Федеральное государственное образовательное бюджетное учреждение

высшего образования

**«Финансовый университет**

**при Правительстве Российской Федерации»**

**(Финансовый университет)**

**Департамент анализа данных, принятия решений и финансовых технологий**

**Практическая работа по дисциплине «Алгоритмы и структуры данных в языке Python» №6**

**Выполнила:**

Студентка группы ПИ19-4

Жевагина А. В.

**Проверил:**

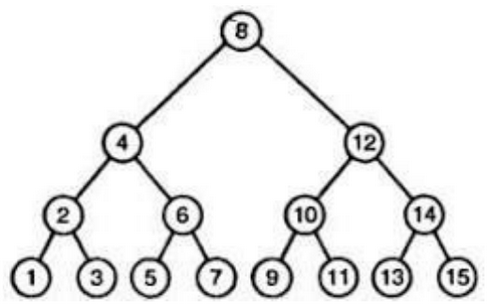
Петросов Д. А.

**Москва 2020**

# Задание №1

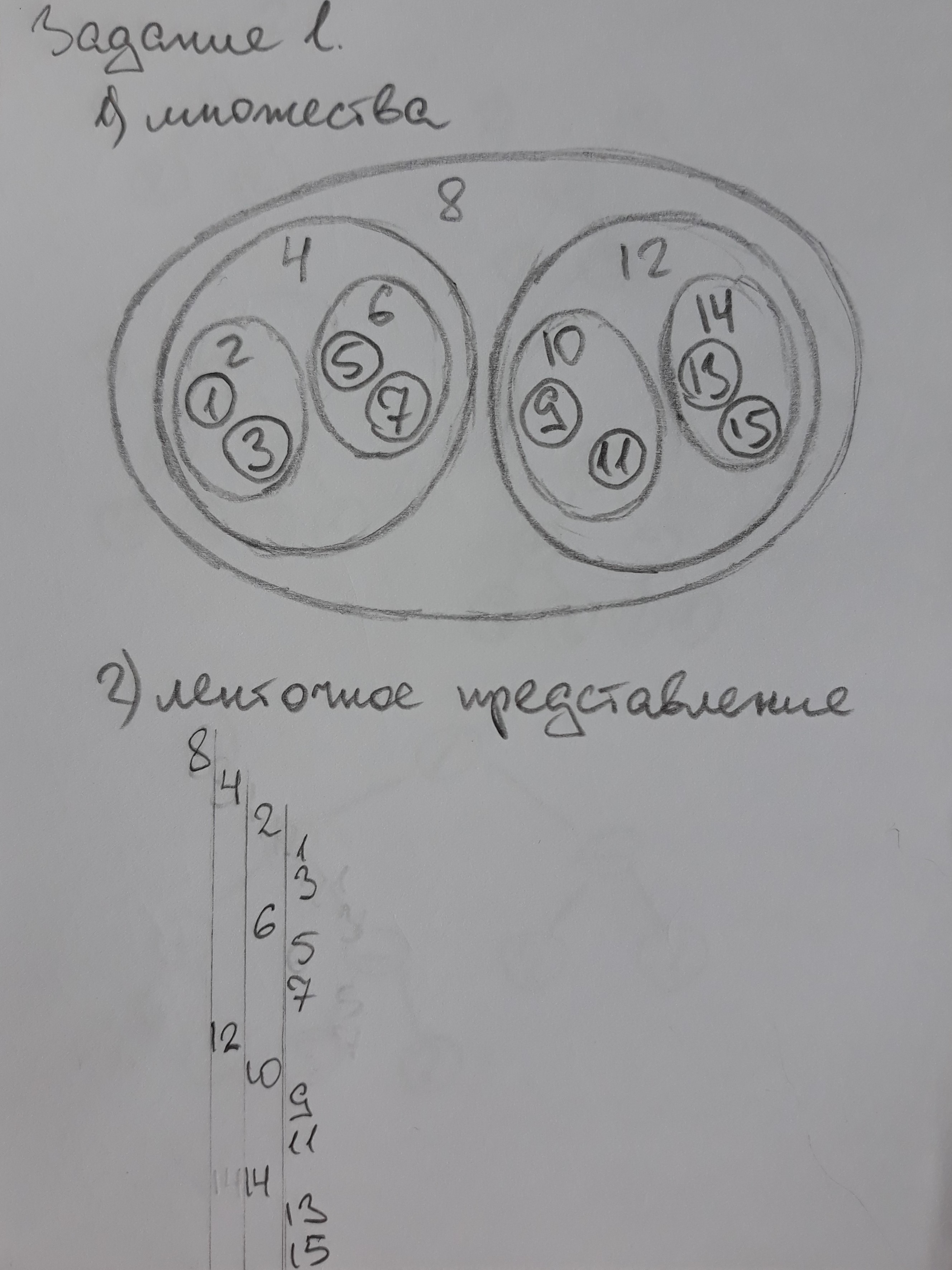
## Условие:

