

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 ПРАКТИЧЕСКОЙ РАБОТЕ №6**  
**дисциплины «Алгоритмизация»**  
**Вариант 7**

Выполнил:  
Горбунов Данила Евгеньевич  
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., канд. технических  
наук, доцент кафедры  
инфокоммуникаций

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

## Ход работы

1. В соответствии с приведённым ниже псевдокодом написал программу на Python, которая на вход принимает множество точек, а выводит минимальное количество отрезков единичной длины, которыми можно покрыть все точки. Алгоритм заключается в том, что пока размер входной массив данных не пуст: мы находим в нём минимальное значение и добавляем к решению отрезок  $[x_{\min}, x_{\min} + 1]$ , а затем удаляем все точки, которые входят в данный отрезок.

Функция  $\text{POINTSCOVER}(x_1, \dots, x_n)$

$S \leftarrow \{x_1, \dots, x_n\}$

пока  $S$  не пусто:

$x_m \leftarrow$  минимальная точка  $S$

    добавить к решению отрезок  $[\ell, r] = [x_m, x_m + 1]$

    выкинуть из  $S$  точки, покрытые отрезком  $[\ell, r]$

вернуть построенное решение

Рисунок 1. Алгоритм PointsCover

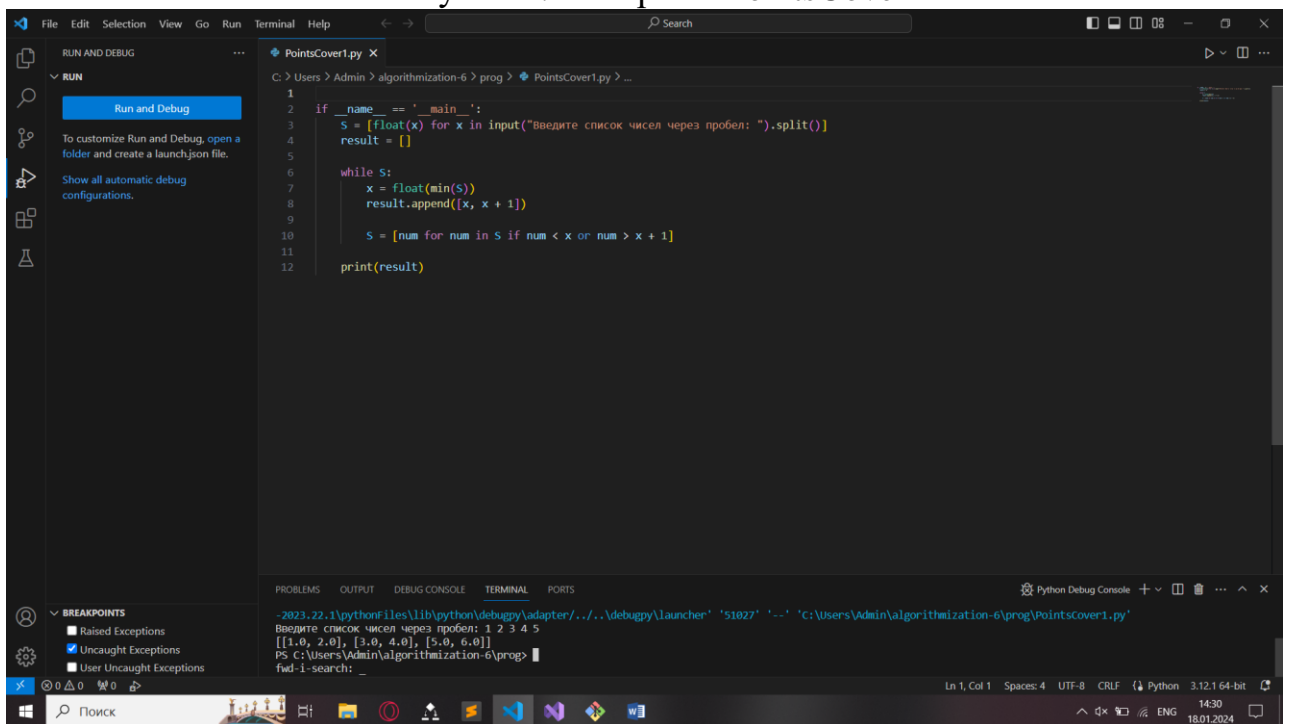


Рисунок 2. Результат работы программы PointsCover

2. В соответствии с приведённым ниже псевдокодом написал программу на Python, которая на вход принимает множество точек, а выводит минимальное количество отрезков единичной длины, которыми можно покрыть все точки. Алгоритм заключается в том, что сначала массив сортируется, далее пока  $i$  меньше размера массива добавляем отрезок  $[S[i], S[i]+1]$  и пока  $S[i]$  меньше предыдущего значения конца отрезка добавляем 1 к  $i$ .

