Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
дисциплины «Алгоритмизация»

Выполнил:
Зармухамбетов Булат Эльдарович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А., доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой_____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

1. Написал программу по задаче покрытия точек отрезками единичной длины двумя способами: обычным (pointscover1) и улучшенным (pointscover2) алгоритмами.

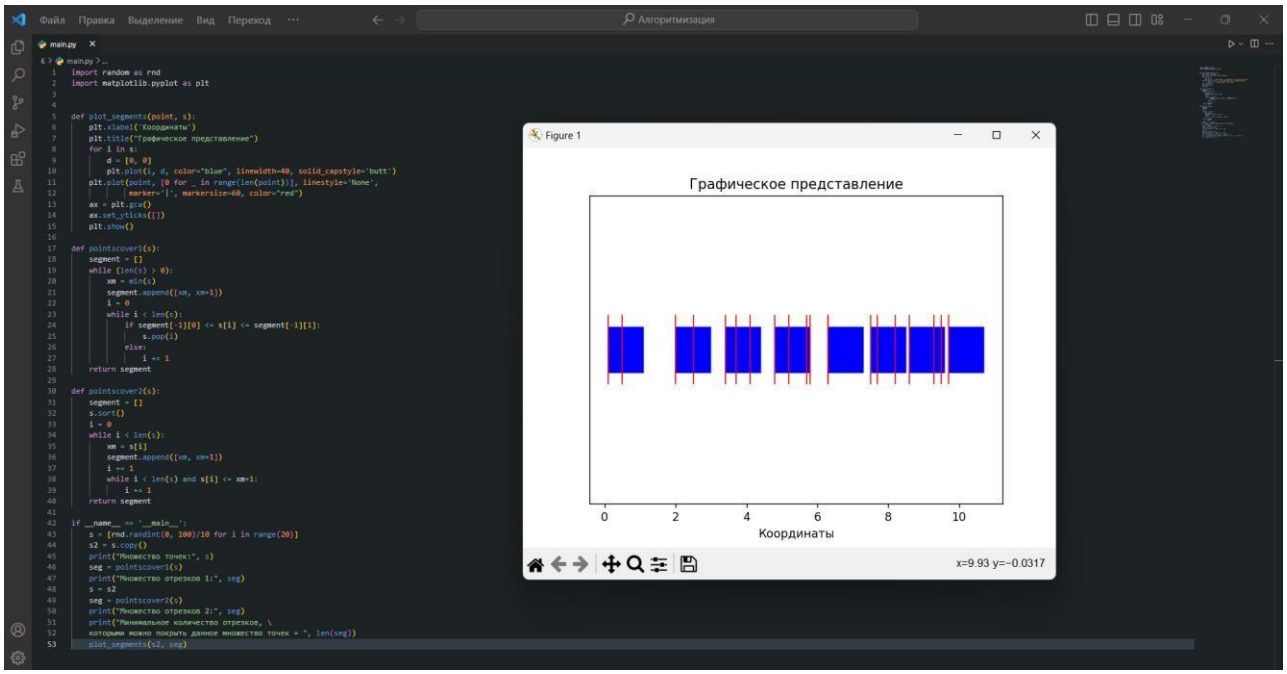
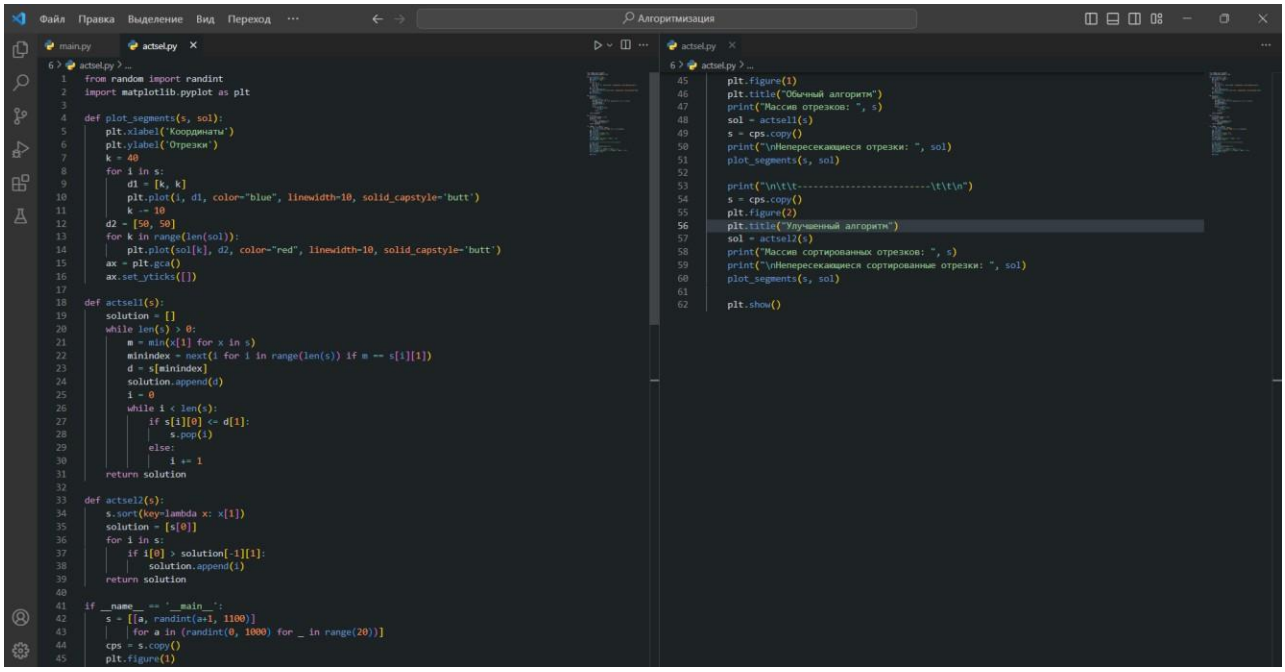