Выполнить представление через множества и ленточное представления бинарного дерева, представленного на рис. 1



*Рисунок 1*

Выполнение:



# Задание №2

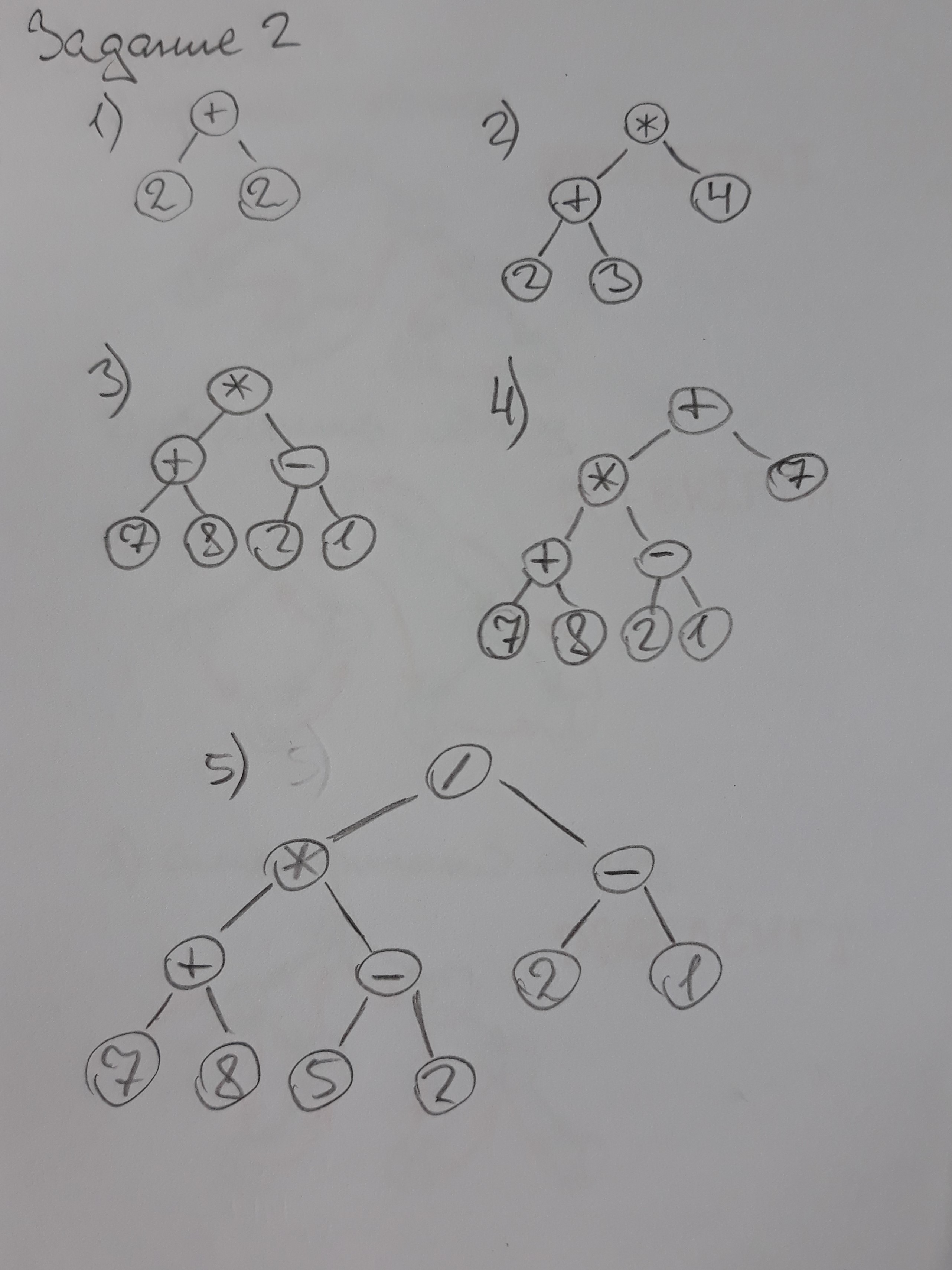
## Условие:

Выполнить графовое представление и программную реализацию с помощью

бинарного дерева следующие вычисления:

1. 2+2
2. (2+3)\*4
3. (7+8)\*(2-1)
4. (7+8)\*(2-1)+7
5. (7+8)\*(5-2)/(2-1)

## Выполнение:



## Исходный код программы:

class Binary\_Tree:

    def \_\_init\_\_(self, value):

        self.root = value

        self.left = None

        self.right = None

    # Задание 2

    def \_processing(self, inp):

        flag = True

        if len(inp) == 1:

            self.root = inp[0]

        else:

            for i in range(len(inp)):

                if inp[i] in ['+', '-'] and (inp[0:i].count('(') == inp[0:i].count(')')):

                    flag = False

                    self.root = inp[i]

                    self.left = Binary\_Tree(None)

                    self.left.\_processing(inp[0:i])

                    self.right = Binary\_Tree(None)

                    self.right.\_processing(inp[i+1:])

            if flag:

                for i in range(len(inp)):

                    if inp[i] in ['\*', '/'] and (inp[0:i].count('(') == inp[0:i].count(')')):

                        flag = False

                        self.root = inp[i]

                        self.left = Binary\_Tree(None)

                        self.left.\_processing(inp[0:i])

                        self.right = Binary\_Tree(None)

                        self.right.\_processing(inp[i+1:])

            if flag:

                inp = inp[1:len(inp)-1]

                self.\_processing(inp)

    def tree\_creating(self, s):

        inp = []

        for i in s:

            inp.append(i)

        i = 0

        while i < len(inp)-1:

            try:

                int(inp[i])

                int(inp[i+1])

            except:

                i+=1

            else:

                inp[i] = str(inp[i])+str(inp[i])

                inp.pop[i+1]

        self.\_processing(inp)

    def calculating(self):

        if self.left == None and self.right == None:

            return (self.root)

        else:

            return (str(eval(self.left.calculating()+self.root+self.right.calculating())))

    def get\_right(self):

        return self.right

    def get\_left(self):

        return self.left

    def set\_root\_val(self, obj):

        self.root = obj

    def get\_root\_val(self):

        return self.root

    def \_\_str\_\_(self):

        return '{} ({}, {})'.format(self.get\_root\_val(), str(self.get\_left()), str(self.get\_right()))

if \_\_name\_\_=='\_\_main\_\_':

    print('ЗАДАНИЕ 2')

    # 2+2

    print('1) 2+2')

    Tree2\_1 = Binary\_Tree(None)

    Tree2\_1.tree\_creating('2+2')

    print(Tree2\_1)

    print(Tree2\_1.calculating())

    # (2+3)\*4

    print('2) (2+3)\*4')

    Tree2\_2 = Binary\_Tree(None)

    Tree2\_2.tree\_creating('(2+3)\*4')

    print(Tree2\_2)

    print(Tree2\_2.calculating())

    # (7+8)\*(2-1)

    print('3) (7+8)\*(2-1)')

    Tree2\_3 = Binary\_Tree(None)

    Tree2\_3.tree\_creating('(7+8)\*(2-1)')

    print(Tree2\_3)

    print(Tree2\_3.calculating())

    # (7+8)\*(2-1)+7

    print('4) (7+8)\*(2-1)+7')

    Tree2\_4 = Binary\_Tree(None)

    Tree2\_4.tree\_creating('(7+8)\*(2-1)+7')

    print(Tree2\_4)

    print(Tree2\_4.calculating())

    # (7+8)\*(5-2)/(2-1)

    print('5) (7+8)\*(5-2)/(2-1)')

    Tree2\_5 = Binary\_Tree(None)

    Tree2\_5.tree\_creating('(7+8)\*(5-2)/(2-1)')

