МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЁТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ №3

по дисциплине «Структуры и алгоритмы обработки данных»

Тема: «Нелинейные структуры данных. Бинарное дерево.»

Отчет представлен к рассмотрению:
Студентка группы ИНБО-01-20 «12» октября 2021 г. _____ Тульцова А.Д.
(подпись)

Преподаватель «12» октября 2021 г. _____ Сорокин А.В.
(подпись)

Москва, 2021 г.

СОДЕРЖАНИЕ

Цель работы

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

Постановка задачи

Разработать программу, которая создает идеально сбалансированное дерево из n узлов и выполняет операции.

Вариант 7.

Значение информационной части	Операции варианта
Символьное значение	Определить уровень, на котором находится заданное значение. Определить количество цифр в левом поддереве исходного дерева. Вывести дерево, располагая элементы вертикально.

Дано:

Идеально сбалансированное бинарное дерево из n узлов.

Структура узла дерева, включающая в себя информационную часть узла, указатель на левое и указатель на правое поддерево. Информационная часть узла – символьное значение.

Результат.

Отображение на экране дерева, повернутого справа налево.

Реализованные операции варианта.

Подход к решению.

1) В ходе работы над задачей был разработан класс идеально сбалансированного бинарного дерева, реализующий, согласно варианту, методы создания бинарного дерева из массива элементов, определения уровня, на котором находится заданное значение, определения количества цифр в левом поддереве исходного дерева, и вывода дерева с вертикальным отображением элементов. Также был разработан метод вывода дерева на экран справа налево.
2) В ходе работы был разработан класс узла бинарного дерева. Он содержит информационную часть и ссылки на левое и правое поддерево. В узле реализованы методы распределения элементов входного массива на свою информационную часть и на правое и левое поддеревья, а также метод вывода дерева на экран.

3) В ходе работы был разработан класс приложения с графическим пользовательским интерфейсом для тестирования работоспособности программы.
4) В ходе решения были разработаны методы обработки бинарного дерева:

1. Обход в «глубину» – метод, обходящий дерево в «глубину», для вывода уровня, на котором находится искомый элемент.
2. Подсчет количества цифр в левом поддереве – метод, перебирающий все элементы левого поддерева, определяющий цифры в каждом из них и возвращающий их количество.
3. Вертикальный вывод дерева – последовательный вывод дерева, начиная с верхнего элемента, в процессе которого визуально сохраняется относительное расположение элементов по горизонтали.
4. Горизонтальный вывод дерева – последовательный вывод дерева, начиная с правого элемента, в процессе которого отступ от левого края выставляется сообразно глубине элемента.

Алгоритмы операций на псевдокоде.

Метод подсчёта кол-ва цифр в левом поддереве:

```
функция count_digits_from_left(elements := массив[строковый тип]):
    elements.сортировать()
    middle := длина(elements) // 2
    sum_d := 0
    для каждого i в elements[от 0 до middle]:
        для каждого j в i:
            если (j <= '9') и (j >= '0'):
                sum_d := sum_d + 1
    возврат sum_d
```

Метод вертикального вывода дерева:

```
функция print_vert():
    если стартовый_узел == пустой: возврат ''
    ls := массив[]
    depth := стартовый_узел.добавить_в_верт_список(ls, 0)
    res := ''
    для каждого line от 0 до depth:
        для каждого pos от 0 до длина(ls):
            если ls[pos][1] == line:
                res := res + в_строковый_тип(ls[pos][0])
            иначе:
                res := res + ' ' * ls[pos][2]
        res := res + 'перевод_на_следующую_строку'
    возврат res
```

Метод вывода уровня:

```
функция level(искомое_значение, elements := массив[строковый тип]):
    если искомое_значение не в elements: возврат '–'
    иначе: возврат стартовый_узел.dfs(0, искомое_значение)
```

Метод поиска в «глубину»:

```
функция dfs(depth, искомое_значение):
    res := 0
    если правый_узел не пустой:
        res := res + правый_узел.dfs(depth + 1, искомое_значение)
    если левый_узел не пустой:
        res := res + левый_узел.dfs(depth + 1, искомое_значение)
    если значение_элемента != искомое значение: возврат res
    иначе: возврат depth
```

Код приложения.

