МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Отчет по выполнению практического задания №3

По дисциплине «Структуры и алгоритмы обработки данных»

Тема:

Нелинейные структуры данных. Бинарное дерево

Выполнил студент Гусейнов Р.Э.

группа ИКБО-20-21

Тема. Нелинейные структуры данных. Бинарное дерево

Цель. Получить навыки по разработке хеш-таблиц и их применении.

Задание: Разработать программу, которая создает идеально сбалансированное

дерево из n узлов и выполняет операции.

1. Реализовать операции общие для всех вариантов

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 на основе меню, позволяющего проверить выполнение всех операций на ваших тестах и тестах преподавателя.

4. Оформить отчет.

1) Для каждой представленной в программе функции предоставить отчет по ее разработке в соответствии с требованиями разработки программы (подпрограммы).

2) Представить алгоритм основной программы и таблицу имен, используемых в алгоритме.

Вариант 4: Значение информационной части - целое число.

Операции варианта:
Определить, в каком поддереве исходного дерева четных чисел больше.
Создать копию исходного двоичного дерева.

Разработка: Было написано больше методов, чем требовалось по заданию (здесь описаны только те, что по заданию)

Класс BinTree {

```
int val; //информационная часть узла
BinTree* right = nullptr; // указатель на правое поддерево
BinTree* left = nullptr; // указатель на левое поддерево
```

}

Метод создания дерева: (было написано три разных функции, здесь описывается одна)

Прототип: void BinTree::fill_random(int n)

Предусловие: int n – количество элементов в дереве

Алгоритм:

```
void BinTree::fill_random(int n) {
   val := СЛУЧАЙНОЕ МЕНЬШЕЕ 100;
   n := n – 1
   ЦЕЛОЕ r = n / 2, l = n - r;
   Если (l > 0) тогда
      left := new BinTree
      left->fill_random(l) // вызов метода
   Конец если
   Если (r > 0) тогда
      right := new BinTree;
      right->fill_random(r); // вызов метода
   Конец если
}
```

Реализация: создание идеально сбалансированного бинарного дерева(листинг 1)

```
void BinTree::fill_random(int n) {
    this->val = rand() % 100;
    --n;
    int r = n / 2, l = n - r;
    if (l > 0) {
        this->left = new BinTree;
        this->left->fill_random(l);
    }
    if (r > 0) {
        this->right = new BinTree;
        this->right->fill_random(r);
    }
}
```

Листинг 1

Метод перевернутого вывода дерева (также были написаны два разных метода для обычного вывода дерева, но не относятся к работе)

Прототип: void BinTree::out_right_left(int h, int lvl)

Предусловие: int h – высота дерева, int lvl – глубина, на которой мы сейчас находимся

Алгоритм:

```
void BinTree::out_right_left(int h, int lvl) {
    ЕСЛИ !(right==nullptr) ТОГДА
        right->out_right_left( h,lvl + 1); // вызов метода
    ИНАЧЕ
        ЕСЛИ !(lvl + 1 == h) ТОГДА
            ВЫВОД << std::setw((lvl + 2) * 4) << "n" << "\n";
            ВЫВОД << std::setw((lvl + 1) * 4) << th  is->val << "\n";

        КОНЕЦ ЕСЛИ

    КОНЕЦ ЕСЛИ
    ЕСЛИ !(left == nullptr) ТОГДА
        left->out_right_left(h, lvl + 1); // вызов метода
    ИНАЧЕ
```

```
    ЕСЛИ !(lvl + 1 == h) ТОГДА
      ВЫВОД << std::setw((lvl + 2) * 4) << "n" << "\n";
    КОНЕЦ ЕСЛИ
  КОНЕЦ ЕСЛИ
```

Реализация: (Листинг 2)

```
void BinTree::out_right_left(int h, int lvl) {
    if (this->right) {
        this->right->out_right_left( h,lvl + 1);
    } else {
        if (lvl + 1 != h) {
            std::cout << std::setw((lvl + 2) * 4) << "n" << "\n";
        }
    }
    std::cout << std::setw((lvl + 1) * 4) << this->val << "\n";
    if (this->left) {
        this->left->out_right_left(h, lvl + 1);
    } else {
        if (lvl + 1 != h) {
            std::cout << std::setw((lvl + 2) * 4) << "n" << "\n";
        }
    }
}
```

Листинг 2

Метод для определения количества четных чисел в дереве

Прототип: int BinTree::even()

Предусловие: считает четные числа в дереве

Алгоритм:

```
int BinTree::even(){
    int count := 0;
    ЕСЛИ (val % 2 == 0) ТОГДА
```

```
    count := count + 1;
КОНЕЦ ЕСЛИ
ЕСЛИ !(left == 0)ТОГДА
    count := count +left->even();
КОНЕЦ ЕСЛИ
ЕСЛИ !(right == 0) ТОГДА
    count := count + right->even();
КОНЕЦ ЕСЛИ
ВЕРНУТЬ count;
}
```