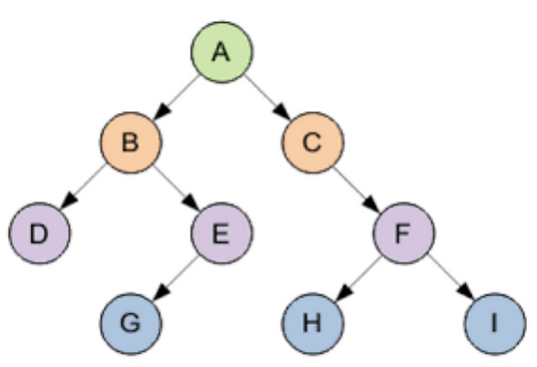
    print(Tree2\_5)

    print(Tree2\_5.calculating())

# Задание №3

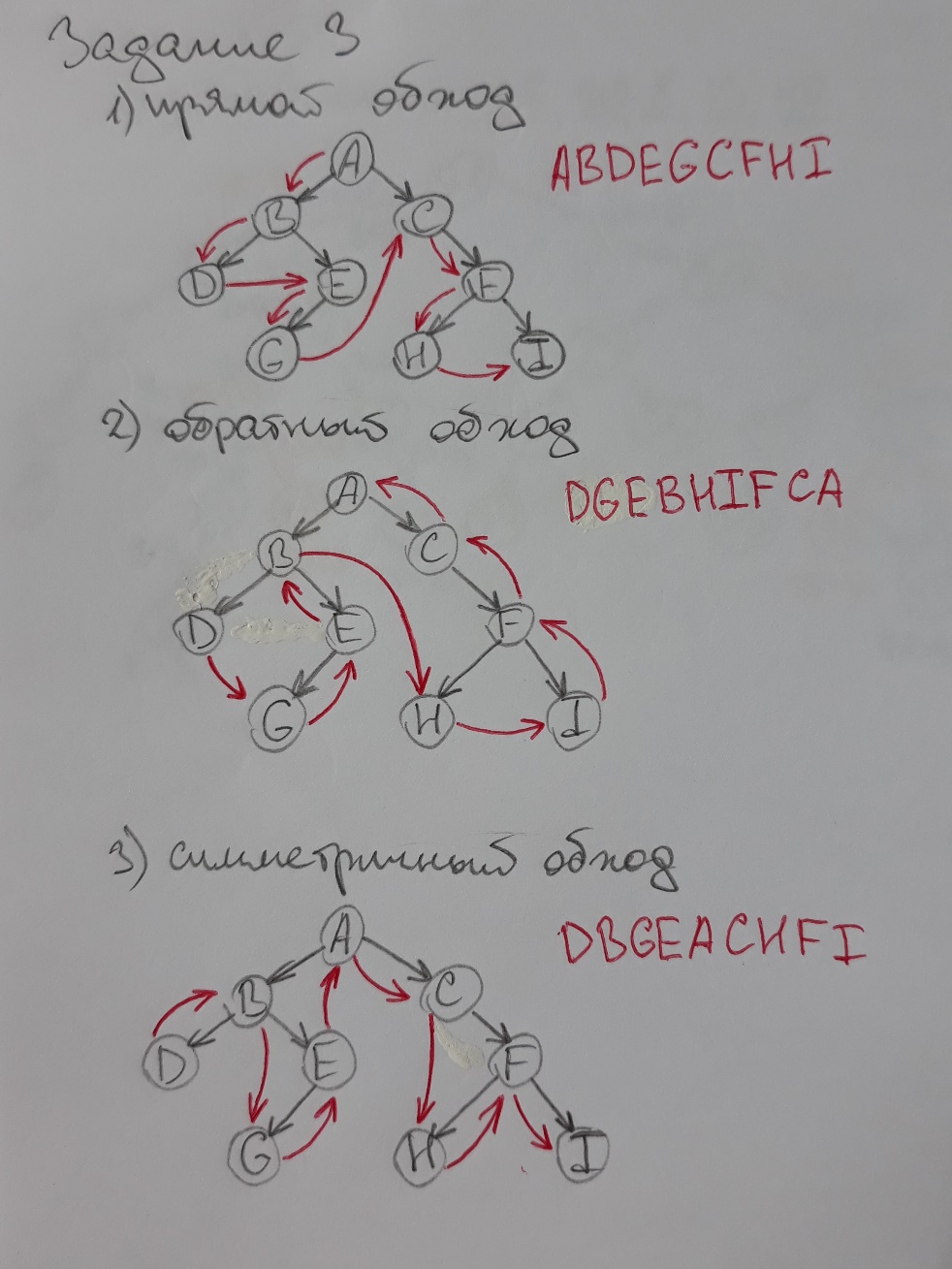
## Условие:

1. Выполнить прямой обход (в ручном режиме и программную реализацию) бинарного дерева, представленного на рисунке 2
2. Выполнить обратный обход (в ручном режиме и программную реализацию) бинарного дерева, представленного на рисунке 2
3. Выполнить симметричный обход (в ручном режиме и программную реализацию) бинарного дерева, представленного на рисунке 2



*Рисунок 2*

## Выполнение:



## Исходный код программы:

class Binary\_Tree:

    def \_\_init\_\_(self, value):

        self.root = value

        self.left = None

        self.right = None

    # Задание 3

    def direct\_bypass(self):

        print(self.root, end='')

        if self.left!=None:

            self.left.direct\_bypass()

        if self.right!=None:

            self.right.direct\_bypass()

    def reverse\_bypass(self):

        if self.left!=None:

            self.left.reverse\_bypass()

        if self.right!=None:

            self.right.reverse\_bypass()

        print(self.root, end='')

    def symmetrical\_bypass(self):

        if self.left!=None:

            self.left.symmetrical\_bypass()

        print(self.root, end='')

        if self.right!=None:

            self.right.symmetrical\_bypass()

    def \_\_str\_\_(self):

        return '{} ({}, {})'.format(self.get\_root\_val(), str(self.get\_left()), str(self.get\_right()))

if \_\_name\_\_=='\_\_main\_\_':

    print('\nЗАДАНИЕ 3')

    # Заполнение  дерева

    Tree3 = Binary\_Tree('A')

    Tree3.left = Binary\_Tree('B')

    Tree3.left.left = Binary\_Tree('D')

    Tree3.left.right = Binary\_Tree('E')

    Tree3.left.right.left = Binary\_Tree('G')

    Tree3.right = Binary\_Tree('C')

    Tree3.right.right = Binary\_Tree('F')

    Tree3.right.right.left = Binary\_Tree('H')

    Tree3.right.right.right = Binary\_Tree('I')

    # Обходы

    print('Прямой обход:')

    Tree3.direct\_bypass()

    print('\nОбратный обход:')

    Tree3.reverse\_bypass()

    print('\nСимметричный обход:')

    Tree3.symmetrical\_bypass()

# Задание №4

## Условие:

Выполнить программную реализацию и сравнительный анализ

быстродействия поиска элементов с помощью бинарного дерева в заданном

массиве элементов: 81 77 79 68 10 12 13 20 15 24 27 42 33 51 57

Для этого:

1. Реализовать представление данных с помощью бинарного дерева;
2. Реализовать поиск элементов в массиве с учетом времени на поиск
3. Реализовать поиск элемента в бинарном дереве с учетом времени на поиск
4. Реализовать поиск элементов в отсортированном массиве с учетом времени на поиск
5. Провести сравнительный анализ представления данных в трех программных реализациях.

## Исходный код программы:

import array

import copy

import timeit

class Binary\_Tree:

    def \_\_init\_\_(self, value):

        self.root = value

        self.left = None

        self.right = None

    # Задание 4

    def tree\_from\_list(self, i, arr):

            self.root = arr[i]

            if i<(len(arr)//2):

                self.left = Binary\_Tree(None)

                self.right = Binary\_Tree(None)

                self.left.tree\_from\_list(i\*2+1, arr)

                self.right.tree\_from\_list(i\*2+2, arr)

    def direct\_serch(self, inp):

        if self.root == inp:

            print('Нашли', end = '')

        if self.left!=None:

            self.left.direct\_serch(inp)

        if self.right!=None:

            self.right.direct\_serch(inp)

    def get\_right(self):

        return self.right

    def get\_left(self):

        return self.left

    def set\_root\_val(self, obj):

        self.root = obj

    def get\_root\_val(self):

        return self.root

    def \_\_str\_\_(self):

        return '{} ({}, {})'.format(self.get\_root\_val(), str(self.get\_left()), str(self.get\_right()))

if \_\_name\_\_=='\_\_main\_\_':

    print('\n\nЗАДАНИЕ 4')

