// NOTE : Copy paste the whole code in the editor.

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define endline cout << endl;

// typedef cout << endl;

class TreeNode

{

public:

TreeNode \*left;

TreeNode \*right;

int data;

TreeNode(int data)

{

this->data = data;

this->left = NULL;

this->right = NULL;

}

};

class Node

{

public:

Node \*next;

int data;

Node()

{

this->next = NULL;

}

Node(int data)

{

this->data = data;

this->next = NULL;

}

};

class Solutions

{

public:

void preorder(TreeNode \*root)

{

if (root == NULL)

return;

cout << root->data << " -> ";

preorder(root->left);

preorder(root->right);

}

void printNodes(Node \*head)

{

while (head != NULL)

{

cout << head->data << " -> ";

head = head->next;

}

cout << "NULL" << endl;

}

void printArray(vector<int> a) {

for(int i : a) cout << i << " ";

}

// QUESTION 1:

Node \*reverseList(Node \*head)

{

if (head == NULL)

{

cout << "head is NULL";

return head;

}

Node \*prev = NULL, \*curr = head, \*fwd = NULL;

while (curr != NULL)

{

fwd = curr->next;

curr->next = prev;

prev = curr;

curr = fwd;

}

return prev;

}

// QUESTION 2:

int findLongestSubstring(string s)

{

int ans = 0;

unordered\_map<char, int> mp;

int length = 0;

int left = 0, right = 0;

while (right < s.size())

{

if (mp[s[right]] and mp[s[right]] >= left)

{

left = mp[s[right]] + 1;

length = 0;

}

else

{

length++;

ans = max(ans, length);

}

mp[s[right]] = right;

right++;

}

return ans;

}

// QUESTION 3:

int maxPathSum(TreeNode \*root)

{

int maxi = INT\_MIN;

solve(root, maxi);

return maxi;

}

int solve(TreeNode \*node, int &maxi)

{

if (node == NULL)

return 0;

int leftSum = max(0, solve(node->left, maxi));

int rightSum = max(0, solve(node->right, maxi));

maxi = max(maxi, leftSum + rightSum + node->data);

return max(leftSum, rightSum) + node->data;

}

// QUESTION 4:

string serializeBST(TreeNode \*root)

{

if (!root)

return "";

queue<TreeNode \*> q;

q.push(root);

string result = "";

while (!q.empty())

{

TreeNode \*temp = q.front();

q.pop();

if (temp != NULL)

{

result.append(to\_string(temp->data));

result.append(",");

q.push(temp->left);

q.push(temp->right);

}

else

{

result.append("#,");

}

}

result.pop\_back();

return result;

}

TreeNode \*deserializeBST(string str)

{

if (str.size() == 0)

return NULL;

stringstream s(str);

string st;

getline(s, st, ',');

TreeNode \*root = new TreeNode(stoi(str));

queue<TreeNode \*> q;

q.push(root);

while (!q.empty())

{

TreeNode \*node = q.front();

q.pop();

// for left node

if(getline(s, st, ',')) {

if (st == "#")

node->left = NULL;

else

{

TreeNode \*leftNode = new TreeNode(stoi(st));

node->left = leftNode;

q.push(leftNode);

}

}

// for right node

if(getline(s, st, ',')){

if (st == "#")

node->right = NULL;

else

{

TreeNode \*rightNode = new TreeNode(stoi(st));

node->right = rightNode;

q.push(rightNode);

}

}

}

return root;

}

// QUESTION 5:

vector<int> solve(vector<int> &a, int n, int k, int b[])

{

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

{

a[(i + k) % n] = b[(i % n) % n];

}

return a;

}

void rotate(vector<int> &nums, int k)

{

int b[nums.size()];

for (int i = 0; i < nums.size(); i++)

{

b[i] = nums[i];

}

solve(nums, nums.size(), k, b);

}

// QUESTION 6 :

int factorial(int n)

{

if (n <= 1)

return 1;

return n \* factorial(n - 1);

