a tree is defined as depth of deepest node.
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

```
12
10 20 -1 30 50 -1 60 -1 -1 40 -1 -1
```

Sample Output

```
2
```

```
#include <bits/stdc++.h>
using namespace std;
struct Node{
    int data;
    vector<Node*>children;
};
```

```
Node *newNode(int key)
{
    Node *temp=new Node;
    temp->data=key;
    return temp;

}

Node *construct(int arr[],int n )
{
    Node *root=NULL;
    stack<Node*>st;
    for(int i=0;i<n;i++){
        if(arr[i]==-1){
            st.pop();
        }else{
            Node *t=newNode(arr[i]);
            //t->data=arr[i];

            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}

int height(Node *node)
{
    //Write your code here
    //pepcoding solution
    int h = -1;
    for(Node* child:node->children){
        int ch = height(child);
        if(ch>h){
            h = ch;
        }
    }
    h += 1;
    //my solution
    // int h = 0;
    // for(Node* child:node->children){
    //     int ch = height(child);
    //     if(ch>h){
    //         h = ch;
    //     }
    // }
    // if(node->children.size() != 0){
    //     h += 1;
```

```
    // }
    return h;
}

int main()
{
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++)
    {
        cin>>arr[i];
    }
    Node *root=construct(arr,n);
    int h=height(root);
    cout << h << endl;
}
```

Generic Tree - Traversals (pre-order, Post-order)

Easy

1. You are given a partially written GenericTree class.
2. You are required to complete the body of traversals function. The function is expected to visit every node. While traversing the function must print following content in different situations.
   2.1. When the control reaches the node for the first time -> "Node Pre" node.data.
   2.2. Before the control leaves for a child node from a node -> "Edge Pre" node.data--child.data.
   2.3. After the control comes back to a node from a child -> "Edge Post" node.data--child.data.
   2.4. When the control is about to leave node, after the children have been visited -> "Node Post" node.data.
3. Input is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

As suggested in Sample Output

Example

Sample Input

```
12
10 20 -1 30 50 -1 60 -1 -1 40 -1 -1
```

Sample Output

```
Node Pre 10
Edge Pre 10--20
Node Pre 20
Node Post 20
Edge Post 10--20
Edge Pre 10--30
```

```
Node Pre 30
Edge Pre 30--50
Node Pre 50
Node Post 50
Edge Post 30--50
Edge Pre 30--60
Node Pre 60
Node Post 60
Edge Post 30--60
Node Post 30
Edge Post 10--30
Edge Pre 10--40
Node Pre 40
Node Post 40
Edge Post 10--40
Node Post 10
```

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node{
    int data;
    vector<Node*>children;
};

Node *newNode(int key){
    Node *temp=new Node;
    temp->data=key;
    return temp;

}

Node *construct(int arr[],int n ){
    Node *root=NULL;
    stack<Node*>st;
    for(int i=0;i<n;i++){
        if(arr[i]==-1){
            st.pop();
        }else{
            Node *t=newNode(arr[i]);
            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}
```

```
void traversals(Node *node)
{
  //write your code here
  cout<<"Node Pre "<<node->data<<endl;
  for(Node *child:node->children){
      cout<<"Edge Pre "<<node->data<<"--"<<child->data<<endl;
      traversals(child);
      cout<<"Edge Post "<<node->data<<"--"<<child->data<<endl;
  }
  cout<<"Node Post "<<node->data<<endl;
}

int main(){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    Node *root=construct(arr,n);
    traversals(root);
}
```

Level-order Of Generic Tree

Easy

1. You are given a partially written GenericTree class.
2. You are required to complete the body of levelorder function. The function is expected to visit every node in "levelorder fashion". Please check the question video for more details.
3. Input is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

All nodes from left to right (level by level) all separated by a space and ending in a "."

Example

Sample Input

```
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1
```

Sample Output

```
10 20 30 40 50 60 70 80 90 100 110 120 .
```

```
#include <bits/stdc++.h>
using namespace std;
struct Node{
    int data;
    vector<Node*>children;
};
```

```
Node *newNode(int key){
  Node *temp=new Node;
  temp->data=key;
  return temp;

}

Node *construct(int arr[],int n ){
    Node *root=NULL;
    stack<Node*>st;
    for(int i=0;i<n;i++){
        if(arr[i]==-1){
            st.pop();
        }else{
            Node *t=newNode(arr[i]);
            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}


void levelorder(Node *node)
{
    //write your code here
    queue<Node*> q;
    q.push(node);

    while(q.size()>0){
        //till not empty-> pop(), display, add its childrens(if
any)
        Node * a = q.front();
        q.pop();
        cout<<a->data<<" ";
        for(Node* child:a->children){
            q.push(child);
        }
    }
    cout<<"."<<endl;

}

int main(){
    int n;
    cin>>n;
    int arr[n];
```

```
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    Node *root=construct(arr,n);
    levelorder(root);
}
```

Levelorder Linewise (generic Tree)

Easy

1. You are given a partially written GenericTree class.
2. You are required to complete the body of levelorderLineWise function. The function is expected to visit every node in "levelorder fashion" and print them from left to right (level by level). All nodes on same level should be separated by a space. Different levels should be on separate lines. Please check the question video for more details.
3. Input is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

All nodes from left to right (level by level) all separated by a space.
All levels on separate lines starting from top to bottom.

Example

Sample Input