### Функция POINTSCOVER( $x_1, \dots, x_n$ )

```

 $x_1, \dots, x_n \leftarrow \text{SORT}(x_1, \dots, x_n)$ 
 $i \leftarrow 1$ 
пока  $i \leq n$ :
    добавить к решению отрезок  $[\ell, r] = [x_i, x_i + 1]$ 
     $i \leftarrow i + 1$ 
    пока  $i \leq n$  и  $x_i \leq r$ :
         $i \leftarrow i + 1$ 
вернуть построенное решение

```

Рисунок 3. Улучшенный алгоритм PointsCover

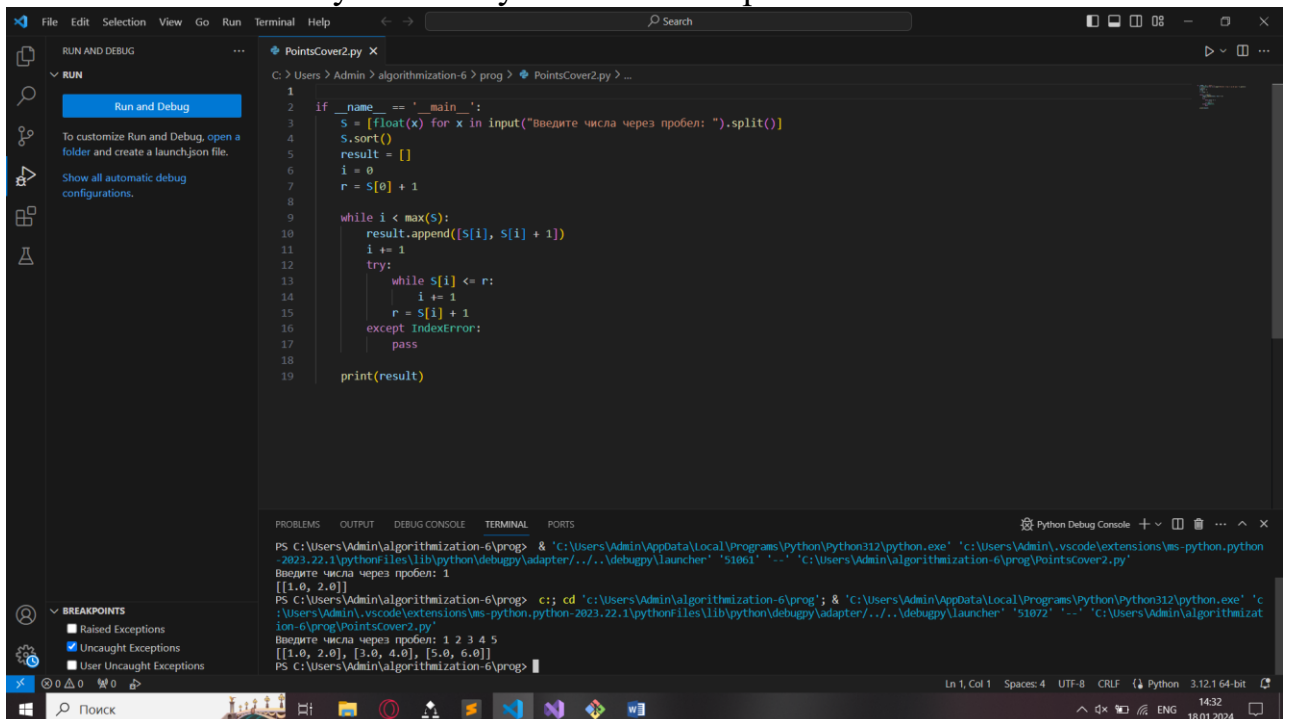


Рисунок 4. Результат работы программы PointsCover2

3. В соответствии с приведённым ниже псевдокодом, написал программу для решения задачи о выборе заявок, в которой требуется найти максимальное количество попарно не пересекающихся отрезков.