    # Создание

    arr = array.array('i', [81, 77, 79, 68, 10, 12, 13, 20, 15, 24, 27, 42, 33, 51, 57])

    print('Заданный массив:', arr)

    sorted\_arr = sorted(copy.copy(arr))

    print('Отсортированный массив:', sorted\_arr)

    Tree4 = Binary\_Tree(None)

    Tree4.tree\_from\_list(0, arr)

    print('Дерево:', Tree4)

    # Поиск

    inp = int(input('Введите, что искать будем: '))

    if inp in arr:

        a = timeit.default\_timer()

        i = arr.index(inp)

        print('Поиск в заданном массиве: Номер элемента - {}, Время - {}'.format(i+1, timeit.default\_timer() - a))

        a = timeit.default\_timer()

        i = sorted\_arr.index(inp)

        print('Поиск в отсортированном массиве: Номер элемента - {}, Время - {}'.format(i+1, timeit.default\_timer() - a))

        i = 1

        print('Поиск в дереве: Элемент - ', end = '')

        a = timeit.default\_timer()

        i = Tree4.direct\_serch(inp)

        print(', Время - {}'.format(timeit.default\_timer() - a))

    else:

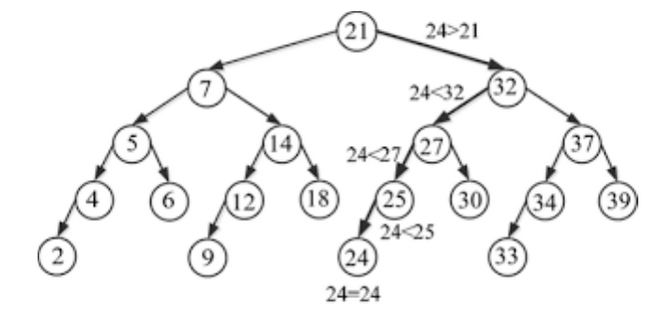
        print('Такого элемента нет')