Класс узла дерева:

```
class Node:
    def __init__(self, members: list):
        if len(members) == 0:
            pass
        elif len(members) == 1:
            self.info = members[0]
            self.right = None
            self.left = None
        elif len(members) == 2:
            self.info = members[1]
            self.left = Node([members[0]])
            self.right = None
        else:
            middle = len(members) // 2
            self.left = Node(members[0: middle])
            self.info = members[middle]
            self.right = Node(members[middle + 1:])

    def add_to_hor_list(self, ls: list, depth: int) -> None:
        if self.right is not None:
            self.right.add_to_hor_list(ls, depth + 1)
        ls.append('        ' * depth + str(self.info) + '(' + str(depth) +
')')
        if self.left is not None:
            self.left.add_to_hor_list(ls, depth + 1)

    def add_to_vert_list(self, ls: list, depth) -> int:
        m1, m2 = 0, 0
        if self.left is not None:
            m1 = self.left.add_to_vert_list(ls, depth + 1)
        ls.append((self.info, depth, len(str(self.info))))
        if self.right is not None:
            m2 = self.right.add_to_vert_list(ls, depth + 1)
        return max(m1, m2, depth)

    def dfs(self, depth, key):
        res = 0
        if self.right is not None:
            res += self.right.dfs(depth + 1, key)
        if self.left is not None:
            res += self.left.dfs(depth + 1, key)
        return res if self.info != key else depth
```

Класс дерева:

```
class Tree:
    def __init__(self, elements: str):
        elements = [str(x) for x in elements.split()]
        elements.sort()
        self.__start = Node(elements)

    def __str__(self):
        if self.__start == None: return ''
        ls = []
        self.__start.add_to_hor_list(ls, 0)
        return '\n'.join(ls)

    def count_digits_from_left(self, elements: str) -> int:
        elements = [str(x) for x in elements.split()]
        elements.sort()
        middle = len(elements) // 2
        sum_d = 0
        for i in elements[0: middle]:
            for j in i:
                if (j <= '9') and (j >= '0'):
                    sum_d += 1
        return sum_d

    def delete(self) -> None:
        self.__start = None

    def print_vert(self):
        if self.__start == None: return ''
        ls = []
        depth = self.__start.add_to_vert_list(ls, 0)
        res = ''
        for line in range(depth + 1):
            for pos in range(len(ls)):
                if ls[pos][1] == line:
                    res += str(ls[pos][0])
                else:
                    res += ' ' * ls[pos][2]
            res += '\n'
        return res

    def level(self, key: str, elements: str):
        elements = [str(x) for x in elements.split()]
        if key not in elements: return('-')
        else: return self.__start.dfs(0, key)
```

Класс для тестирования:

```
import PySimpleGUI as PSG
from Tree import Tree

class App:
    def __init__(self):
        default_tree = 'a&b hello x0y0z0 c0x0 abc012 34de'
        self.tree = Tree(default_tree)
        self.level = PSG.Text(font=("Times New Roman", 12),
text_color='black', background_color='#ffb300')
        self.count_digits_from_left = PSG.Text(font=("Times New Roman", 12),
text_color='black', background_color='#ffb300')
        self.show_hor = PSG.Text(str(self.tree), font=("Times New Roman",
12), text_color='black', background_color='#ffb300')
        self.show_vert = PSG.Text(str(self.tree.print_vert()), font=("Times
New Roman", 12), text_color='black', background_color='#ffb300')
        self.layout = [
            [PSG.In(key='_ELEMENTS_', default_text=default_tree, font=("Times
New Roman", 12), text_color='black'), PSG.Button(key='_CREATE_',
button_text='Создать дерево', font=("Times New Roman", 12),
button_color=('orange', 'brown'))],
            [PSG.Text('Кол-во цифр в левом поддереве:', font=("Times New
Roman", 12), text_color='black', background_color='#ffb300'),
self.count_digits_from_left],
            [PSG.In(key='_ELEMENT_'), PSG.Button(key='_LEVEL_',
button_text='Найти уровень элемента', font=("Times New Roman", 12),
button_color=('orange', 'brown')), PSG.Text('Элемент найден на уровне:',
font=("Times New Roman", 12), text_color='black',
background_color='#ffb300'), self.level],
            [PSG.Button(key='_DELETE_', button_text='Удалить дерево',
font=("Times New Roman", 12), button_color=('orange', 'brown'))],
            [PSG.Text('Дерево (гор.):', font=("Times New Roman", 12),
text_color='black', background_color='#ffb300'), self.show_hor],
            [PSG.Text('Дерево (верт.):', font=("Times New Roman", 12),
text_color='black', background_color='#ffb300'), self.show_vert]
        ]
    def run(self):
        app = PSG.Window('Идеальное сбалансированное дерево', self.layout,
background_color='#ffb300')
        while True:
            event, values = app.read()

            if event == PSG.WIN_CLOSED:
                break
            elif event == '_CREATE_':
                self.tree = Tree(values['_ELEMENTS_'])

self.count_digits_from_left.update(str(self.tree.count_digits_from_left(value
s['_ELEMENTS_'])))
                self.show_hor.update(str(self.tree))
                self.show_vert.update(str(self.tree.print_vert()))
            elif self.tree is not None:
                if event == '_LEVEL_':

self.level.update(str(self.tree.level(values['_ELEMENT_'],
values['_ELEMENTS_'])))
                if event == '_DELETE_':
                    self.tree.delete()
                    self.show_hor.update(str(self.tree))
                    self.show_vert.update(str(self.tree))
application = App()
application.run()
```