Функция  $\text{ACTSEL}(\ell_1, r_1, \dots, \ell_n, r_n)$

$S \leftarrow \{[\ell_1, r_1], \dots, [\ell_n, r_n]\}$

пока  $S$  не пусто:

$[\ell_m, r_m] \leftarrow$  отрезок из  $S$  с мин. правым концом

добавить  $[\ell_m, r_m]$  к решению

выкинуть из  $S$  отрезки, пересекающиеся с  $[\ell_m, r_m]$

вернуть построенное решение

Рисунок 5. Алгоритм ActSel

The screenshot shows a VS Code editor with a file named `ActSel.py`. The code implements the ActSel algorithm. It takes user input for the number of intervals and then for each interval, its start and end points. It then finds the maximum number of non-overlapping intervals by selecting the interval with the minimum right endpoint and removing all intervals that overlap with it. The output shows the selected intervals: `[[1, 3], [5, 6]]`.

```

1
2 if __name__ == '__main__':
3     num_intervals = int(input("Введите количество интервалов: "))
4     S = []
5     for _ in range(num_intervals):
6         start, end = input("Введите интервал [L, R] через пробел: ").split()
7         S.append([int(start), int(end)])
8
9     result = []
10    m = 0
11
12    while len(S) != 0:
13        minimum = 10000
14        for index, interval in enumerate(S):
15            if interval[1] < minimum:
16                minimum = interval[1]
17                m = interval[0]
18        result.append([m, minimum])
19        m_range = list(range(m, minimum))
20        S = [interval for interval in S if interval[0] not in m_range]
21
22    print(result)

```

The terminal output shows the execution of the program:

```

PS C:\Users\Admin\algorithmization-6> & 'C:\Users\Admin\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\Admin\.vscode\extensions\ms-pyt
len.python-2023.22.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '51096' '-.' 'C:\Users\Admin\algorithmization-6\prog\ActSel1.py'
Введите количество интервалов: 3
Введите интервал [L, R] через пробел: 1 3
Введите интервал [L, R] через пробел: 2 4
Введите интервал [L, R] через пробел: 5 6
[[1, 3], [5, 6]]
PS C:\Users\Admin\algorithmization-6\prog>
fud-i-search: _

```

Рисунок 6. Результат работы программы ActSel

4. В соответствии с приведённым ниже псевдокодом, написал улучшенную программу для решения задачи о выборе заявок, в которой требуется найти максимальное количество попарно не пересекающихся отрезков.

Функция  $\text{ACTSEL}(\ell_1, r_1, \dots, \ell_n, r_n)$

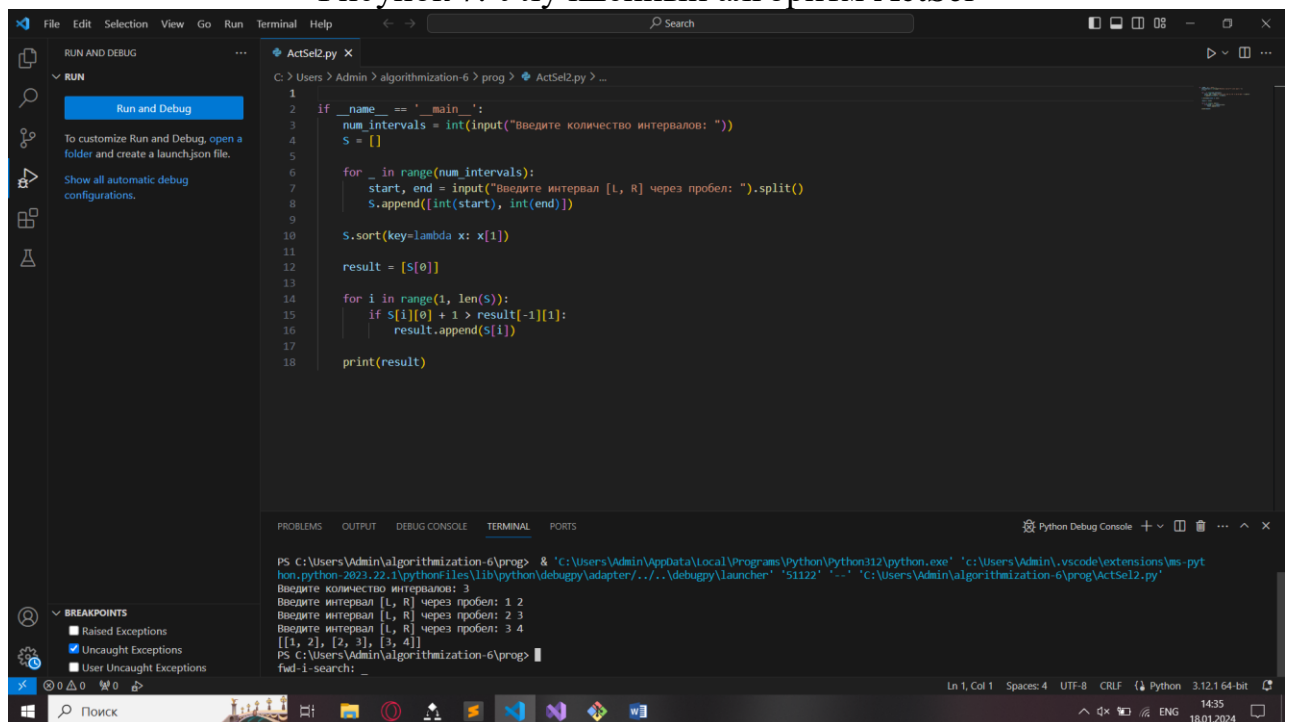
отсортировать  $n$  отрезков по правым концам  
для всех отрезков в полученном порядке:

если текущий отрезок не пересекает  
последний добавленный:

взять его в решение

вернуть построенное решение

Рисунок 7. Улучшенный алгоритм ActSel



```
1
2 if __name__ == '__main__':
3     num_intervals = int(input("Введите количество интервалов: "))
4     S = []
5
6     for _ in range(num_intervals):
7         start, end = input("Введите интервал [L, R] через пробел: ").split()
8         S.append([int(start), int(end)])
9
10    S.sort(key=lambda x: x[1])
11
12    result = [S[0]]
13
14    for i in range(1, len(S)):
15        if S[i][0] > result[-1][1]:
16            result.append(S[i])
17
18    print(result)
```

PS C:\Users\Admin\algorithmization-6> & 'C:\Users\Admin\AppData\Local\Programs\Python\Python312\python.exe' 'C:\Users\Admin\.vscode\extensions\ms-python.python-2023.22.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '51122' '-x' 'C:\Users\Admin\algorithmization-6\prog\ActSel2.py'

Введите количество интервалов: 3  
Введите интервал [L, R] через пробел: 1 2  
Введите интервал [L, R] через пробел: 2 3  
Введите интервал [L, R] через пробел: 3 4  
[[1, 2], [2, 3], [3, 4]]  
PS C:\Users\Admin\algorithmization-6> fud-i-search: \_

Рисунок 8. Результат работы программы ActSel2

5. В соответствии с приведённым ниже псевдокодом написал программу, которая получает на вход дерево, а на выходе независимое множество. Сначала находится максимальное число в массиве состоящем из ребер графа, потом пока этот массив не пуст, создаётся множество локальных решений, в который добавляются элементы графа которые имеют 1 связь, а эта связь проверяется, которая возвращает количество элементов соответствующих значению num. Если вершина имеет 1 связь – это значит, что лист найдет и он добавляется в массив локальных решений. Те ребра вершин, которые находятся в локальном решении удаляются из входного массива и цикл проходит до того момента, пока число элементов входного массива не станет равно 0.

