

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:
Епифанов Алексей Александрович
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 Роман Александрович

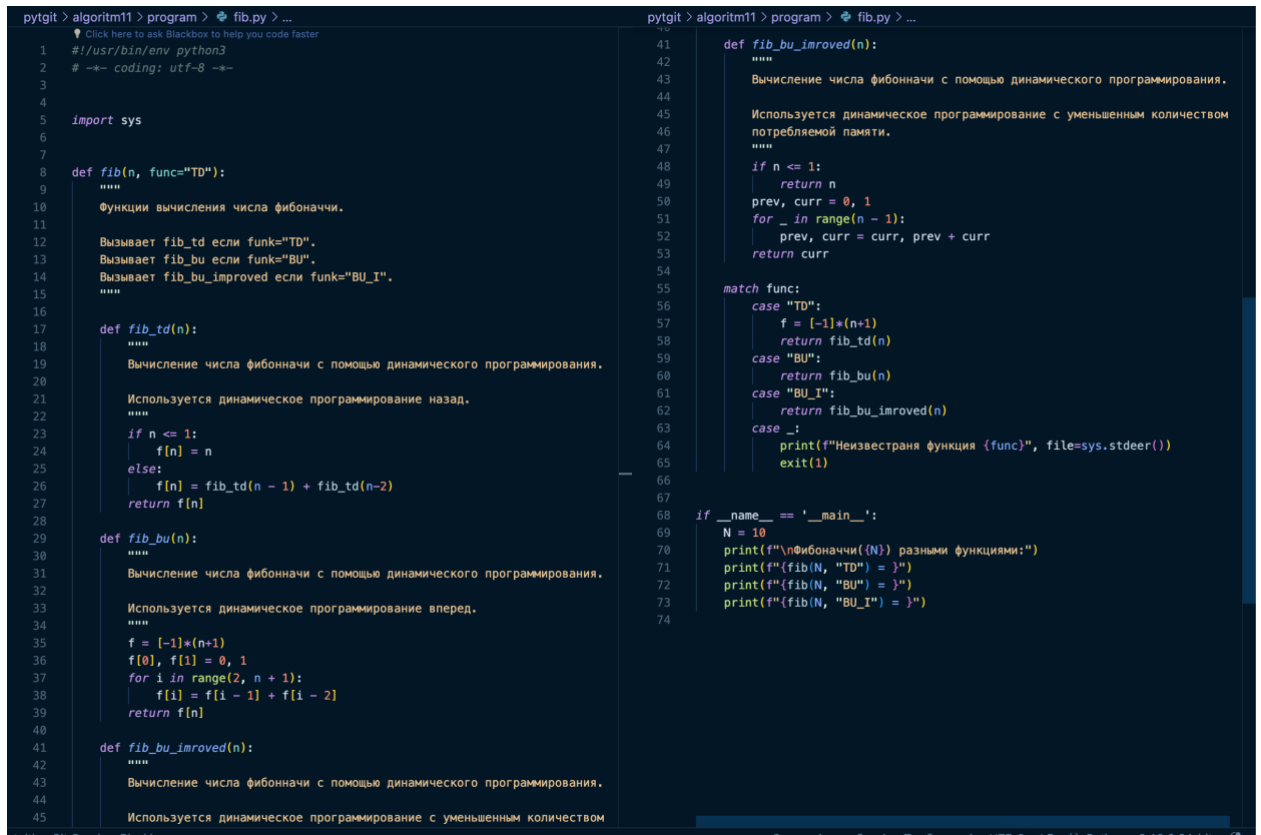
(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

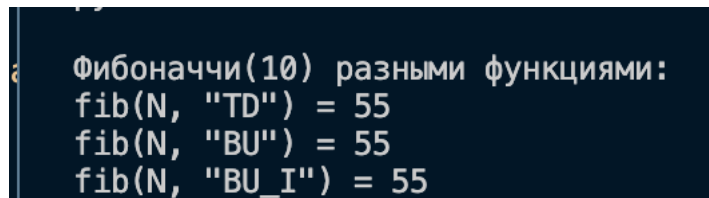
Порядок выполнения работы:

1. Написал программу по примерам из видео для нахождения числа Фибоначчи с использованием динамического программирования



```
pythgit > algorithm1 > program > fib.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  import sys
6
7
8  def fib(n, func="TD"):
9      """
10     Функция вычисления числа фибонначи.
11
12     Вызывает fib_td если func="TD".
13     Вызывает fib_bu если func="BU".
14     Вызывает fib_bu_improved если func="BU_I".
15     """
16
17     def fib_td(n):
18         """
19         Вычисление числа фибонначи с помощью динамического программирования.
20
21         Используется динамическое программирование назад.
22         """
23         if n <= 1:
24             f[n] = n
25         else:
26             f[n] = fib_td(n - 1) + fib_td(n-2)
27         return f[n]
28
29     def fib_bu(n):
30         """
31         Вычисление числа фибонначи с помощью динамического программирования.
32
33         Используется динамическое программирование вперед.
34         """
35         f = [-1]*(n+1)
36         f[0], f[1] = 0, 1
37         for i in range(2, n + 1):
38             f[i] = f[i - 1] + f[i - 2]
39         return f[n]
40
41     def fib_bu_improved(n):
42         """
43         Вычисление числа фибонначи с помощью динамического программирования.
44
45         Используется динамическое программирование с уменьшенным количеством
46
47
48     if n <= 1:
49         return n
50     prev, curr = 0, 1
51     for _ in range(n - 1):
52         prev, curr = curr, prev + curr
53     return curr
54
55     match func:
56         case "TD":
57             f = [-1]*(n+1)
58             return fib_td(n)
59         case "BU":
60             return fib_bu(n)
61         case "BU_I":
62             return fib_bu_improved(n)
63         case _:
64             print(f"Неизвестная функция {func}", file=sys.stderr())
65             exit(1)
66
67
68 if __name__ == '__main__':
69     N = 10
70     print(f"\nФибоначчи({N}) разными функциями:")
71     print(f"fib(N, \"TD\") = {fib(N, \"TD\")}")
72     print(f"fib(N, \"BU\") = {fib(N, \"BU\")}")
73     print(f"fib(N, \"BU_I\") = {fib(N, \"BU_I\")}")
74
```

Рисунок 1. Код программы fib.py



```
Фибоначчи(10) разными функциями:
fib(N, "TD") = 55
fib(N, "BU") = 55
fib(N, "BU_I") = 55
```

Рисунок 2. Результат работы программы fib.py

2. Написал программу по примерам из видео для нахождения длины НВП в списке, а также для нахождения самой НВП

```
pytglt > algorithm1 > program > list.py > ...
1 Click here to ask Blackbox to help you code faster
2 #!/usr/bin/env python3
3 # -*- coding: utf-8 -*-
4
5 def list_bottom_up(a):
6     """
7     Поиск длины НВП.
8     """
9     d = []
10    for i, _ in enumerate(a):
11        d.append(1)
12        for j in range(i):
13            if a[j] < a[i] and d[j] + 1 > d[i]:
14                d[i] = d[j] + 1
15
16    ans = max(d)
17    return ans
18
19
20 def list_bottom_up2(a):
21     """
22     Поиск длины и самой НВП.
23
24     Сама НВП ищется двумя способами:
25     с помощью дополнительного списка prev;
26     без помощи дополнительного списка.
27     """
28    def restore_using_prev(m_index):
29        """
30        Восстановление НВП с помощью списка prev
31        """
32        l = []
33        while True:
34            l.append(m_index)
35            if prev[m_index] == -1:
36                break
37            m_index = prev[m_index]
38
39        l.reverse()
40        return l
41
42    def restore_without_prev(ans, m_index):
43        """
44        Восстановление НВП без помощи списка prev
45        """
46
47    def restore_without_prev(ans, m_index):
48        """
49        Восстановление НВП без помощи списка prev
50        """
51        l = []
52        while True:
53            l.append(m_index)
54            if ans == 1:
55                break
56            ans -= 1
57            while True:
58                m_index -= 1
59                if d[m_index] == ans and a[m_index] < a[l[-1]]:
60                    break
61            l.reverse()
62            return l
63
64    d, prev = [], []
65    for i, _ in enumerate(a):
66        d.append(1)
67        prev.append(-1)
68        for j in range(i):
69            if a[j] < a[i] and d[j] + 1 > d[i]:
70                d[i] = d[j] + 1
71                prev[i] = j
72
73    ans, max_index = 0, 0
74    for i, item in enumerate(d):
75        if ans < item:
76            ans, max_index = item, i
77
78    list_using_prev = restore_using_prev(max_index)
79    list_without_prev = restore_without_prev(ans, max_index)
80
81    return ans, (list_using_prev, list_without_prev)
82
83
84 if __name__ == '__main__':
85     a = [7, 2, 1, 3, 8, 4, 9, 1, 2, 6, 5, 9, 3, 8, 1]
86     print(list_bottom_up(a))
87     print(list_bottom_up2(a))
```

