|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Нелинейные структуры данных. Бинарное дерево.»** | |
|  | |
| Выполнил студент группы ИКБО-07-21 | Дамарад Д.В. |
| Принял преподаватель | Скворцова Л.А. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

**СОДЕРЖАНИЕ**

[**ЦЕЛЬ РАБОТЫ** 3](#_Toc118843025)

[**ПОСТАНОВКА ЗАДАЧИ** 3](#_Toc118843026)

[**РЕШЕНИЕ** **Ошибка! Закладка не определена.**](#_Toc118843027)

[**Ответы на вопросы:** **Ошибка! Закладка не определена.**](#_Toc118843028)

[**Функции задания** **Ошибка! Закладка не определена.**](#_Toc118843029)

[**Интерфейс** **Ошибка! Закладка не определена.**](#_Toc118843030)

[**Тестирование** **Ошибка! Закладка не определена.**](#_Toc118843031)

[**ВЫВОД** **Ошибка! Закладка не определена.**](#_Toc118843032)

[**ПОЛНЫЙ КОД ПРОГРАММЫ** **Ошибка! Закладка не определена.**](#_Toc118843033)

# **ЦЕЛЬ РАБОТЫ**

Получение умений и навыков разработки и реализаций операций над структурой данных бинарное дерево.

# **ПОСТАНОВКА ЗАДАЧИ**

Вид дерева: идеально сбалансированное из n узлов (не AVL).

1. Реализовать операции общие для вариантов c 1 по 7
   1. Создать идеально сбалансированное бинарное дерево из n узлов. Структура узла дерева включает: информационная часть узла, указатель на левое и указатель на правое поддерево. Информационная часть узла определена вариантом.
   2. Отобразить дерево на экране, повернув его справа налево.
2. Реализовать операции варианта.
3. Разработать программу, демонстрирующую выполнение всех операций на ваших тестах и тестах преподавателя.
4. Оформить отчет.

|  |  |  |
| --- | --- | --- |
| Вариант | Значение информационной части | Операции варианта |
| 6 | Целое число | Используя рекурсивный алгоритм определить количество уровней в дереве.  Вернуть узел с максимальным значением, обходя дерево в ширину.  Определить максимальное и минимальное значения. |

# **РЕШЕНИЕ**

Бинарное дерево — это иерархическая структура данных, в которой каждый узел имеет значение (оно же является в данном случае и ключом) и ссылки на левого и правого потомка. Узел, находящийся на самом верхнем уровне (не являющийся чьим либо потомком) называется корнем. Узлы, не имеющие потомков (оба потомка которых равны NULL) называются листьями.

В варианте 6 требуется создать идеально сбалансированное бинарное дерево из n узлов.

Бинарное дерево называют идеально сбалансированным, если для каждой его вершины количество вершин в левом и правом поддереве различаются не более чем на 1.

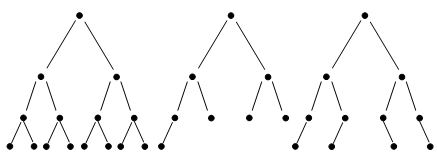


Рисунок 1 - Примеры идеально сбалансированных деревьев

У такого дерева только одна вершина (а именно корень) может находиться на нулевом расстоянии от корня; не более двух вершин могут находиться на расстоянии 1 от корня; не более четырех вершин могут находиться от корня на расстоянии, равном 2 и так далее.

Алгоритм построения идеально сбалансированного дерева при известном числе вершин n лучше всего формулируется с помощью рекурсии.

При этом необходимо лишь учесть, что для достижения минимальной высоты при заданном числе вершин, нужно располагать максимально возможное число вершин на всех уровнях, кроме самого нижнего.

Структура узла дерева включает информационную часть узла, указатель на левое и указатель на правое поддерево. Информационная часть узла определена вариантом.

struct point {

int data = 0;

point\* left = nullptr;

point\* right = nullptr;

};

Функция BuildTree создает идеально сбалансированное бинарное дерево из n узлов, принимает в качестве параметров число элементов дерева, а также текущую структуру point\* p, далее. Создается новый элемент дерева, инициализируются переменные nl и nr, которые при условии - если n не равно 0 – принимают значение n/2 и n-nl-1 соответственно. То есть количество элементов делятся пополам (nl>=nr). Под созданный на предыдущем шаге элемент дерева выделяется память. Рекурсивно заполняется сначала левое поддерево, затем заполняется правое поддерево. Это происходит, пока мы не заполним все n элементов дерева. Изначально переданная структура ‘p’ в функцию принимает значение созданной и измененной структуры ‘r’. Структура ‘p’ возвращается.

point\* BuildTree(int n, point\* p){

point\* r;

int nl, nr;

if (n == 0) {

p = NULL;

return p;

}

nl = n / 2;

nr = n - nl - 1;

r = new point;

inputElement(r->data);

r->left = BuildTree(nl, r->left);

r->right = BuildTree(nr, r->right);

p = r;

return p;

}

Функция PrintTree выводит дерево, повернутое против часовой стрелки. принимает в качестве параметров целочисленную переменную со значением - число элементов дерева и будет использоваться как переменная текущего уровня, а также текущую структуру point\* p. Если текущий элемент дерева не пустой (не равен nullptr), начинаем вывод с самого правого листа, то есть рекурсивно спускаемся к самому правому элементу и выводим его, если мы его вывели, возвращаемся к предыдущему элементу, далее вызываем функцию Run и в качестве параметров передаем ссылку на элемент левого поддерева.

void PrintTree(point\* p, int level) {

if (p) {

PrintTree(p->right, level + 1);

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

cout << " ";

}

cout << p->data << endl;

PrintTree(p->left, level + 1);

}

}

Функция TreeHeight используя рекурсивный алгоритм определяет количество уровней в дереве, принимает в качестве параметра текущую структуру point\* p. Есть два узла, левый и правый. Высота дерева на 1 больше максимальной высоты поддеревьев. Сначала считаем высоту левого поддерева, потом правого. Как только опускаемся до узла, у которого нет потомка, рекурсия останавливается.

int TreeHeight(point\* p) {

if (!p) {

return 0;

}

int h\_l = TreeHeight(p->left);

int h\_r = TreeHeight(p->right);

return max(h\_l, h\_r) + 1;

