

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И
ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ**

**Отчет по лабораторной работе №7
по курсу «Алгоритмы и структуры данных»**

**Тема: Динамическое программирование №1
Вариант 1**

**Выполнил:
Бен Шамех Абделазиз
Группа: К3239**

**Проверила:
Ромакина Оксана Михайловна**

**Санкт-Петербург
2025 г.**

Содержание

1 Задача 1. Обмен монет	2
1.1 Условие	2
1.2 Листинг кода	2
1.3 Объяснение решения	2
1.4 Результаты выполнения	2
2 Задача 2. Примитивный калькулятор	3
2.1 Листинг кода	3
2.2 Объяснение решения	3
2.3 Результаты выполнения	3
3 Задача 3. Редакционное расстояние	4
3.1 Листинг кода	4
3.2 Объяснение решения	4
3.3 Результаты выполнения	4
4 Задача 5. Наибольшая общая подпоследовательность трех последовательностей	5
4.1 Листинг кода	5
4.2 Объяснение решения	5
4.3 Результаты выполнения	5
5 Вывод	6

1 Задача 1. Обмен монет

1.1 Условие

Используя динамическое программирование, решите задачу об обмене монет для произвольного набора номиналов. Необходимо найти минимальное количество монет для размена суммы *money*.

1.2 Листинг кода

```
1 def get_min_cnt(amount, coin_arr):
2     dp = [float('inf')] * (amount + 1)
3     dp[0] = 0
4
5     for item in coin_arr:
6         for i in range(item, amount + 1):
7             dp[i] = min(dp[i - item] + 1, dp[i])
8
9     return dp[amount]
```

1.3 Объяснение решения

Создаем одномерную таблицу динамики. По порядку от меньшей суммы к большей пересчитываем минимально возможное количество монет для каждой ячейки на основе доступных номиналов.

1.4 Результаты выполнения

Пример из задачи (*money=6, coins={1,3,4}*):

- Результат: 2 (монеты 3 + 3).

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=5)	0.00078 sec	15.36 Mb
Пример из задачи	0.00082 sec	15.38 Mb
Верхняя граница (n=100000)	0.54783 sec	38.51 Mb

2 Задача 2. Примитивный калькулятор

2.1 Листинг кода

```
1 def prim_calc(n):
2     order_ans = [[] for i in range(n + 1)]
3     order_ans[1] = [1]
4     cnt_oper_arr = [0] * (n + 1)
5
6     for i in range(2, n + 1):
7         min_cnt = float('inf')
8         ind_min_cnt = -1
9
10        if i % 3 == 0 and cnt_oper_arr[i // 3] + 1 < min_cnt:
11            min_cnt = cnt_oper_arr[i // 3] + 1
12            ind_min_cnt = i // 3
13
14        if i % 2 == 0 and cnt_oper_arr[i // 2] + 1 < min_cnt:
15            min_cnt = cnt_oper_arr[i // 2] + 1
16            ind_min_cnt = i // 2
17
18        if cnt_oper_arr[i - 1] + 1 < min_cnt:
19            min_cnt = cnt_oper_arr[i - 1] + 1
20            ind_min_cnt = i - 1
21
22        cnt_oper_arr[i] = min_cnt
23        order_ans[i] = order_ans[ind_min_cnt] + [i]
24
25    return cnt_oper_arr[-1], order_ans[-1]
```

2.2 Объяснение решения

Создаем таблицу для динамики и на каждом шаге пересчитываем минимальный путь (количество операций), с которого можно прийти к текущему числу через допустимые операции (*2, *3, +1).

2.3 Результаты выполнения

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=5)	0.00128 sec	15.11 Mb
Пример из задачи	0.00137 sec	15.21 Mb
Верхняя граница (n=100000)	0.68457 sec	37.84 Mb

3 Задача 3. Редакционное расстояние

3.1 Листинг кода

```
1 def lev_dist(st1, st2):
2     now_row = [i for i in range(len(st1) + 1)]
3     for i in range(1, len(st2) + 1):
4         prev_row, now_row = now_row, [i] + [0] * len(st1)
5         for j in range(1, len(st1) + 1):
6             now_row[j] = min(prev_row[j] + 1, now_row[j - 1] + 1,
7                               prev_row[j - 1] + (st1[j - 1] != st2[i - 1]))
8     return now_row[-1]
```

3.2 Объяснение решения

Находим расстояние Левенштейна по специальному динамическому алгоритму. Используем оптимизацию памяти, храня только две строки таблицы одновременно.

3.3 Результаты выполнения

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=5)	0.00087 sec	14.72 Mb
Пример из задачи	0.00093 sec	14.82 Mb
Верхняя граница (n=100000)	0.56134 sec	36.34 Mb

4 Задача 5. Наибольшая общая подпоследовательность трех последовательностей

4.1 Листинг кода

```
1 def get_longest_sub(arr1, arr2, arr3):
2     dp = [[[0] * (len(arr3) + 1) for j in range(len(arr2) + 1)]
3           for i in range(len(arr1) + 1)]
4
5     for i in range(1, len(arr1) + 1):
6         for j in range(1, len(arr2) + 1):
7             for k in range(1, len(arr3) + 1):
8                 if arr1[i - 1] == arr2[j - 1] == arr3[k - 1]:
9                     dp[i][j][k] = dp[i - 1][j - 1][k - 1] + 1
10                else:
11                    dp[i][j][k] = max(dp[i - 1][j][k],
12                                      dp[i][j - 1][k],
13                                      dp[i][j][k - 1])
14
15     return dp[-1][-1][-1]
```

4.2 Объяснение решения

Создаем трехмерную динамику. Если текущие элементы всех трех последовательностей совпадают, увеличиваем значение по диагонали. В противном случае берем максимум из трех соседних состояний.

4.3 Результаты выполнения

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=3)	0.00095 sec	14.87 Mb
Пример из задачи	0.00107 sec	14.98 Mb
Верхняя граница (n=100000)	0.65414 sec	35.42 Mb

5 Вывод

1. В этой лабораторной работе мы изучили мощный инструмент — динамическое программирование, а также реализовывали алгоритмы, основанные на динамическом подходе (задачи о размене, калькуляторе и редакционном расстоянии).
2. Алгоритмы $O(n)$ и $O(n \log n)$ выполняются достаточно быстро. Затраты памяти при использовании динамических таблиц также прямо пропорциональны линейному росту данных или размеру таблиц.
3. С помощью методов `time.perf_counter()` и `psutil.Process().memory_info().rss` можно эффективно отслеживать ресурсозатратность реализованных алгоритмов.