Реализация: подсчет количества четных чисел в дереве

```
int BinTree::even(){
    int count = 0;
    if (this->val % 2 == 0){
        count += 1;
    }
    if (this->left){
        count += this->left->even();
    }
    if (this->right){
        count += this->right->even();
    }
    return count;
}
```

Листинг 3

Метод для подсчета высоты дерева:

Прототип: int BinTree::height()

Предусловие: определяет высоту дерева

Алгоритм:

```
УКАЗАТЕЛЬ НА БИНАРНОЕ ДЕРЕВО *tmp = this;
ЦЕЛОЕ count = 0;
ПОКА !(tmp == 0) НЦ
```

```
    count := count + 1;
    tmp := tmp->left;
КЦ
ВЕРНУТЬ count;
```

Реализация: рекурсивный подсчёт высоты дерева (Листинг 4)

```
int BinTree::height() {
    BinTree *tmp = this;
    int count = 0;
    while (tmp) {
        ++count;
        tmp = tmp->left;
    }
    return count;
}
```

Листинг 4

Метод для определения поддерева с максимальным числом четных чисел:

Прототип: void BinTree::count_even()

Алгоритм:

```
void BinTree::count_even() {
    ЦЕЛОЕ c_e_r = 0, c_e_l = 0;
    ЕСЛИ !(left == 0) ТОГДА
        c_e_l += this->left->even();
    КОНЕЦ ЕСЛИ
    ЕСЛИ !(right == 0) ТОГДА
        c_e_r = c_e_r + right->even()
    КОНЕЦ ЕСЛИ
    ЕСЛИ (c_e_r > c_e_l) ТОГДА
        ВЫВОД << "right: " << c_e_r << std::endl;
    ИНАЧЕ ЕСЛИ (c_e_r < c_e_l) ТОГДА
        ВЫВОД << "left: " << c_e_l << std::endl;
    ИНАЧЕ
        ВЫВОД << "equality: " << c_e_l << std::endl;
```

```
    КОНЕЦ ЕСЛИ
}
```

Реализация: метод поиска поддерева с максимальным количеством четных чисел (Листинг 5)

```
void BinTree::count_even() {
    int c_e_r = 0, c_e_l = 0;
    if (this->left){
        c_e_l += this->left->even();
    }
    if (this->right){
        c_e_r += this->right->even();
    }
    if (c_e_r > c_e_l) {
        std::cout << "right: " << c_e_r << std::endl;
    } else if (c_e_r < c_e_l) {
        std::cout << "left: " << c_e_l << std::endl;
    } else {
        std::cout << "equality: " << c_e_l << std::endl;
    }
}
```

Листинг 5

Конструктор копирования бинарного дерева

Прототип: BinTree(BinTree* &other)

Предусловие: BinTree* &other – копируемое дерево

Алгоритм:

```
this->val := other->val;
ЕСЛИ (other->left) ТОГДА
    this->left := new BinTree(other->left);
КОНЕЦ ЕСЛИ
ЕСЛИ (other->right) ТОГДА
    this->right := new BinTree(other->right);
КОНЕЦ ЕСЛИ
```

Реализация: метод копирования дерева(Листинг 6).

```
BinTree::BinTree(BinTree *&other) {
    this->val = other->val;
    if (other->left){
        this->left = new BinTree(other->left);
    }
    if (other->right){
        this->right = new BinTree(other->right);
    }
};
```

Листинг 6

Тестирование:

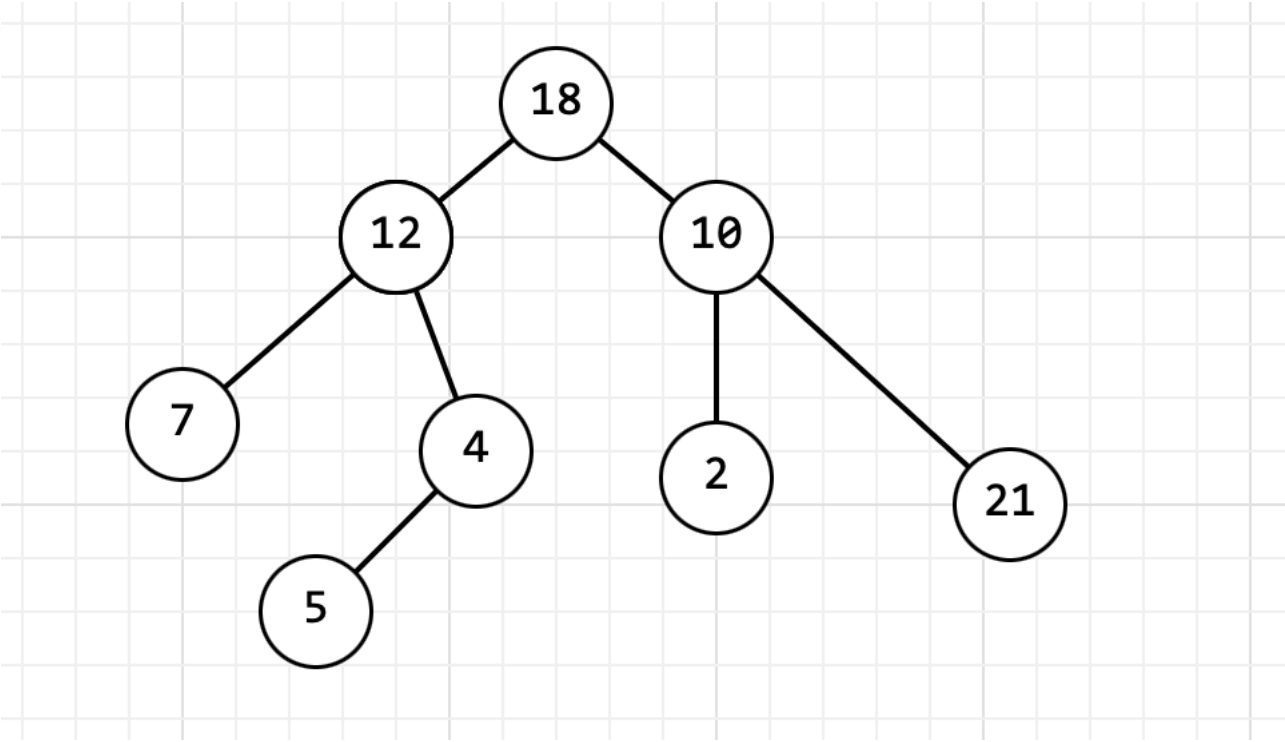
```
/Users/aurumnleaf/CLionProjects/BIN_TREE/cmake-build-debug/BIN_TREE
Select operation:
1. Create a binary tree yourself
2. Create a binary tree
3. Create a binary tree with random
4. Output a binary tree (not by option)
5. Output an inverted binary tree
6. Insert element into binary tree
7. Find out which subtree has more even numbers
8. Create a copy of the binary tree
9. Output a copy of the binary tree (not by option)
1
Input enter the number of elements: 15
Enter tree elements:
1 2 3 4 5 6 7 8 0 0 0 0 0 0 0
Tree completed successfully
Would you like to continue? 1 --- yes, 0 --- no: 1
4
The binary tree:
                1
        2               0
    3       6       0       0
  4   5   7   8   0   0   0   0
```

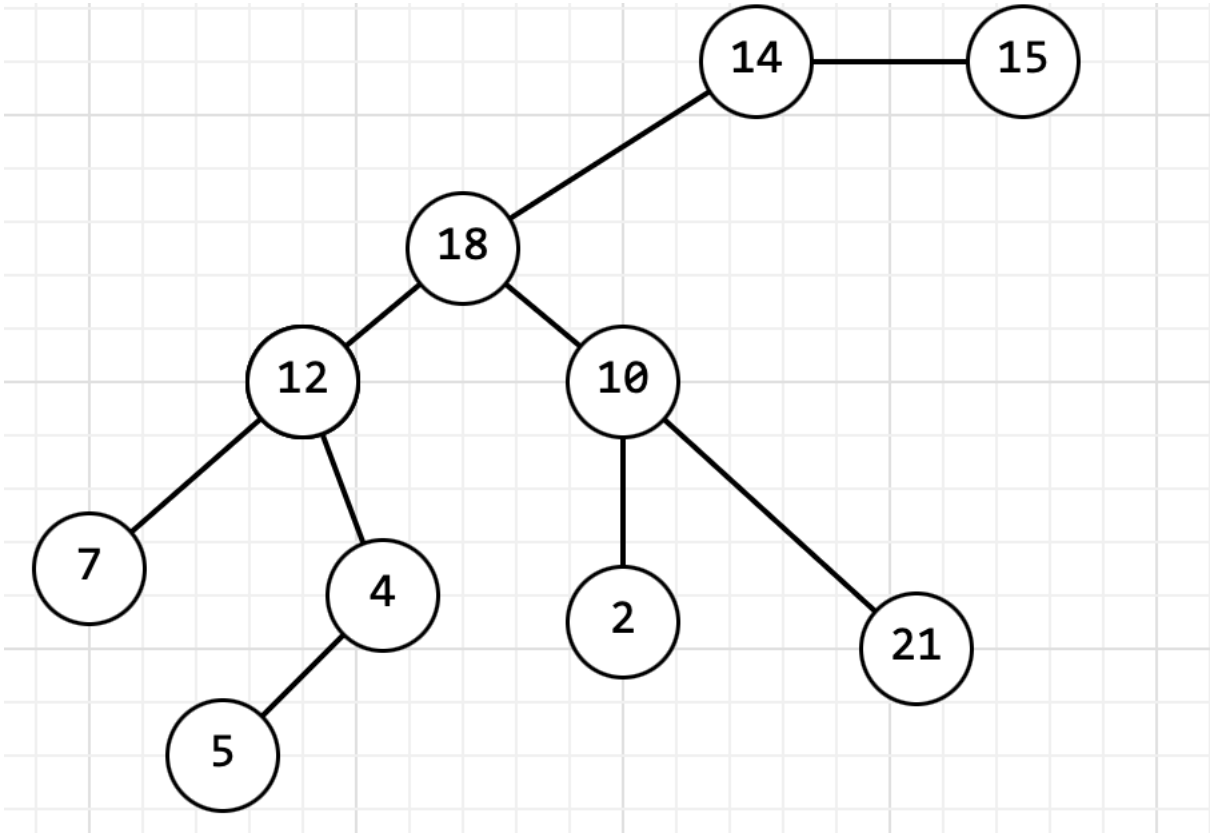
```
5
The binary tree:
                0
            0
                0
        0
                0
            0
                0
    1
                8
            6
                7
        2
                5
            3
                4
Would you like to continue? 1 --- yes, 0 --- no: 1
6
Input enter the element:
123
Item added successfully
Would you like to continue? 1 --- yes, 0 --- no: 1
8
Tree copy create
Would you like to continue? 1 --- yes, 0 --- no: 1
9
Copy of the binary tree:
                                1
                2                               0
          3               6               0               0
      4       5       7       8       0       0       0       0
  123   n   n   n   n   n   n   n   n   n   n   n   n   n   n   n
Would you like to continue? 1 --- yes, 0 --- no: 1
7
right: 7
```