```
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1
```

Sample Output

```
10
20 30 40
50 60 70 80 90 100
110 120
```

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

class Node

{

public:

  int data = 0;

  vector<Node *> children;


  Node(int data)
```

```
  {

    this->data = data;

  }

};

void display(Node *node)
{
  string s = "";
  s += to_string(node->data) + " -> ";
  for (Node *child : node->children)
  {
    s += to_string(child->data) + ", ";
  }

  cout << s << "." << endl;

  for (Node *child : node->children)
  {
    display(child);
  }
}

Node *constructor01(vector<int> &arr)

{

  if (arr.size() == 0)

    return NULL;



  vector<Node *> stack;

  stack.push_back(new Node(arr[0]));



  Node *root = stack[0];



  for (int i = 1; i < arr.size(); i++)

  {

    if (arr[i] != -1)
```

```
    {
      Node *node = stack.back();
      Node *nnode = new Node(arr[i]);

      node->children.push_back(nnode);
      stack.push_back(nnode);
    }
    else
      stack.pop_back();
  }
  return root;
}
//MY SOLUTION //see pepcoding solution in video
//change the name of funtion for specific function

// void levelorderLineWise(Node *node)
// {
// //   write code here
//
//     queue<Node*> q;
//     q.push(node);
//     int r{1};
//     int b{};

//     while(q.size() != 0){
//         Node* a = q.front();
//         q.pop();

//         for(Node* child:a->children){
//           q.push(child);
//           b++;
//         }
//         cout<<a->data <<" ";
//         r--;
//         if(r == 0){
//           r = b;
//           b = 0;
//           cout<<endl;
//         }

//     }
```

```
// }



//PEPCODING SOLUTION //change the name of funtion for specific
function
//main queue and child queue approach
void levelorderLineWiseP(Node *node){
  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;
    }
  }
}



// ANOTHER APPROACH
//delimiter approach
//change the name of funtion for specific function

// void levelorderLineWise1(Node *node)
// {
// //   write code here
// //MY SOLUTION //see pepcoding solution in video
//     queue<Node*> q;
//     q.push(node);
//     q.push(NULL);

//     while(q.size() != 0){
//         Node* a = q.front();
//         q.pop();

//         if(a != NULL){
//             for(Node* child:a->children){
//             q.push(child);
```

```
//             }
//             cout<<a->data <<" ";
//         }
//         else if(a == NULL && q.size() > 0){
//             cout<<endl;
//             q.push(NULL);
//         }


//     }

// }


//ANOTHER APPROACH 2
//SIZE , COUNT APPROACH
void levelorderLineWiseS(Node *node) {
  queue<Node*> q;
  q.push(node);

  while(q.size() > 0) {
    int cicl = q.size();//children in current level
    for(int i {};i<cicl;i++){
      //remove print add
      Node * n = q.front();
      q.pop();
      cout<<n->data<<" ";
      for(Node * child:n->children) {
        q.push(child);
      }
    }
    cout<<endl;
  }
}

class Pair{
    public:
    Node* node;
    int level;
    Pair(Node * n, int lev) {
        this->node = n;
        this->level = lev;
    }
};
void levelorderLineWisePair(Node *node) {
  queue<Pair> q;
  q.push(Pair(node,1));
  int level {1};
  while(q.size() > 0) {
      Pair rem = q.front(); q.pop();
    if(rem.level > level){
        level = rem.level;
```

```
            cout<<endl;
        }
        cout<<rem.node->data<<" ";
        for(Node * child : rem.node->children){
            q.push(Pair(child,rem.level+1));
        }
    }

}
void solve()
{
    int n;
    cin>>n;
    vector<int>arr(n);
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    Node * root=constructor01(arr);

    levelorderLineWisePair(root);
    //  display(root);

}

int main()
{
    solve();
    return 0;
}
```

Levelorder Linewise Zig Zag

Medium

1. You are given a partially written GenericTree class.
2. You are required to complete the body of levelorderLineWiseZZ function. The function is expected to visit every node in "levelorder fashion" but in a zig-zag manner i.e. 1st level should be visited from left to right, 2nd level should be visited from right to left and so on. All nodes on same level should be separated by a space. Different levels should be on separate lines. Please check the question video for more details.
3. Input is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

All nodes on the same level should be separated by a space.
1st level should be visited left to right, 2nd from right to left and so on alternately.
All levels on separate lines starting from top to bottom.

