Лабораторная работа №4

ПОЛНЫЙ ПЕРЕБОР

Задание: решить методом полного p-ичного перебора следующие задачи:

Вариант 6: строки.

Текстовый файл состоит из двух строк, длинами не более N (N <= 20). Выяснить, можно ли получить вторую строку из первой путем вычеркивания некоторых символов. Прописные и строчные буквы не различаются, т.е. строки abcd и AbcD считаются одинаковыми. Программа должна находить номера вычеркнутых символов и выводить в текстовый файл все возможные варианты.

ХОД РАБОТЫ

1. Определяется структура Node, которая содержит целочисленное значение data и указатели на левого и правого потомков узла. Эта структура используется для представления каждого узла в бинарном дереве.
2. Функция add (int x, Node\*&node):

Если переданный узел node равен NULL, то создается новый узел с значением x и указателями на потомков, и узел node указывает на этот новый узел.

Иначе, если значение x меньше значения текущего узла, функция вызывается рекурсивно для левого потомка, иначе для правого потомка.

1. Функция buildRandomTree (Node\*& root, int size):

Генерируется случайное значение для каждого узла от 0 до 99 и вызывается функция add, чтобы добавить это значение в дерево.

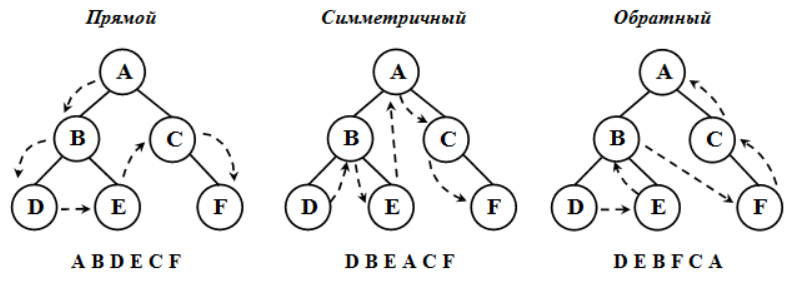
В итоге, строится случайное бинарное дерево заданного размера size, используя случайно сгенерированные значения.

1. Три функции обхода дерева:

Прямой обход (straightPrintTree): выводит значение текущего узла, затем вызывает себя рекурсивно для левого поддерева, затем для правого.

Обратный обход (reversePrintTree): вызывает себя рекурсивно для левого поддерева, затем для правого, и после этого выводит значение текущего узла.

Симметричный обход (symmetricalPrintTree) - вызывает себя рекурсивно для левого поддерева, затем выводит значение текущего узла, затем вызывается для правого поддерева.



1. Функция countingLeaves(Node\* node):

Рекурсивно подсчитывает количество листьев в дереве.

Если узел node является листом (не имеет левого и правого потомков), функция возвращает 1.

Для каждого узла считается количество листьев в левом и правом поддеревьях, и результат слагается.

1. Функция printLeaves(Node\* node):

Рекурсивно проверяет, является ли узел node листом в дереве и, соответственно, печатает его. Выполняется для каждого узла.

1. Внутри функции main():

Инициализируется указатель root дерева.

Генерируется случайное количество элементов для дерева.

Строится случайное дерево с заданным количеством элементов.

Выводятся элементы дерева в прямом, обратном и симметричном обходах.

Вызывается функция подсчёта количества листьев в дереве и выводится на экран.

Вызывается функция печати листьев.

Результат работы программы представлен в соответствии с рисунками 1-2.

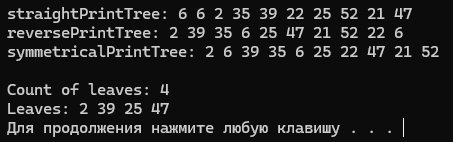


Рис. 1. Результаты работы программы.

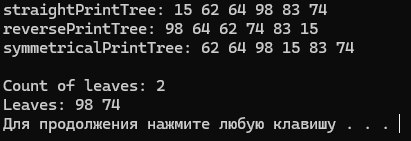


Рис. 2. Результаты работы программы.

