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:
Середа Кирилл Витальевич
1 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Ход выполнения заданий

1) Написал программу по задаче покрытия точек отрезками единичной длины двумя способами

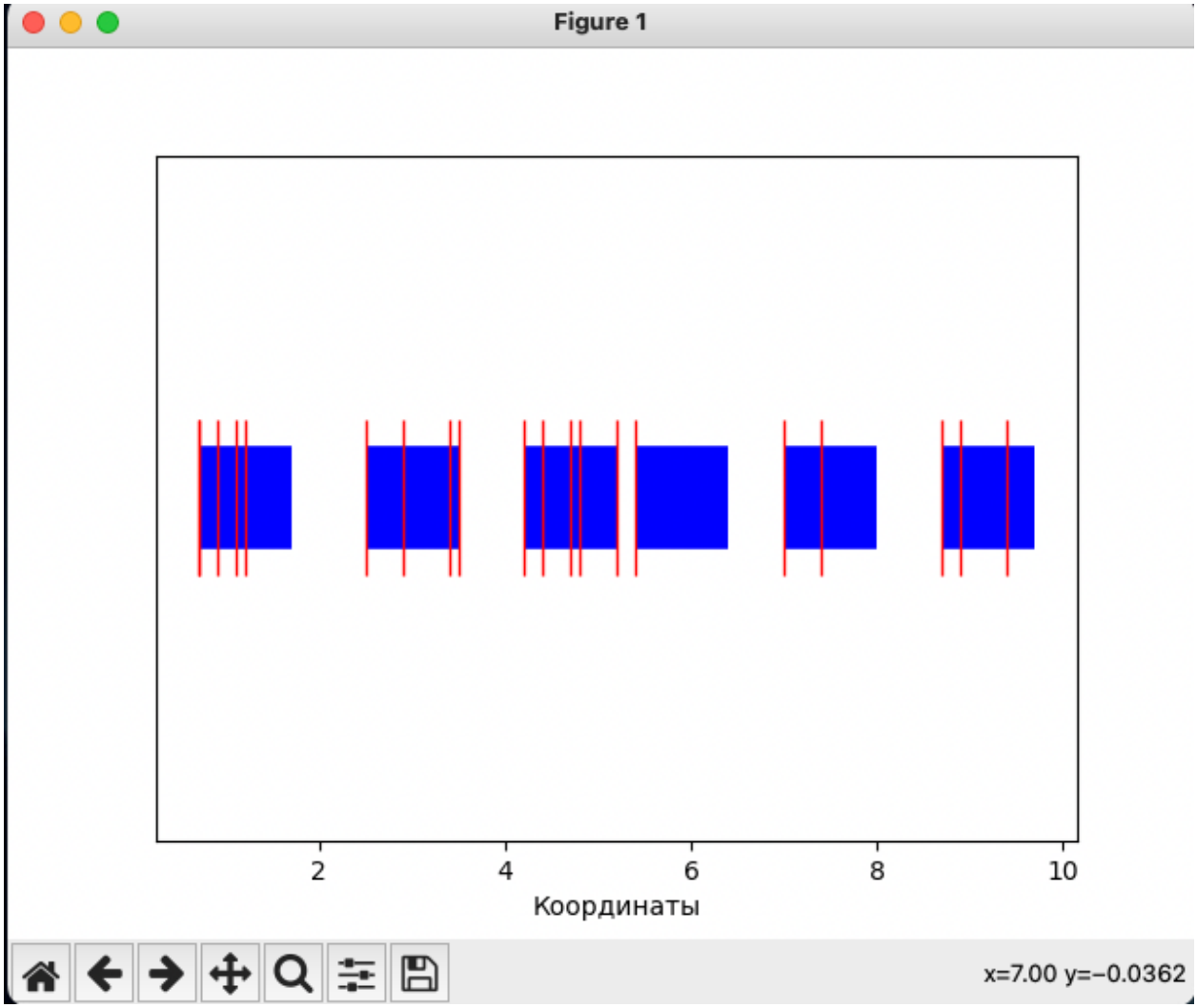

Рисунок 1 – Графическое представление

```
Множество точек: [2.9, 4.8, 5.2, 9.4, 1.1, 8.9, 0.7, 4.2, 3.4, 1.2, 2.5, 5.4, 7.4, 3.5, 8.7, 0.7, 0.9, 7.0, 4.7, 4.4]
Множество отрезков 1: [[0.7, 1.7], [2.5, 3.5], [4.2, 5.2], [5.4, 6.4], [7.0, 8.0], [8.7, 9.7]]
Множество отрезков 2: [[0.7, 1.7], [2.5, 3.5], [4.2, 5.2], [5.4, 6.4], [7.0, 8.0], [8.7, 9.7]]
Минимальное количество отрезков, которыми можно покрыть данное множество точек =  6
```