Example

Sample Input

```
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1
```

Sample Output

```
10
40 30 20
50 60 70 80 90 100
120 110
```

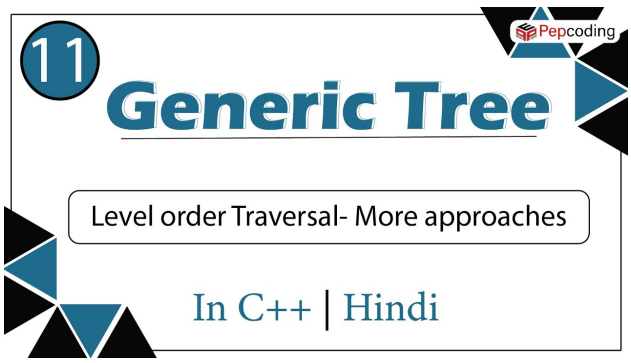
```
#include <iostream>
#include <vector>
#include <string>
#include <queue>
#include <stack>
#include <cstdlib>
#include<climits>
using namespace std;

class Node

{

public:

  int data = 0;

  vector<Node *> children;


  Node(int data)

  {

    this->data = data;

  }

};

Node *constructor01(vector<int> &arr)

{

  if (arr.size() == 0)

    return NULL;


  vector<Node *> stack;
```

```
  stack.push_back(new Node(arr[0]));


  Node *root = stack[0];


  for (int i = 1; i < arr.size(); i++)
  {
    if (arr[i] != -1)
    {
      Node *node = stack.back();
      Node *nnode = new Node(arr[i]);


      node->children.push_back(nnode);
      stack.push_back(nnode);
    }
    else
      stack.pop_back();
  }
  return root;
}
//MY SOLUTION
void levelOderZigZag(Node *node)

{
  //write code here
  queue <Node*> q;
  q.push(node);
  int a {1};
  int b{};
  bool flag{false};
  string str {""};
  while(q.size() > 0){
      Node* node = q.front();
      q.pop();
```

```
      for(Node* child:node->children){
        q.push(child);
        b++;
      }

      if(flag == true){
        if(a != 0){
          str =  " " + str;
        }
        str = to_string(node->data) +""+ str;
        a--;
        if(a == 0) {
          cout<<str<<endl;
          a = b;
          b = 0;
          str = "";
          flag = false;
        }
      }else{
        cout<<node->data << " ";
        a--;
        if(a == 0) {
          a = b;
          b = 0;
          flag = true;
          cout<<endl;
        }
      }
  }

}
//PEPCODING SOLUTION
// void levelOderZigZag(Node *node){
//     stack<Node *> main;
//     main.push(node);

//     stack<Node *> ch;
//     int level{};////////taking initial level as 0
//     while(main.size() > 0){
//       Node * n = main.top();
//       main.pop();
//       cout<<n->data<<" ";
//       if(level % 2 == 0) {
//         for(Node * child : n->children){
//           ch.push(child);
//         }
//       }else{
//         int si = n->children.size() - 1;
//         for(int i {si};i >=0 ; i--) {
//           ch.push(n->children[i]);
//         }
//       }
```

```
//         if(main.size() == 0){
//           main = ch;
//           stack <Node*> temp;
//           ch = temp;
//           level++;
//           cout<<endl;
//         }
//       }
// }

void solve()
{
    int n;
    cin>>n;
    vector<int>arr(n);
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    Node * root=constructor01(arr);
    levelOderZigZag(root);
}

int main()
{
    solve();
    return 0;
}
```

Level order traversal - more approaches



Find In Generic Tree

Easy

1. You are given a partially written GenericTree class.
2. You are required to complete the body of find function. The function is expected to find the given data in the tree, if found it should return true or return false.
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

```
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1
120
```

Sample Output

```
true
```

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

using namespace std;

class Node
{
public:
  int data = 0;
  vector<Node *> children;

  Node(int data)
  {
    this->data = data;
  }
};


void display(Node *node)
{
  string s = "";
  s += to_string(node->data) + " Child: ";
  for (Node *child : node->children)
  {
    s += to_string(child->data) + ", ";
  }

  cout << s << "." << endl;
```

```
  for (Node *child : node->children)
  {
    display(child);
  }
}


Node *constructor01(vector<int> &arr)
{
  if (arr.size() == 0)
    return NULL;

  vector<Node *> stack;
  stack.push_back(new Node(arr[0]));

  Node *root = stack[0];

  for (int i = 1; i < arr.size(); i++)
  {
    if (arr[i] != -1)
    {
      Node *node = stack.back();
      Node *nnode = new Node(arr[i]);

      node->children.push_back(nnode);
      stack.push_back(nnode);
    }
    else
      stack.pop_back();
  }
  return root;
}


bool find(Node *node, int data)
{
  // Write your code here
  if(node->data == data){
    return true;
  }
  bool flag = false;
  for(Node * child:node->children) {
    flag = find(child,data);
    if(flag == true){
      break;
    }
  }

  return flag;
}
```

```
void solve()
{
    int n;
    cin>>n;
    vector<int>arr(n,0);
    for(int i = 0; i < arr.size(); i++)
    {
        cin>>arr[i];
    }

    int data;
    cin>>data;

    Node *root = constructor01(arr);
    bool flag = find(root,data);
    if(flag == true){
        cout <<"true"<<endl;
    }
    else{
        cout <<"false"<<endl;
    }

}

int main()
{
    solve();
    return 0;
}
```

Node To Root Path In Generic Tree

Easy

1. You are given a partially written GenericTree class. 2. You are required to complete the body of nodeToRootPath function. The function is expected to return in form of linked list the path from element to root, if the element with data is found. 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

```
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1
120
```

Sample Output

```
[120, 80, 30, 10]
```

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

using namespace std;

class Node
{
public:
  int data = 0;
  vector<Node *> children;

  Node(int data)
  {
    this->data = data;
  }
};


void display(Node *node)
{
  string s = "";
  s += to_string(node->data) + " Child: ";
  for (Node *child : node->children)
  {
    s += to_string(child->data) + ", ";
  }

  cout << s << "." << endl;

  for (Node *child : node->children)
  {
    display(child);
  }
}
```

```
Node *constructor01(vector<int> &arr)
{
  if (arr.size() == 0)
    return NULL;

  vector<Node *> stack;
  stack.push_back(new Node(arr[0]));

  Node *root = stack[0];

  for (int i = 1; i < arr.size(); i++)
  {
    if (arr[i] != -1)
    {
      Node *node = stack.back();
      Node *nnode = new Node(arr[i]);

      node->children.push_back(nnode);
      stack.push_back(nnode);
    }
    else
      stack.pop_back();
  }
  return root;
}

vector<int> nodeToRootPath(Node *node, int data)
{
    // Write your code here
  if(node->data == data){
    vector<int> vec;
    vec.push_back(node->data);
    return vec;
  }
  vector<int> vec;
  for(Node * child:node->children) {
    vec = nodeToRootPath(child,data);
    if(vec.size() > 0){
      vec.push_back(node->data);
      break;
    }
  }
  return vec;
}


void solve()
{
    int n;
    cin>>n;
    vector<int>arr(n,0);
    for(int i = 0; i < arr.size(); i++)
```

```
    {
        cin>>arr[i];
    }

    int data;
    cin>>data;

    Node *root = constructor01(arr);
    vector<int> ans = nodeToRootPath(root,data);

    cout<<"[";
    for(int i =0 ; i < ans.size(); i++)
    {
        cout<<ans[i];
        if(i!=ans.size()-1)
            cout<<", ";
    }
    cout<<"]";

}

int main()
{
    solve();
    return 0;
}
```