# Задание №5

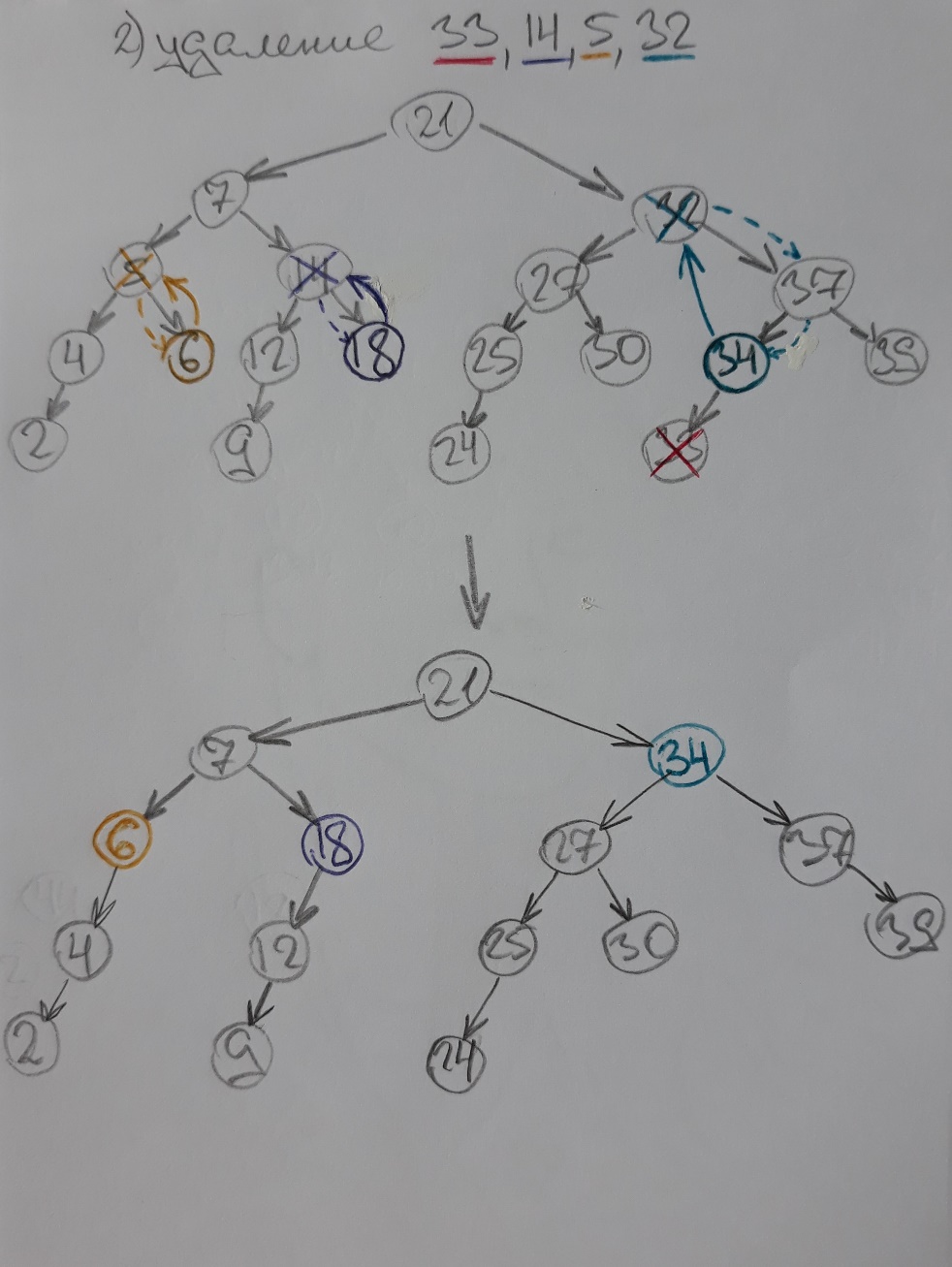
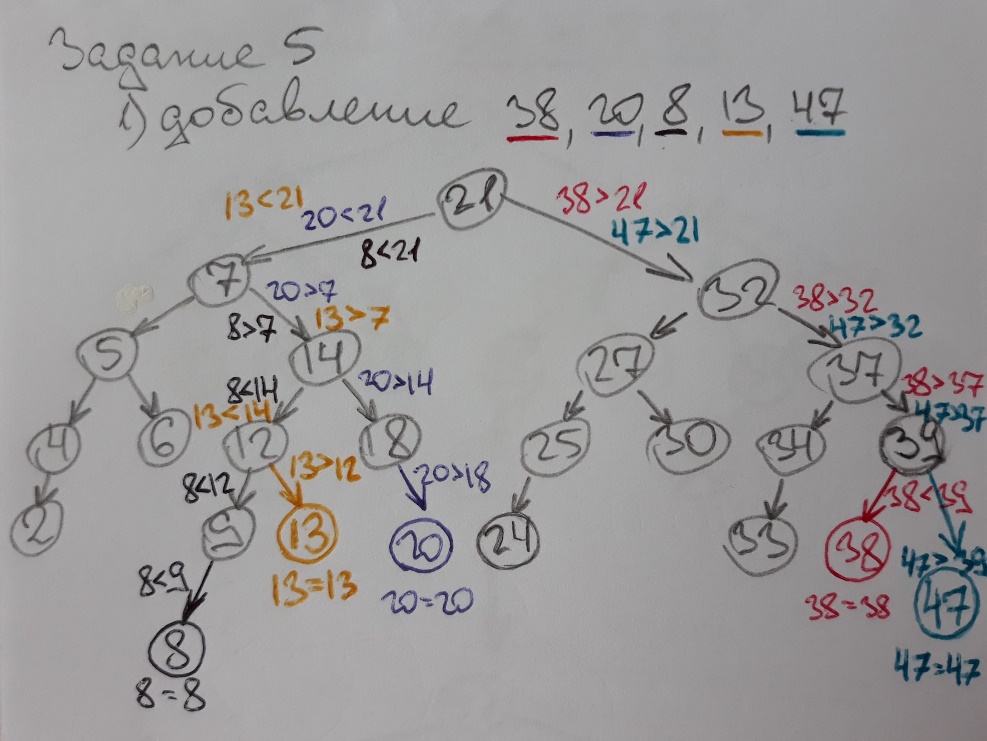
## Условие:

В заданном на рисунке 3 бинарном дереве реализовать:

1. графически для заданных элементов добавление чисел: 38, 20, 8, 13, 47
2. программную реализацию добавления элементов.
3. графически для заданных элементов удаления чисел: 33, 14, 5, 32
4. программную реализацию удаления элементов.