}

Функция breadthFirst определяет максимальное значение обходя дерево в ширину, принимает в качестве параметра текущую структуру point\* p. Обход в ширину подразумевает, что сначала мы посещаем корень, затем, слева направо, все ветви первого уровня, затем все ветви второго уровня и т.д. Пусть мы находимся в корне дерева. Далее необходимо посетить всех наследников корня. Таким образом, нужно засунуть в контейнер сначала узел, затем его наследников, при этом узел далее должен быть обработан первым. То есть, элемент, который вошёл первым должен быть обработан первым.

void breadthFirst(point\* p) {

int max\_data = INT\_MIN;

list <point\*> q;

q.push\_back(p);

while (!q.empty()) {

point\* tmp = (point\*)q.front();

q.pop\_front();

if (tmp->left) {

q.push\_back(tmp->left);

}

if (tmp->right) {

q.push\_back(tmp->right);

}

if (tmp->data > max\_data) {

max\_data = tmp->data;

}

}

cout << "Максимальное значение = " << max\_data << endl;

}

Функции MinValue и MaxValue определяют минимальное и максимальное значения в дереве соответственно, принимает в качестве параметра текущую структуру point\* p. Для нахождения максимального значения создаем переменную max\_data, инициализируя ее INT\_MIN. Если дерево не пустое, то присваиваем max\_data текущее значение узла. Далее с помощью циклов ищем максимальное значение сначала в правом, потом левом поддеревьях. Алгоритм нахождения минимального значения аналогичен.

int MaxValue(point\* p) {

int max\_data = INT\_MIN;

if (!p) {

return 0;

}

if (p->data > max\_data) {

max\_data = p->data;

}

while (p->right != nullptr) {

p = p->right;

if (p->data > max\_data) {

max\_data = p->data;

}

}

while (p->left != nullptr) {

p = p->left;

if (p->data > max\_data) {

max\_data = p->data;

}

}

return max\_data;

}

int MinValue(point\* p){

int min\_data = INT\_MAX;

if (!p) {

return 0;

}

if (p->data < min\_data) {

min\_data = p->data;

}

while (p->right != nullptr) {

p = p->right;

if (p->data < min\_data) {

min\_data = p->data;

}

}

while (p->left != nullptr) {

p = p->left;

if (p->data < min\_data) {

min\_data = p->data;

}

}

return min\_data;

}

Функция DeleteTree удаляет бинарное дерево, принимает в качестве параметра текущую структуру point\* p. Удаление происходит рекурсивно, начиная с самого левого нижнего элемента. Сначала он удаляется с помощью delete, затем данная структура принимает значение NULL. Так происходит, пока не удалится и не обнулится корень.

void DeleteTree(point\*& p) {

if (p) {

DeleteTree(p->left);

DeleteTree(p->right);

delete p;

p = nullptr;

}

}

# **ИНТЕРФЕЙС**

При запуске программы пользователю предоставляется выбор одного из предложенных заданий (Рисунок 2), указанных в постановке задачи. Любой другой введенный символ или число не входящее в диапазон [0;6] спровоцирует повторный ввод.

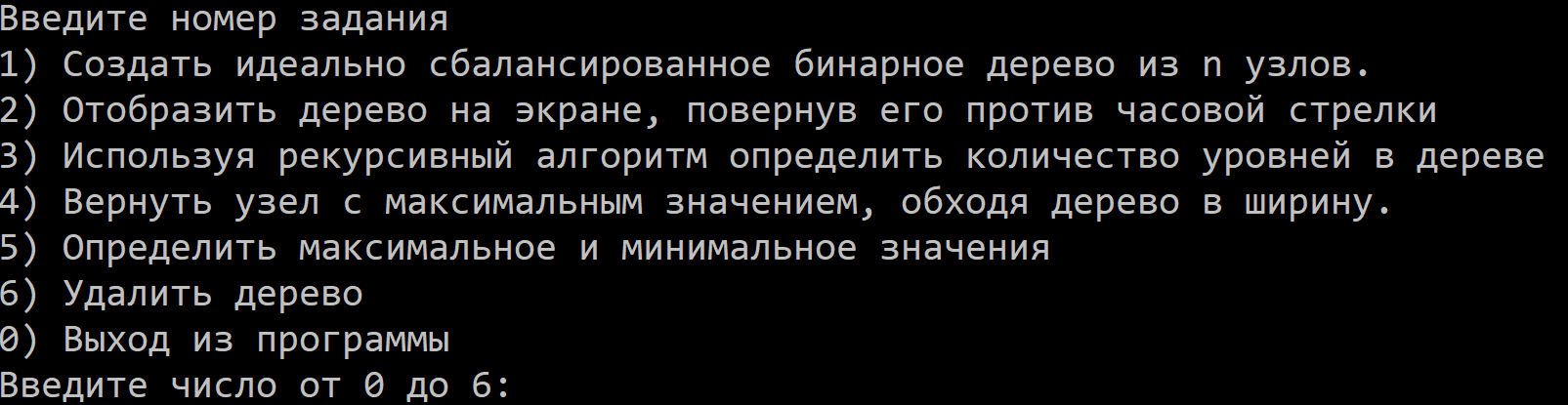


Рисунок 2 - Интерфейс программы

# **ТЕСТИРОВАНИЕ**

При тестировании первого задания мы вводим количество элементов дерева, а затем эти элементы. Элементы проверяются на корректность. Для проверки работоспособности программы воспользуемся вторым заданием, заодно протестировав его (Рисунок 3). Если мы в приложении выберем пункт 2, не создав перед этим дерево, то выведется соответствующее сообщение.

