

Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Кафедра Систем Управления и Информатики

Лабораторная работа №6

Выполнил(и):

Доценников Н.А.

Проверил:

Мусаев А.А.

Санкт-Петербург,
2025

Введение

В данной лабораторной работе рассматриваются задачи динамического программирования, включающие оптимизацию выбора предметов, минимизацию количества операций умножения матриц и нахождение наибольшей возрастающей подпоследовательности в массиве. Цель работы – закрепить навыки разработки алгоритмов с использованием методов динамического программирования.

Задача 1. Задача о воре

Условие: Вор пробрался в музей и хочет украсть N экспонатов. У каждого экспоната есть свой вес и цена. Вор может сделать M заходов, каждый раз унося K кг веса. Необходимо определить, что должен унести вор, чтобы сумма украденного была максимальной.

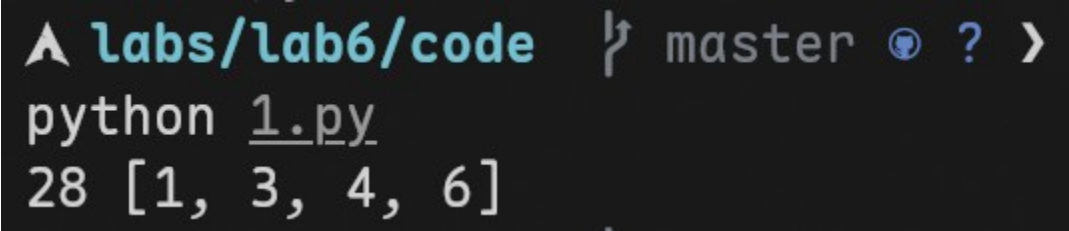
Решение: Задача является вариацией задачи о рюкзаке. Будем использовать подход динамического программирования. Составим таблицу, где строки будут соответствовать экспонатам, а столбцы – возможному весу до $M \cdot K$. Оптимальное решение ищется с использованием перехода: $dp[i][w] = \max(dp[i-1][w], dp[i-1][w - \text{weight}[i]] + \text{value}[i])$.

Реализация (Python) для задачи 1

```
def thief_knapsack(weights, values, M, K):
    cap = M * K
    n = len(weights)
    dp = [0] * (cap + 1)
    take = [[False] * (cap + 1) for _ in range(n)]
    for i in range(n):
        w = weights[i]
        v = values[i]
        for c in range(cap, w - 1, -1):
            if dp[c - w] + v > dp[c]:
                dp[c] = dp[c - w] + v
                take[i][c] = True
    c = cap
    items = []
    for i in range(n - 1, -1, -1):
        if take[i][c]:
            items.append(i)
            c -= weights[i]
    items.reverse()
```

```
return dp[cap], items
```

```
if __name__ == "__main__":  
    weights = [2, 3, 4, 5, 9, 7, 3]  
    values = [3, 4, 5, 8, 10, 7, 6]  
    M = 2  
    K = 10  
    best, items = thief_knapsack(weights, values, M, K)  
    print(best, items)
```



```
^ labs/lab6/code master ? >  
python 1.py  
28 [1, 3, 4, 6]
```

Рисунок 1 – Скриншот работы кода для задачи 1

Задача 2. Умножение матриц

Условие: Дана последовательность матриц $A, B, C, \dots, Z$. Необходимо определить порядок перемножения, минимизирующий количество скалярных операций.

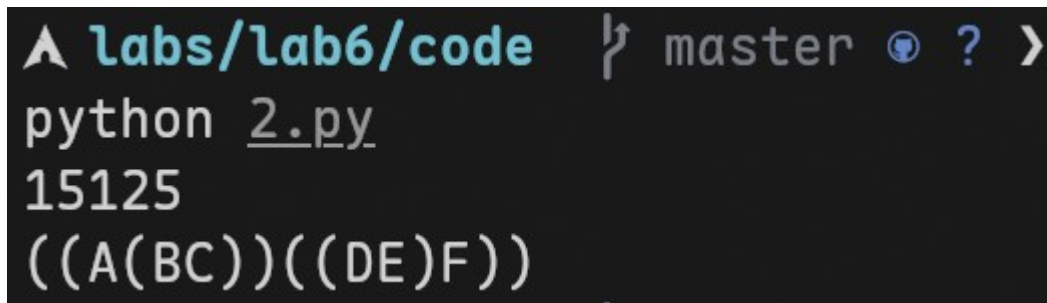
Решение: Опять же будем использовать подход динамического программирования. Определим таблицу минимальных затрат на умножение подпоследовательности матриц. Оптимальное разбиение находится по формуле: $dp[i][j] = \min(dp[i][k] + dp[k+1][j] + p[i-1]*p[k]*p[j])$, где p – размеры матриц.