Lowest Common Ancestor (generic Tree)

Easy

1. You are given a partially written GenericTree class. 2. You are required to complete the body of lca function. The function is expected to return the lowest common ancestor of two data values that are passed to it. Please watch the question video to understand what lca is. 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

```
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1
120
80
```

Sample Output

```
80
```

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

using namespace std;

class Node
{
public:
  int data = 0;
  vector<Node *> children;

  Node(int data)
  {
    this->data = data;
  }
};


void display(Node *node)
{
  string s = "";
  s += to_string(node->data) + " Child: ";
  for (Node *child : node->children)
  {
    s += to_string(child->data) + ", ";
  }

  cout << s << "." << endl;

  for (Node *child : node->children)
  {
    display(child);
  }
}
```

```
Node *constructor01(vector<int> &arr)
{
  if (arr.size() == 0)
    return NULL;

  vector<Node *> stack;
  stack.push_back(new Node(arr[0]));

  Node *root = stack[0];

  for (int i = 1; i < arr.size(); i++)
  {
    if (arr[i] != -1)
    {
      Node *node = stack.back();
      Node *nnode = new Node(arr[i]);

      node->children.push_back(nnode);
      stack.push_back(nnode);
    }
    else
      stack.pop_back();
  }
  return root;
}

vector<int> *rootNodeToPath(Node *node, int data)
{

  if (node->data == data)
  {
    vector<int> *base = new vector<int>();
    base->push_back(data);
    return base;
  }

  vector<int> *ans = new vector<int>();
  for (Node *child : node->children)
  {
    vector<int> *recAns = rootNodeToPath(child, data);
    if (recAns->size() > 0)
    {
      recAns->push_back(node->data);
      return recAns;
    }
  }
  return ans;
}
```

```
int lca(Node *node, int data1, int data2){

    // Write your code here
    vector<int>* v1 = rootNodeToPath(node,data1);
    vector<int>* v2 = rootNodeToPath(node,data2);
    int i = v1->size()-1;
    int j = v2->size()-1;
    while(i >= 0 && j >= 0 && v1->at(i) == v2->at(j)){
      i--;
      j--;
    }
    i++;
    j++;
    return v1->at(i);
}

void solve()
{
    int n;
    cin>>n;
    vector<int>arr(n,0);
    for(int i = 0; i < arr.size(); i++)
    {
        cin>>arr[i];
    }

    int data1;
    cin>>data1;
    int data2;
    cin>>data2;

    Node *root = constructor01(arr);
    int Lca = lca(root,data1,data2);
    cout<<Lca<<endl;
}

int main()
{
    solve();
    return 0;
}
```

Distance Between Two Nodes In A Generic Tree

Easy

1. You are given a partially written GenericTree class.
2. You are required to complete the body of distanceBetweenNodes function. The function is expected to return the distance (in terms of number of edges) between two nodes in a generic tree.
Please watch the question video to understand what lca is.
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

```
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1
100
110
```

Sample Output

```
5
```

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

using namespace std;

class Node
{
public:
  int data = 0;
  vector<Node *> children;

  Node(int data)
  {
    this->data = data;
  }
};


void display(Node *node)
{
  string s = "";
  s += to_string(node->data) + " Child: ";
  for (Node *child : node->children)
  {
    s += to_string(child->data) + ", ";
  }

  cout << s << "." << endl;

  for (Node *child : node->children)
  {
    display(child);
  }
}
```

```
Node *constructor01(vector<int>arr)
{
  if (arr.size() == 0)
    return NULL;

  vector<Node *> stack;
  stack.push_back(new Node(arr[0]));

  Node *root = stack[0];

  for (int i = 1; i < arr.size(); i++)
  {
    if (arr[i] != -1)
    {
      Node *node = stack.back();
      Node *nnode = new Node(arr[i]);

      node->children.push_back(nnode);
      stack.push_back(nnode);
    }
    else
      stack.pop_back();
  }
  return root;
}


vector<int> *rootNodeToPath(Node *node, int data)
{

  if (node->data == data)
  {
    vector<int> *base = new vector<int>();
    base->push_back(data);
    return base;
  }

  vector<int> *ans = new vector<int>();
  for (Node *child : node->children)
  {
    vector<int> *recAns = rootNodeToPath(child, data);
    if (recAns->size() > 0)
    {
      recAns->push_back(node->data);
      return recAns;
    }
  }
  return ans;
}


int lca(Node *node, int data1, int data2)
```

```
{
  vector<int> onePath = *rootNodeToPath(node, data1);
  vector<int> twoPath = *rootNodeToPath(node, data2);

  int i = onePath.size() - 1;
  int j = twoPath.size() - 1;
  bool flag = false;
  while (i >= 0 && j >= 0 && onePath[i] == twoPath[j])
  {
    flag = true;
    i--;
    j--;
  }

  int ans = 0;
  if (flag)
  {
    ans = onePath[i++];
  }
  return ans;
}


int distance(Node *node, int data1, int data2)
{
   // Write your code here
  vector<int> *v1 = rootNodeToPath(node, data1);
  vector<int> *v2 = rootNodeToPath(node, data2);
  int i = v1->size()-1;
  int j = v2->size()-1;
  while(i >= 0 && j >= 0 && v1->at(i) == v2->at(j)){
    i--;
    j--;
  }
  i++;j++;
  return i+j;
}

void solve()
{
    int n;
    cin>>n;
    vector<int>arr(n,0);
    for(int i = 0; i < n; i++)
    {
        cin>>arr[i];
    }

    int data1;
    cin>>data1;
    int data2;
    cin>>data2;
```

```
    Node *root = constructor01(arr);
    int dist =  distance(root,data1,data2);
    cout<<dist<<endl;

}

int main()
{
    solve();
    return 0;
}
```