Рисунок 2 – Полученные множества

Код программы:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import random as rnd
import matplotlib.pyplot as plt


def visualize_segments(points, segments):
```

```
    plt.xlabel('Координаты')
    for segment in segments:
        plt.plot(segment, [0, 0], color="blue", linewidth=40,
solid_capstyle='butt')
    plt.plot(points, [0 for _ in range(len(points))], linestyle='None',
             marker='|', markersize=60, color="red")
    # Убрать ось y
    ax = plt.gca()
    ax.set_yticks([])
    plt.show()


def find_segments_method1(points):
    result_segments = []
    while len(points) > 0:
        x_min = min(points)
        result_segments.append([x_min, x_min + 1])
        i = 0
        while i < len(points):
            if result_segments[-1][0] <= points[i] <= result_segments[-1][1]:
                points.pop(i)
            else:
                i += 1
    return result_segments


def find_segments_method2(points):
    result_segments = []
    points.sort()
    i = 0
    while i < len(points):
        x_min = points[i]
        result_segments.append([x_min, x_min + 1])
        i += 1
        while i < len(points) and points[i] <= x_min + 1:
            i += 1
    return result_segments


if __name__ == '__main__':
    input_points = [rnd.randint(0, 100) / 10 for _ in range(20)]
    original_points = input_points.copy()
    print("Множество точек:", input_points)

    segments_method1 = find_segments_method1(input_points)
    print("Множество отрезков 1:", segments_method1)

    input_points = original_points
    segments_method2 = find_segments_method2(input_points)
    print("Множество отрезков 2:", segments_method2)

    print("Минимальное количество отрезков, "
          "которыми можно покрыть данное множество точек = ",
len(segments_method2))