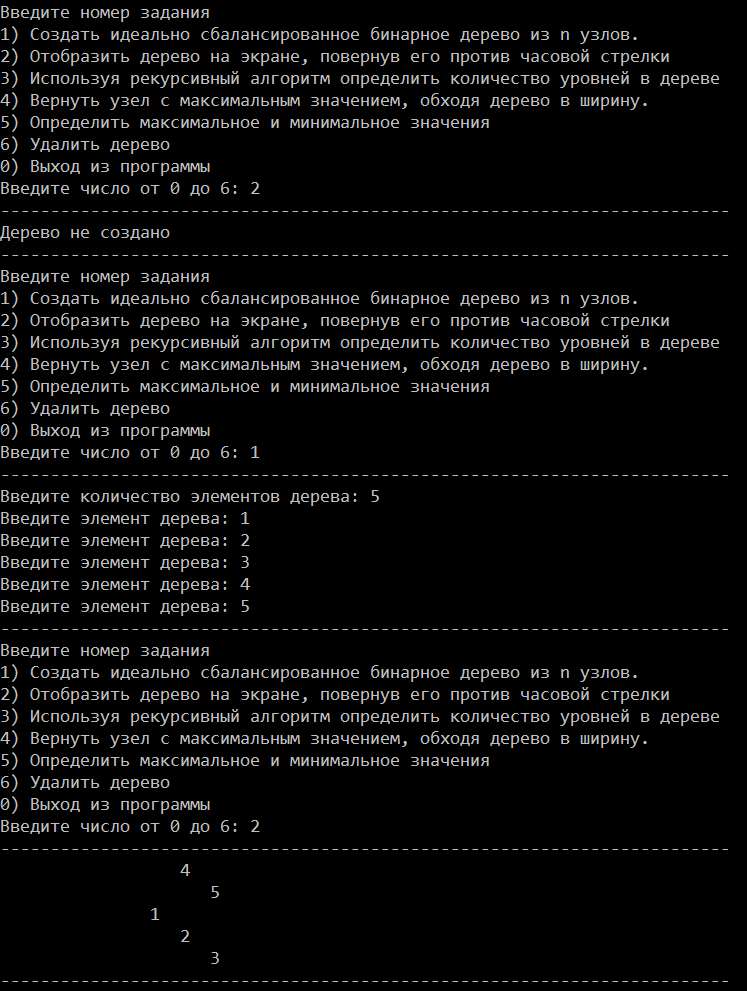


Рисунок 3 - Тестирование первого и второго заданий

Для тестирования функции определения количества уровней в дереве создадим дерево из пяти элементов и заполним его. Функция работает корректно (Рисунок 4). Если мы не создадим перед использованием функции дерево, то выведется соответствующее сообщение.

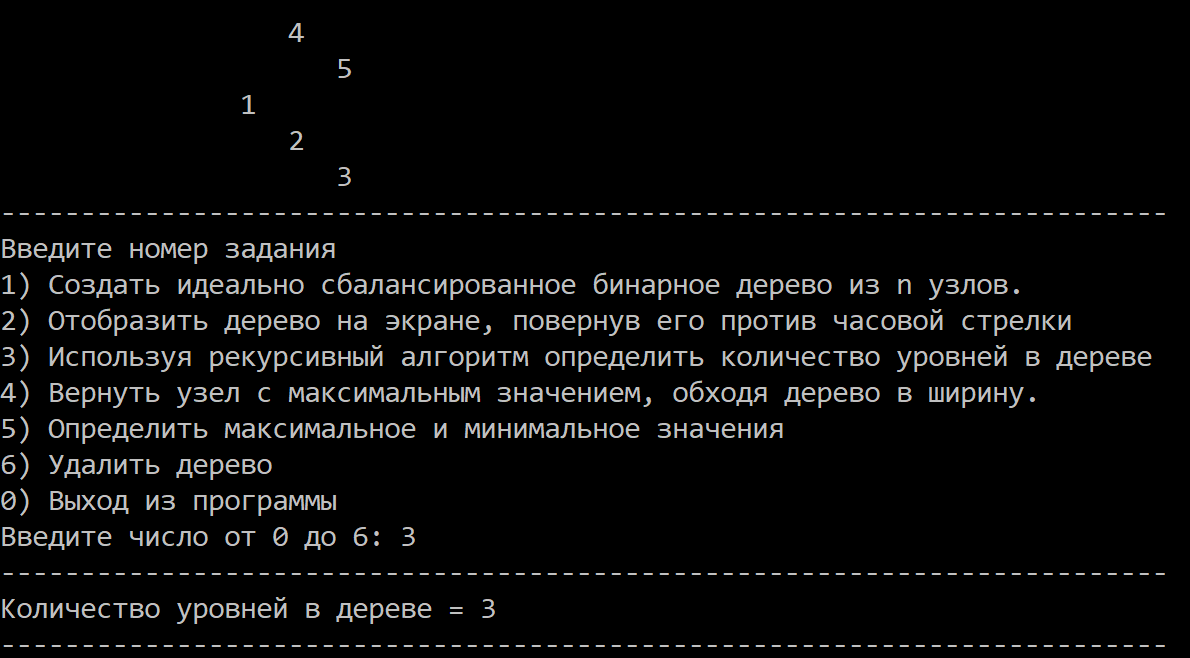


Рисунок 4 - Тестирование третьего задания

Для тестирования четвертого задания создадим два дерева, с 5 и 6 узлами. Функция работает корректно (Рисунок 5, Рисунок 6). Если мы не создадим перед использованием функции дерево, то выведется соответствующее сообщение.

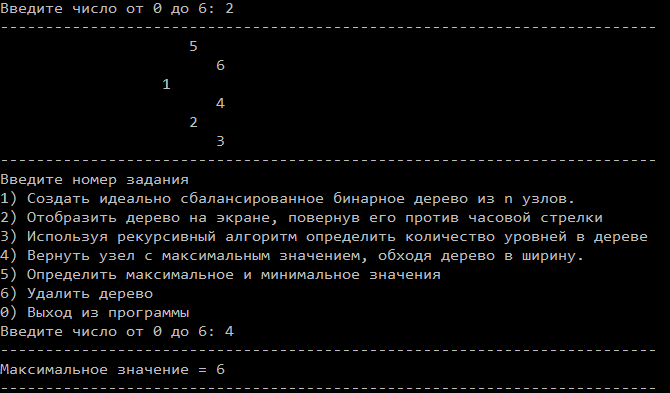


Рисунок 5 - Тестирование четвертого задания (1)

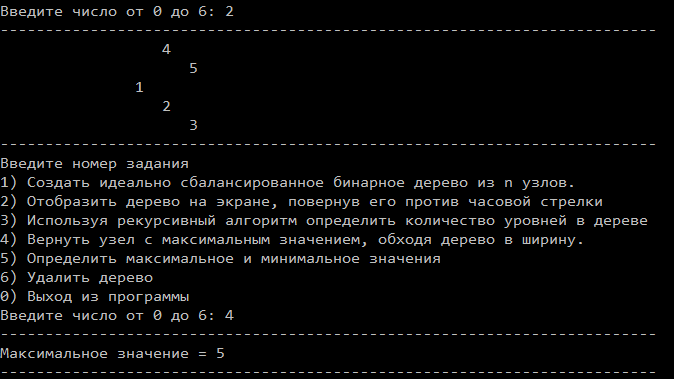


Рисунок 6 - Тестирование четвертого задания (2)

Аналогично четвертому заданию для тестирования пятого задания создадим два дерева, с 5 и 6 узлами. Функция работает корректно (Рисунок 7, Рисунок 8). Если мы не создадим перед использованием функции дерево, то выведется соответствующее сообщение.