Ответы на вопросы:

1. Степень дерева - это максимальная степень вершин, входящих в дерево.
2. Степень сильноветвящегося дерева - произвольная.
3. Путь в дереве - последовательность узлов от корневого до искомого узла.
4. Длину пути в дереве == сумме длин путей его ребер.
5. Степень бинарного дерева – 0, 1 или 2.
6. Дерево может быть пустым, если степень вершины 0.
7. Бинарное дерево – дерево, степень которого не больше 2
8. Обход дерева - вид обхода графа, обуславливающий процесс посещения каждого узла структуры дерева ровно один раз.
9. Высота равна 0, если дерево пустое
   Высота равна 1 + максимум(высота правого поддерева, высота левого поддерева)

10. В этой таблице какой-то бред, если рисовать дерево целиком, картинка ниже прилагается





11. Прямой ход: ABDGCEHIF, Обратный ход: GDBHIEFCA, Симметричный: DGBAHEICF

12. Очередь

13.

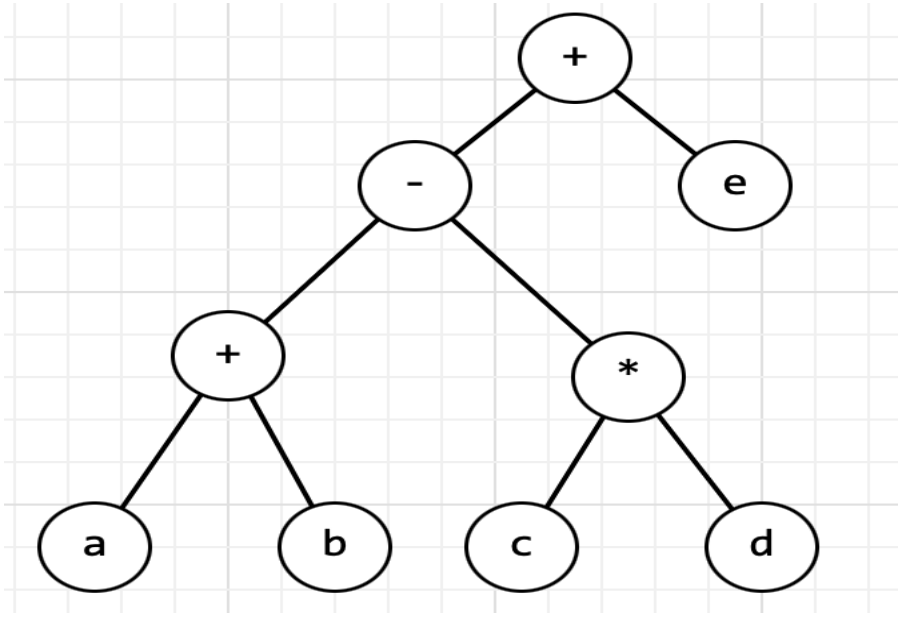
Очередь	Вывод
A	
BC	A
CD	AB
DEF	ABC
EFG	ABCD
FGHI	ABCDE
GHI	ABCDEF
HI	ABCDEF
I	ABCDEFH
	ABCDEFHI