    visualize_segments(original_points, segments_method2)
```

2) Написал программу по задаче о выборе заявок, в которой требуется найти максимальное количество попарно не пересекающихся отрезков двумя способами

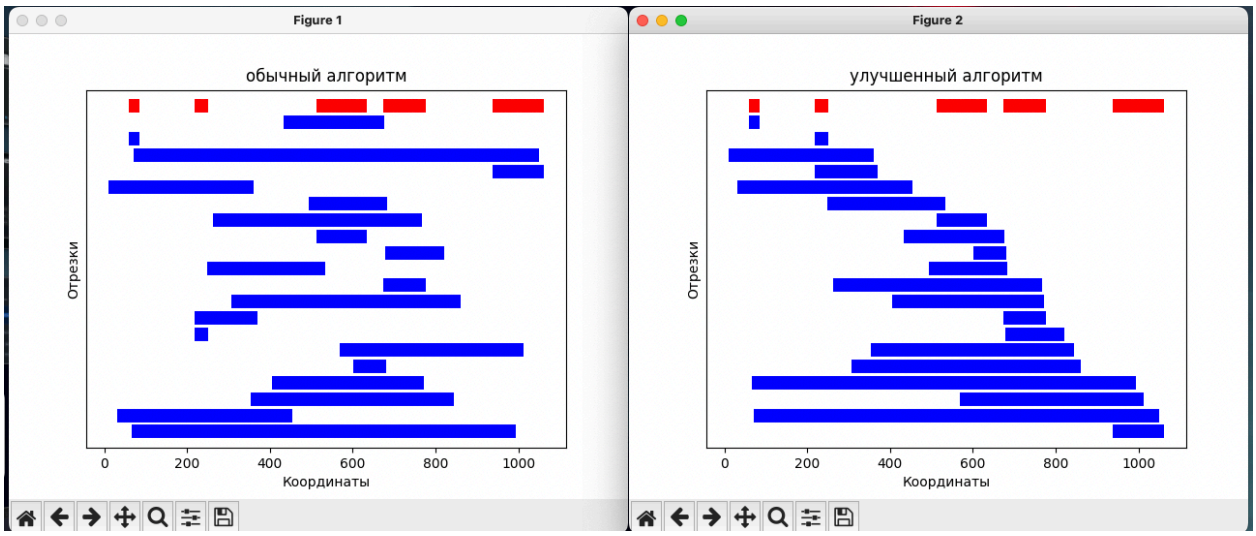

Рисунок 3 – Графики

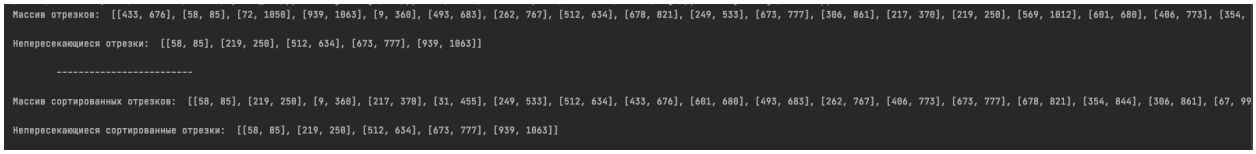

Рисунок 4 – Полученный данные

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from random import randint
import matplotlib.pyplot as plt


def visualize_solution(segments, selected_segments):
    plt.xlabel('Координаты')
    plt.ylabel('Отрезки')
    offset1 = 40
    for segment in segments:
        plt.plot(segment, [offset1, offset1], color="blue", linewidth=10,
solid_capstyle='butt')
        offset1 -= 10
    offset2 = 50
    for selected_segment in selected_segments:
        plt.plot(selected_segment, [offset2, offset2], color="red",
linewidth=10, solid_capstyle='butt')
    ax = plt.gca()
    ax.set_yticks([])


def select_segments_method1(segments):
    solution = []
    while len(segments) > 0:
        min_right = min(x[1] for x in segments)
        min_index = next(i for i in range(len(segments)) if min_right ==
segments[i][1])
        current_segment = segments[min_index]
        solution.append(current_segment)
        i = 0
        while i < len(segments):
            if segments[i][0] <= current_segment[1]:
```

```
                segments.pop(i)
            else:
                i += 1
    return solution


def select_segments_method2(segments):
    segments.sort(key=lambda x: x[1])
    solution = [segments[0]]
    for segment in segments:
        if segment[0] > solution[-1][1]:
            solution.append(segment)
    return solution


if __name__ == '__main__':
    segments = [[a, randint(a+1, 1100)] for a in (randint(0, 1000) for _ in
range(20))]
    original_segments = segments.copy()

    plt.figure(1)
    plt.title("обычный алгоритм")
    print("Массив отрезков: ", segments)
    selected_segments_method1 = select_segments_method1(segments)
    segments = original_segments.copy()
    print("\nНепересекающиеся отрезки: ", selected_segments_method1)
    visualize_solution(segments, selected_segments_method1)

    print("\n\t\t------------------------\t\t\n")
    segments = original_segments.copy()
    plt.figure(2)
    plt.title("улучшенный алгоритм")
    selected_segments_method2 = select_segments_method2(segments)
    print("Массив сортированных отрезков: ", segments)
    print("\nНепересекающиеся сортированные отрезки: ",
selected_segments_method2)
    visualize_solution(segments, selected_segments_method2)

    plt.show()
```

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

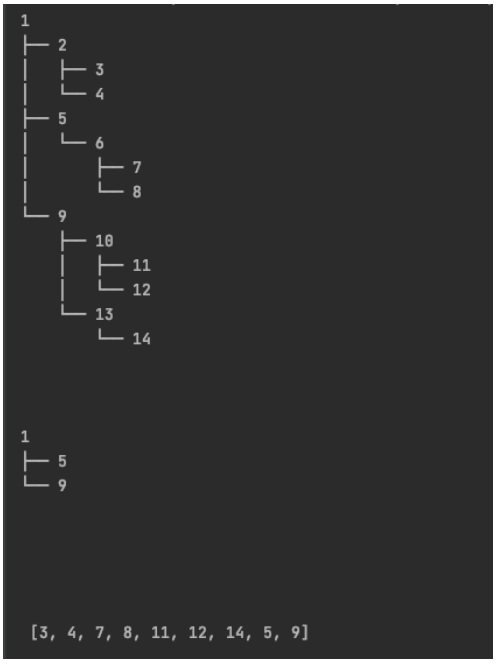

Рисунок 5 – Результат работы программы

Код программы:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import random


def generate_random_tree_structure(depth, max_children, used_nodes=None):
    if used_nodes is None:
        used_nodes = list()
    if depth == 0:
        return {}

    tree = {}

    if not used_nodes:
        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_structure(
            depth - 1, max_children, used_nodes)

        tree[node] = child

    return tree


def display_tree_structure(tree, level=0, levels=None):
    if levels is None:
```