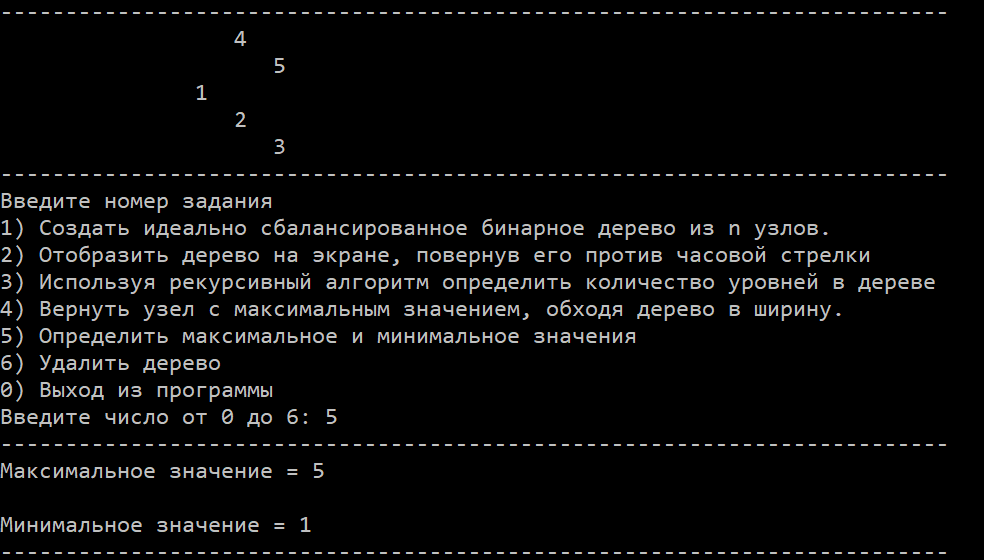


Рисунок 7 - Тестирование пятого задания (1)

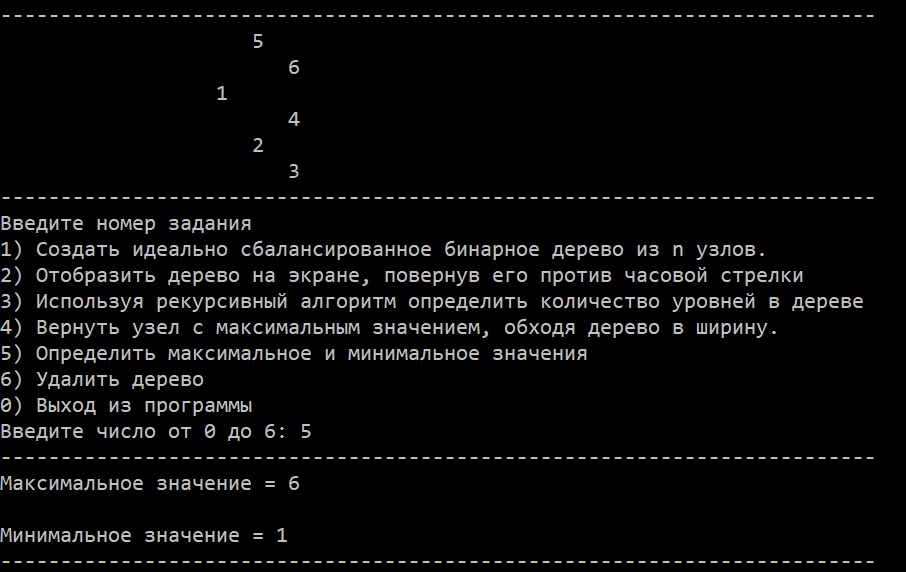


Рисунок 8 - Тестирование пятого задания (2)

# **ВЫВОД**

В результате выполнения работы были получены умения и навыки разработки и реализаций операций над структурой данных бинарное дерево.

# **ПОЛНЫЙ КОД ПРОГРАММЫ**

Файл siaod4\_2part.cpp:

#include <iostream>

#include "BinaryTree.h"

using namespace std;

int inputMenu(int& number) {

cout << "Введите номер задания" << endl;

cout << "1) Создать идеально сбалансированное бинарное дерево из n узлов." << endl;

cout << "2) Отобразить дерево на экране, повернув его против часовой стрелки" << endl;

cout << "3) Используя рекурсивный алгоритм определить количество уровней в дереве" << endl;

cout << "4) Вернуть узел с максимальным значением, обходя дерево в ширину. " << endl;

cout << "5) Определить максимальное и минимальное значения" << endl;

cout << "6) Удалить дерево" << endl;

cout << "0) Выход из программы" << endl;

while (1) {

cout << "Введите число от 0 до 6: ";

if ((cin >> number).good() && (number >= 0) && (number <= 6)) {

return number;

}

if (cin.fail()) {

cin.clear();

cout << endl << "Неверный ввод, повторите." << endl << endl;

}

else {

cout << endl << "Число вне допустимого диапазона значений. Повторите ввод." << endl << endl;

}

cin.ignore(100, '\n');

}

}

int main() {

system("chcp 1251 > null");

int menu;

point\* p = 0;

int n = -1;

while (true) {

inputMenu(menu);

if (menu == 0) {

cout << "Задание не выбрано\nВыход из программы..." << endl;

break;

