**Звіт**

**Зайцев А. М., гр. КІ-23у-1**

**Завдання: «Дерева».**

**Варіант №2**

1. Користувач вводить дані для створення бінарного дерева пошуку та два цілих числа, які будуть визначати межі діапазону . Визначте суму значень усіх вузлів із значенням у діапазоні [low, high].

Приклад:

Input: root = [10,5,15,3,7, null,18], low = 7, high = 15

Output: 32

Explanation: Nodes 7, 10, and 15 are in the range [7, 15]. 7 + 10 + 15 = 32.

1. Однією з класичних задач на структури даних є порядок обходу вершин бінарного дерева. Існує три стандартні варіанти обходу:

Прямий: відвідується корінь, ліве піддерево, праве піддерево;

Центрований: відвідується ліве піддерево, корінь, праве піддерево;

Обернений: відвідується ліве піддерево, праве піддерево, корінь;

При прямому, центрованому та оберненому обході ми отримаємо відповідно ABCDEF, CBAEDF та CBEFDA. В задачі необхідно знайти послідовність вершин при оберненому обході, якщо відомі прямий та центрований.

Вхідні дані

Перший рядок містить кількість тестів c (c ≤ 2000). Кожний наступний рядок є окремим тестом та містить кількість вершин у бінарному дереві n (1 ≤ n ≤ 52) та два рядки S[1] та S[2], що містять відповідно прямий та центрований обхід дерева. Вершини дерева пронумеровані різними символами з множин a..z та A..Z. Значення n, S[1] та S[2] розділені пропуском.

Вихідні дані

Для кожного тесту вивести послідовність вершин при оберненому обході дерева.

Приклади

Вхідні дані #1

3 Відповідь #1

3 xYz Yxz Yzx

3 abc cba cba

6 ABCDEF CBAEDF CBEFDA

Код програми написаний на мові програмування **С#**.

1. ***Лістинг програми:***internal class Program

{

static void Main(string[] args)

{

// Генеруємо дерево з випадковою кількістю вузлів

Console.WriteLine("Enter the number of nodes for the tree:");

int nodeCount = int.Parse(Console.ReadLine());

TreeNode root = GenerateRandomBST(nodeCount, 1, 100); // Значення вузлів у діапазоні [1, 100]

Console.WriteLine("\nGenerated Binary Search Tree:");

PrintTree(root);

// Введення меж діапазону

Console.WriteLine("\nEnter low value:");

int low = int.Parse(Console.ReadLine());

Console.WriteLine("Enter high value:");

int high = int.Parse(Console.ReadLine());

// Обчислення суми значень вузлів у діапазоні

int result = RangeSumBST(root, low, high);

Console.WriteLine($"\nSum of nodes in range [{low}, {high}]: {result}");

}

static Random random = new Random();

// Генерує рандомне дерево пошуку

static TreeNode GenerateRandomBST(int nodeCount, int minValue, int maxValue)

{

if (nodeCount == 0) return null;

if (minValue > maxValue)

{

int abs = minValue;

minValue = maxValue;

maxValue = abs+5;

};

int rootValue = random.Next(minValue, maxValue + 1);

TreeNode root = new TreeNode(rootValue);

int leftCount = random.Next(0, nodeCount); // Кількість вузлів у лівому піддереві

int rightCount = nodeCount - 1 - leftCount;

root.Left = GenerateRandomBST(leftCount, minValue, rootValue - 1); // Ліве піддерево

root.Right = GenerateRandomBST(rightCount, rootValue +1, maxValue); // Праве піддерево

return root;

}

// Виводить дерево у вигляді ієрархії

static void PrintTree(TreeNode root, string indent = "", bool isLeft = true)

{

if (root != null)

{

Console.WriteLine($"{indent}{(isLeft ? "├──" : "└──")}{root.Val}");

PrintTree(root.Left, indent + (isLeft ? "│ " : " "), true);

PrintTree(root.Right, indent + (isLeft ? "│ " : " "), false);

}

}

static int RangeSumBST(TreeNode root, int low, int high)

{

if (root == null)

return 0;

int sum = 0;

if (root.Val >= low && root.Val <= high)

{

sum += root.Val;

}

if (root.Val > low)

{

sum += RangeSumBST(root.Left, low, high);

}

if (root.Val < high)

{

sum += RangeSumBST(root.Right, low, high);

}

return sum;

}

}

class TreeNode

{

public int Val;

public TreeNode Left;

public TreeNode Right;

public TreeNode(int val = 0, TreeNode left = null, TreeNode right = null)

{

Val = val;

