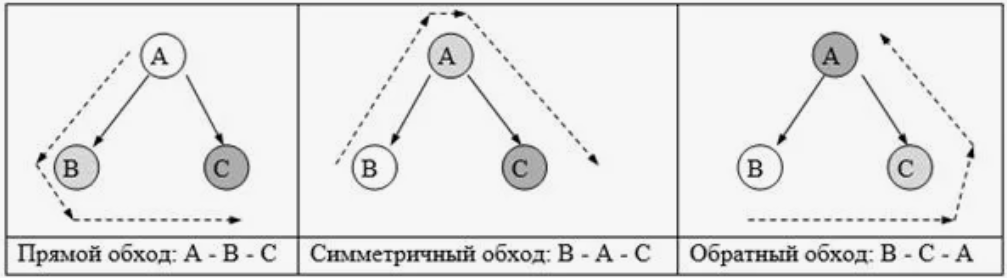
**Лабоpатоpная pабота №2**

**Бинарные деревья**

**11 вариант**

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:

Написать рекурсивную функцию, которая считает количество внутренних вершин дерева.



Алгоритм:

Дерево – реализована при помощи структуры Node, полями которой являются: data, left, right. data – целочисленная переменная, означающая вес вершин. left – указатель на узел, являющийся левым потомком. right – указатель на узел, являющийся правым потомком.

)Создание корня бинарного дерева

Добавление значения в корень дерева при помощи метода add:

если узел пуст, то создается новый узел, его полю data присваивается вес новой вершины.

)Создание бинарного дерева со случайным количеством элементов и структурой при помощи метода maketree:

с шансом 50% создается, или не создается левый потомок с рандомным весом методом add, метод maketree вызывается к этому левому потомку.

с шансом 50% создается, или не создается правый потомок с рандомным весом методом add, метод maketree вызывается к этому правому потомку.

)Функция countInternalNodes :

Если элемент не равен нулю и он является родителем, то функция возвращает 1 + countInternalNodes (узел-потомок слева) + countInternalNodes (узел-потомок справа).

Иначе возвращается 0.

3)Вывод на экран трех обходов дерева и количества внутренних вершин дерева.

#include <iostream>

#include <time.h>

using namespace std;

#include <iostream>

#include <time.h>

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;

}

}

void directbypass(Node\* node) {

if (node != NULL) {

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

directbypass(node->left);

directbypass(node->right);

}

}

void symmetricbypass(Node\* node) {

if (node != NULL) {

symmetricbypass(node->left);

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

symmetricbypass(node->right);

}

}

void reveversebypass(Node\* node) {

if (node != NULL) {

reveversebypass(node->left);

reveversebypass(node->right);

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

}

}

int countInternalNodes(Node\* node) {

if (node != NULL && (node->left != NULL || node->right != NULL)) {

return 1 + countInternalNodes(node->left) + countInternalNodes(node->right);

}

else return 0;

}

void maketree(Node\*& node) {

if (rand() % 2) {

add(rand() % 100, node->left);

maketree(node->left);

}

if (rand() % 2) {

add(rand() % 100, node->right);

maketree(node->right);

}

}

int main() {

srand(time(0));

Node\* root = NULL;

int x = rand() % 100;

add(x, root);

maketree(root);

cout << "direct: "; directbypass(root); cout << endl;

cout << "symmetric: "; symmetricbypass(root); cout << endl;

cout << "reverse: "; reveversebypass(root); cout << endl;

int numInternalNodes = countInternalNodes(root);

cout << numInternalNodes;

cout << endl;

system("pause");

}