### Функция `MAXINDEPENDENTSET( $T$ )`

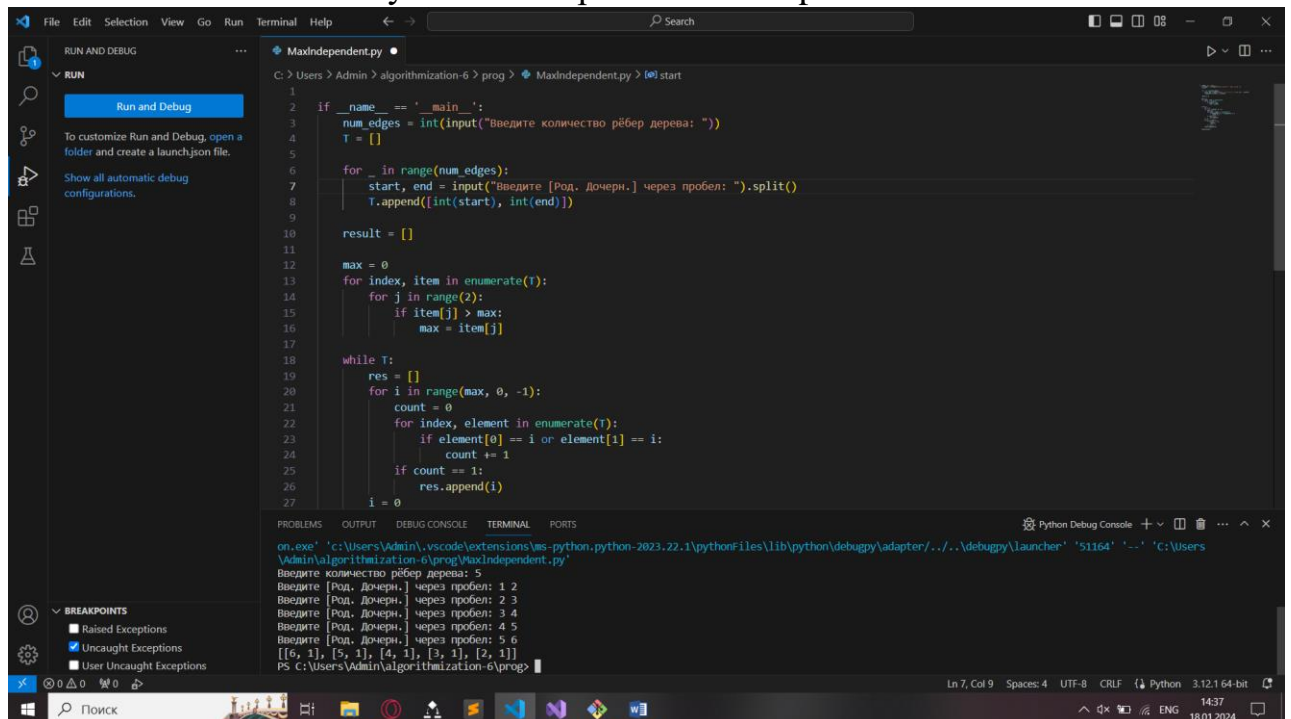
пока  $T$  не пусто:

    взять в решение все листья

    выкинуть их и их родителей из  $T$

    вернуть построенное решение

Рисунок 9.Алгоритм MaxIndependentSet



```
1 if __name__ == '__main__':
2     num_edges = int(input("Введите количество ребер дерева: "))
3     T = []
4
5     for _ in range(num_edges):
6         start, end = input("Введите [Род. Дочери.] через пробел: ").split()
7         T.append([int(start), int(end)])
8
9     result = []
10
11     max = 0
12     for index, item in enumerate(T):
13         for j in range(2):
14             if item[j] > max:
15                 max = item[j]
16
17     while T:
18         res = []
19         for i in range(max, 0, -1):
20             count = 0
21             for index, element in enumerate(T):
22                 if element[0] == i or element[1] == i:
23                     count += 1
24             if count == 1:
25                 res.append(i)
26         i = 0
27
```

Python Debug Console

```
on.exe' 'c:\Users\Admin\vscode\extensions\ms-python.python-2023.22.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '51164' '--' 'c:\Users\Admin\algorithmization-6\prog\MaxIndependent.py'
Введите количество ребер дерева: 5
Введите [Род. Дочери.] через пробел: 1 2
Введите [Род. Дочери.] через пробел: 2 3
Введите [Род. Дочери.] через пробел: 3 4
Введите [Род. Дочери.] через пробел: 4 5
Введите [Род. Дочери.] через пробел: 5 6
[[6, 1], [5, 1], [4, 1], [3, 1], [2, 1]]
PS C:\Users\Admin\algorithmization-6\prog>
```