Реализация (Python) для задачи 2

```
def matrix_chain_order(p):
    n = len(p) - 1
    m = [[0] * n for _ in range(n)]
    s = [[0] * n for _ in range(n)]
    for L in range(2, n + 1):
        for i in range(n - L + 1):
            j = i + L - 1
            m[i][j] = 10**18
            for k in range(i, j):
                q = m[i][k] + m[k + 1][j] + p[i] * p[k + 1] * p[j + 1]
                if q < m[i][j]:
                    m[i][j] = q
                    s[i][j] = k
    return m, s

def build_parenthesization(s, i, j, names):
    if i == j:
        return names[i]
    k = s[i][j]
    return '(' + build_parenthesization(s, i, k, names) + build_parenthesization(s, k + 1, j,
names) + ')'
```

```
if __name__ == "__main__":  
    p = [30, 35, 15, 5, 10, 20, 25]  
    names = ["A", "B", "C", "D", "E", "F"][:len(p)-1]  
    m, s = matrix_chain_order(p)  
    expr = build_parenthesization(s, 0, len(p) - 2, names)  
    print(m[0][len(p) - 2])  
    print(expr)
```



```
^ labs/lab6/code master ? >  
python 2.py  
15125  
((A(BC))((DE)F))
```

Рисунок 2 – Скриншот работы кода для задачи 2

Задача 3. Наибольшая возрастающая последовательность

Условие: Дан массив из n случайных чисел от -100 до 100. Найти наибольшую непрерывную возрастающую подпоследовательность.

Решение: Снова будем использовать динамическое программирование. Если $N[i] < N[i+1]$, увеличиваем текущую длину. Иначе – начинаем новую последовательность. Результат – максимальное значение длины, полученное в процессе.

Реализация (Python) для задачи 3

```
def longest_increasing_run(a):
```

```
    if not a:
```

```
        return 0, -1, -1
```

```
    best_len = 1
```

```
    best_l = 0
```

```
    best_r = 0
```

```
    cur_len = 1
```

```
    cur_l = 0
```

```
    for i in range(1, len(a)):
```

```
        if a[i] > a[i - 1]:
```

```
            cur_len += 1
```

```
        else:
```

```
            if cur_len > best_len:
```

```
                best_len = cur_len
```

```
                best_l = cur_l
```

```
                best_r = i - 1
```

```
            cur_len = 1
```

```
            cur_l = i
```

```
    if cur_len > best_len:
```

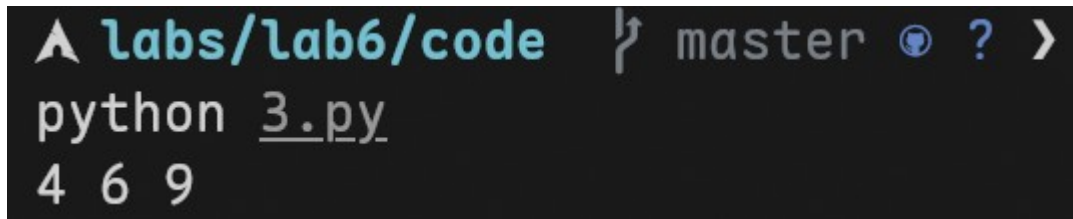
```
        best_len = cur_len
```

```
        best_l = cur_l
```

```
        best_r = len(a) - 1
```

```
return best_len, best_l, best_r

if __name__ == "__main__":
    a = [1, 2, 3, -1, 0, 1, 1, 2, 3, 4, -5]
    length, l, r = longest_increasing_run(a)
    print(length, l, r)
```



```
^ labs/lab6/code ↵ master ? >
python 3.py
4 6 9
```

Рисунок 3 – Скриншот работы кода для задачи 3

Заключение

В ходе выполнения лабораторной работы были изучены и реализованы методы динамического программирования. Рассмотренные задачи продемонстрировали универсальность данного подхода.

Список литературы

1. Википедия. Динамическое программирование. [Электронный ресурс] – https://ru.wikipedia.org/wiki/Динамическое_программирование; (10.05.2025);
2. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Addison-Wesley, 1983. – 427 p. [Электронный ресурс] – <https://doc.lagout.org/Alfred%20V.%20Aho%20-%20Data%20Structures%20and%20Algorithms.pdf>. (Глава 10.2 Dynamic Programming) (10.05.2025).