Скриншоты результатов.

Было запущено тестирование и создано дерево из элементов: ‘a&b’, ‘hello’, ‘x0y0z0’, ‘c0x0’, ‘abc012’, ‘34de’.

Результат:

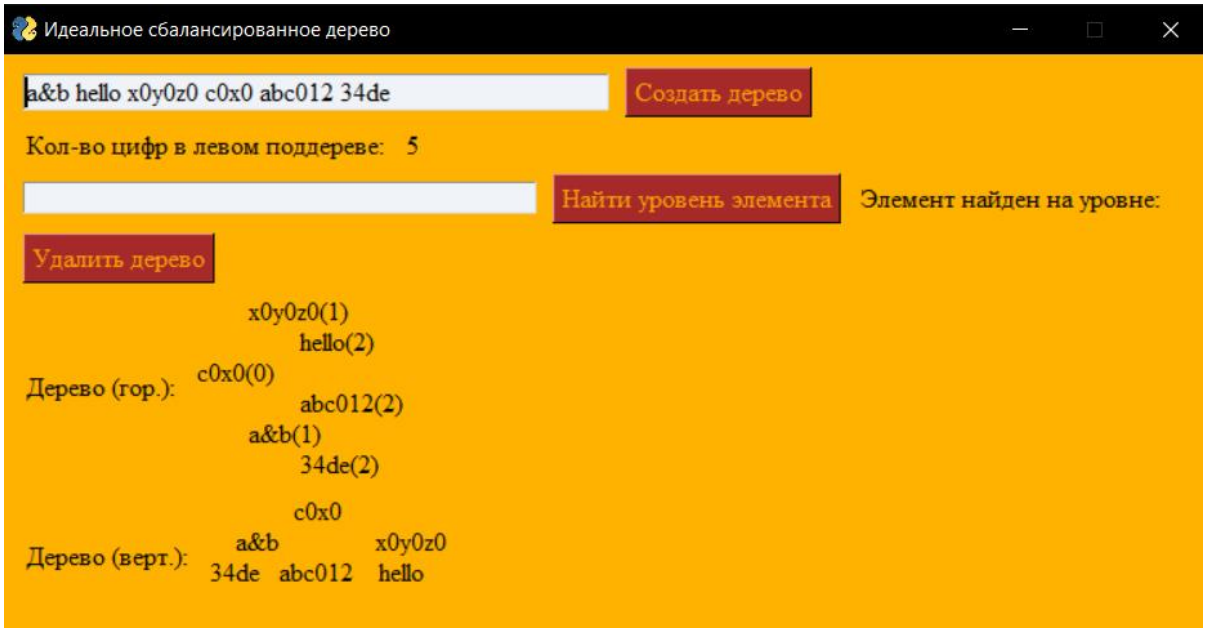

Рисунок 1 – Ввод элементов.

В результате работы программы было выведено дерево из данных элементов (в вертикальном и горизонтальном видах) и общее количество цифр в левом поддереве.

Продолжая тестирование программы, найдём уровень элементов ‘hello’, ‘c0x0’, а также попробуем найти уровень элемента ‘ab’, не существующего в дереве.