14. Стек

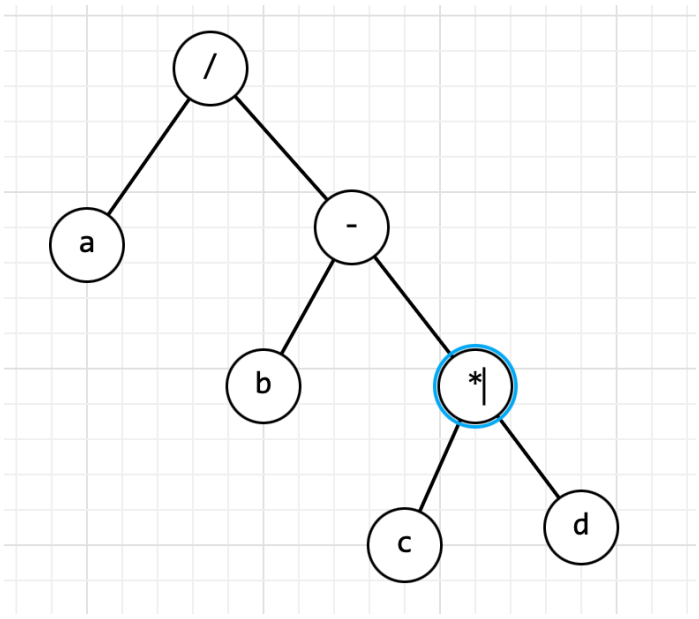
15. Прямой обход: **-+a/*bcde**, симметричный обход: **a+b*c/d-e,** обратный обход**: abc*d/+e-**

16.

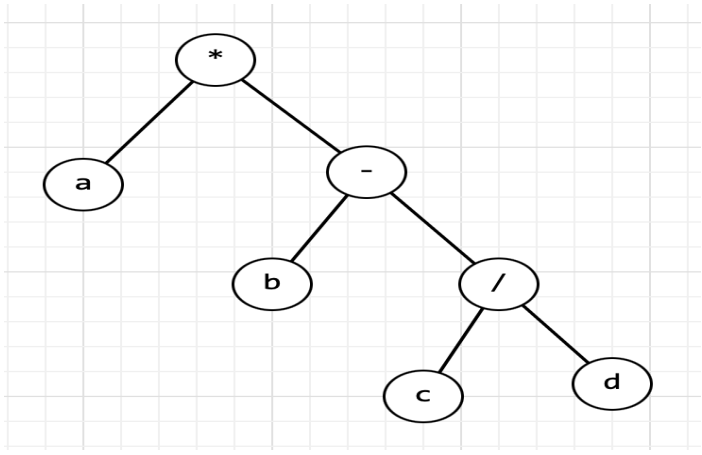
1.



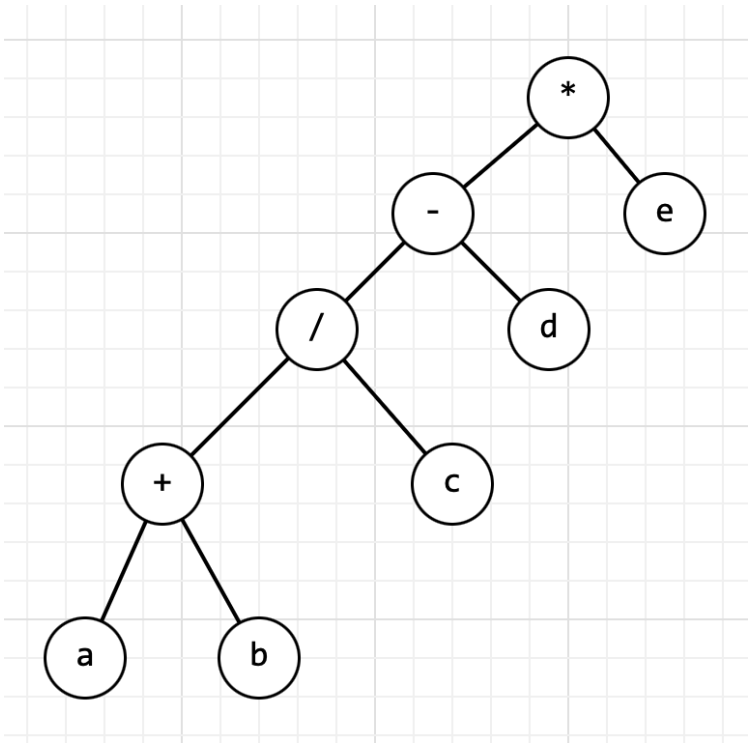
2.



3.