Are Trees Similar In Shape

Easy

1. You are given a partially written GenericTree class.
2. You are required to complete the body of areSimilar function. The function is expected to check if the two trees passed to it are similar in shape or not.
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

```
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1
24
1 2 5 -1 6 -1 -1 3 7 -1 8 11 -1 12 -1 -1 9 -1 -1 4 10 -1 -1 -1
```

Sample Output

```
true
```

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

using namespace std;

class Node
{
public:
  int data = 0;
  vector<Node *> children;

  Node(int data)
  {
    this->data = data;
  }
```

```
};


void display(Node *node)
{
  string s = "";
  s += to_string(node->data) + " Child: ";
  for (Node *child : node->children)
  {
    s += to_string(child->data) + ", ";
  }

  cout << s << "." << endl;

  for (Node *child : node->children)
  {
    display(child);
  }
}

Node *constructor01(vector<int>arr)
{
  if (arr.size() == 0)
    return NULL;

  vector<Node *> stack;
  stack.push_back(new Node(arr[0]));

  Node *root = stack[0];

  for (int i = 1; i < arr.size(); i++)
  {
    if (arr[i] != -1)
    {
      Node *node = stack.back();
      Node *nnode = new Node(arr[i]);

      node->children.push_back(nnode);
      stack.push_back(nnode);
    }
    else
      stack.pop_back();
  }
  return root;
}


int sizeRec(Node *node)
{
  int size = 0;
  for (Node *child : node->children)
  {
```

```
    size += sizeRec(child);
  }
  return size + 1;
}


int maxEle(Node *node)
{
  int OverMax = node->data;
  for (Node *child : node->children)
  {
    OverMax = max(OverMax, maxEle(child));
  }
  return OverMax;
}


int height(Node *node)
{
  int ht = 0;
  for (Node *child : node->children)
  {
    ht = max(ht, height(child));
  }
  return ht + 1;
}


bool areSimilar(Node *n1, Node *n2)
{
    // Write your code here
    if(n1->children.size() != n2->children.size()){
        return false;
    }

    for(int i{};i < n1->children.size();i++){
      bool a =  areSimilar(n1->children[i],n2->children[i]);
      if(a == false) {
        return a;
      }
    }

    return true;
}


void solve()
{
    int n1;
    cin>>n1;
    vector<int>arr1(n1,0);
    for(int i = 0; i < n1; i++)
```

```
    {
        cin>>arr1[i];
    }
     Node *root1 = constructor01(arr1);

     int n2;
    cin>>n2;
    vector<int>arr2(n2,0);
    for(int i = 0; i < n2; i++)
    {
        cin>>arr2[i];
    }

    Node *root2 = constructor01(arr2);

     bool similar = areSimilar(root1, root2);
    if(similar == true)
    {
        cout<<"true"<<endl;
    }
    else
    {
        cout<<"false"<<endl;
    }
}

int main()
{
    solve();
    return 0;
}
```

Are Trees Mirror In Shape

Easy

1. You are given a partially written GenericTree class.
2. You are required to complete the body of areMirror function. The function is expected to check if the two trees passed to it are mirror images of each other in shape (data not to be checked, just the shape of tree).
Note -> Watch the 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

```
12
10 20 -1 30 50 -1 60 -1 -1 40 -1 -1
12
100 200 -1 300 500 -1 600 -1 -1 400 -1 -1
```

Sample Output