Результат:

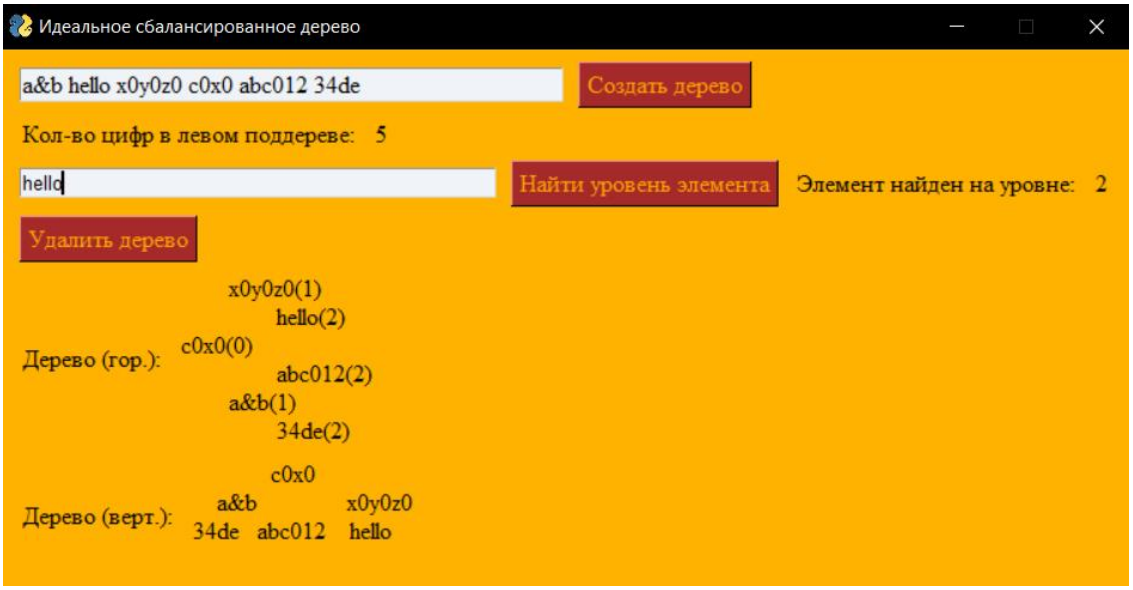

Рисунок 2 – Результаты тестирования.