Рисунок 1. Код и результат работы программы main

```
PS C:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Алгоритмизация> python -u "c:\Users\dalam\OneDrive\Рабочий стол\
Множество точек: [2.0, 0.5, 5.7, 4.1, 5.8, 7.5, 3.4, 5.2, 7.7, 0.1, 3.7, 2.5, 9.3, 6.3, 9.7, 8.2, 6.3, 9.5, 8.6, 4.8]
Множество отрезков 1: [[0.1, 1.1], [2.0, 3.0], [3.4, 4.4], [4.8, 5.8], [6.3, 7.3], [7.5, 8.5], [8.6, 9.6], [9.7, 10.7]]
Множество отрезков 2: [[0.1, 1.1], [2.0, 3.0], [3.4, 4.4], [4.8, 5.8], [6.3, 7.3], [7.5, 8.5], [8.6, 9.6], [9.7, 10.7]]
Минимальное количество отрезков,     которыми можно покрыть данное множество точек =  8
```

Рисунок 2. Вывод программы main в терминал

2. Написал программу по задаче о выборе заявок, в которой требуется найти максимальное количество попарно не пересекающихся отрезков двумя способами: обычным (actsel1) и улучшенным (actsel2) алгоритмами.



```python
from random import randint
import matplotlib.pyplot as plt

def plot_segments(s, sol):
    plt.xlabel('Координаты')
    plt.ylabel('Отрезки')
    k = 40
    for i in s:
        d1 = [k, k]
        plt.plot(i, d1, color="blue", linewidth=10, solid_capstyle='butt')
        k -= 10
    d2 = [50, 50]
    for k in range(len(sol)):
        plt.plot(sol[k], d2, color="red", linewidth=10, solid_capstyle='butt')
    ax = plt.gca()
    ax.set_yticks([])

def actsel1(s):
    solution = []
    while len(s) > 0:
        m = min(x[1] for x in s)
        minindex = next(i for i in range(len(s)) if m == s[i][1])
        d = s[minindex]
        solution.append(d)
        i = 0
        while i < len(s):
            if s[i][0] <= d[1]:
                s.pop(i)
            else:
                i += 1
    return solution

def actsel2(s):
    s.sort(key=lambda x: x[1])
    solution = [s[0]]
    for i in s:
        if i[0] > solution[-1][1]:
            solution.append(i)
    return solution

if __name__ == '__main__':
    s = [[a, randint(a+1, 1100)]
         for a in (randint(0, 1000) for _ in range(20))]
    cps = s.copy()
    plt.figure(1)
    plt.figure(1)
    plt.title("Обычный алгоритм")
    print("Массив отрезков: ", s)
    sol = actsel1(s)
    s = cps.copy()
    print("\nНепересекающиеся отрезки: ", sol)
    plot_segments(s, sol)

    print("\n\t\t----------\t\t\n")
    s = cps.copy()
    plt.figure(2)
    plt.title("Улучшенный алгоритм")
    sol = actsel2(s)
    print("Массив сортированных отрезков: ", s)
    print("\nНепересекающиеся сортированные отрезки: ", sol)
    plot_segments(s, sol)

    plt.show()
```

