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:
Михеева Елена Александровна
2 курс, группа ИВТ-б-з-20-1,
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. техн. наук, доцент, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Динамическое программирование

Цель работы: приобретение навыков по работе со списками при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы.

1. Была создана программа для нахождения требуемого числа в последовательности Фибоначчи на основе примеров динамического программирования. Были разработаны три функции: FibTD – динамическое программирование вперед, FibBU – динамическое программирование назад, FibImproved – уменьшение количества памяти

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

Рисунок 1. Функция FibTD

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

Рисунок 2. Функция FibBU

```
def FibBUImproved(n):
    if n <= 1:
        return n

    prev = 0
    curr = 1

    for _ in range(n-1):
        next = prev + curr
        prev = curr
        curr = next

    return curr
```

Рисунок 2. Функция FibImproved

2. Была создана программа для нахождения списка НВП и самой НВП на основе примеров динамического программирования. Были разработаны три функции: LISBottomUp и LISBottomUp2.

```python
def LISBottomUp(a):
    n = len(a)
    D = []
    for i in range(n):
        D.append(1)
        for j in range(1, i-1):
            if a[j] < a[i] and D[j] + 1 > D[i]:
                D[i] = D[j] + 1

    ans = 0
    for i in range(n):
        ans = max(ans, D[i])

    return ans
```

Рисунок 3. Функция LISBottomUp

```python
def LISBottomUp2(a):
    n = len(a)
    D = []
    prev = []
    for i in range(n):
        D.append(1)
        prev.append(-1)
        for j in range(1, i-1):
            if a[j] < a[i] and D[j] + 1 > D[i]:
                D[i] = D[j] + 1
                prev[i] = j

    ans = 0
    maxi = 0
    for i in range(n):
        if ans < D[i]:
            ans = D[i]
            maxi = i
    print("reconstructed sequence without prev: ")
    print(replyANS(D, ans, maxi))
    print("reconstructed sequence: ")
    print(replyANSwith(maxi, prev))

    return ans


def replyANSwith(maxi, prev):
    L = []
    while True:
        L.append(maxi)
        if prev[maxi] == -1:
            break
        maxi = prev[maxi]

    L.reverse()
    return L


def replyANS(d, ans, maxi):
    L = []
    while True:
        L.append(maxi)
        if ans == 1:
            break
        ans -= 1
        while True:
            maxi -= 1
            if d[maxi] == ans and a[maxi] < a[L[-1]]:
                break
    L.reverse()
    return L
```

Рисунок 4. Функция LISBottomUp2

```
Maximum sequence length is  5
reconstructed sequence without prev:
[2, 3, 5, 10, 11]
reconstructed sequence:
[1, 3, 5, 9, 11]
Maximum sequence length is  5
```

Рисунок 5. Результат работы программы

3. Была написана программа для нахождения максимальной стоимости рюкзака в двух случаях: когда предметы повторяются и когда каждый предмет может быть использован только один раз.

```
def knapSackWithRepetitions(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)
    print(f"Maximum value without repetitions: {D[w]}\n")
```

Рисунок 6. Функция knapSackWithRepetitions

```
def reincarnate(d, weights, cell):
    solution = []
    w = W
    item = len(weights)

    for i in range(len(weights) - 1, -1, -1):
        current_weight = weights[i]
        if d[w][item] == d[w - current_weight][item - 1] + cell[i]:
            solution.append(1)
            w -= current_weight
        else:
            solution.append(0)
        item -= 1

    solution.reverse()
    return solution


def knapSackWithoutRepetitions(weights, cell):
    d = [[0] * (len(weights) + 1) for _ in range(W + 1)]

    for i in range(len(weights) + 1):
        d[0][i] = 0

    for weight in range(W + 1):
        d[weight][0] = 0

    for item in range(1, len(weights) + 1):
        for weight in range(1, W + 1):
            d[weight][item] = d[weight][item - 1]
            cur_weight = weights[item - 1]

            if cur_weight <= weight:
                d[weight][item] = max(d[weight][item],
                                      d[weight - cur_weight][item - 1] +
                                      cell[item - 1])

    reincarnation = reincarnate(d, weights, cell)
    print(f"Method without repetitions counted price = {d[W][len(weights)]}")
    print("Included elements are:")

    for i in range(len(reincarnation)):
        if reincarnation[i] == 1:
            print(f"Element {i + 1}, weights {weights[i]} and costs {cell[i]}")
```

Рисунок 7. Функция knapSackWithoutRepetitions и reincarnate

```
Maximum value without repetitions: 48

Method without repetitions counted price = 46
Included elements are:
Element 1, weights 6 and costs 30
Element 3, weights 4 and costs 16
```

Рисунок 8. Результат работы программы

Вывод: в ходе выполнения практической работы были рассмотрены примеры применения динамического программирования для решения трех типов задач: нахождение значения числа Фибоначчи на заданной позиции, поиск наибольшей возрастающей последовательности и нахождение оптимального заполнения рюкзака в задаче комбинаторной оптимизации. В процессе работы освоены основные принципы динамического программирования, который представляет собой эффективный метод решения сложных задач путем их разбиения на более простые подзадачи. Этот подход позволяет избежать повторных вычислений и значительно улучшить эффективность решения задачи.