```
        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))
        display_tree_structure(child, level + 1, levels + [True])


def find_maximum_independent_set(tree):
    global temp, node
    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 branch[i] not 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")
    display_tree_structure(tree)
    leav = find_maximum_independent_set(tree)
```

```
    leaves.extend(leav)

    return leaves


if __name__ == '__main__':
    random_tree = generate_random_tree_structure(4, 3)
    display_tree_structure(random_tree)
    print("\n\n", find_maximum_independent_set(random_tree))
    display_tree_structure(random_tree)
```

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

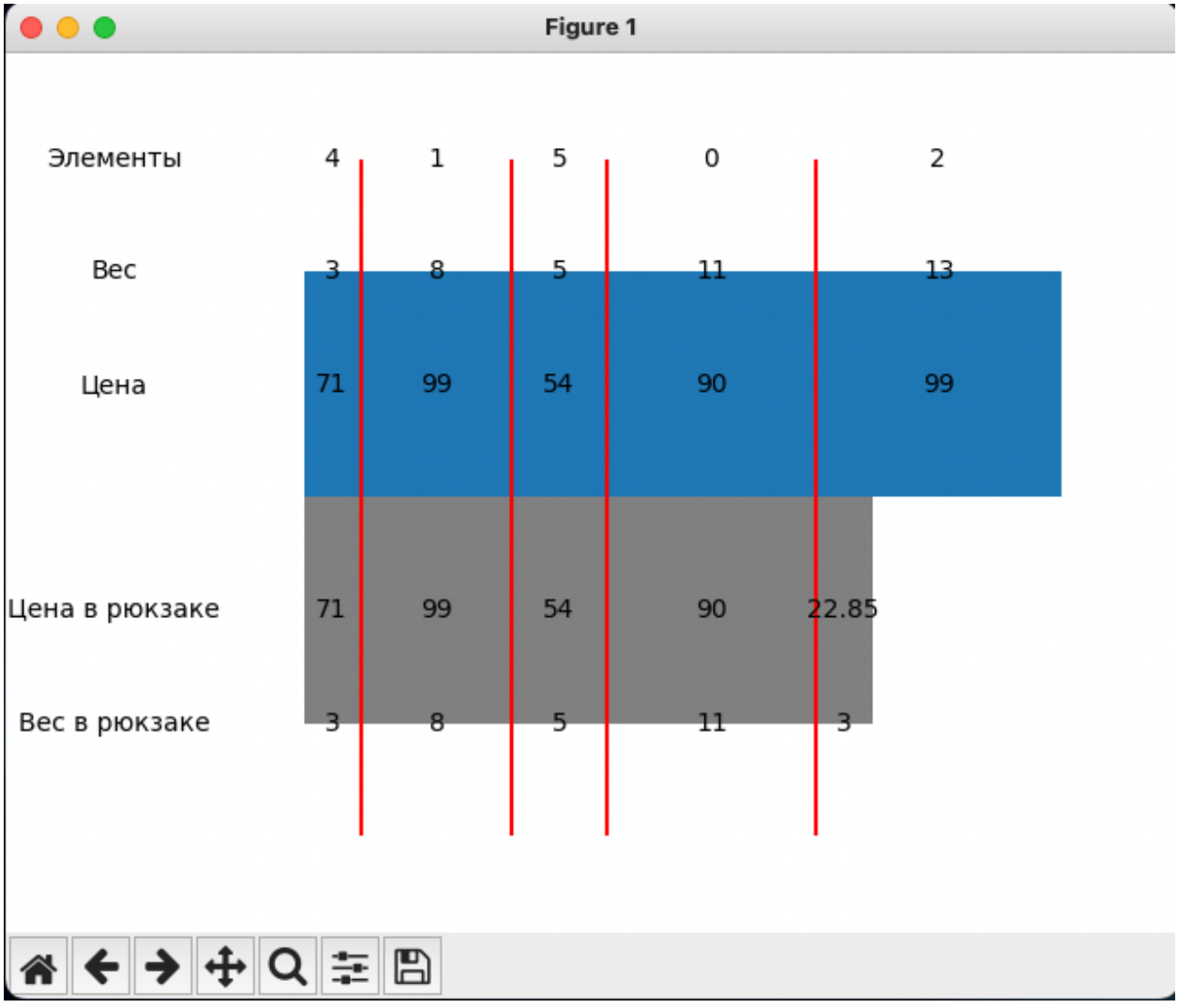

Рисунок 6 – Графическое представление

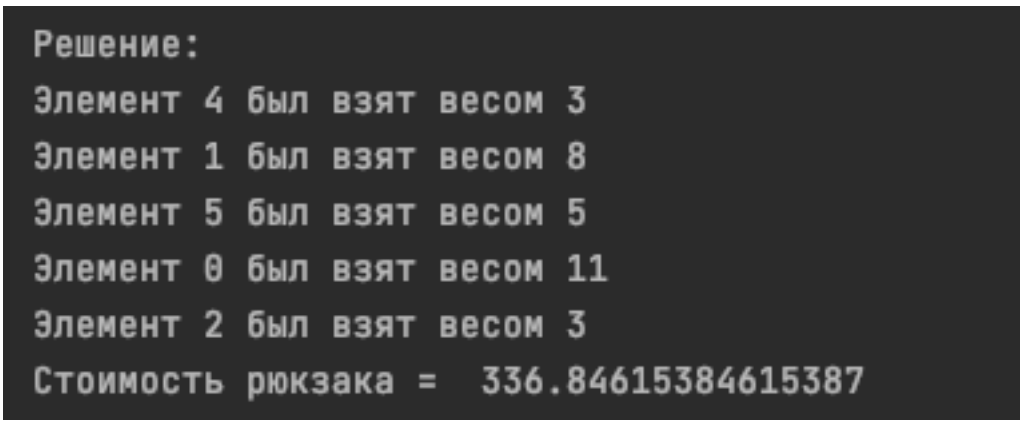

Рисунок 7 – Результат работы программы

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

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


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

    for item in items:
        remaining_capacity = capacity - total_weight
        if remaining_capacity / item[2] > 1:
            selected_items[item[0]] = item[2]
            total_weight += item[2]
            total_value += item[1]
        else:
            selected_items[item[0]] = remaining_capacity
            total_value += item[1] / item[2] * remaining_capacity
            break

    return selected_items, total_value


if __name__ == '__main__':
    num_items = 10
    items_list = [[i, random.randint(1, 100), random.randint(3, 20)]
                  for i in range(num_items)]
    backpack_capacity = 30

    print("| {:^10} | {:^10} | {:^10} |".format('Элемент', 'Стоимость',
'Вес'))
    for item in items_list:
        print("├{:^12}┼{:^12}┼{:^12}┤".format('', '', ''))
        print("| {:^10} | {:^10} | {:^10} |".format(*item))

    selected_items, total_value = solve_fractional_knapsack(items_list,