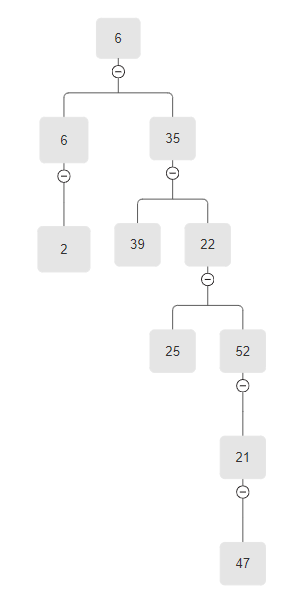
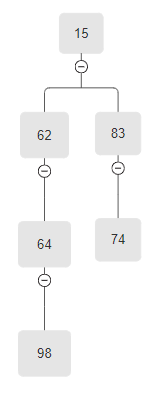
 

Рис. 3. Бинарные деревья по выводу программы

Текст программы:

#include<iostream>

using namespace std;

// В этой структуре определены три поля: data для хранения значения узла и указатели left и right для связи с левым и правым потомком.

struct Node {

int data;

Node\* left, \* right;

};

// Функция add отвечает за добавление нового узла в бинарное дерево.

// Если переданный узел node равен NULL, создается новый узел с данными x и указателями на потомков. Иначе функция случайным образом выбирает, добавлять новый узел в левую или правую ветвь, и рекурсивно вызывает себя для соответствующего поддерева.

void add(int x, Node\*& node) {

if (node == NULL) {

node = new Node;

node->data = x;

node->left = NULL;

node->right = NULL;

} else {

int rand\_branch = rand() % 2;

if (rand\_branch)

add(x, node->right);

else

add(x, node->left);

}

}

// В данной функции генерируется случайное бинарное дерево заданного размера size.

// Для каждого узла в цикле генерируется случайное значение randomValue от 0 до 100, и затем это значение добавляется в дерево с помощью вызова функции add.

void buildRandomTree(Node\*& root, int size) {

srand(time(0));

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

int randomValue = rand() % 100;

add(randomValue, root);

}

}

// Прямой обход:

// Функция straightPrintTree выводит значения узлов дерева в порядке прямого обхода:

// сначала текущий узел, затем левое поддерево и правое поддерево.

// Рекурсивно обрабатываются левое и правое поддеревья.

void straightPrintTree(Node\* node) {

if (node != NULL) {

cout << node->data << " ";

straightPrintTree(node->left);

straightPrintTree(node->right);

}

}

// Обратный обход:

// В функции reversePrintTree значения узлов выводятся в обратном порядке обхода:

// сначала левое поддерево, затем правое поддерево и текущий узел.

// Рекурсивно обрабатываются левое и правое поддеревья.

void reversePrintTree(Node\* node) {

if (node != 0) {

reversePrintTree(node->left);

reversePrintTree(node->right);

cout << node->data << " ";

}

}

// Cимметричный обход:

// Функция symmetricalPrintTree осуществляет симметричный обход дерева, выводя значения узлов в порядке:

// левое поддерево, текущий узел, правое поддерево.

// Рекурсивно обрабатываются левое и правое поддеревья.

void symmetricalPrintTree(Node\* node)

{

if (node != 0) {

symmetricalPrintTree(node->left);

cout << node->data << " ";

symmetricalPrintTree(node->right);

}

}

// Количество листьев в дереве:

// Если узел равен NULL, возвращается 0.

// Если узел является листовым (не имеет потомков), возвращается 1.

// Иначе рекурсивно вызывается для левого и правого поддерева, и возвращается их сумма.

int countLeaves(Node\* node) {

if (node == NULL) return 0;

else if (node->left == NULL && node->right == NULL)

return 1;

return countLeaves(node->left) + countLeaves(node->right);

}

// Вывод листьев в дереве:

// Если узел является листовым (не имеет потомков), выводится значение узла.

// Иначе рекурсивно вызывается для левого и правого поддерева

void printLeaves(Node\* node) {

if (node != NULL) {

if (node->left == NULL && node->right == NULL)

cout << node->data << " ";

else {

printLeaves(node->left);

printLeaves(node->right);

}

}

}

int main()

{

srand(time(0));

Node\* root = NULL;

int elements = rand() % 10 + 1;

buildRandomTree(root, elements);

// Прямой обход

cout << "straightPrintTree: ";

straightPrintTree(root);

cout << endl;

// Обратный обход

cout << "reversePrintTree: ";

reversePrintTree(root);

cout << endl;

// Симметричный обход

cout << "symmetricalPrintTree: ";

symmetricalPrintTree(root);

cout << endl << endl;

// Функция количества листьев в дереве:

int leaves = countLeaves(root);

cout << "Count of leaves: " << leaves << endl;

// Функция Вывести листья в дереве:

cout << "Leaves: ";

printLeaves(root);

cout << endl;

system("pause");

}

ВЫВОД

В ходе выполнения лабораторной работы мы научились формировать бинарные деревья и выводить элементы дерева тремя видами обхода.