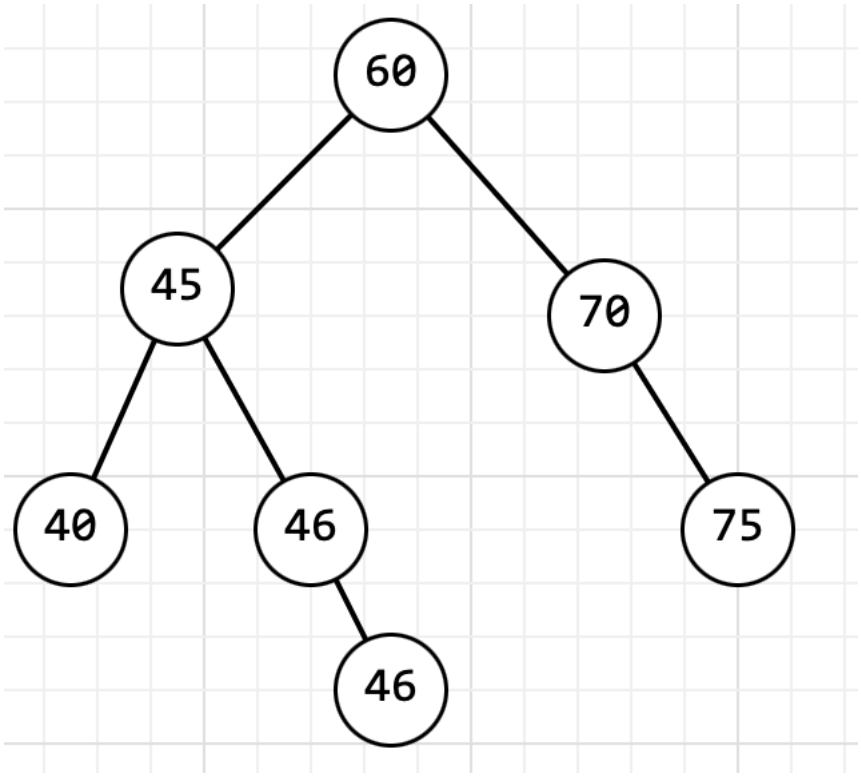


4.

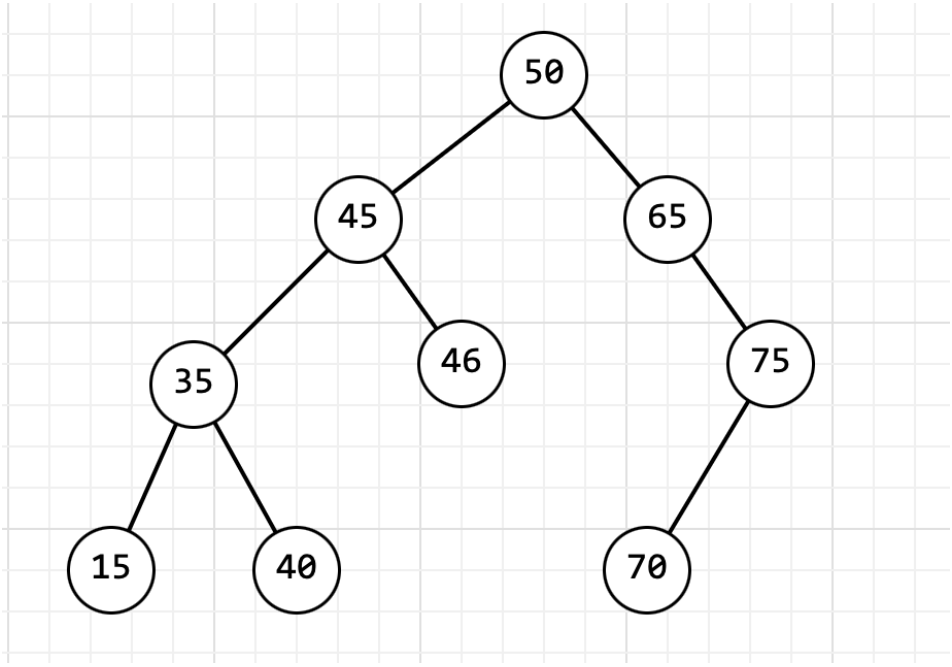


17. Бинарное дерево будет обходится в прямом порядке

18.

