

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 №11
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.

Порядок выполнения работы:

1. Нахождение числа фиббоначи:

Динамическое программирование назад:

Инициализация

$$F[0 \dots n] = [-1, -1, \dots, -1]$$

Функция FIBTD(n)

```
если  $F[n] = -1$ :  
    если  $n \leq 1$ :  
         $F[n] \leftarrow n$   
    иначе:  
         $F[n] \leftarrow \text{FIBTD}(n - 1) + \text{FIBTD}(n - 2)$   
вернуть  $F[n]$ 
```

Рисунок 1 – Алгоритм нахождения числа Фибоначчи

```
5 def fib(n, calculation_method=0):  
6     """  
7     Функции вычисления числа фибоначчи.  
8     """  
9     def fib_td(n):  
10        """  
11        Динамическое программирование назад.  
12        """  
13        if n <= 1:  
14            f[n] = n  
15        else:  
16            f[n] = fib_td(n - 1) + fib_td(n-2)  
17        return f[n]  
18
```

Рисунок 2 – Фрагмент кода файла fib.py

Динамическое программирование вперед:

Функция $\text{FIBBU}(n)$

создать массив $F[0 \dots n]$
 $F[0] \leftarrow 0, F[1] \leftarrow 1$
для i от 2 до n :
 $F[i] \leftarrow F[i - 1] + F[i - 2]$
вернуть $F[n]$

Рисунок 3 – Алгоритм нахождения числа Фибоначчи

```
19 def fib_bu(n):
20     """
21     Динамическое программирование вперед.
22     """
23     f = [-1] * (n+1)
24     f[0] = 0
25     f[1] = 1
26
27     for i in range(2, n + 1):
28         f[i] = f[i - 1] + f[i - 2]
29     return f[n]
```

Рисунок 4 – Фрагмент кода файла fib.py

Уменьшение количества потребления памяти:

Функция FIBBUIMPROVED(n)

```
если  $n \leq 1$ :  
    вернуть  $n$   
  
 $prev \leftarrow 0$   
 $curr \leftarrow 1$   
  
повторить  $(n - 1)$  раз:  
     $next \leftarrow prev + curr$   
     $prev \leftarrow curr$   
     $curr \leftarrow next$   
вернуть  $curr$ 
```

Рисунок 5 – Алгоритм нахождения числа Фибоначчи

```
31 def fib_bu_improved(n, calculation_method):  
32     """  
33     Уменьшенным памяти.  
34     """  
35     if n <= 1:  
36         return n  
37  
38     prev, curr = 0, 1  
39  
40     for _ in range(n - 1):  
41         prev, curr = curr, prev + curr  
42     return curr  
43
```

Рисунок 6 – Фрагмент кода файла fib.py

Результат:

```
Число Фибоначчи(10):  
fib(N, 0) = 55  
fib(N, 1) = 55  
fib(N, 2) = 55
```

Рисунок 7 – Результат выполнения программы fib.py

2. Нахождение в списка НВП и самой НВП:

Поиск длины наибольшей возрастающей подпоследовательности

Функция LISBOTTOMUP($A[1 \dots n]$)

```
создать массив  $D[1 \dots n]$ 
для  $i$  от 1 до  $n$ :
     $D[i] \leftarrow 1$ 
    для  $j$  от 1 до  $i - 1$ :
        если  $A[j] < A[i]$  и  $D[j] + 1 > D[i]$ :
             $D[i] \leftarrow D[j] + 1$ 
ans  $\leftarrow 0$ 
для  $i$  от 1 до  $n$ :
    ans  $\leftarrow \max(\text{ans}, D[i])$ 
вернуть ans
```

Рисунок 8 – Алгоритм нахождения длины НВП

```
5 def list_bottom_up(a):
6     """
7     Поиск длины наибольшей возрастающей подпоследовательности.
8     """
9     n = len(a)
10    D = []
11
12    for i in range(n):
13        D.append(1)
14        for j in range(i):
15            if a[j] < a[i] and D[j] + 1 > D[i]:
16                D[i] = D[j] + 1
17
18    ans = max(D)
19
20    return ans
21
```

Рисунок 9 – Фрагмент кода файла (list.py)

Восстановление НВП с помощью списка prev:

Восстановление ответа

```
создать массив  $L[1 \dots ans]$    {индексы НВП}
 $k \leftarrow 1$ 
для  $i$  от 2 до  $n$ :
    если  $D[i] > D[k]$ :
         $k \leftarrow i$ 
 $j \leftarrow ans$ 
пока  $k > 0$ :
     $L[j] \leftarrow k$ 
     $j \leftarrow j - 1$ 
     $k \leftarrow prev[k]$ 
```

Рисунок 10 – Алгоритм нахождения НВП

```
23 def using_prev(prev, m_index):
24     """
25     Восстановление НВП с помощью списка prev
26     """
27     l = []
28     while True:
29         l.append(m_index)
30         if prev[m_index] == -1:
31             break
32         m_index = prev[m_index]
33
34     l.reverse()
35     return l
```

Рисунок 11 – Фрагмент кода файла (list.py)

Восстановление НВП без помощи списка prev:

Функция LISBOTTOMUP2($A[1 \dots n]$)

```
создать массивы  $D[1 \dots n]$  и  $prev[1 \dots n]$ 
для  $i$  от 1 до  $n$ :
     $D[i] \leftarrow 1$ ,  $prev[i] \leftarrow -1$ 
    для  $j$  от 1 до  $i - 1$ :
        если  $A[j] < A[i]$  и  $D[j] + 1 > D[i]$ :
             $D[i] \leftarrow D[j] + 1$ ,  $prev[i] \leftarrow j$ 
 $ans \leftarrow 0$ 
для  $i$  от 1 до  $n$ :
     $ans = \max(ans, D[i])$ 
вернуть  $ans$ 
```