Рисунок 3. Код программы ActSel

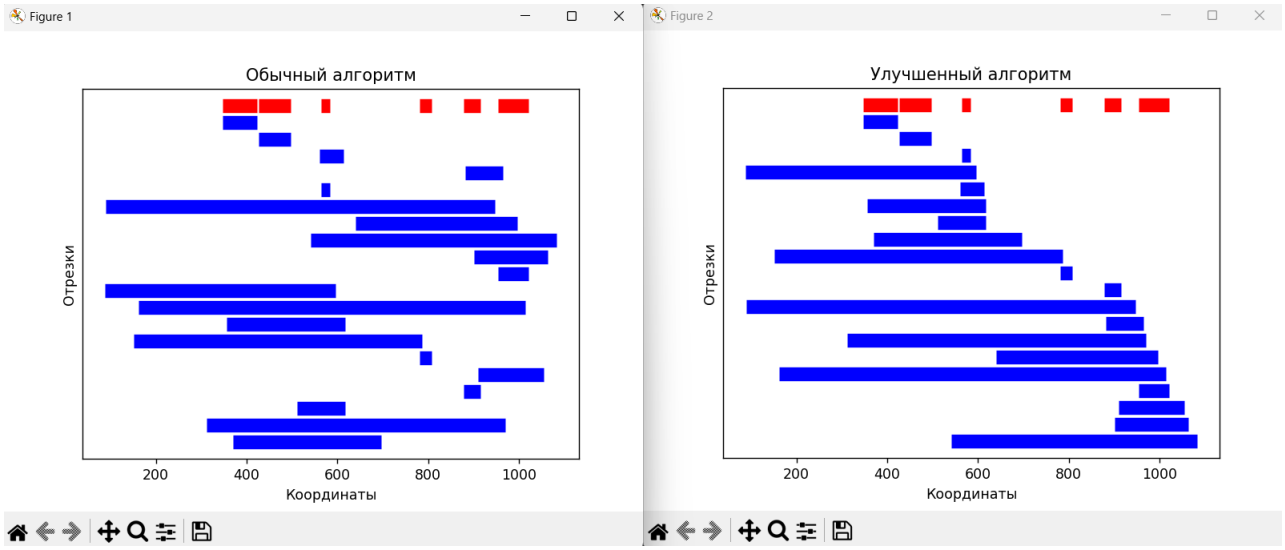
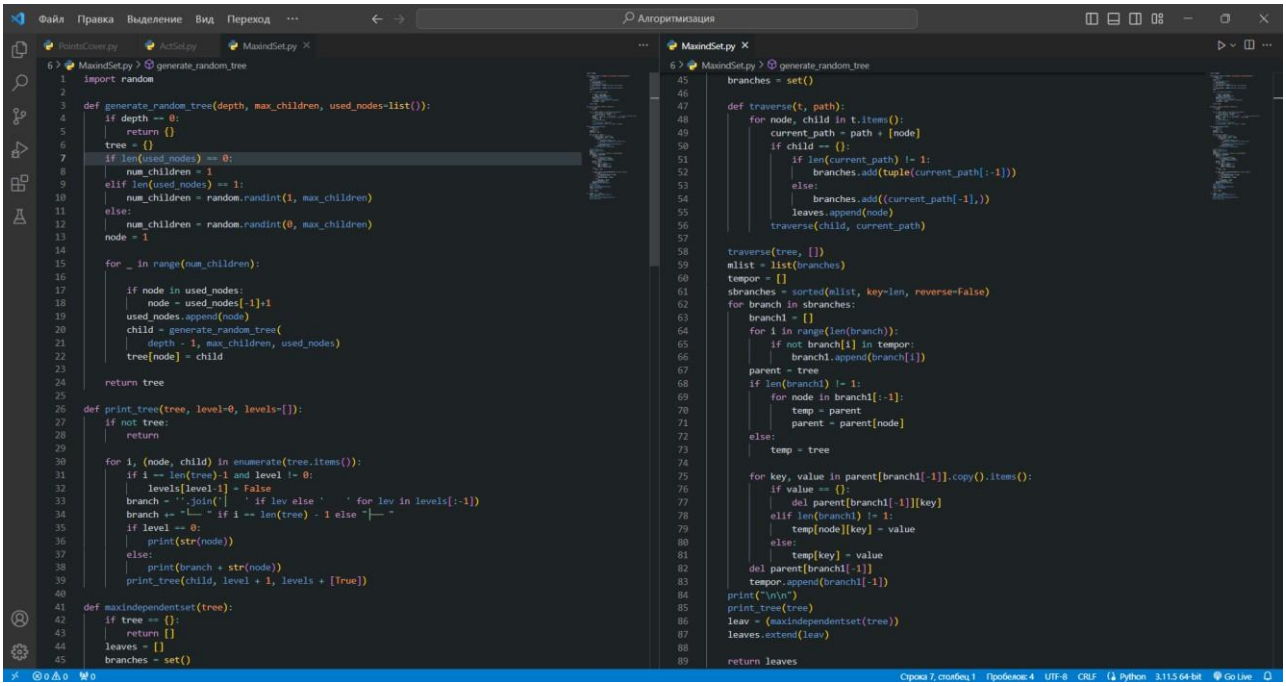