}

switch (menu) {

case 1: {

cout << "-------------------------------------------------------------------------" << endl;

cout << "Введите количество элементов дерева: ";

cin >> n;

p = BuildTree(n, p);

cout << "-------------------------------------------------------------------------" << endl;

break;

}

case 2: {

cout << "-------------------------------------------------------------------------" << endl;

if (p) {

PrintTree(p, n);

}

else {

cout << "Дерево не создано\n";

}

cout << "-------------------------------------------------------------------------" << endl;

break;

}

case 3: {

cout << "-------------------------------------------------------------------------" << endl;

if (p) {

cout << "Количество уровней в дереве = " << TreeHeight(p) << endl;

}

else {

cout << "Дерево не создано\n";

}

cout << "-------------------------------------------------------------------------" << endl;

break;

}

case 4: {

cout << "-------------------------------------------------------------------------" << endl;

if (p) {

breadthFirst(p);

}

else {

cout << "Дерево не создано\n";

}

cout << "-------------------------------------------------------------------------" << endl;

break;

}

case 5: {

cout << "-------------------------------------------------------------------------" << endl;

if (p) {

cout << "Максимальное значение = " << MaxValue(p) << endl;

cout << endl;

cout << "Минимальное значение = " << MinValue(p) << endl;

}

else {

cout << "Дерево не создано\n";

}

cout << "-------------------------------------------------------------------------" << endl;

break;

}

case 6: {

DeleteTree(p);

cout << "-------------------------------------------------------------------------" << endl;

cout << "Дерево удалено" << endl;

cout << "-------------------------------------------------------------------------" << endl;

break;

}

default: {

cout << "Задание не выбрано\nВыход из программы..." << endl;

break;

}

}

}

return 0;

}

Файл BinaryTree.h:

#ifndef BINARYTREE\_H

#define BINARYTREE\_H

#include <iostream>

#include <list>

using namespace std;

struct point {

int data = 0;

point\* left = nullptr;

point\* right = nullptr;

};

int inputElement(int& number);

point\* BuildTree(int n, point\* p);

void PrintTree(point\* p, int level);

int TreeHeight(point\* p);

void breadthFirst(point\* p);

int MaxValue(point\* p);

int MinValue(point\* p);

void DeleteTree(point\*& p);

#endif

Файл BinaryTree.cpp:

#include "BinaryTree.h"

int inputElement(int& number) {

while (1) {

cout << "Введите элемент дерева: ";

if ((cin >> number).good() && (number > 0)) {

return number;

}

if (cin.fail()) {

cin.clear();

cout << endl << "Неверный ввод, повторите." << endl << endl;

}

else {

cout << endl << "Число вне допустимого диапазона значений. Повторите ввод." << endl << endl;

}

cin.ignore(100, '\n');

}

}

point\* BuildTree(int n, point\* p){

point\* r;

int nl, nr;

if (n == 0) {

p = nullptr;

return p;

}

nl = n / 2;

nr = n - nl - 1;

r = new point;

inputElement(r->data);

r->left = BuildTree(nl, r->left);

r->right = BuildTree(nr, r->right);

p = r;

return p;

}

void PrintTree(point\* p, int level) {

if (p) {

PrintTree(p->right, level + 1);

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

cout << " ";

}

cout << p->data << endl;

PrintTree(p->left, level + 1);

}

}

int TreeHeight(point\* p) {

if (!p) {

return 0;

}

int h\_l = TreeHeight(p->left);

int h\_r = TreeHeight(p->right);

return max(h\_l, h\_r) + 1;

}

void breadthFirst(point\* p) {

int max\_data = INT\_MIN;

list <point\*> q;

q.push\_back(p);

while (!q.empty()) {

point\* tmp = (point\*)q.front();

q.pop\_front();

if (tmp->left) {

q.push\_back(tmp->left);

}

if (tmp->right) {

q.push\_back(tmp->right);

}

if (tmp->data > max\_data) {

max\_data = tmp->data;

}

}

cout << "Максимальное значение = " << max\_data << endl;

}

int MaxValue(point\* p) {

int max\_data = INT\_MIN;

if (!p) {

return 0;

}

if (p->data > max\_data) {

max\_data = p->data;

}

while (p->right != nullptr) {

p = p->right;

if (p->data > max\_data) {

max\_data = p->data;

}

}

while (p->left != nullptr) {

p = p->left;

if (p->data > max\_data) {

max\_data = p->data;

}

}

return max\_data;

}

int MinValue(point\* p){

int min\_data = INT\_MAX;

if (!p) {

return 0;

}

if (p->data < min\_data) {

min\_data = p->data;

}

while (p->right != nullptr) {

p = p->right;

if (p->data < min\_data) {

min\_data = p->data;

}

}

while (p->left != nullptr) {

p = p->left;

if (p->data < min\_data) {

min\_data = p->data;

}

}

return min\_data;

}

void DeleteTree(point\*& p) {

if (p) {

DeleteTree(p->left);

DeleteTree(p->right);

delete p;

p = nullptr;

}

}

# **ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Что определяет степень дерева?

Ответ: Максимальное значение среди степеней всех узлов определяет степень дерева.

1. Какова степень сильноветвящегося дерева?

Ответ: Если сильноветвящееся дерево содержит n-1 ключей и n указателей, то степень узлов n>2.

1. Что определяет путь в дереве?

Ответ: Путь в дереве – это последовательность, в которой чередуются вершины и рёбра графа, начиная с вершины и заканчивая вершиной.

1. Как рассчитать длину пути в дереве?

Ответ: Длина пути дерева определяется как сумма длин путей ко всем его вершинам. Рассчитывается по рекурсивной формуле: Длина пути = длина пути в левом поддереве + длина пути в правом поддереве + количество узлов в дереве – 1.

1. Какова степень бинарного дерева?

Ответ: В бинарном дереве степени вершин не превосходят 2.

1. Может ли дерево быть пустым?

Да, ровно, как и любое множество. Такое дерево является nullptr.

1. Дайте определение бинарного дерева?

Ответ: Двоичное дерево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками. Двоичное дерево является упорядоченным ориентированным деревом.

1. Дайте определение алгоритму обхода.

Ответ: Алгоритм обхода – такой алгоритм, который осуществляет посещение каждой вершины дерева один раз.