```
true
```

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

using namespace std;

class Node
{
public:
  int data = 0;
  vector<Node*> children;

  Node(int data)
  {
    this->data = data;
  }
};


void display(Node* node)
{
  string s = "";
  s += to_string(node->data) + " Child: ";
  for (Node* child : node->children)
  {
    s += to_string(child->data) + ", ";
  }

  cout << s << "." << endl;

  for (Node* child : node->children)
  {
    display(child);
  }
}

Node* constructor01(vector<int>arr)
{
  if (arr.size() == 0)
    return NULL;

  vector<Node*> stack;
  stack.push_back(new Node(arr[0]));

  Node* root = stack[0];
```

```
  for (int i = 1; i < arr.size(); i++)
  {
    if (arr[i] != -1)
    {
      Node* node = stack.back();
      Node* nnode = new Node(arr[i]);

      node->children.push_back(nnode);
      stack.push_back(nnode);
    }
    else
      stack.pop_back();
  }
  return root;
}


int sizeRec(Node* node)
{
  int size = 0;
  for (Node* child : node->children)
  {
    size += sizeRec(child);
  }
  return size + 1;
}


int maxEle(Node* node)
{
  int OverMax = node->data;
  for (Node* child : node->children)
  {
    OverMax = max(OverMax, maxEle(child));
  }
  return OverMax;
}


int height(Node* node)
{
  int ht = 0;
  for (Node* child : node->children)
  {
    ht = max(ht, height(child));
  }
  return ht + 1;
}
```

```
bool areMirrorInShape(Node* n1, Node* n2){

  // Write your code here
    if(n1->children.size() != n2->children.size()){
        return false;
    }

    int size = n1->children.size();
    for(int i{},j{size - 1};i < size && j >= 0 ;i++,j--){
      bool a =  areMirrorInShape(n1->children[i],n2->children[j]);
      if(a == false) {
        return a;
      }
    }

    return true;
}


void solve()
{
  int n1;
  cin >> n1;
  vector<int>arr1(n1, 0);
  for (int i = 0; i < n1; i++)
  {
    cin >> arr1[i];
  }
  Node* root1 = constructor01(arr1);

  int n2;
  cin >> n2;
  vector<int>arr2(n2, 0);
  for (int i = 0; i < n2; i++)
  {
    cin >> arr2[i];
  }

  Node* root2 = constructor01(arr2);

  bool similar = areMirrorInShape(root1, root2);
  if (similar == true)
  {
    cout << "true" << endl;
  }
  else
  {
    cout << "false" << endl;
  }
}
```

```
int main()
{
  solve();
  return 0;
}
```

Is Generic Tree Symmetric

Easy

1. You are given a partially written GenericTree class.
2. You are required to complete the body of IsSymmetric function. The function is expected to check if the tree is symmetric, if so return true otherwise return false. For knowing symmetricity think of face and hand. Face is symmetric while palm is not. Also, we are check only smmetricity of shape and not content. Check the 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

```
20
10 20 50 -1 60 -1 -1 30 70 -1 80 -1 90 -1 -1 40 100 -1 110 -1 -1 -1
```

Sample Output

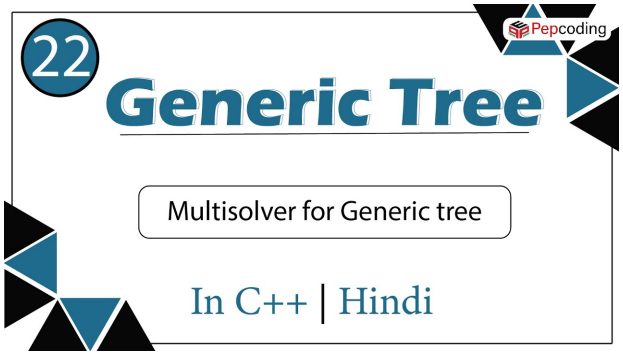
```
true
```

```
#include<bits/stdc++.h>
#include<iostream>
using namespace std;
int c= INT_MAX;
int flloor= INT_MIN;

struct Node{
    int data;
    vector<Node*>children;
};

Node *newNode(int key){
    Node *temp=new Node;
    temp->data=key;
    return temp;

}

Node *construct(int arr[],int n ){
    Node *root=NULL;
    stack<Node*>st;
    for(int i=0;i<n;i++){
```

```
        if(arr[i]==-1){
            st.pop();
        }else{
            Node *t=newNode(arr[i]);


            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}

bool issym(Node *node)
{
    //Write your code here
    //
    //SYMETTRIC THINGS ARE MIRROR OF EACH OTHER
    //so you can use the areMirror function(used earliar) and pass
node two times
    //see the video

    //OR ANOTHER SOLUTION (MY)
    if(node->children.size() == 0) {
        return true;
    }

    int r {};
    int s = node->children.size() - 1;
    while(r < s) {
        if(node->children[r]->children.size() != node-
>children[s]->children.size()){
            return false;
        }
        r++;
        s--;
    }

    for(int i{};i < node->children.size();i++){
      bool a =  issym(node->children[i]);
      if(a == false) {
        return a;
      }
    }

    return true;
}
```

```cpp
int main(){

    int n;
    cin>>n;

    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    Node *root=construct(arr,n);
    bool sym = issym(root);
    if(sym == 1){
    cout<<"true";
    }
    else
    {
        cout<<"false";
    }
}
```

Generic Tree - Multisolver



Predecessor And Successor Of An Element

Easy

1. You are given a partially written GenericTree class.
2. You are required to find the preorder predecessor and successor of a given element. Use the "travel and change" strategy explained in the earlier video. The static properties have been declared for you. You can declare more if you want.
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

```
24
10 20 -50 -1 60 -1 -1 30 70 -1 -80 110 -1 -120 -1 -1 90 -1 -1 40 -100 -1 -1
-1
-120
```

Sample Output

```
Predecessor = 110
Successor = 90
```

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



struct Node{
    int data;
    vector<Node*>children;
};

Node *newNode(int key){
    Node *temp=new Node;
    temp->data=key;
    return temp;

}

Node *construct(int arr[],int n ){
    Node *root=NULL;
    stack<Node*>st;
    for(int i=0;i<n;i++){
        if(arr[i]==-1){
            st.pop();
        }else{
            Node *t=newNode(arr[i]);
```

```
            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}

Node *pre ;
Node *suc ;
int state=0 ;

//they are NULL initially while declaring
void prnsc(Node *node, int data){
    //Write your code here
    if(state == 0){
        if(node->data == data){
            state = 1;
        }else{
            pre = node;
        }
    }else if (state == 1){
        suc = node;
        state = 2;
    }