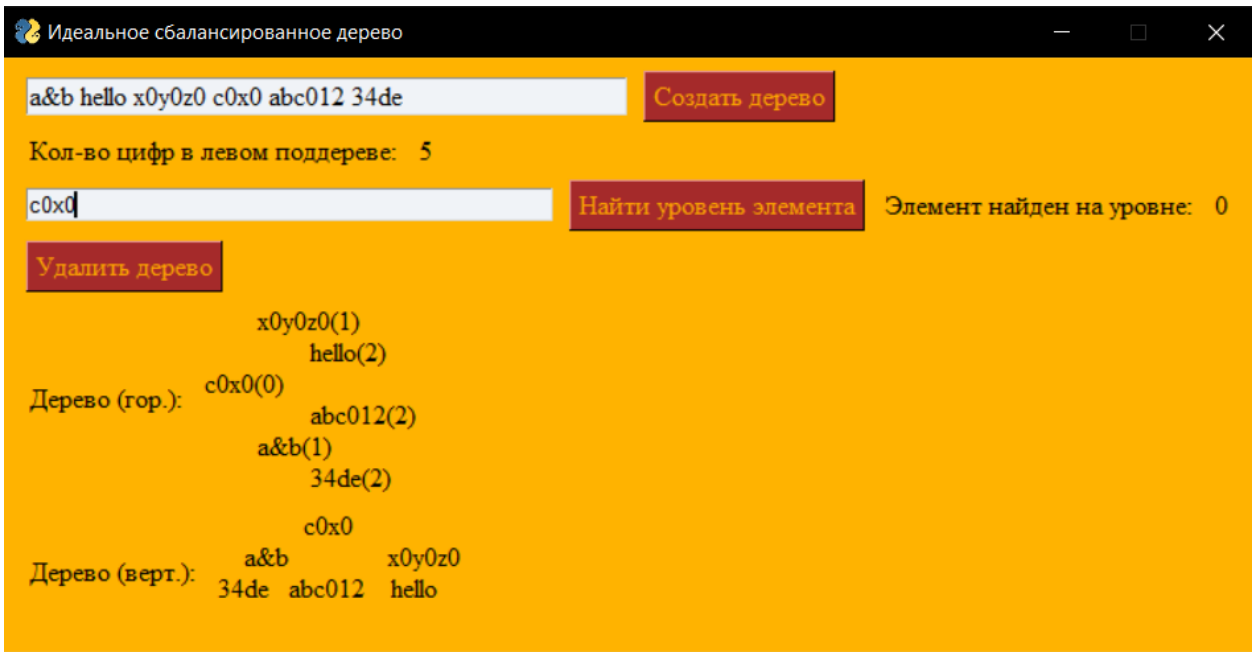

Рисунок 3 – Результаты тестирования.

Рисунок 4 – Результаты тестирования.

Элемент ‘ab’ не был найден в дереве, поэтому был выведен ‘–’.

Ответы на вопросы

1. Что определяет степень дерева?

Степень дерева – максимальная степень его узлов.

Степень узла – число непосредственных потомков.

2. Какова степень сильноветвящегося дерева?

Степень сильноветвящегося дерева больше второй.

3. Что определяет путь в дереве?

Путь в дереве – последовательность узлов от корня к указанному узлу.

4. Как рассчитать длину пути в дереве?

Длина пути в дереве равно сумме длин всех его ребер.

5. Какова степень бинарного дерева?

Степень бинарного дерева не больше второй.

6. Может ли дерево быть пустым?

Дерево – совокупность узлов, один из которых корень, и отношений, образующих иерархическую структуру, поэтому дерево может быть:

- пустым;
- состоящим из одного узла, который является корнем своего поддерева;
- связанным с несколькими узлами – корнями деревьев (поддеревьев данного дерева).

7. Дайте определение бинарного дерева.

Бинарное (двоичное) дерево – дерево, степень которого не больше двух.

8. Дайте определение алгоритму обхода.

Обход дерева – алгоритм, обеспечивающий посещение каждого узла дерева с целью выполнить операцию с данными текущего узла.

9. Приведите рекуррентную зависимость для вычисления высоты дерева:

$$Hight(T) = \begin{cases} -1 \text{ если } T \text{ пусто} \\ 1 + \max(Hight(LeftTree(T)), Hight(RightTree(T))) \text{ иначе} \end{cases}$$

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

Таблица 3

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

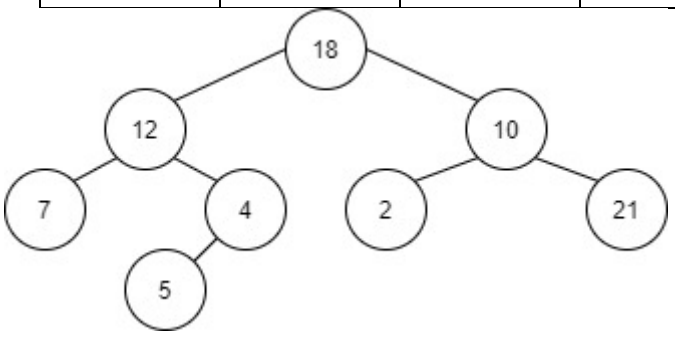

Рисунок 6 – Дерево по таблице 3

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

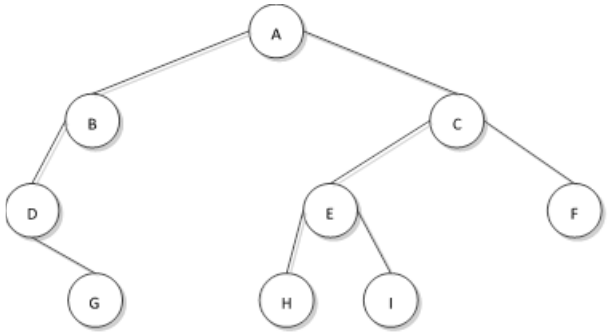

Рисунок 7 – Дерево по 11 вопросу

Прямой: ABDGCEHIF.

Обратный: GDBHIEFCA.

Симметричный: GDBAHEICF.

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

В алгоритме обхода дерева методом в «ширину» используется структура «очередь».

