**Лабоpатоpная pабота №2**

**Бинарные деревья**

Написать функцию формирования бинарного дерева, состоящего из целых чисел. Для представления дерева использовать динамические структуры данных. Количество элементов дерева, а также его вид задаются случайным образом. Произвести вывод элементов дерева тремя видами обхода. Используя информацию о выведенном дереве изобразить структуру одного из построенных деревьев в отчете. Выполнить по вариантам следующие задания:

Вариант 12

12. Написать рекурсивную функцию, которая находит элемент дерева, встречающийся чаще всего.

Код:

# include <iostream>

# include <fstream>

# include <string>

# include <time.h>

#include <set>

using namespace std;

struct Node {

int data;

Node\* left, \* right;

};

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

if (node == NULL) {

node = new Node;

node->data = x;

node->left = NULL;

node->right = NULL;

}

else {

int b = rand() % 2;

if (b == 0) add(x, node->left);

else add(x, node->right);

}

}

void createTree(Node\*& root) {

int capacity = 15;

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

int value = -(rand() % 20);

add(value, root);

}

}

// Обход дерева прямым способом

void prePrint(Node\* node) {

if (node != NULL) {

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

prePrint(node->left);

prePrint(node->right);

}

}

// Обход дерева симметричным способом

void inPrint(Node\* node) {

if (node != NULL) {

inPrint(node->left);

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

inPrint(node->right);

}

}

// Обход дерева обратным способом

void postPrint(Node\* node) {

if (node != NULL) {

postPrint(node->left);

postPrint(node->right);

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

}

}

void FindMost(const Node\* root, int& maxCount, int& mostFrequent, int& currentCount, multiset <int>& count, bool& multipleMaxFreq)

{

if (!root)

{

return;

}

FindMost(root->left, maxCount, mostFrequent, currentCount, count, multipleMaxFreq);

currentCount = root->data;

count.insert(currentCount);

int x = count.count(currentCount);

if (x > maxCount)

{

maxCount = x;

mostFrequent = currentCount;

multipleMaxFreq = false;

}

else if (x==maxCount)

multipleMaxFreq = true;

FindMost(root->right, maxCount, mostFrequent, currentCount, count, multipleMaxFreq);

}

int Search(Node\* root) {

int currentCount = 0;

int maxCount = 0;

int mostFrequent = -1;

bool multipleMaxFreq = false;

multiset <int> count;

FindMost(root, maxCount, mostFrequent, currentCount, count, multipleMaxFreq) ;

if (multipleMaxFreq) {

return -1;

}

return mostFrequent;

}

int main() {

srand(time(NULL));

Node\* root = NULL;

createTree(root);

cout << "prePrint" << endl;

prePrint(root);

cout << endl;

cout << "inPrint" << endl;

inPrint(root);

cout << endl;

cout << "postPrint" << endl;

postPrint(root);

cout << endl;

int mostFrequent = Search(root);

if (mostFrequent == -1) {

cout << "No single most frequently occurring element." << endl;

}

else {

cout << "The frequently occurring element is: " << mostFrequent << endl;

}

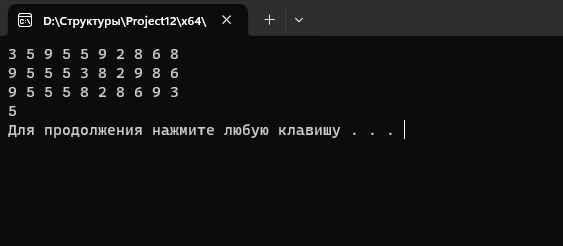
system("pause");

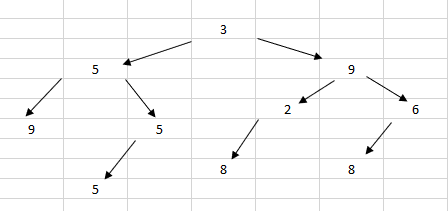
return 0;