//sad
    // if(pre == NULL){
    //     cout<<"nice"<<endl;
    // }
    // if (state == 2) {
    //     suc = node;
    //     state = 3;
    // }else if(state == 0 ){
    //     pre = NULL;
    //     state = 1;
```

```
    // }
    // if(node-> data == data) {
    //     state = 2;
    // }
    // else if(node->data != data && state != 3) {
    //     pre = node ;
    // }
    // if(state != 3) {
    //     suc = NULL;
    // }

    for(Node * child: node ->children) {
        prnsc(child,data);
    }
}

int main(){

    int n;
    cin>>n;

    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int val;
    cin>>val;
    Node *root=construct(arr,n);
    prnsc(root,val);
    if(pre==NULL){
      cout<<"Predecessor = Not found"<<endl;
    } else {
      cout<<"Predecessor = "<< pre->data<<endl;
    }

    if(suc==NULL){
      cout<<"Successor = Not found";
    } else {
      cout<<"Successor = " << suc->data;
    }
}
```

Ceil And Floor In Generic Tree

Easy

1. You are given a partially written GenericTree class.
2. You are required to find the ceil and floor value of a given element. Use the "travel and change" strategy explained in the earlier video. The static properties - ceil and floor have been declared for you. You can declare more if you want. If the element is largest ceil will be largest integer value (32 bits), if element is smallest floor will be smallest integer value (32 bits). Watch the 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

```
24
10 20 -50 -1 60 -1 -1 30 70 -1 -80 110 -1 -120 -1 -1 90 -1 -1 40 -100 -1 -1
-1
70
```

Sample Output

```
CEIL = 90
FLOOR = 60
```

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


struct Node{
    int data;
    vector<Node*>children;
};

Node *newNode(int key){
    Node *temp=new Node;
    temp->data=key;
    return temp;

}

Node *construct(int arr[],int n ){
    Node *root=NULL;
    stack<Node*>st;
```

```
    for(int i=0;i<n;i++){
        if(arr[i]==-1){
            st.pop();
        }else{
            Node *t=newNode(arr[i]);

            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}

int c = INT_MAX;
int flloor= INT_MIN;

void cnf(Node *node, int data){
    //Write your code here
    if(node->data < data && node->data > flloor){
        flloor = node->data;
    }else if(node->data > data && node->data < c){
        c = node->data;
    }

    for(Node * child: node->children) {
        cnf(child,data);
    }

}

int main(){

    int n;
    cin>>n;

    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int data;
    cin>>data;
    Node *root=construct(arr,n);
    cnf(root,data);
    cout<<"CEIL = "<<c<<endl;
    cout<<"FLOOR = "<<flloor<<endl;
}
```

Kth Largest Element In Tree

Medium

1. You are given a partially written GenericTree class.
2. You are given a number k. You are required to find and print the kth largest value in 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

```
24
10 20 -50 -1 60 -1 -1 30 70 -1 -80 110 -1 -120 -1 -1 90 -1 -1 40 -100 -1 -1
-1
8
```

Sample Output

```
10
```

```
#include<bits/stdc++.h>
#include<iostream>
using namespace std;
int c= INT_MAX;
int flloor= INT_MIN;

struct Node{
    int data;
    vector<Node*>children;
};

Node *newNode(int key){
    Node *temp=new Node;
    temp->data=key;
    return temp;

}

Node *construct(int arr[],int n ){
    Node *root=NULL;
    stack<Node*>st;
    for(int i=0;i<n;i++){
        if(arr[i]==-1){
            st.pop();
```

```
        }else{
            Node *t=newNode(arr[i]);
            //t->data=arr[i];

            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}



void cnf(Node *node, int data){
    if(node->data > data)
    {
        if(node->data <c)
        {
            c = node->data;
        }
    }

    if(node->data < data)
    {
        if(node->data > flloor)
        {
            flloor = node->data;
        }
    }

    for(Node *child: node->children)
    {
        cnf(child,data);
    }

}

int kthlarge(Node *node, int k)
{
    //Write your code here
    flloor = INT_MIN;
    int a = INT_MAX;
    for(int i {};i < k;i++){
        cnf(node,a);
        a = flloor;
        flloor = INT_MIN;

    }
```

```
    return a;
}


//bekaar solution
// vector<int> v;

// void fill_vector(Node * node){
//     queue<Node*> q;
//     q.push(node);

//     while(q.size() > 0){
//         Node * a = q.front();
//         v.push_back(a->data);
//         q.pop();
//         for(Node* child:a->children){
//             q.push(child);
//         }
//     }

// }
// int kthlarge(Node *node, int k){
//     //Write your code here
//     fill_vector(node);
//     sort(v.begin(), v.end());

//     return v[v.size() - k ];

// }
int main(){

    int n;
    cin>>n;

    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int k;
    cin>>k;
    Node *root=construct(arr,n);
    int kthlrg = kthlarge(root,k);
    cout<<kthlrg;
}
```

Node With Maximum Subtree Sum

Medium

1. You are given a partially written GenericTree class.
2. You are required to find and print the node which has the subtree with largest sum. Also print the sum of the concerned subtree separated from node's value by an '@'. Refer the question video for clarity.
3. Input is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

@

Example

Sample Input

```
20
10 20 -50 -1 60 -1 -1 30 -70 -1 80 -1 90 -1 -1 40 -100 -1 -1 -1
```