}

// QUESTION 7 :

int sumOfDigits(int num)

{

int sum = 0;

while (num > 0)

{

sum += num % 10;

num /= 10;

}

return sum;

}

// QUESTION 8 :

int gcd(int a, int b)

{

while (b != 0)

{

int temp = b;

b = a % b;

a = temp;

}

return a;

}

// QUESTION 9 :

int maxDifference(vector<int> &nums)

{

if (nums.size() < 2)

return 0;

int minElement = nums[0];

int maxDiff = 0;

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

{

maxDiff = max(maxDiff, nums[i] - minElement);

minElement = min(minElement, nums[i]);

}

return maxDiff;

}

// QUESTION 10 :

bool isAlphabetic(string &s)

{

for (char c : s)

{

if (!isalpha(c))

return false;

}

return true;

}

};

int main()

{

Solutions sol;

Node \*head = new Node(10);

Node \*num1 = new Node(20);

Node \*num2 = new Node(30);

Node \*num3 = new Node(40);

Node \*num4 = new Node(50);

head->next = num1;

num1->next = num2;

num2->next = num3;

num3->next = num4;

// num4->next = NULL;

Node \*curr = head;

cout << "SOLUTION 1 : " << endl;

cout << "Before Reverse : " << endl;

sol.printNodes(curr);

curr = head;

Node \*newhead = sol.reverseList(curr);

cout << "After Reverse : " << endl;

sol.printNodes(newhead);

cout << endl;

string ques2 = "cadbzabcd";

cout << "SOLUTION 2 : " << endl;

cout << "The longest substring without repeating character in "<< ques2 << " is : " << sol.findLongestSubstring(ques2) << endl;

cout << endl;

TreeNode \*root = new TreeNode(10);

TreeNode \*node1 = new TreeNode(20);

TreeNode \*node2 = new TreeNode(30);

TreeNode \*node3 = new TreeNode(40);

TreeNode \*node4 = new TreeNode(50);

root->left = node1;

root->right = node2;

node2->left = node3;

node2->right = node4;

cout << "SOLUTION 3 : " << endl;

cout <<"Max Path sum : " << sol.maxPathSum(root) << endl;

cout << endl;

cout << "SOLUTION 4 : " << endl;

cout << "Original tree : ";

sol.preorder(root);

cout << endl;

string serializeString = sol.serializeBST(root);

cout <<"Serialized Tree : "<< serializeString << endl;

TreeNode \*newNode = sol.deserializeBST(serializeString);

cout <<"Deserialized Tree : ";

sol.preorder(newNode);

endline

cout << endl;

cout << "SOLUTION 5 : " << endl;

vector<int> ques5 = {1,2,3,4,5};

cout << "Original Array : " << endl;

sol.printArray(ques5);

sol.rotate(ques5, 3);

cout << endl;

cout << "After k roatation : " << endl;

sol.printArray(ques5);

cout << endl;

cout << endl;

cout << "SOLUTION 6 : " << endl;

cout << "Factorial of 5 is : " << sol.factorial(5) << endl;

cout << endl;

cout << "SOLUTION 7 : " << endl;

cout << "Sum of digits of nums 12345 is : " << sol.sumOfDigits(12345) << endl;

cout << endl;

cout << "SOLUTION 8 : " << endl;

cout << "GCD of 12, 18 is : " << sol.gcd(12,18) << endl;

cout << endl;

cout << "SOLUTION 9 : " << endl;

vector<int> ques9 = {3,5,6,9,2,8};

cout << "The max difference is : "<< sol.maxDifference(ques9) << endl;

cout << endl;

cout << "SOLUTION 10 : " << endl;

string ques10 = "am1t";

string ans10 = (sol.isAlphabetic(ques10)) ? "Yes, It has all alphabets" : "No, It contains alphanumeric characters.";

cout << "Is " << ques10 << " Alplhabet ?? " << ans10 << endl;

}

**SOLUTIONS**



