Рисунок 10. Результат работы программы MaxIndependentSet

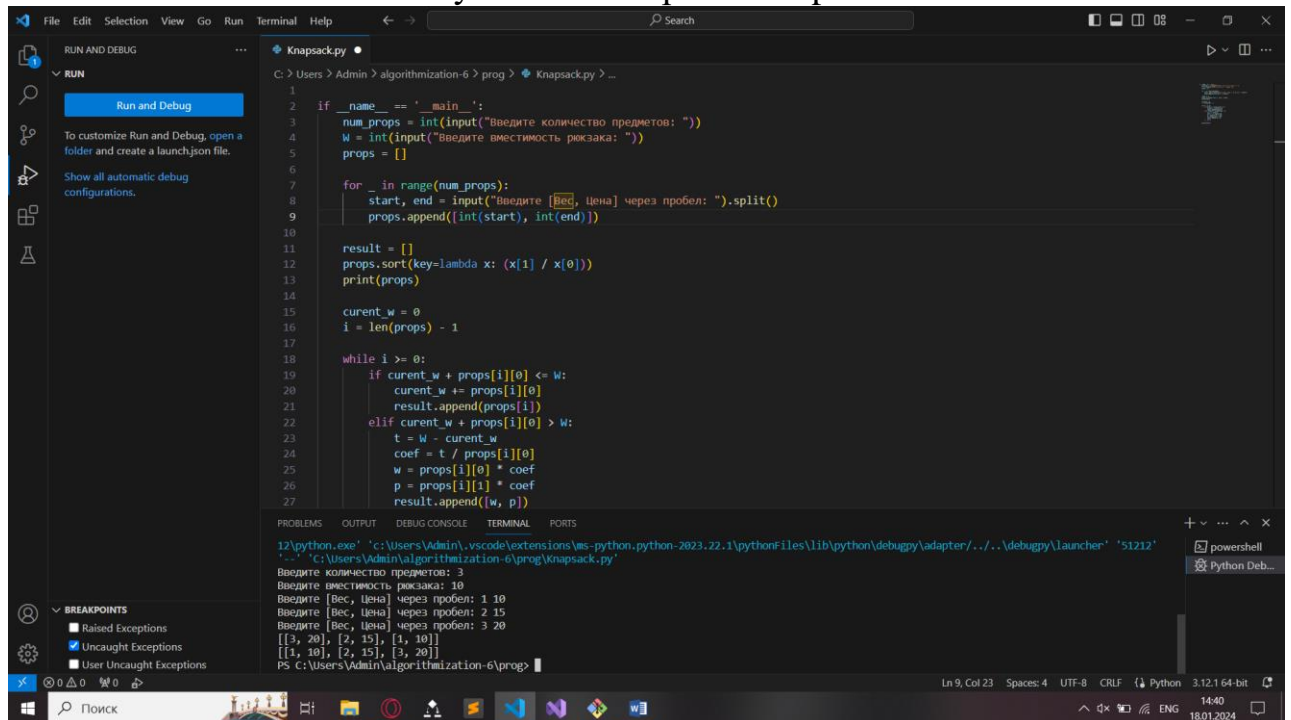


6. В соответствии с приведённым ниже псевдокодом написал программу по задаче о непрерывном рюкзаке, в которой требуется частями предметов с весами и их стоимостью заполнить рюкзак определённого размера так, чтобы стоимость помещённых в него предметов была максимальной.

### Функция $\text{KNAPSACK}(w_1, c_1, \dots, w_n, c_n)$

отсортировать предметы по убыванию  $c/w$   
для всех предметов в полученном порядке:  
взять по максимуму текущего предмета  
вернуть построенное решение

Рисунок 11. Алгоритм Knapsack



```
1 if __name__ == '__main__':
2     num_props = int(input("Введите количество предметов: "))
3     W = int(input("Введите вместимость рюкзака: "))
4     props = []
5
6     for _ in range(num_props):
7         start, end = input("Введите [Вес, Цена] через пробел: ").split()
8         props.append((int(start), int(end)))
9
10    result = []
11    props.sort(key=lambda x: (x[1] / x[0]))
12    print(props)
13
14    current_w = 0
15    i = len(props) - 1
16
17    while i >= 0:
18        if current_w + props[i][0] <= W:
19            current_w += props[i][0]
20            result.append(props[i])
21        elif current_w + props[i][0] > W:
22            t = W - current_w
23            coef = t / props[i][0]
24            w = props[i][0] * coef
25            p = props[i][1] * coef
26            result.append((w, p))
27            break
28        i -= 1
29
30    print(result)
```

12\python.exe 'c:\Users\Admin\vscode\extensions\ms-python.python-2023.22.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '51212' '-c' 'c:\Users\Admin\algorithmization-6\prog\knapsack.py'

Введите количество предметов: 3  
Введите вместимость рюкзака: 10  
Введите [Вес, Цена] через пробел: 1 10  
Введите [Вес, Цена] через пробел: 2 15  
Введите [Вес, Цена] через пробел: 3 20  
[[3, 20], [2, 15], [1, 10]]  
[[1, 10], [2, 15], [3, 20]]  
PS C:\Users\Admin\algorithmization-6\prog>

Рисунок 12. Результат работы программы Knapsack

Вывод: в ходе выполнения лабораторной работы были исследованы некоторые примеры жадных алгоритмов, решающих различные задачи. На основании этих примеров можно сказать, что жадные алгоритмы действительно строят оптимальное решение благодаря понятиям надёжного шага и оптимальности подзадач.