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

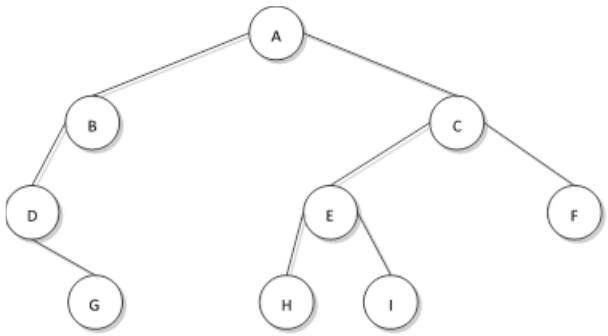

Рисунок 8 – Дерево по 13 вопросу

1) В очередь Q помещается первый узел с меткой A.

Q: A

2) Из очереди извлекается узел A для обработки и в очередь помещаются его сыновья.

Q: BC Вывод A

3) Из очереди извлекается узел Вдля обработки и в очередь помещаются его сыновья.

Q: C D Вывод B

4) Из очереди извлекается узел C для обработки и в очередь помещаются его сыновья.

Q:DEF Вывод C

5) Из очереди извлекается узел D для обработки и в очередь помещаются его сыновья.

Q: EFG Вывод D

6) Из очереди извлекается узел E для обработки и в очередь помещаются его сыновья.

Q: FG HI Вывод E

7) Из очереди извлекается узел F для обработки и в очередь должны помещаться его сыновья, но их нет, значит ничего не помещается.

Q: G HI Вывод F

8) Из очереди извлекается узел G для обработки и в очередь должны помещаться его сыновья, но их нет, значит ничего не помещается.

Q: HI Вывод G

9) Из очереди извлекается узел H для обработки и в очередь должны помещаться его сыновья, но их нет, значит ничего не помещается.

Q: I Вывод H

10) Из очереди извлекается узел I для обработки и в очередь должны помещаться его сыновья, но их нет, значит ничего не помещается.

Q: очередь пуста, дерево пройдено Вывод I

Путь обхода в «ширину»: ABCDEFGHI.

14. Какая структура используется в не рекурсивном обходе дерева методов в «глубину»?

Используется структура «стек».

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

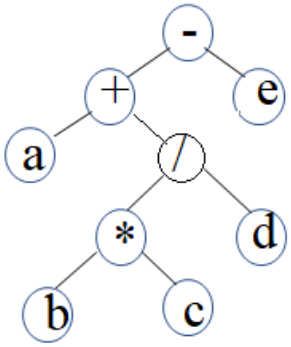

Рисунок 9 – Дерево по 15 вопросу

Прямой: – + a / * b c d e.

Обратный: abc * d / + e –.

Симметричный: a + b * c / d – e.