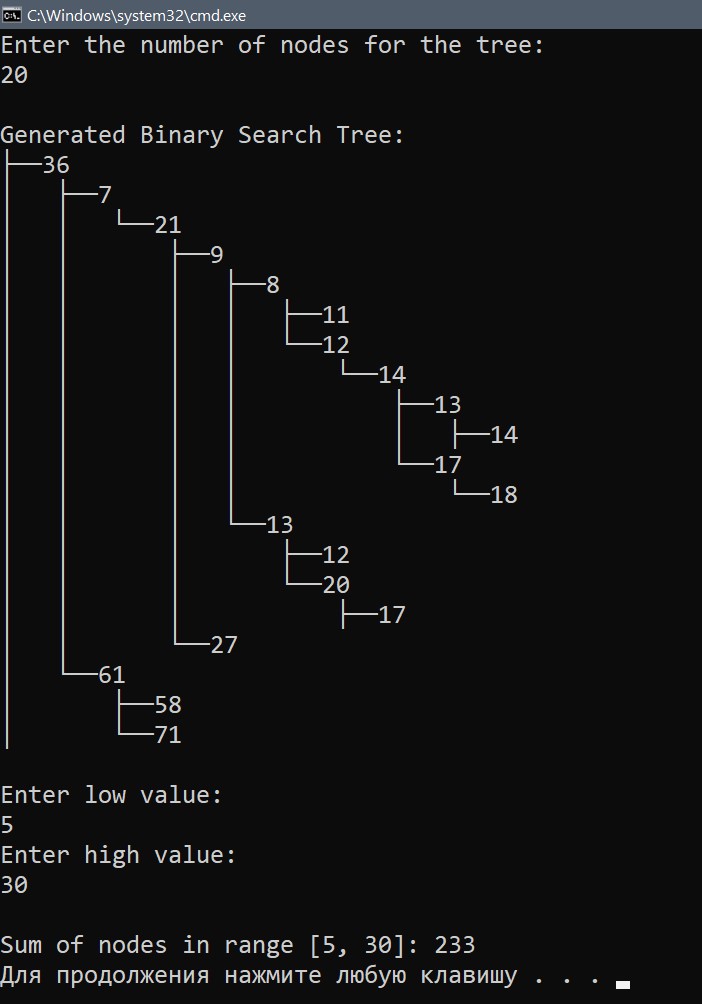
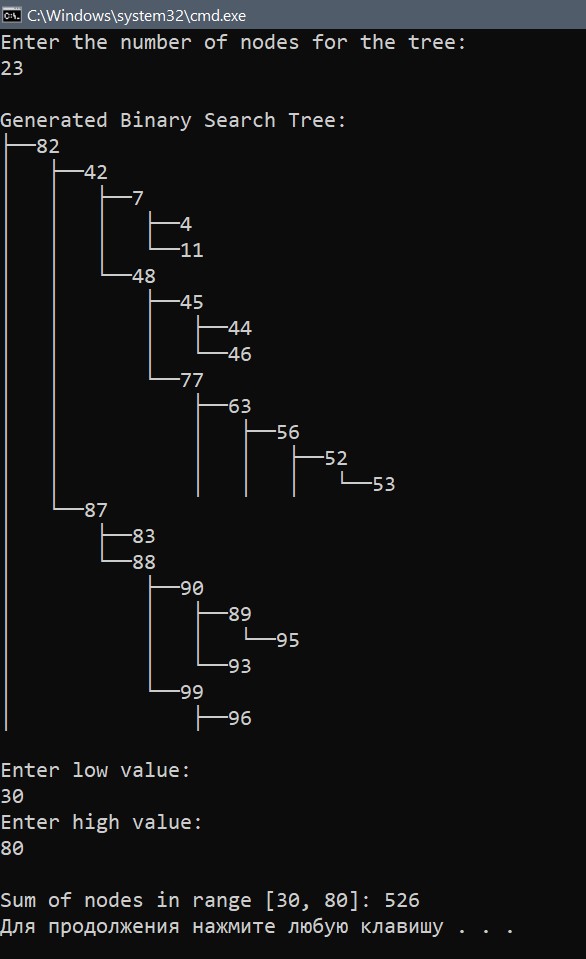
Left = left;

Right = right;

}

}

***Результати роботи програми:***

1. ***Лістинг програми:***internal class Program

{

static void Main(string[] args)

{

Console.Write("Кiлькiсть тестiв: ");

int c = int.Parse(Console.ReadLine()); // Кількість тестів

Console.WriteLine("----------------------------------------");

for (int t = 0; t < c; t++)

{

// Зчитуємо дані для кожного тесту

Console.WriteLine($"Даннi для {t+1} тесту:");

string[] input = Console.ReadLine().Split();

int n = int.Parse(input[0]); // Кількість вершин

string preorder = input[1]; // Прямий обхід

string inorder = input[2]; // Центрований обхід

// Відновлюємо обернений обхід

string postorder = GetPostorder(preorder, inorder);

Console.WriteLine($"Обернений обхiд: {postorder}");

Console.WriteLine("----------------------------------------");

}

}

static string GetPostorder(string preorder, string inorder)

{

if (string.IsNullOrEmpty(preorder) || string.IsNullOrEmpty(inorder))

return "";

// Кореневий вузол дерева — це перший символ у прямому обході

char root = preorder[0];

// Знаходимо позицію кореня в центрованому обході

int rootIndex = inorder.IndexOf(root);

// Ліва частина центрованого обходу — це ліве піддерево

string leftInorder = inorder.Substring(0, rootIndex);

// Права частина центрованого обходу — це праве піддерево

string rightInorder = inorder.Substring(rootIndex + 1);

// Відповідні частини прямого обходу для лівого та правого піддерев

string leftPreorder = preorder.Substring(1, leftInorder.Length);

string rightPreorder = preorder.Substring(1 + leftInorder.Length);

// Рекурсивно обробляємо ліве і праве піддерева

string leftPostorder = GetPostorder(leftPreorder, leftInorder);

string rightPostorder = GetPostorder(rightPreorder, rightInorder);

// Формуємо обернений обхід: ліва частина + права частина + корінь

return leftPostorder + rightPostorder + root;

}

}

***Результати роботи програми:***