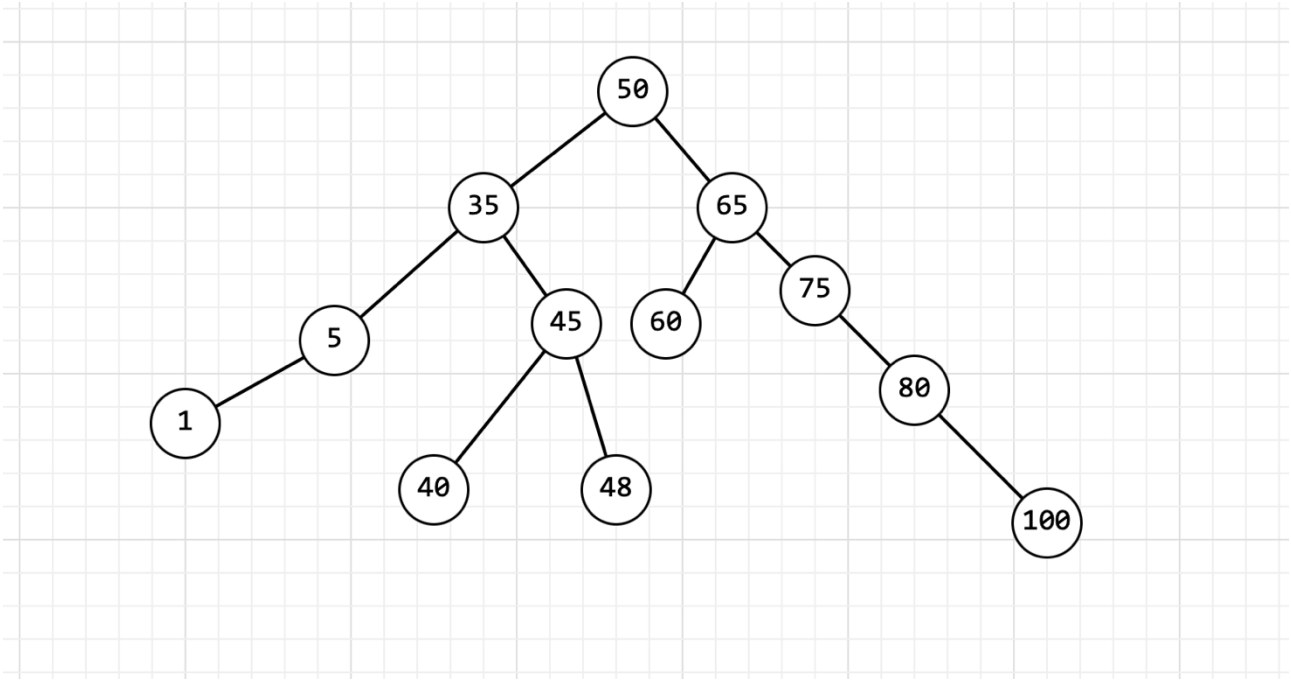


19.

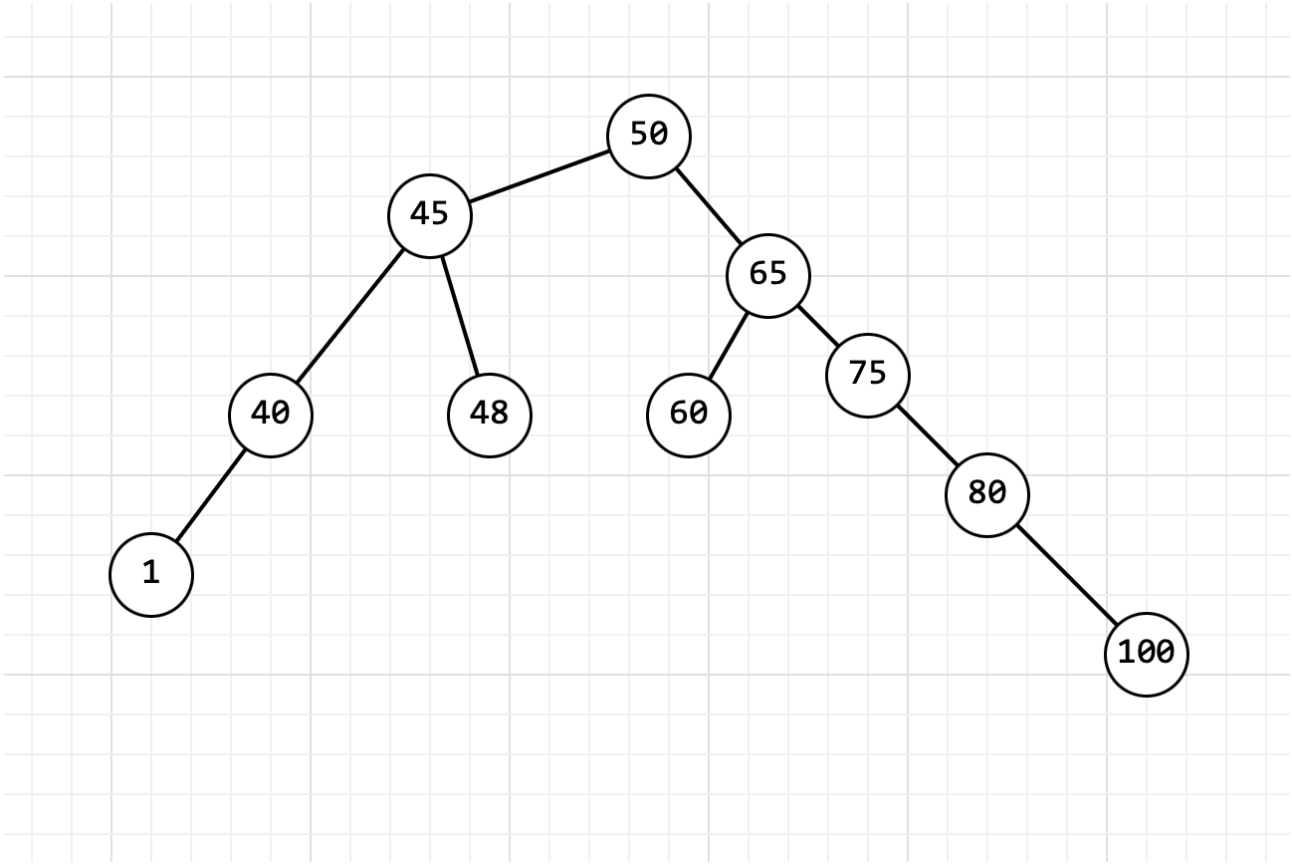


20.