Рисунок 12 – Алгоритм нахождения НВП

```
38 def without_prev(d, ans, m_index):
39     """
40     Восстановление НВП без помощи списка prev
41     """
42     l = []
43     while True:
44         l.append(m_index)
45         if ans == 1:
46             break
47         ans -= 1
48         while True:
49             m_index -= 1
50             if d[m_index] == ans and a[m_index] < a[l[-1]]:
51                 break
52     l.reverse()
53
54     return l
55
```

Рисунок 13 – Фрагмент кода файла (list.py)

Поиск длины и самой НВП:

```

57 def list_bottom_up_2(a):
58     """
59     Поиск длины и самой НВП.
60     """
61     n = len(a)
62     d, prev = [], []
63     for i in range(n):
64         d.append(1)
65         prev.append(-1)
66         for j in range(i):
67             if a[j] < a[i] and d[j] + 1 > d[i]:
68                 d[i] = d[j]+1
69                 prev[i] = j
70
71     ans, max_index = 0, 0
72     for i, item in enumerate(d):
73         if ans < item:
74             ans, max_index = item, i
75
76     list_using_prev = using_prev(prev, max_index)
77     list_without_prev = without_prev(d, ans, max_index)
78
79     return ans, (list_using_prev, list_without_prev)
80

```

Рисунок 14 – Фрагмент кода файла (list.py)

Результат:

```

5
(5, ([1, 3, 5, 9, 11], [2, 3, 5, 10, 11]))

```

Рисунок 15 – Результат выполнения кода (list.py)

3. Поиск максимальной стоимости предметов в рюкзаке

Предметы могут повторяться:

Функция

KNAPSACKWITHREPSBU($W, w_1, \dots, w_n, c_1, \dots, c_n$)

создать массив $D[0 \dots W] = [0, 0, \dots, 0]$

для w от 1 до W :

для i от 1 до n :

если $w_i \leq w$:

$D[w] \leftarrow \max(D[w], D[w - w_i] + c_i)$

вернуть $D[W]$

Рисунок 16 – Алгоритм поиска максимальной стоимости предметов в рюкзаке

```

5  def knapsack_with_reps(W, weight, cell):
6      """
7      Поиск максимальной стоимости предметов в рюкзаке.
8      Предметы могут повторяться.
9      """
10     d = [0] * (W+1)
11     for w in range(1, W+1):
12         for weight_i, cell_i in zip(weight, cell):
13             if weight_i <= w:
14                 d[w] = max(d[w], d[w - weight_i] + cell_i)
15     return d[W]
16
17

```

Рисунок 17 – Фрагмент кода файла (knapsack.py)

Предметы не могут повторяться:

KNAPSACKWITHOUTREPSBU($W, w_1, \dots, w_n, c_1, \dots, c_n$)

создать массив $D[0 \dots W, 0 \dots n]$

для w от 0 до W :

$D[w, 0] \leftarrow 0$

для i от 0 до n :

$D[0, i] \leftarrow 0$

для i от 1 до n :

- для w от 1 до W :
 - $D[w, i] \leftarrow D[w, i - 1]$
 - если $w_i \leq w$:

$$D[w, i] = \max(D[w, i], D[w - w_i, i - 1] + c_i)$$

вернуть $D[W, n]$

Handwritten notes:
 $D[i, w_i]$
 пере
 $D[w, i - 1]$
 $D[w - w_i, i - 1]$

Рисунок 18 – Алгоритм поиска максимальной стоимости предметов в рюкзаке

```

18 def knapsack_without_reps(W, weight, cell):
19     """
20     Поиск максимальной стоимости предметов в рюкзаке.
21     Предметы не могут повторяться.
22     """
23     def restore(d, weight_rev, cell_rev):
24         """
25         Восстановление предметов в рюкзаке.
26         """
27         solution = []
28         w = W
29         elem = len(weight_rev)
30         for weight_i, cell_i in zip(weight_rev, cell_rev):
31             if d[w][elem] == d[w - weight_i][elem-1] + cell_i:
32                 solution.append(1)
33                 w -= weight_i
34             else:
35                 solution.append(0)
36             elem -= 1
37         solution.reverse()
38         return solution
39
40     d = [[0] for _ in range(W+1)]
41     d[0] = [0] * (len(weight) + 1)
42     for weight_i, cell_i in zip(weight, cell):
43         for w in range(1, W+1):
44             d[w].append(d[w][-1])
45             if weight_i <= w:
46                 d[w][-1] = max(d[w][-1], d[w - weight_i][-2] + cell_i)
47
48     solution = restore(d, weight[::-1], cell[::-1])
49
50     return d[W][-1], solution
51

```

Рисунок 19 – Фрагмент кода файла (knapsack.py)

Результат:

```

with_rep_bu = 48
without_rep_bu = (46, [1, 0, 1, 0])

```

Рисунок 20 – Результат выполнения кода (knapsack.py)

Вывод: в процессе работы мы ознакомились с методами динамического программирования, такими как нахождение чисел Фибоначчи, поиск наибольшей возрастающей подпоследовательности (НВП) и алгоритм расчета максимальной стоимости предметов в рюкзаке.