Рисунок 4. Графическое представление решения задачи обоих алгоритмов

```
PS C:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Алгоритмизация> python -u "c:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Алгоритмизация\6\ActSel.py"
Массив отрезков:  [[347, 423], [427, 498], [560, 613], [882, 965], [564, 583], [90, 948], [640, 997], [542, 1083], [901, 1064], [954, 1022], [88, 596], [162, 1014], [356, 617
12, 618], [311, 970], [371, 696]]

Непересекающиеся отрезки:  [[347, 423], [427, 498], [564, 583], [782, 808], [878, 915], [954, 1022]]

                ----------

Массив сортированных отрезков:  [[347, 423], [427, 498], [564, 583], [88, 596], [560, 613], [356, 617], [512, 618], [371, 696], [151, 787], [782, 808], [878, 915], [90, 948],
, 1022], [911, 1055], [901, 1064], [542, 1083]]

Непересекающиеся сортированные отрезки:  [[347, 423], [427, 498], [564, 583], [782, 808], [878, 915], [954, 1022]]
```

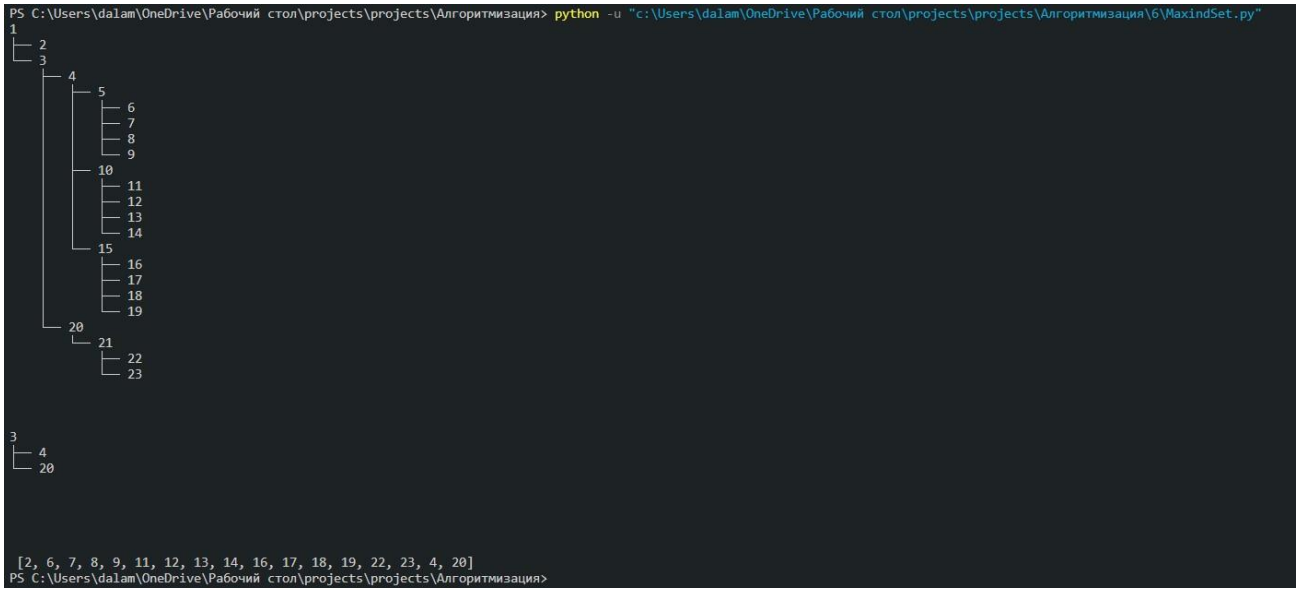
Рисунок 5. Вывод программы ActSel в терминал

3. Написал программу по задаче планирования вечеринки в кампании, в которой требуется по заданному дереву определить независимое множество (множество не соединённых друг с другом вершин) максимального размера.



```python
import random

def generate_random_tree(depth, max_children, used_nodes=list()):
    if depth == 0:
        return {}
    tree = {}
    if len(used_nodes) == 0:
        num_children = 1
    elif len(used_nodes) == 1:
        num_children = random.randint(1, max_children)
    else:
        num_children = random.randint(0, max_children)
    node = 1

    for _ in range(num_children):

        if node in used_nodes:
            node = used_nodes[-1]+1
        used_nodes.append(node)
        child = generate_random_tree(
            depth - 1, max_children, used_nodes)
        tree[node] = child

    return tree

def print_tree(tree, level=0, levels=[]):