backpack_capacity)

    print("\nРешение:")
    for item, weight in selected_items.items():
```

```
        print("Элемент", item, "был взят весом", weight)
    print("Стоимость рюкзака = ", total_value)

    fig, ax = plt.subplots()

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

    x_position = 0
    y_position = 0.5

    text_properties = {'verticalalignment': 'center', 'horizontalalignment':
'center'}

    for item, value, weight in items_list:
        if item in selected_items:
            ax.add_patch(patches.Rectangle((x_position, y_position), weight,
1))
            ax.text(x_position + weight / 2, 1, str(value),
**text_properties)
            if selected_items[item] == weight:
                ax.text(x_position + weight / 2, -0.5,
str(selected_items[item]), **text_properties)
                ax.text(x_position + weight / 2, 0, str(value),
**text_properties)
            else:
                ax.text((backpack_capacity + x_position) / 2, -0.5,
str(selected_items[item]), **text_properties)
                ax.text((backpack_capacity + x_position) / 2, 0,
                        f"{value / weight * selected_items[item]:.2f}",
**text_properties)

            ax.text(x_position + weight / 2, 2, str(item), **text_properties)
            ax.text(x_position + weight / 2, 1.5, str(weight),
**text_properties)
            if x_position > 0:
                ax.axvline(x=x_position, ymin=0, ymax=2, color='red')
            x_position += weight

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

    ax.axis('off')

    plt.show()
```

Вывод:

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. На основании этих примеров можно сделать следующий вывод: жадные

алгоритмы действительно строят оптимальное решение благодаря таким идеям, как:

- Надёжный шаг. Существует оптимальное решение, согласованное с локальным жадным шагом.

- Оптимальность подзадач. Задача, остающаяся после жадного шага, имеет тот же тип.