}

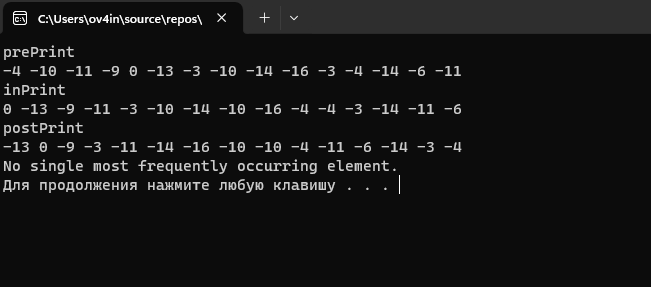
Тесты:

1)

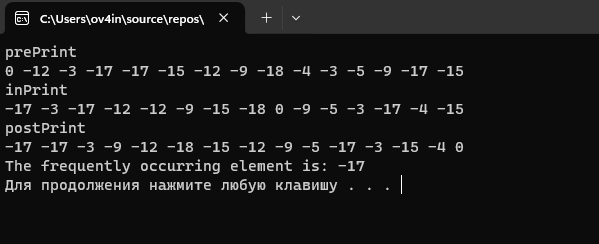




2)



3)



Алгоритм:

1. Определение структуры **Node** и функций для бинарного дерева:
   * Создание структуры **Node**, содержащей данные и указатели на левый и правый узлы.
   * Определение функции **add**, добавляющей новый узел в левое или правое поддерево с вероятностью 50%.
   * Определение функции **Search,** вызывает функцию **FindMost**, чтобы найти наиболее часто встречающийся элемент в дереве. Если таких элементов несколько, возвращается **-1**. Иначе возвращается значение наиболее часто встречающегося элемента.
2. Объявление корня бинарного дерева **root.**
3. Инициализация дерева с помощью функции **createTree:**
   * Создание случайного бинарного дерева с ограниченной вместимостью.
   * Генерация случайных значений и добавление их в дерево.
4. Вывод элементов дерева прямым обходом с помощью функции **prePrint:**
   * Рекурсивный обход дерева в прямом порядке (узел, левое поддерево, правое поддерево).
   * Вывод значений узлов в порядке их посещения.
5. Вывод элементов дерева симметричным обходом с помощью функции **inPrint:**
   * Рекурсивный симметричный обход дерева (левое поддерево, узел, правое поддерево).
   * Вывод значений узлов в порядке их посещения.
6. Вывод элементов дерева обратным обходом с помощью функции **postPrint:**
   * Рекурсивный обратный обход дерева (левое поддерево, правое поддерево, узел).
   * Вывод значений узлов в порядке их посещения.
7. Поиск наиболее часто встречающегося элемента в дереве с помощью функции **Search:**
   * Вызов функции **FindMost**, находящей узел с наибольшей частотой встречаемости.
   * Возврат значения наиболее часто встречающегося элемента или индикатора, если таких элементов несколько.

Алгоритм функции **FindMost:**

1. **Рекурсивный проход по дереву:**
   * Функция **FindMost** работает рекурсивно, начиная с корня дерева и спускаясь вниз по каждой ветви дерева (путем вызова самой себя для левого и правого поддеревьев).
2. **Обход узлов дерева:**
   * Каждый узел посещается в процессе обхода. Сначала функция переходит к левому поддереву, затем обрабатывает текущий узел, а затем переходит к правому поддереву.
3. **Обновление статистики:**
   * При обработке каждого узла:
     + **currentCount** получает значение текущего узла (**root->data**).
     + Значение **currentCount** добавляется в **multiset** **count**, чтобы подсчитать частоту его встречаемости в дереве.
     + Проверяется количество встреч **x** для **currentCount** в **multiset**.
     + Если **x** больше, чем текущее максимальное количество встреч **maxCount**:
       - Обновляются переменные **maxCount** (максимальное количество встреч) и **mostFrequent** (значение наиболее часто встречающегося элемента) значениями **x** и **currentCount**.
       - Переменная **multipleMaxFreq** устанавливается в **false**, так как пока обнаружен только один элемент с максимальной частотой.
     + Если **x** равно **maxCount**:
       - Переменная **multipleMaxFreq** устанавливается в **true**, что указывает на то, что есть несколько элементов с одинаковой максимальной частотой.
4. **Возврат результата:**
   * Функция завершает свою работу, обойдя все узлы дерева. Результаты, такие как наиболее часто встречающийся элемент и информация о наличии нескольких элементов с максимальной частотой, передаются через параметры функции.