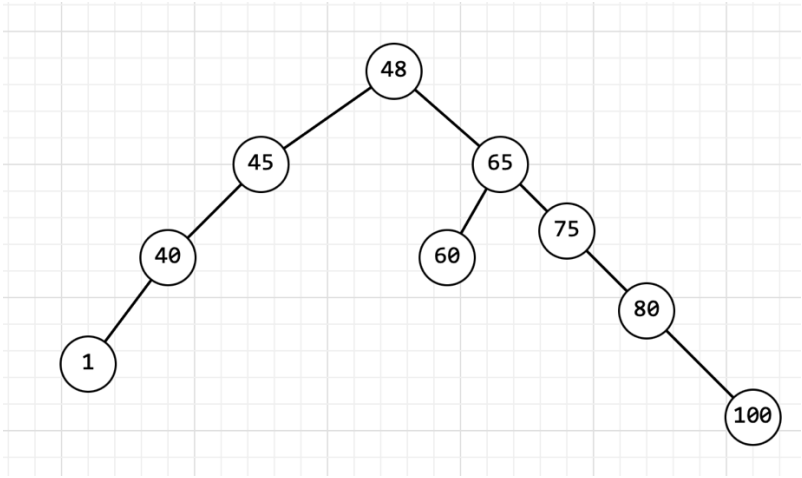
20.1.



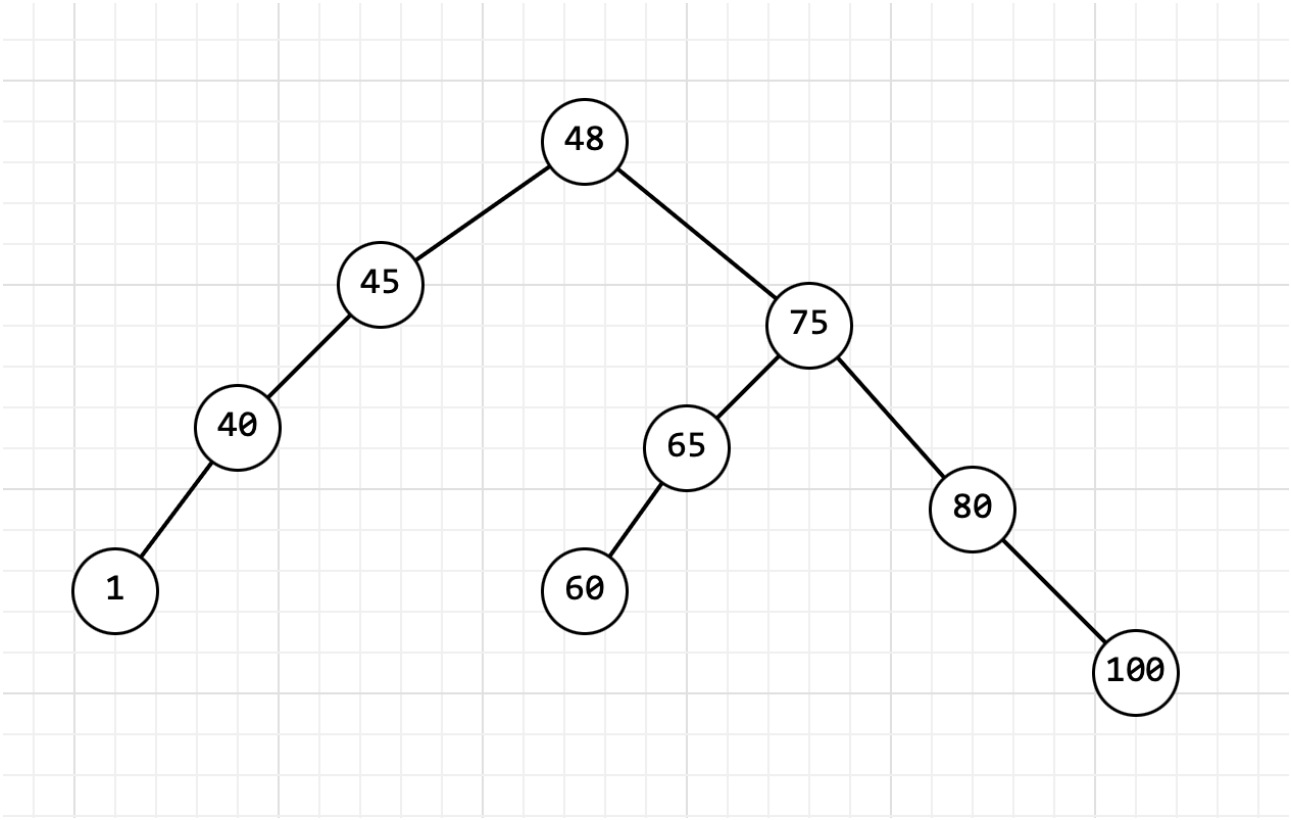
20.2.



20.3.



20.4.


48
45
75
40
65
80
1
60
100

Выводы:

В ходе выполнения данной работай был реализован класс представляющий собой структуру хранения - идеально сбалансированное дерево. Также были получены навыки и умения разработки и реализации операций над данной структурой.

Список литературы:

Методическое пособие по выполнению задания 2