Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»  
(ВлГУ)

Институт информационных технологий и электроники

Кафедра физики и прикладной математики

ЛАБОРАТОРНАЯ РАБОТА №9  
по дисциплине  
«Основы программирования»  
на тему:  
«Нелинейные динамические структуры»

Выполнил:  
ст. гр. ПМИ-123

Рушев А.М.  
  
  
Принял:  
ст. преподаватель   
каф. ФиПМ  
Шишкина М.В.

Владимир,2024

**Цель работы:**

Изучение алгоритмов создания нелинейных динамических структур. Изучение и реализация алгоритмов обхода бинарного дерева.

**Задание:**

Написать функцию добавления элементов в бинарное дерево (предусмотрев проверку на пустоту), реализовать функции симметричного и прямого обхода. При обходе, соответствующий элемент выводить на экран. Информационное поле задать целочисленным. Создать нелинейную динамическую структуру двоичное дерево, добавив в него элементы целочисленного массива. Вызвать поочередно функции обхода дерева.

**Теоретическая часть**

Под нелинейными динамическими структурами данных понимают такой способ организации хранения данных, при котором количество данных в структуре может изменяться во время работы программы и порядок расположения элементов в памяти может не совпадать с порядком их добавления в структуру. При решении ряда задач, например поиска, такой подход к хранению данных эффективнее. К нелинейным динамическим структурам данных относят: графы, различные виды деревьев, мультисписки. В этом разделе в качестве примера нелинейной динамической структуры будут рассмотрены бинарные деревья.

Бинарным деревом называют определённым образом организованную динамическую структуру данных, позволяющую хранить большие объёмы данных с возможностью их быстрого поиска. Далее рассмотрены двоичные деревья поиска. Элементы такой структуры называют узлами (вершинами). На каждый узел имеется только одна ссылка. Узел содержит как минимум одно информационное поле и два поля-указателя на левое и правое поддеревья.

Добавление элементов в такую динамическую структуру подчинено следующему правилу: для каждого узла все ключи его левого поддерева меньше ключа рассматриваемого узла, а ключи правого поддерева больше ключа рассматриваемого узла. Одинаковые ключи не допускаются. Поиск элемента в такой структуре осуществляется гораздо быстрее, чем в линейном списке. При работе с двоичными деревьями применяют следующую терминологию.

Начальный узел принято называть корнем дерева. Узел, не имеющий поддерева, называют листом. Если существует путь от узла а до узла b, то говорят, что узел а – предок узла b, а узел b – потомок узла а.

Самый длинный путь от корня до листа называют высотой дерева. Так как каждое поддерево представляет собой дерево, очевидно, что эта структура рекурсивная, поэтому и действия с ней удобнее реализовывать при помощи рекурсивных функций, передавая в качестве параметра указатель на корень дерева.

Для бинарных деревьев определены следующие операции:

- включение узла в дерево;

- удаление узла из дерева;

- поиск значения по ключу;

- обход дерева, при обходе дерева узлы не удаляются.

Рассмотрим три порядка обхода дерева: обход в симметричном порядке (симметричный обход (inorder – LCR)), обход в прямом порядке (обход сверху, или обход в ширину (preorder – CLR)), обход в обратном порядке (обход в глубину, обратный обход, обход снизу (postorder – LRC)). При обходе в симметричном порядке (LCR) обрабатывается сначала левое поддерево, потом корень, а затем правое поддерево. При обходе в прямом порядке (CLR) обрабатывается сначала корень, потом левое поддерево, а затем правое. При обходе в обратном порядке (LRC) сначала обрабатывается левое поддерево, потом правое и, наконец, корень.

Алгоритмы обхода дерева

Обход в прямом порядке. Каждый узел посещается до того, как были посещены его потомки. Для корня дерева рекурсивно выполняются следующие действия: посетить узел; обойти левое поддерево; обойти правое поддерево. Симметричный обход. Посещаем сначала левое поддерево, потом узел, затем правое поддерево. Для корня дерева рекурсивно выполняются следующие действия: обойти левое поддерево; посетить узел; обойти правое поддерево. Обход в обратном порядке. Узлы посещаются снизу вверх. Для корня дерева рекурсивно выполняются следующие действия: обойти левое поддерево; обойти правое поддерево; посетить узел.