 *Рисунок 3*

## Выполнение:



## Исходный код программы:

import copy

class Binary\_Tree:

    def \_\_init\_\_(self, value):

        self.root = value

        self.left = None

        self.right = None

    # Задание 5

    def insert\_elements(self, a):

        if self.root == None:

            self.root = a

        elif a<self.root:

            if self.left == None:

                self.left = Binary\_Tree(None)

            self.left.insert\_elements(a)

        elif a>=self.root:

            if self.right == None:

                self.right = Binary\_Tree(None)

            self.right.insert\_elements(a)

    def \_searching(self, a):

        if self.left == None and self.right == None:

            b = self.root

            self.root = None

            return b

        elif self.left == None:

            b = self.root

            self.root = self.right

            return b

        else:

            return self.left.\_searching(a)

    def \_deleting(self, a):

        if self.left == None and self.right == None:

            self.root = None

        elif self.left == None or self.right == None:

            if self.left == None:

                self.root = self.right

            elif self.right == None:

                self.root = self.left

        else:

            self.root = self.right.\_searching(a)

    def delete\_elements(self, a):

        if self.root == a:

            self.\_deleting(a)

        elif a<self.root:

            self.left.delete\_elements(a)

        elif a>self.root:

            self.right.delete\_elements(a)

    def \_cleaning(self):

        if self.left != None:

            if self.left.root == None and self.left.left == None and self.left.right == None:

                self.left = None

            else:

                self.left.\_cleaning()

        if self.right != None:

            if self.right.root == None and self.right.left == None and self.right.right == None:

                self.right = None

            else:

                self.right.\_cleaning()

    def get\_right(self):

        return self.right

    def get\_left(self):

        return self.left

    def set\_root\_val(self, obj):

        self.root = obj

    def get\_root\_val(self):

        return self.root

    def \_\_str\_\_(self):

        return '{} ({}, {})'.format(self.get\_root\_val(), str(self.get\_left()), str(self.get\_right()))

if \_\_name\_\_=='\_\_main\_\_':

    print('\nЗАДАНИЕ 5')

    # Создание

    Tree5\_1 = Binary\_Tree(21)

    Tree5\_1.left = Binary\_Tree(7)

    Tree5\_1.left.left = Binary\_Tree(5)

    Tree5\_1.left.left.left = Binary\_Tree(4)

    Tree5\_1.left.left.left.left = Binary\_Tree(2)

    Tree5\_1.left.left.right = Binary\_Tree(6)

    Tree5\_1.left.right = Binary\_Tree(14)

    Tree5\_1.left.right.left = Binary\_Tree(12)

    Tree5\_1.left.right.left.left = Binary\_Tree(9)

    Tree5\_1.left.right.right = Binary\_Tree(18)

    Tree5\_1.right = Binary\_Tree(32)

    Tree5\_1.right.left = Binary\_Tree(27)

    Tree5\_1.right.left.left = Binary\_Tree(25)

    Tree5\_1.right.left.left.left = Binary\_Tree(24)

    Tree5\_1.right.left.right = Binary\_Tree(30)

    Tree5\_1.right.right = Binary\_Tree(37)

    Tree5\_1.right.right.left = Binary\_Tree(34)

    Tree5\_1.right.right.left.left = Binary\_Tree(33)

    Tree5\_1.right.right.right = Binary\_Tree(39)

    print('Исходное дерево:\n', Tree5\_1)

    Tree5\_2 = copy.deepcopy(Tree5\_1)

    # Вставка

    insert = [38, 20, 8, 13, 47]

    delete = [33, 14, 5, 32]

    for a in insert:

        Tree5\_1.insert\_elements(a)

    print('Вставка элементов (38, 20, 8, 13, 47):\n', Tree5\_1)

    for a in delete:

        Tree5\_2.delete\_elements(a)

        Tree5\_2.\_cleaning()

    print('Удаление элементов (33, 14, 5, 32):\n', Tree5\_2)