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

1) a + b – c * d + e

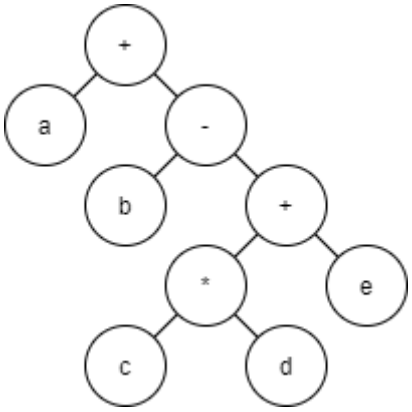

Рисунок 10 – Дерево по 16 вопросу. Выражение 1

2) / a – b * c d

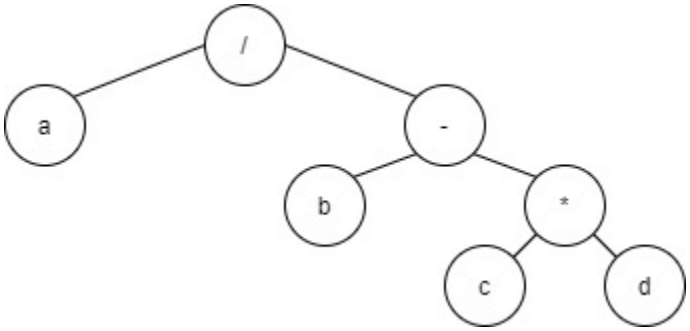

Рисунок 11 – Дерево по вопросу 16. Выражение 2

3) a b c d / – *

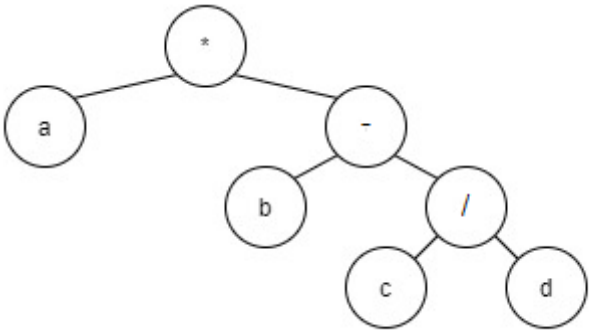

Рисунок 12 – Дерево по вопросу 16. Выражение 3

4) * – / + a b c d e

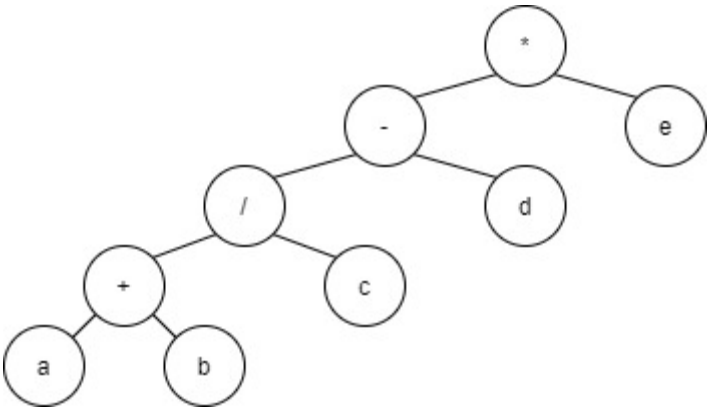

Рисунок 13 – Дерево по вопросу 16. Выражение 4

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

В таком случае будет использован метод обхода в «глубину».

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

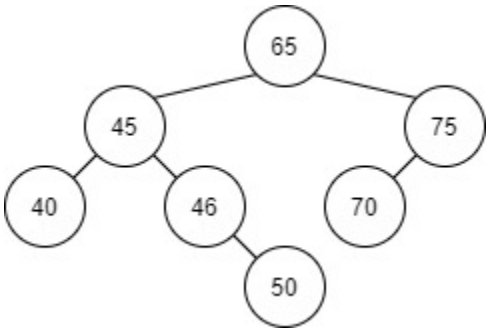

Рисунок 14 – Дерево по 18 вопросу

19. Приведенная ниже последовательность получена путем прямого обхода бинарного дерева поиска. Постройте это дерево.

50 45 35 15 40 46 65 75 70

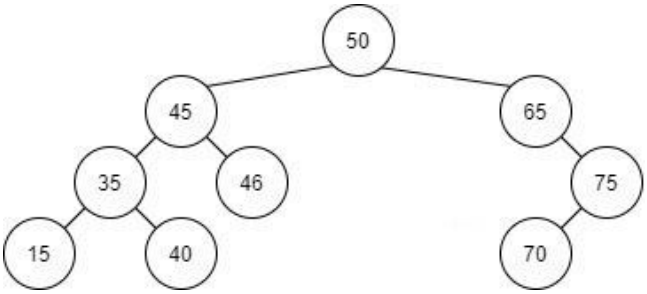

Рисунок 15 – Дерево по 19 вопросу

20. Дано бинарное дерево поиска, представленное на рисунке 21. Выполните действия **над исходным** дерево и покажите дерево:

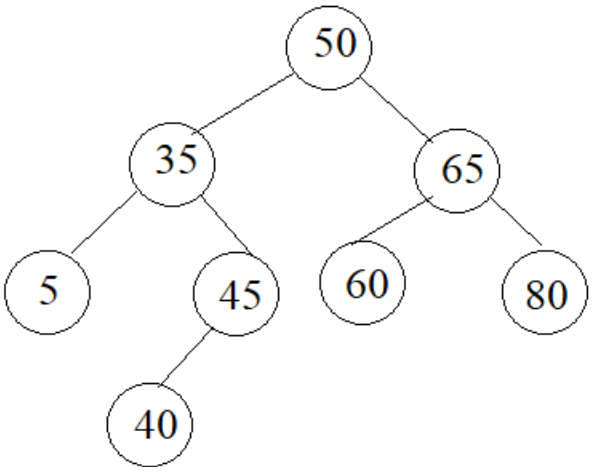

Рисунок 16 – Исходное бинарное дерево поиска по 20 вопросу

1) После включения узлов 1, 48, 75, 100

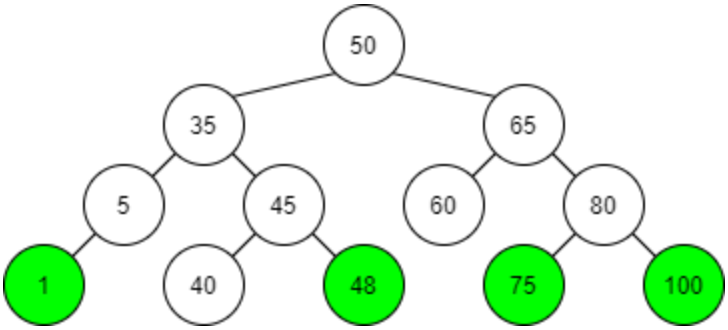

Рисунок 17 – Дерево поиска по 20 вопросу

2) После удаления узлов 5, 35

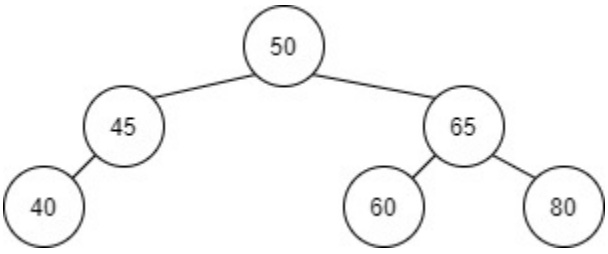

Рисунок 18 – Дерево поиска по 20 вопросу

3) После удаления узла 45

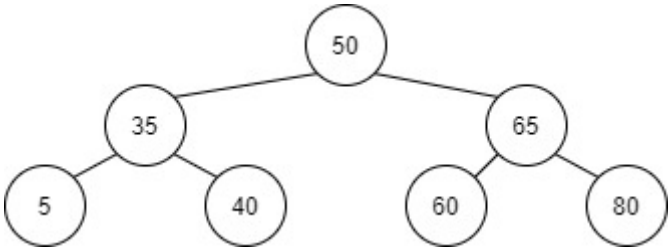

Рисунок 19 – Дерево поиска по 20 вопросу

4) После удаления узла 50

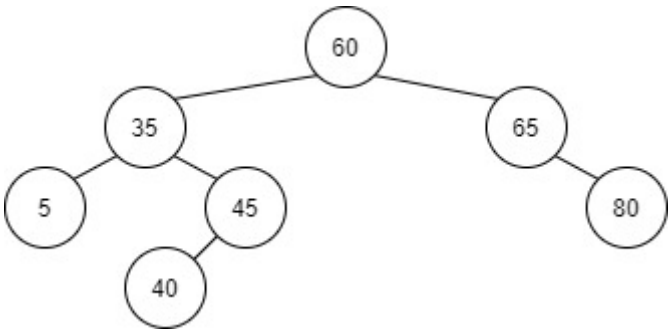

Рисунок 20 – Дерево поиска по 20 вопросу

5) После удаления узлов 5, 35

Ответ показан на рисунке 18

6) После удаления узла 65 и вставки его снова

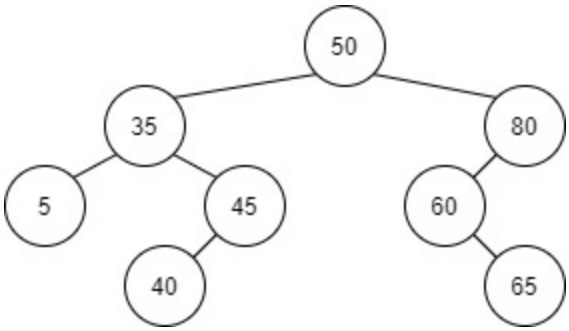

Рисунок 21 – Дерево поиска по 20 вопросу

7) После удаления узлов 5, 35

Ответ на рисунке 18

Вывод

В ходе работы были приобретены умения и навыки разработки и реализации операций над структурой данных «бинарное дерево».

Список информационных источников

1. Лекции по дисциплине «Структуры и алгоритмы обработки данных» / Л. А. Скворцова, МИРЭА – Российский технологический университет, 2021.