Sample Output

```
30@130
```

```
#include<bits/stdc++.h>
#include<iostream>
using namespace std;
int c= INT_MAX;
int flloor= INT_MIN;

struct Node{
    int data;
    vector<Node*>children;
};

Node *newNode(int key){
    Node *temp=new Node;
    temp->data=key;
    return temp;

}

Node *construct(int arr[],int n ){
    Node *root=NULL;
    stack<Node*>st;
    for(int i=0;i<n;i++){
        if(arr[i]==-1){
            st.pop();
        }else{
            Node *t=newNode(arr[i]);

            if(st.size()>0){
                st.top()->children.push_back(t);
            }else{
```

```
                root=t;
            }
            st.push(t);
        }
    }
    return root;
}

int msum = INT_MIN;
int msumnode = 0;

int subsumtree(Node *node)
{
    //Write your code here
    /*by travel and change
    return something and changing something else */
    if(node->children.size() == 0){
        if(node->data > msum){
            msum = node->data;
            msumnode = node->data;
        }
        return node->data;
    }

    int c_sum = 0;
    for(Node * child:node->children){
        c_sum += subsumtree(child);
    }

    int n_sum = node->data + c_sum;
    if(n_sum > msum){
        msum = n_sum ;
        msumnode = node->data;
    }
    return n_sum;
}

int main(){

    int n;
    cin>>n;

    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int k;
    cin>>k;
    Node *root=construct(arr,n);
    subsumtree(root);
    cout<<msumnode<<"@"<<msum<<endl;
}
```

Diameter Of Generic Tree

Medium

1. You are given a partially written GenericTree class.
2. You are required to find and print the diameter of tree. THe diameter is defined as maximum number of edges between any two nodes in the tree. Check the question video for clarity.
3. Input is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

diameter

Example

Sample Input

```
20
10 20 -50 -1 60 -1 -1 30 -70 -1 80 -1 90 -1 -1 40 -100 -1 -1 -1
```

Sample Output

```
4
```

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

class Node {
    public:
    int data;
    vector<Node*>children;
};



Node* construct(vector<int>&arr) {
    Node* root = nullptr;

    stack <Node*> st;
    for (int i = 0; i < arr.size(); i++) {
      if (arr[i] == -1) {
        st.pop();
      } else {
        Node* t = new Node();
        t->data = arr[i];

        if (st.size() > 0) {
          st.top()->children.push_back(t);
        } else {
          root = t;
        }
```

```
        st.push(t);
      }
    }
    return root;
  }
int dia = 0;
int diameter(Node* root){
//write your code here
/*
by travel and change
    return something and changing something else
     */

    int maxhic =0;
     for(Node *child:root->children){
        int height = diameter(child);
        if(height + maxhic > dia){
          dia = height + maxhic;
        }
        if(height > maxhic){
            maxhic = height;
        }
    }
    return maxhic + 1;

}
int main(){
    int n,x;
    cin>>n;
    vector<int>arr;
    for(int i=0;i<n;i++){
        cin>>x;
        arr.push_back(x);
    }
    Node *root = construct(arr);
    dia=0;
    diameter(root);
    cout<<dia<<endl;

}
```

Iterative Preorder And Postorder Of Generic Tree

Medium

1. You are given a partially written GenericTree class. 2. You are require to complete the body of function IterativePreandPostOrder. The function does not use recursion and prints preorder and postorder of the tree. Both orders are printed in separate lines (preorder first, followed by post order in next line). Elements in an order are separated by space. 3. Input and Output is managed for you.

Constraints

None

Format

Input

Input is managed for you

Output

Elements in preorder separated by a space
Elements in postorder separated by a space

Example

Sample Input

```
24
10 20 -50 -1 60 -1 -1 30 70 -1 -80 110 -1 -120 -1 -1 90 -1 -1 40 -100 -1 -1
-1
```

Sample Output

```
10 20 -50 60 30 70 -80 110 -120 90 40 -100
-50 60 20 70 110 -120 -80 90 30 -100 40 10
```

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

class Node {
    public:
    int data;
    vector<Node*>children;
};


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

void iterativePreAndPostOrder(Node* root){
  //write your code here
```

```
  stack<pair<Node *,int >> st ;

  pair<Node *,int> r;
  r.first = root;
  r.second = -1;
  st.push(r);
// cout<<root
  string pre ;
  string post ;
  while (st.size() > 0){
    pair <Node *, int >top = st.top();
    st.pop();
    if(top.second == -1){
      pre += (to_string)(top.first->data) + " ";
      top.second += 1;
      st.push(top);
    }else if (top.second == top.first->children.size()){
      post += (to_string)(top.first->data) + " ";
    }else{
      pair<Node *,int> c(top.first->children[top.second],-1);
      top.second += 1;
      st.push(top);
      st.push(c);
    }

  }

  cout<<pre<<endl;
  cout<<post<<endl;
}

Node* construct(vector<int>&arr) {
    Node* root = nullptr;

    stack <Node*> st;
    for (int i = 0; i < arr.size(); i++) {
      if (arr[i] == -1) {
        st.pop();
      } else {
        Node* t = new Node();
        t->data = arr[i];

        if (st.size() > 0) {
          st.top()->children.push_back(t);
        } else {
          root = t;
        }
        st.push(t);
      }
    }
    return root;
  }
```

```
int main(){
    int n,x;
    cin>>n;
    vector<int>arr;
    for(int i=0;i<n;i++){
        cin>>x;
        arr.push_back(x);
    }
    Node *root = construct(arr);
    iterativePreAndPostOrder(root);
}
```

Iterator and utterable | Java |