    if not tree:
        return

    for i, (node, child) in enumerate(tree.items()):
        if i == len(tree)-1 and level != 0:
            levels[level-1] = False
        branch = ''.join('│   ' if lev else '    ' for lev in levels[:-1])
        branch += "└── " if i == len(tree) - 1 else "├── "
        if level == 0:
            print(str(node))
        else:
            print(branch + str(node))
        print_tree(child, level + 1, levels + [True])

def maxindependentset(tree):
    if tree == {}:
        return []
    leaves = []
    branches = set()

    def traverse(t, path):
        for node, child in t.items():
            current_path = path + [node]
            if child == {}:
                if len(current_path) != 1:
                    branches.add(tuple(current_path[:-1]))
                else:
                    branches.add((current_path[-1],))
                leaves.append(node)
            traverse(child, current_path)

    traverse(tree, [])
    mlist = list(branches)
    tempor = []
    sbranches = sorted(mlist, key=len, reverse=False)
    for branch in sbranches:
        branch1 = []
        for i in range(len(branch)):
            if not branch[i] in tempor:
                branch1.append(branch[i])
        parent = tree
        if len(branch1) != 1:
            for node in branch1[:-1]:
                temp = parent
                parent = parent[node]
        else:
            temp = tree

        for key, value in parent[branch1[-1]].copy().items():
            if value == {}:
                del parent[branch1[-1]][key]
            elif len(branch1) != 1:
                temp[node][key] = value
            else:
                temp[key] = value
        del parent[branch1[-1]]
        tempor.append(branch1[-1])
    print("\n\n")
    print_tree(tree)
    leav = (maxindependentset(tree))
    leaves.extend(leav)

    return leaves
```