Пример описания элемента динамической структуры двоичное дерево: struct elem {

int data;

elem\* left;

elem\* right;

};

Для организации обращения к дереву необходим указатель на его начало – корень.

elem\* root=NULL;

Операции над деревом реализуют в виде рекурсивных функций, передавая в них в качестве параметра ссылку на корень дерева. Передача этого параметра по значению не позволит изменить структуру за пределами функции. В функцию добавления элемента в дерево также необходимо передать значение информационного поля. Функции работы с деревом не должны возвращать значения, т. е. перед именем функции нужно указать void.

Фрагмент кода добавления нового элемента в дерево:

elem\* tmp=new elem; //захвачена память под новый элемент

tmp->data=a; //заполняем информационное поле

tmp->left= NULL; //записываем в указатели NULL

tmp->right= NULL;

\*root=tmp; //настраиваем корень на новый элемент

**Практическая часть**

#include <iostream>

using namespace std;

struct Tree {

int field; //поле данных

Tree\* left; //левый потомок

Tree\* right; //правый потомок

};

void InitMassiv(int\* arr, int size) {

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

arr[i] = 1 + rand() % 10;

}

}

void PrintMassiv(int\* arr, int size) {

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

cout << arr[i] << "\t";

}

cout << endl;

}

void AddNode(int x, Tree\*& \_tree) {

if (\_tree == NULL) { //проверяем на пустоту, если дерева нет - формируем его

\_tree = new Tree;

\_tree->field = x;

\_tree->left = NULL;

\_tree->right = NULL;

}

else if (x < \_tree->field) { //условие добавления левого потомка

AddNode(x, \_tree->left);

}

else { //условие добавления правого потомка

AddNode(x, \_tree->right);

}

}

//префиксная форма

void TreePrintPrefix(Tree\* \_tree) {

if (\_tree != NULL) { //Пока не встретится пустой узел

cout << \_tree->field << "\t"; //Отображаем корень дерева

TreePrintPrefix(\_tree->left); //Рекурсивная функция для левого поддерева

TreePrintPrefix(\_tree->right);

}

}

//инфиксная форма

void TreePrintInfix(Tree\* \_tree) {

if (\_tree != NULL) { //Пока не встретится пустой узел

TreePrintInfix(\_tree->left); //Рекурсивная функция для левого поддерева

cout << \_tree->field << "\t"; //Отображаем корень дерева

TreePrintInfix(\_tree->right);

}

}

//постфиксная форма

void TreePrintPostfix(Tree\* \_tree) {

if (\_tree != NULL) { //Пока не встретится пустой узел

TreePrintPostfix(\_tree->left); //Рекурсивная функция для левого поддерева

TreePrintPostfix(\_tree->right);

cout << \_tree->field << "\t"; //Отображаем корень дерева

}

}

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Лабораторная работа №9 (Бинарное дерево)" << endl;

const int n = 5;

int arr[n];

Tree\* \_tree = NULL;

InitMassiv(arr, n);

cout << "Исходный массив:\n";

PrintMassiv(arr, n);

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

AddNode(arr[i], \_tree);

}

cout << "Обход дерева в прямом порядке:" << endl;

TreePrintPrefix(\_tree);

cout << "\nОбход дерева в симметричном порядке:" << endl;

TreePrintInfix(\_tree);

cout << "\nОбход дерева в обратном порядке:" << endl;

TreePrintPostfix(\_tree);

cout << endl;

system("pause");

return 0;

}

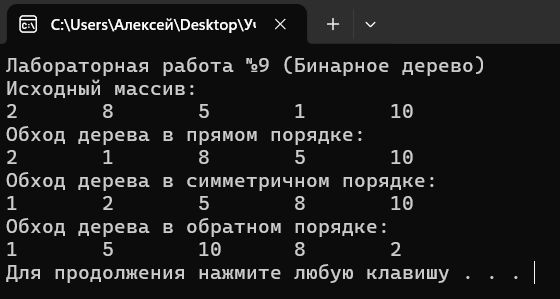


Рисунок 1 – Результат выполнения кода

**Вывод:**

В ходе лабораторной работы были получены навыки организации бинарных деревьев и работы с ними.

Преимущества бинарных деревьев: эффективный поиск информации по ним; упорядоченность обхода; используемость при сортировке.

Недостатки: вероятность несбалансированных деревьев, которые затруднят поиск по ним; низкая производительность в худших сценариях.