Рисунок 3. Код программы list.py

```
11/program/list.py
5
(5, ([1, 3, 5, 9, 11], [2, 3, 5, 10, 11]))
```

Рисунок 4. Результат работы программы list.py

3. Написал программу по примерам из видео для решения задачи о рюкзаке в двух случаях: когда предметов неограниченное количество, и когда каждый предмет может быть использован только один раз

```
pythit > algorithm1 > program > knapsack.py > ...
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 import re
4
5
6 def knapsack_bu(W, weight, cell):
7     """
8     Задача о рюкзаке
9
10    Применяется динамическое программирование снизу вверх
11    """
12    def knapsack_with_reps(W, weight, cell):
13        """
14        Поиск максимальной стоимости предметов в рюкзаке.
15
16        Предметы могут повторяться сколько угодно раз.
17        """
18        d = [0] * (W+1)
19        for w in range(1, W+1):
20            for weight_i, cell_i in zip(weight, cell):
21                if weight_i <= w:
22                    d[w] = max(d[w], d[w - weight_i] + cell_i)
23        return d[W]
24
25    def knapsack_without_reps(W, weight, cell):
26        """
27        Поиск максимальной стоимости предметов в рюкзаке.
28
29        Предметы не могут повторяться.
30        Так же функция возвращает solution, в которой 1 помечены элементы,
31        которые были добавлены в рюкзак.
32        """
33    def restore(d, weight_rev, cell_rev):
34        """
35        Восстановление предметов в рюкзаке.
36
37        Возвращает массив, где i элемент = 1,
38        если предмет i был в рюкзаке,
39        и 0, если нет.
40        """
41        solution = []
42        w = W
43        elem = len(weight_rev)
44        for weight_i, cell_i in zip(weight_rev, cell_rev):
45            if d[w][elem] == d[w - weight_i][elem-1] + cell_i:
46                solution.append(1)
47                w -= weight_i
48            else:
49                solution.append(0)
50            elem -= 1
51        solution.reverse()
52        return solution
53
54        d = [[0] for _ in range(W+1)]
55        d[0] = [0] * (len(weight) + 1)
56        for weight_i, cell_i in zip(weight, cell):
57            for w in range(1, W+1):
58                d[w].append(d[w-1])
59                if weight_i <= w:
60                    d[w][-1] = max(d[w-1], d[w - weight_i][-2] + cell_i)
61
62        solution = restore(d, weight[::-1], cell[::-1])
63
64        return d[W][-1], solution
65
66    with_rep = knapsack_with_reps(W, weight, cell)
67    without_rep = knapsack_without_reps(W, weight, cell)
68
69    return with_rep, without_rep
70
71
72 if __name__ == '__main__':
73     W = 10
74     weight = [6, 3, 4, 2]
75     cell = [30, 14, 16, 9]
76     with_rep_bu, without_rep_bu = knapsack_bu(W, weight, cell)
77     print(f"with_rep_bu = {with_rep_bu}\nwithout_rep_bu = {without_rep_bu}")
```

Рисунок 7. Код программы knapsack.py

```
sack.py
with_rep_bu = 48
without_rep_bu = (46, [1, 0, 1, 0])
```

Рисунок 8. Результат работы программы knapsack.py

В ходе выполнения лабораторной работы было исследовано 3 разных задачи, решение которых реализовано с помощью динамического программирования. Данный метод написания алгоритма подходит, когда задача имеет рекуррентную структуру, при которой каждое решение состоит из решений более мелких подзадач. Также при различных задачах может потребоваться использовать различное динамическое программирование, например динамическое программирование назад, вперед, снизу вверх или сверху вниз.