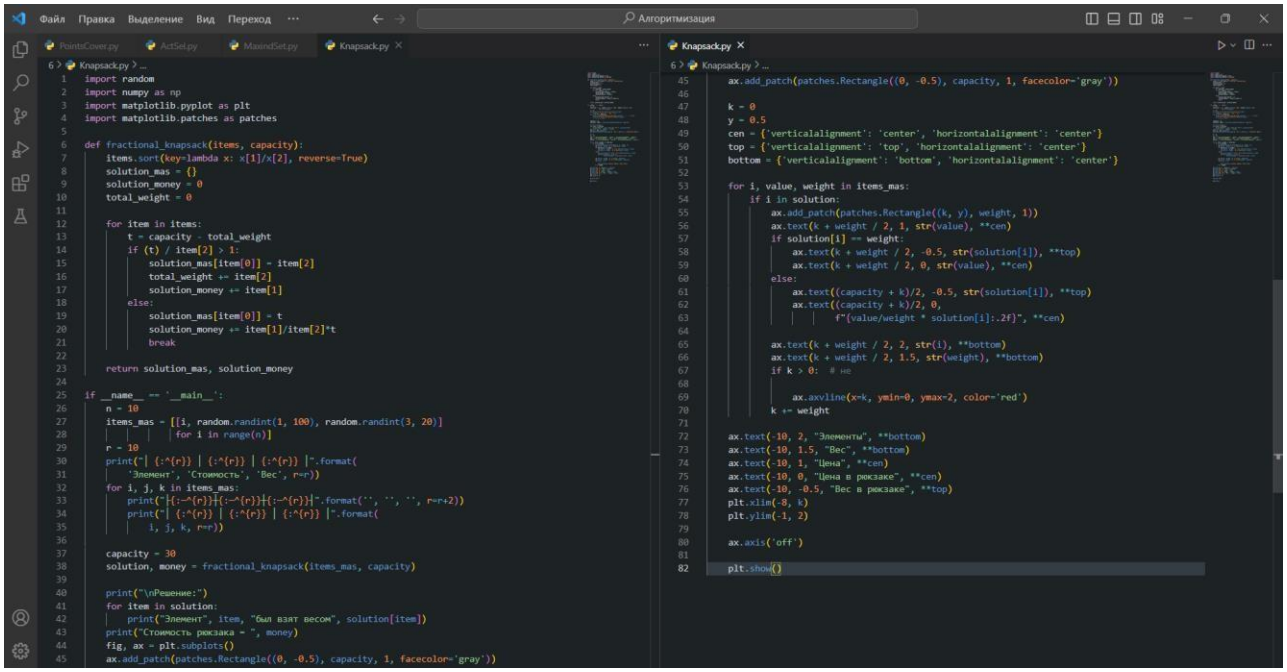
Рисунок 6. Код программы MaxindSet



```
PS C:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Алгоритмизация> python -u "c:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Алгоритмизация\6\MaxindSet.py"
1
├── 2
└── 3
    ├── 4
    │   ├── 5
    │   │   ├── 6
    │   │   ├── 7
    │   │   ├── 8
    │   │   └── 9
    │   └── 10
    │       ├── 11
    │       ├── 12
    │       ├── 13
    │       └── 14
    └── 15
        ├── 16
        ├── 17
        ├── 18
        └── 19
    └── 20
        └── 21
            ├── 22
            └── 23



3
├── 4
└── 20


 [2, 6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 18, 19, 22, 23, 4, 20]
PS C:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Алгоритмизация>
```