1. Приведите рекуррентную зависимость для вычисления высоты дерева.

Ответ:

int TreeHeight(point\* p) {

if (!p) {

return 0;

}

int h\_l = TreeHeight(p->left);

int h\_r = TreeHeight(p->right);

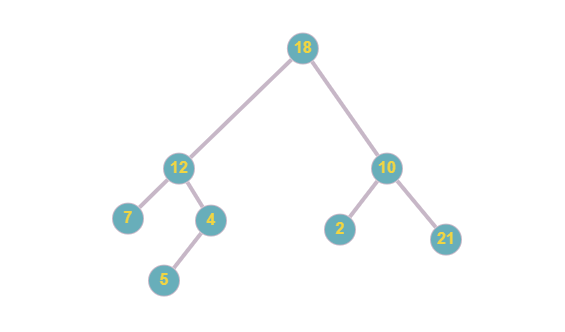
return max(h\_l, h\_r) + 1;

}

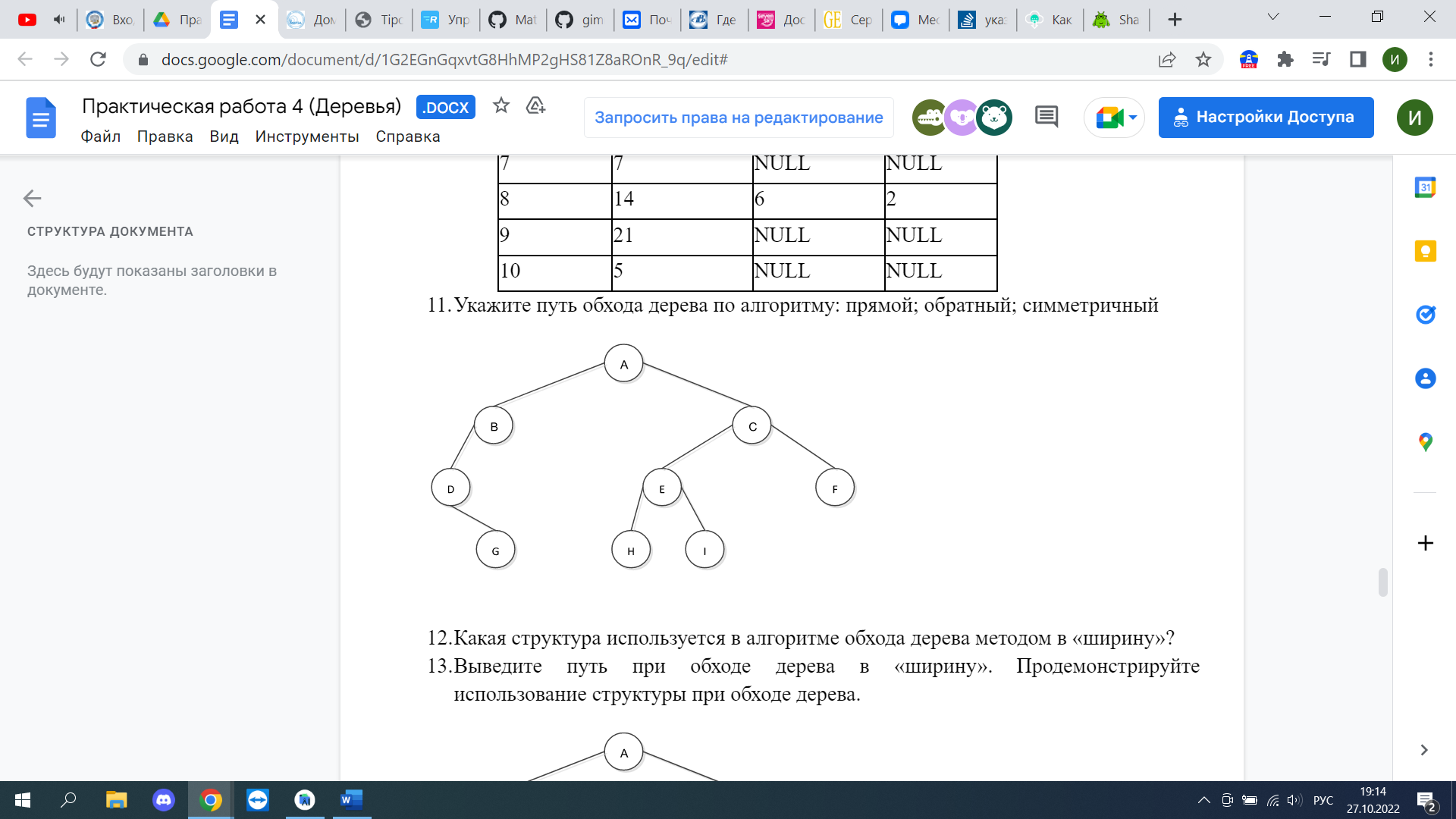
1. Изобразите бинарное дерево, корень которого имеет индекс 6, и которое представлено в памяти таблицей вида

|  |  |  |  |
| --- | --- | --- | --- |
| *Индекс* | *key* | *left* | *right* |
| 1 | 12 | 7 | 3 |
| 2 | 15 | 8 | NULL |
| 3 | 4 | 10 | NULL |
| 4 | 10 | 5 | 9 |
| 5 | 2 | NULL | NULL |
| 6 | 18 | 1 | 4 |
| 7 | 7 | NULL | NULL |
| 8 | 14 | 6 | 2 |
| 9 | 21 | NULL | NULL |
| 10 | 5 | NULL | NULL |

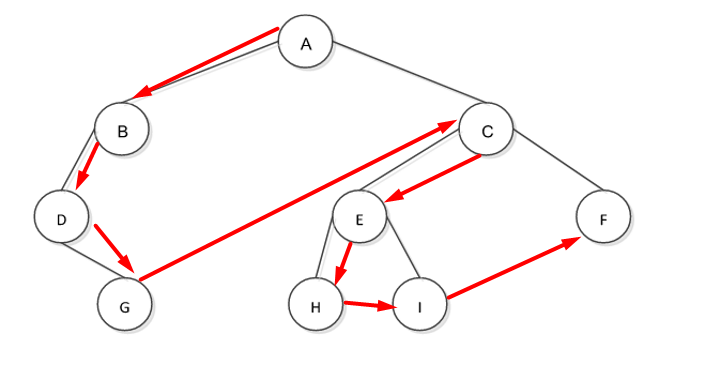
Ответ:



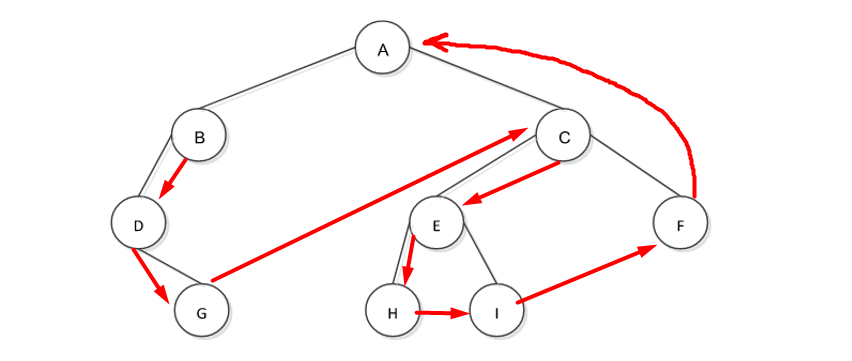
1. Укажите путь обхода дерева по алгоритму: прямой; обратный; симметричный.



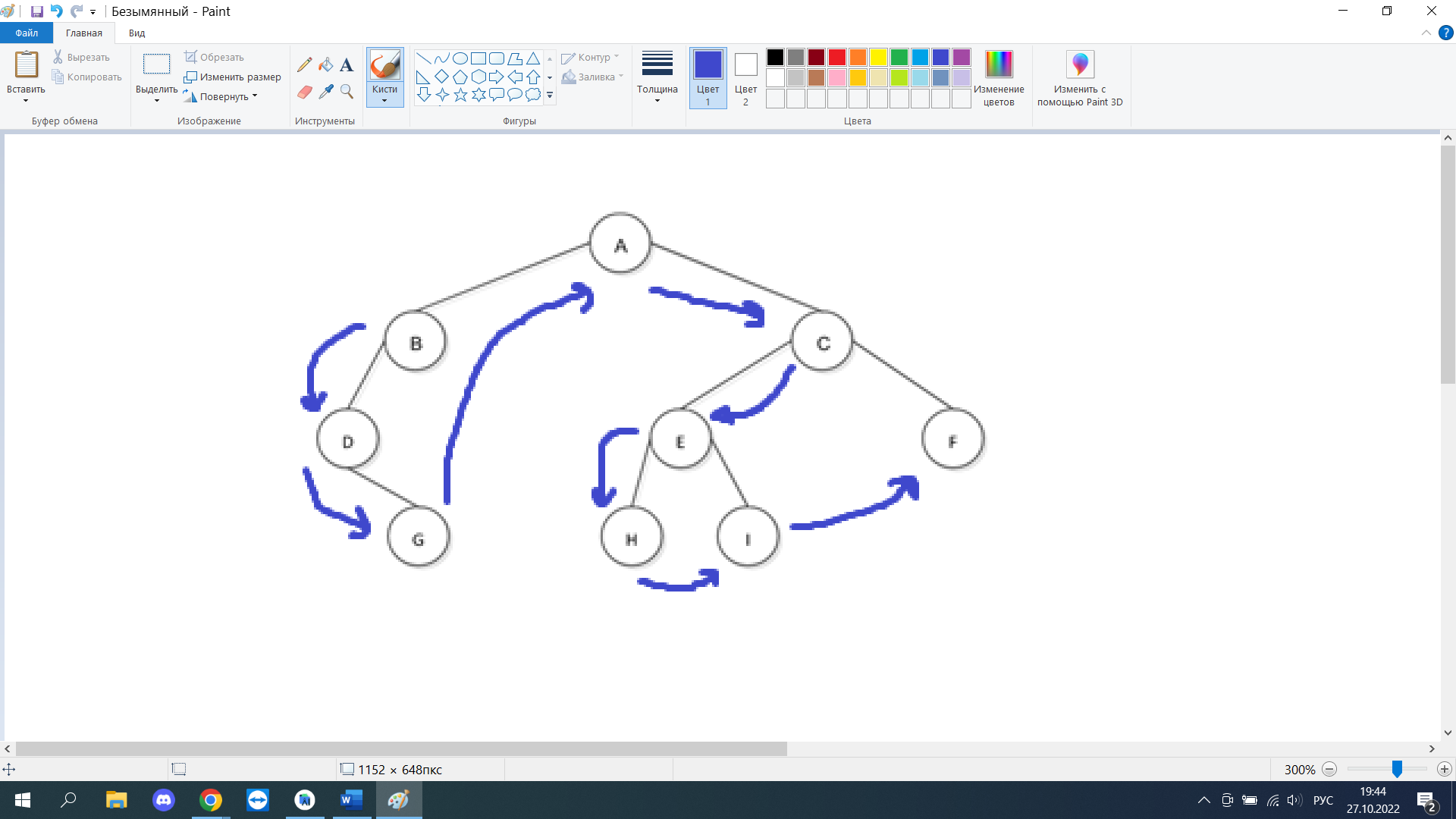
Прямой (Начинается с А):



Обратный (Начинается с B):



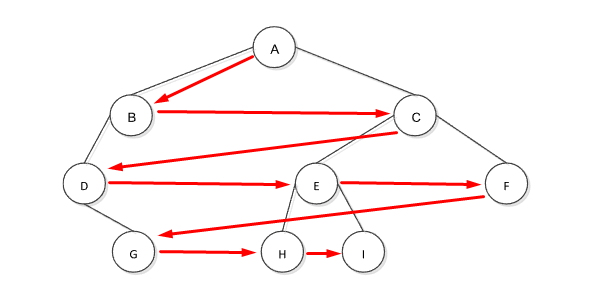
Симметричный обход (Начиная с B):



1. Какая структура используется в алгоритме обхода дерева методом в «ширину»?

Ответ: Используется структура «Очередь». Она же – std::list, она же – двунаправленный список.

