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»

Выполнила:
Беседина Инга Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических наук, доцент, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

Использование динамического программирования для вычисления чисел Фибоначчи:

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

def fib(n, func="TD"):

    def fib_td(n):
        if n <= 1:
            f[n] = n
        else:
            f[n] = fib_td(n - 1)+fib_td(n - 2)
        return f[n]

    def fib_bu(n):
        f = [0]*(n+1)
        f[0], f[1] = 0, 1
        for i in range(2, n + 1):
            f[i] = f[i - 1] + f[i - 2]
        return f[n]

    def fib_bu_i(n):
        if n <= 1:
            return n
        prev, curr = 0, 1
        for i in range(n - 1):
            prev, curr = curr, prev + curr
        return curr

    match func:
        case "TD":
            f = [-1] * (n + 1)
            return fib_td(n)
        case "BU":
            return fib_bu(n)
        case "BU_I":
            return fib_bu_i(n)


if __name__ == "__main__":
    n = 13
    print("Динамическое программирование назад: ", fib(n, 'TD'))
    print("Динамическое программирование вперёд: ", fib(n, 'BU'))
    print("Уменьшение памяти: ", fib(n, "BU_I"))
```

```
Динамическое программирование назад:  233
Динамическое программирование вперёд:  233
Уменьшение памяти:  233
```

Рисунок 1. Результат работы программы

Нахождения длины НВП в списке, нахождения НВП:

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

```python
def lis_bottom_up(a):
    n = len(a)
    d = [0]*n
    for i in range(1, n):
        d[i] = 1
        for j in range(i):
            if a[j] < a[i] and d[j] + 1 > d[i]:
                d[i] = d[j] + 1

    ans = max(d)
    return ans


def lis_bottom_up2(a):
    def restore_with_prev(m_index):
        l = []
        while True:
            l.append(m_index)
            if prev[m_index] == -1:
                break
            m_index = prev[m_index]

        l.reverse()
        return l

    def restore_without_prev(ans, m_index):
        l = []
        while True:
            l.append(m_index)
            if ans == 1:
                break
            ans -= 1
            while True:
                m_index -= 1
                if d[m_index] == ans and a[m_index] < a[l[-1]]:
                    break
        l.reverse()
        return l

    n = len(a)
    d, prev = [0]*n, [0]*n

    for i in range(1, n):
        d[i] = 1
        prev[i] = -1
        for j in range(i):
            if a[j] < a[i] and d[j] + 1 > d[i]:
                d[i] = d[j] + 1
                prev[i] = j

    ans, max_index = 0, 0
    for i, item in enumerate(d):
        if ans < item:
            ans, max_index = item, i

    return ans, restore_with_prev(max_index), restore_without_prev(ans,
max_index)


if __name__ == "__main__":
    a = [7, 2, 1, 3, 8, 4, 9, 1, 2, 6, 5, 9, 3, 8, 1]
    print(lis_bottom_up(a))
    print(lis_bottom_up2(a))
```

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

Рисунок 2. Результат работы программы

Нахождение расстояния редактирования с использованием динамического программирования:

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


def edit_dist_td(a, b, i, j, d):
    if d[i][j] == -1:
        if i == 0:
            d[i][j] = j
        elif j == 0:
            d[i][j] = i
        elif a[i - 1] == b[j - 1]:
            d[i][j] = edit_dist_td(a, b, i - 1, j - 1, d)
        else:
            insert_cost = 1 + edit_dist_td(a, b, i, j - 1, d)
            delete_cost = 1 + edit_dist_td(a, b, i - 1, j, d)
            replace_cost = 1 + edit_dist_td(a, b, i - 1, j - 1, d)
            d[i][j] = min(insert_cost, delete_cost, replace_cost)
    return d[i][j]


def initialize_matrix(n, m):
    return [[-1 for _ in range(m)] for _ in range(n)]


def edit_dist_bu(a, b):
    n = len(a)
    m = len(b)
    d = [[0] * (m+1) for _ in range(n+1)]

    for i in range(n+1):
        d[i][0] = i
    for j in range(m+1):
        d[0][j] = j

    for i in range(1, n+1):
        for j in range(1, m+1):
            if a[i - 1] == b[j - 1]:
                d[i][j] = d[i-1][j-1]
            else:
                insert_cost = 1 + d[i][j-1]
                delete_cost = 1 + d[i-1][j]
                replace_cost = 1 + d[i-1][j-1]
                d[i][j] = min(insert_cost, delete_cost, replace_cost)

    return d[n][m]


if __name__ == "__main__":
    a = "kitten"
    b = "sitting"
    n = len(a)
    m = len(b)
```

```
    d = initialize_matrix(n + 1, m + 1)
    print("Динамическое программирование сверху вниз:", edit_dist_td(a, b, n,
m, d))
    print("Динамическое программирование снизу вверх:", edit_dist_bu(a, b))
```

```
Динамическое программирование сверху вниз: 3
Динамическое программирование снизу вверх: 3
```

Рисунок 3. Результат работы программы

Решение задачи о рюкзаке в двух случаях: когда предметов неограниченное количество, и когда каждый предмет может быть использован только один раз:

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


def knapsack_bu(W, weight, cell):
    def knapsack_with_reps(W, weight, cell):
        d = [0]*(W+1)
        for w in range(1, W+1):
            for weight_i, cell_i in zip(weight, cell):
                if weight_i <= w:
                    d[w] = max(d[w], d[w - weight_i] + cell_i)
        return d[W]

    def knapsack_without_reps(W, weight, cell):
        def restore(d, weight_rev, cell_rev):
            sol = []
            w = W
            el = len(weight_rev)
            for weight_i, cell_i in zip(weight_rev, cell_rev):
                if d[w][el] == d[w - weight_i][el - 1] + cell_i:
                    sol.append(1)
                    w -= weight_i
                else:
                    sol.append(0)
                el -= 1
            sol.reverse()
            return sol

        d = [[0] for e in range(W+1)]
        d[0] = [0] * (len(weight) + 1)
        for weight_i, cell_i in zip(weight, cell):
            for w in range(1, W+1):
                d[w].append(d[w][-1])
                if weight_i <= w:
                    d[w][-1] = max(d[w][-1], d[w - weight_i][-2] + cell_i)

        sol = restore(d, weight[::-1], cell[::-1])

        return d[W][-1], sol

    return knapsack_with_reps(W, weight, cell), knapsack_without_reps(W,
weight, cell)


if __name__ == "__main__":
```

```
W = 10
weight = [6, 3, 4, 2]
cell = [30, 14, 16, 9]
with_rep_bu, without_rep_bu = knapsack_bu(W, weight, cell)
print(with_rep_bu)
print(without_rep_bu)
```

```
48
(46, [1, 0, 1, 0])
```

Рисунок 4. Результат работы программы