Рисунок 7. Вывод программы MaxindSet в терминал

4. Написал программу по задаче о непрерывном рюкзаке, в которой требуется частями предметов с весами w_i, стоимостями c_i набрать рюкзак фиксированного размера на максимальную стоимость.



```python
import random
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches

def fractional_knapsack(items, capacity):
    items.sort(key=lambda x: x[1]/x[2], reverse=True)
    solution_mas = {}
    solution_money = 0
    total_weight = 0

    for item in items:
        t = capacity - total_weight
        if (t) / item[2] > 1:
            solution_mas[item[0]] = item[2]
            total_weight += item[2]
            solution_money += item[1]
        else:
            solution_mas[item[0]] = t
            solution_money += item[1]/item[2]*t
            break

    return solution_mas, solution_money

if __name__ == '__main__':
    n = 10
    items_mas = [[i, random.randint(1, 100), random.randint(3, 20)]
                 for i in range(n)]
    r = 10
    print("| {:^{r}} | {:^{r}} | {:^{r}} |".format(
        'Элемент', 'Стоимость', 'Вес', r=r))
    for i, j, k in items_mas:
        print("|{:-^{r}}|{:-^{r}}|{:-^{r}}|".format('', '', '', r=r+2))
        print("| {:^{r}} | {:^{r}} | {:^{r}} |".format(
            i, j, k, r=r))

    capacity = 30
    solution, money = fractional_knapsack(items_mas, capacity)

    print("\nРешение:")
    for item in solution:
        print("Элемент", item, "был взят весом", solution[item])
    print("Стоимость рюкзака = ", money)
    fig, ax = plt.subplots()
    ax.add_patch(patches.Rectangle((0, -0.5), capacity, 1, facecolor='gray'))
```

```python
    ax.add_patch(patches.Rectangle((0, -0.5), capacity, 1, facecolor='gray'))

    k = 0
    y = 0.5
    cen = {'verticalalignment': 'center', 'horizontalalignment': 'center'}
    top = {'verticalalignment': 'top', 'horizontalalignment': 'center'}
    bottom = {'verticalalignment': 'bottom', 'horizontalalignment': 'center'}

    for i, value, weight in items_mas:
        if i in solution:
            ax.add_patch(patches.Rectangle((k, y), weight, 1))
            ax.text(k + weight / 2, 1, str(value), **cen)
            if solution[i] == weight:
                ax.text(k + weight / 2, -0.5, str(solution[i]), **top)
                ax.text(k + weight / 2, 0, str(value), **cen)
            else:
                ax.text((capacity + k)/2, -0.5, str(solution[i]), **top)
                ax.text((capacity + k)/2, 0,
                        f"{value/weight * solution[i]:.2f}", **cen)

            ax.text(k + weight / 2, 2, str(i), **bottom)
            ax.text(k + weight / 2, 1.5, str(weight), **bottom)
            if k > 0:  # не

                ax.axvline(x=k, ymin=0, ymax=2, color='red')
            k += weight

    ax.text(-10, 2, "Элементы", **bottom)
    ax.text(-10, 1.5, "Вес", **bottom)
    ax.text(-10, 1, "Цена", **cen)
    ax.text(-10, 0, "Цена в рюкзаке", **cen)
    ax.text(-10, -0.5, "Вес в рюкзаке", **top)
    plt.xlim(-8, k)
    plt.ylim(-1, 2)

    ax.axis('off')

    plt.show()
```