1. Выведите путь при обходе дерева в «ширину». Продемонстрируйте использование структуры при обходе дерева.



typedef int Data;

typedef struct Node {

Data data;

struct Node \* left;

struct Node \* right;

} Node;

Node \* tree\_add(Node \* tree, Data data){

if(tree == NULL){

Node \* t = malloc(sizeof(Node));

t->data = data;

t->left = t->right = NULL;

return t;

}

if(data<tree->data){

tree->left = tree\_add(tree->left,data);

}

if(data>tree->data){

tree->right = tree\_add(tree->right,data);

}

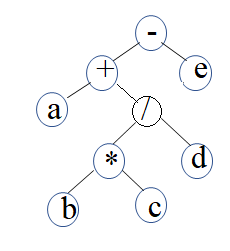
return tree;

}

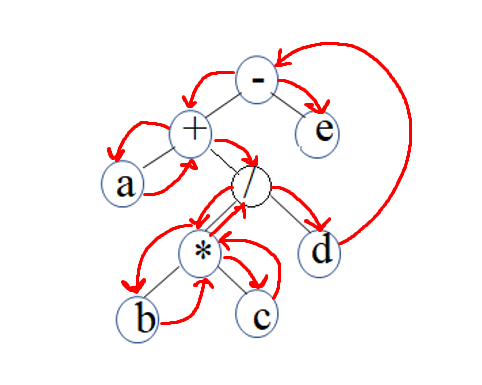
1. Какая структура используется в не рекурсивном обходе дерева методом в «глубину»?

Ответ: Используется структура «Очередь». Она же – std::list, она же – двунаправленный список.

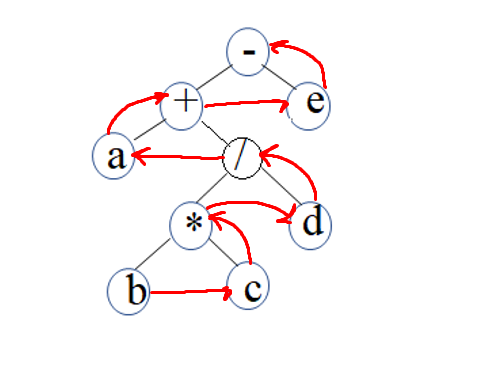
1. Выполните прямой, симметричный, обратный методы обхода дерева выражений.



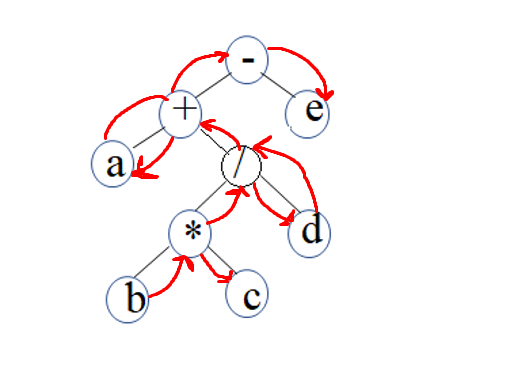
Прямой:



Обратный:

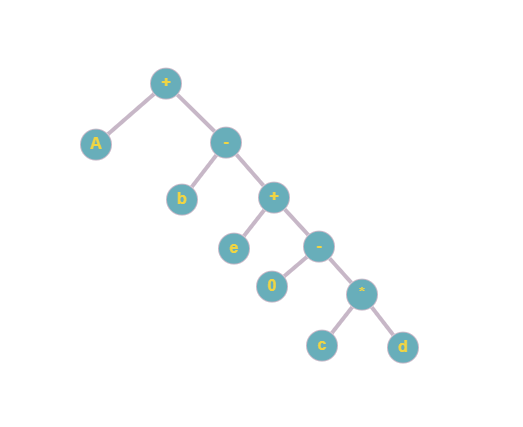


Симметричный:

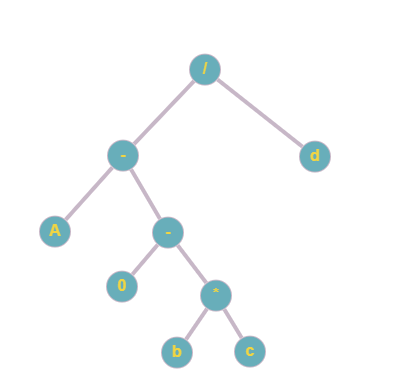


1. Для каждого заданного арифметического выражения постройте бинарное дерево выражений:

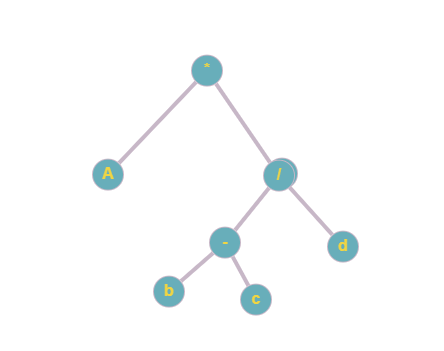
a+b-c\*d+e



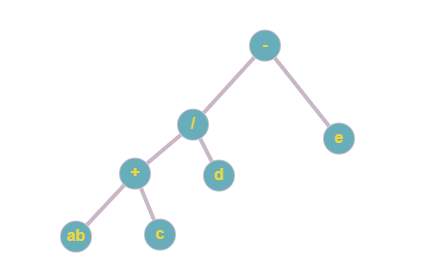
/a-b\*c d



a b c d / - \*



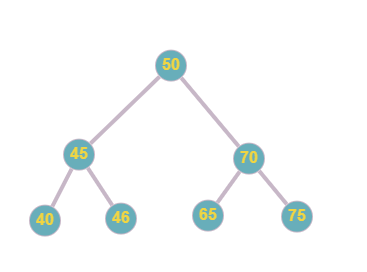
- / + a b c d e



1. В каком порядке будет проходиться бинарное дерево, если алгоритм обхода в ширину будет запоминать узлы не в очереди, а в стеке?

Ответ: обходит узлы как Post-Order (обойти левое поддерево-> обойти правое поддерево-> посетить корень), только задом наперёд.

1. Постройте бинарное дерево поиска, которое в результате симметричного обхода дало бы следующую последовательность узлов 40 45 46 50 65 70 75.



1. Приведенная ниже последовательность получена путем прямого обхода бинарного дерева поиска. Постройте это дерево. 50 45 35 15 40 46 65 75 70

