CONTEST

UPSOLVING

ARBOLES

Ft. Elisa

Height (Altura de un árbol binario)

Encontrar la altura máxima del árbol binario.

```
int height (Node *root){
  if(root == nullptr){
    return -1;
  }
  return max(height(root->left)+1, height(root->right)+1);
}
```

TreeTraversal1 (Recorrido)

Implementar recorrido Pre-Orden.

```
void preOrder(Node *root) {
  if(root == nullptr){
    return;
  }

  cout << root->data <<" ";
  preOrder(root->left);
  preOrder(root->right);
}
```

TreeTraversal2 (Recorrido)

Implementar recorrido Post-Orden.

```
void postOrder(Node *root) {
        if(root == nullptr){
            return;
        }

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

TreeTraversal3 (Recorrido)

Implementar recorrido In-Orden.

```
void inOrder(Node *root) {
        if(root == nullptr){
            return;
        }

        inOrder(root->left);
        cout << root->data <<" ";
        inOrder(root->right);

 }
```

Add Node (función inserter en árbol binario)

Fácil ir al github, copiar código BST y adaptar la entrada.

Opción 2:

```
Node * insert(Node * root, int data) {
        if(root == nullptr) {
            return new Node(data);
        }

        Node* currNode;
        if(data < root->data) {

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

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

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

        return root;
    }
```

Is BST? (Validar si es un árbol binario)

```
vector<int>valuesBST;
void inOrdenCheckBST(Node *root){
    if(root == nullptr){
        return;
    }

    inOrdenCheckBST(root->left);
    valuesBST.push_back(root->data);
    inOrdenCheckBST(root->right);
}

bool checkBST(Node* root) {
    inOrdenCheckBST(root);
    for(int i = 1; i < valuesBST.size(); i++){
        if(valuesBST[i-1] >= valuesBST[i]){
            return false;
        }
    }

    return true;
}
```

TreeTraversal4 (BFS en árboles)

```
void levelOrder(Node * root) {
    queue<Node*>nodesPrint;
    nodesPrint.push(root);

    while(!nodesPrint.empty()){
        Node * currNode = nodesPrint.front();
        cout << currNode->data << " ";

        if(currNode->left != nullptr){
            nodesPrint.push(currNode->left);
        }

        if(currNode->right != nullptr){
            nodesPrint.push(currNode->right);
        }

        nodesPrint.pop();
    }


}
```

Maximum Tree

```
long long n; cin >> n;
vector<long long> valores;

for(long long i = 0; i < n; i++){
    long long aux; cin >> aux;
    valores.push_back(aux);
}

sort(valores.rbegin(), valores.rend());

long long mul = valores[0];
long long suma = valores[0];

for(int i = 1; i < valores.size(); i++){
    mul = mul * valores[i];
    suma += mul;
}

cout<< suma+1 <<"\n";
```

Another String Problem (Implementación)

```cpp
string cad;
getline(cin, cad);

bool flag = false;

int i = 0;
while(i <= cad.size()){
    if(cad[i]=='H' || cad[i]=='Q' || cad[i]=='9'){
        flag = true; break;
    }
    i++;
}
  cout << (flag? "YES\n": "NO\n");
```

String Solitaire (Implementación)

```cpp
void solve(){
string s; cin >> s;
  vector<int>hash(3,0);

  for(int i = 0; i < s.size(); i++){
    if(s[i]=='A'){
        hash[0]++;
    }
    if(s[i]=='B'){
        hash[1]++;
    }

    if(s[i]=='C'){
        hash[2]++;
    }
  }

  if(hash[1] == 0){
     cout <<"NO\n";
     return;
  }

  int posible_res = 0;

  if(hash[1] >= hash[0] && hash[0] != 0){
    posible_res = hash[1] - hash[0];
    hash[1] = posible_res;
    hash[0] = 0;
  }

  if(hash[1] >= hash[2] && hash[2] != 0){
    posible_res = hash[1] - hash[2];
    hash[1] = posible_res;
    hash[2] = 0;
  }

    if(posible_res == 0){
    for(int i = 0; i < hash.size(); i++){
        if(hash[i] != 0){
        cout <<"NO\n";
        return;
      }
    }
    cout <<"YES\n";
    return;
    }

  cout <<"NO\n";
  return;
}
```

Duplicates? (Implementación)

```cpp
int n;
cin >> n;

map<int, bool>mapa;
vector<int> numeros(n);

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

int numero_de_elementos_unicos = 0;

for(int i = n-1; i >= 0; --i){
    if(mapa[numeros[i]]){
        numeros[i] = -1;
    }else{
        mapa[numeros[i]] = 1;
        numero_de_elementos_unicos++;
    }
}

cout << numero_de_elementos_unicos<<"\n";
for(int i = 0; i < n; i++){
    if(numeros[i] != -1){
        cout << numeros[i] <<" ";
    }
}
```

Links de los problemas

- https://www.hackerrank.com/challenges/tree-height-of-a-binary-tree/problem
- https://www.hackerrank.com/challenges/tree-preorder-traversal/problem
- https://www.hackerrank.com/challenges/tree-postorder-traversal/problem
- https://www.hackerrank.com/challenges/tree-inorder-traversal/problem
- https://www.hackerrank.com/challenges/even-tree/problem
- https://www.hackerrank.com/challenges/binary-search-tree-insertion/problem
- https://www.hackerrank.com/challenges/tree-level-order-traversal/problem
- https://codeforces.com/gym/101466/problem/B
- https://codeforces.com/problemset/problem/133/A
- https://codeforces.com/problemset/problem/1579/A
- https://codeforces.com/problemset/problem/978/A

Contacto:

gomezerms@gmail.com

@elissabjg

@elissabj