Рисунок 8. Код программы Knapsack

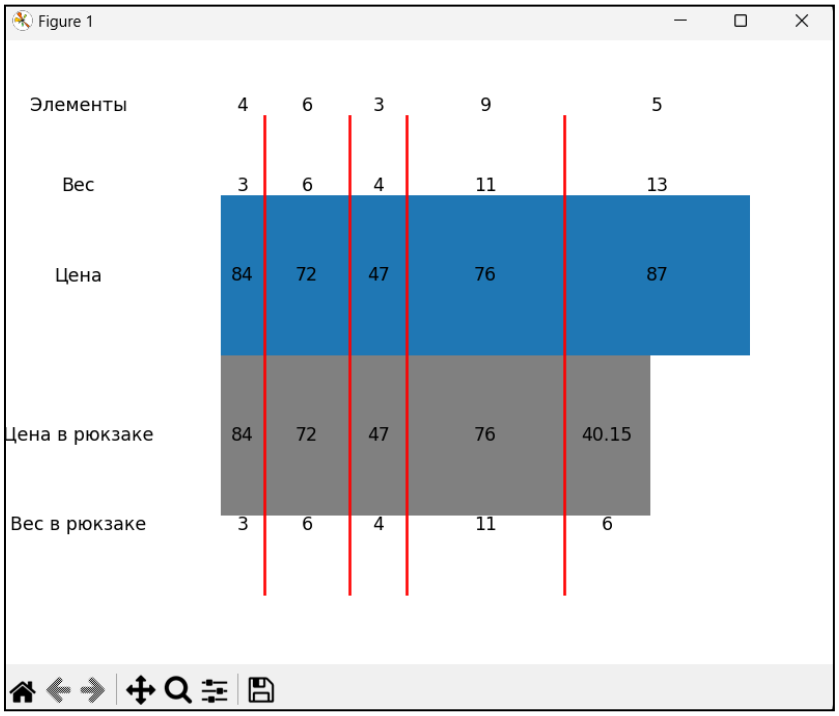


Рисунок 9. Графическое представление решения

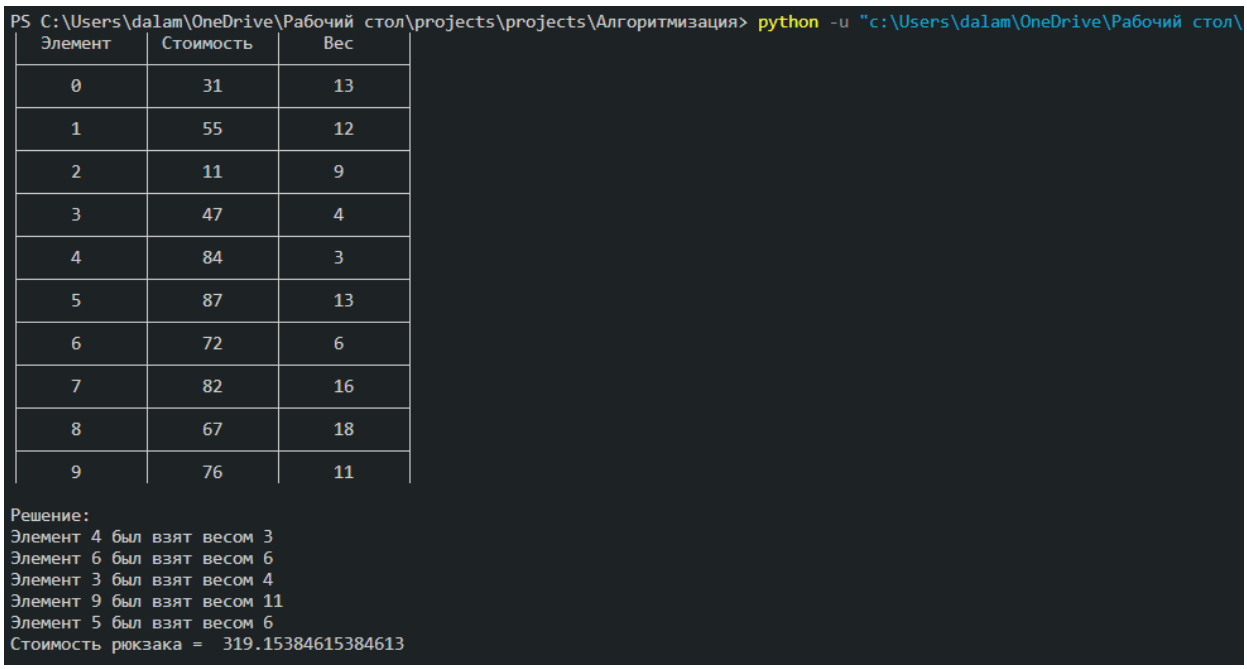


Рисунок 10. Вывод программы Knapsack в консоль

В ходе выполнения лабораторной работы были исследованы некоторые из примеров жадных алгоритмов, решающих такие задачи как: задача о покрытии точек минимальным количеством отрезков, задача о нахождении максимального количества попарно непересекающихся отрезков и др.