БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра программного обеспечения информационных технологий

Факультет ФНиДО

Специальность ПОИТ

Индивидуальная практическая работа №2

по дисциплине «Дискретная математика»

Тема: «Графы»

Вариант 4

Выполнила: Карпеко Н. Г.

Договор № 941 от 20.02.2019 г.

Минск 2020

**Задание**. Вывести номера вершин, у которых количество потомков в левом поддереве не равно количеству потомков в правом поддереве.

**Решение**.

Листинг программы:

**program** Project1;

**uses** crt;

**type**

TData = Integer; //Тип ключа узла дерева.

TPNode = ^TNode; //Тип указателя на узел.

TNode = **record** //Тип узла дерева.

Data: TData; //Ключ узла дерева.

PLeft, PRight: TPNode; //Указатели на левый и правый узел.

**end**;

//--- Процедура освобождения памяти ---------------------

**procedure** TreeFree(**var** aPNode: TPNode);

**begin**

**if** aPNode = nil **then Exit**;

TreeFree(aPNode^.PLeft); //Рекурсивный вызов освобождения памяти в левой ветви.

TreeFree(aPNode^.PRight); //Рекурсивный вызов освобождения памяти в правой ветви.

Dispose(aPNode); //Освобождение памяти, занятой текущего узла.

aPNode := nil; //Обнуление указателя на текущий узел.

**end**;

//--- Добавление узла с ключом aData в двоичное дерево -----

**procedure** AddNode(**var** aPNode: TPNode; **const** aData: TData);

**begin**

**if** aPNode = nil **then** //Вставка узла

**begin**

**New**(aPNode); //Выделяем память для узла

aPNode^.Data := aData; //Записываем в узел значение ключа

aPNode^.PLeft := nil; //Обнуление указателя на левого потомка

aPNode^.PRight := nil; //Обнуление указателя на правого потомка

**end**

**else if** aData <= aPNode^.Data **then** //Поиск места вставки в левой ветви

AddNode(aPNode^.PLeft, aData)

**else if** aData > aPNode^.Data **then** //Поиск места вставки в правой ветви

AddNode(aPNode^.PRight, aData);

**end**;

//--- Процедура распечатки структуры дерева и значений узлов (прямой обход) ---

**procedure** TreeWriteln(**const** aPNode: TPNode; **const** aName: String);

**begin**

**if** aPNode <> nil **then**

**begin**

Writeln('Распечатка текущего узла: ', aName, ': ', aPNode^.Data);

TreeWriteln(aPNode^.PLeft, aName + '-1'); //Рекурсивный вызов для левой ветви

TreeWriteln(aPNode^.PRight, aName + '-2'); //Рекурсивный вызов для правой ветви.

**end**;

**end**;

//--- Процедура проверки условия: высоты лев и прав поддерева =, но количество потомков разное ---

**procedure** TreeCalc(**const** aPNode: TPNode; **const** aName: String; **var** aC, aCnt: Integer);

**var**

CL, CR: Integer;

**begin**

**if** aPNode = nil **then** //Если узла нет

aC := 0 //Количество потомков узла

**else** //Если узел есть

**begin** //Количества потомков для текущего узла =

TreeCalc(aPNode^.PLeft, aName + '-1', CL, aCnt); //Для левой ветви

TreeCalc(aPNode^.PRight, aName + '-2', CR, aCnt); //Для правой ветви

**if** CL <> CR **then**//Если условие выполняется, то

**begin** //учитываем текущий узел в подсчёте и печать сведений о нём

Inc(aCnt);

Writeln(aName, ' (', aPNode^.Data, '): CL = ', CL, ' <> CR = ', CR);

**end**;

//Количество узлов в ветви с корнем aPNode

aC := 1 + CL + CR;

**end**;

**end**;

**VAR**

PTree: TPNode;

Data: TData;

Code, C, Cnt: Integer;

S: String;

**BEGIN**

clrscr;

PTree := nil; //Начальная инициализация дерева.

**repeat**

//Диалог создания дерева.

Writeln(' Добавление узлов в двоичное дерево поиска.');

Writeln(' Для завершения ввода: оставьте пустую строку и нажмите Enter.');

**repeat**

Write(' Введите ключ: '); Readln(S);

**if** S <> '' **then**

**begin**

Val(S, Data, Code);

**if** Code = 0 **then**

AddNode(PTree, Data) //Добавляем узел в дерево

**else**

Writeln(' Неверный ввод! Введите целое число: ');

**end**;

**until** S = '';

Write(' Ввод дерева завершён.');

Writeln(' Созданное дерево:');

TreeWriteln(PTree, '0');

Writeln(' Поиск узлов, для которых количество потомков в левом поддереве не равно количеству потомков в правом поддереве:');

// C := 0; //Необязательное обнуление, т. к., третий параметр в TreeCalc() является выходным.

Cnt := 0; //Обязательное обнуление перед вызовом TreeCalc().

TreeCalc(PTree, '0', C, Cnt);

Writeln(' Всего узлов, для которых выполняется данное условие: ', Cnt);

TreeFree(PTree); //Освобождение памяти

// Writeln('Память, выделенная для дерева - освобождена.');

Writeln('----------------'); Writeln;

Write('Повторить - Enter, выход - любой символ + Enter. ');

Readln(S);

**until** S <> '';

**END**.

Результат работы программы:

