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: K3239

Проверила:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание

1 Задача 1. Обмен монет

1.1 Условие

Используя динамическое программирование, найти минимальное количество монет для размена суммы *money* при заданном наборе номиналов.

1.2 Листинг кода

```python
def get_min_cnt(amount, coin_arr):
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0
    for coin in coin_arr:
        for i in range(coin, amount + 1):
            dp[i] = min(dp[i], dp[i - coin] + 1)
    return dp[amount]
```

1.3 Текстовое объяснение

Решение строится на заполнении массива `dp`, где индекс — это сумма, а значение — минимальное число монет. Для каждой монеты мы обновляем все возможные суммы в массиве. Итоговая сложность составляет $O(amount \cdot k)$, где k — количество номиналов.

1.4 Результаты выполнения

Ввод: `money=34, coins={1, 3, 4}`. **Вывод:** `9`.

Сценарий	**Время выполнения**	**Затраты памяти**
Пример из задачи	0.00082 sec	15.38 Mb
Верхняя граница (n=10^5)	0.54783 sec	38.51 Mb

2 Задача 2. Примитивный калькулятор

2.1 Листинг кода

```python
def primitive_calculator(n):
    dp = [0] * (n + 1)
    for i in range(2, n + 1):
        variants = [dp[i-1]]
        if i % 2 == 0: variants.append(dp[i//2])
        if i % 3 == 0: variants.append(dp[i//3])
        dp[i] = min(variants) + 1

    curr = n; sequence = [curr]
    while curr > 1:
        if curr % 3 == 0 and dp[curr//3] == dp[curr] - 1: curr //= 3
        elif curr % 2 == 0 and dp[curr//2] == dp[curr] - 1: curr //= 2
        else: curr -= 1
        sequence.append(curr)
    return dp[n], reversed(sequence)
```

2.2 Текстовое объяснение

Для каждого числа от 2 до n мы вычисляем минимальное количество операций, основываясь на уже вычисленных оптимальных значениях для $i-1$, $i/2$ и $i/3$. После нахождения минимального количества шагов выполняется обратный проход для восстановления самой последовательности.

2.3 Результаты выполнения

Ввод: `5`. **Вывод:** `3; 1 2 4 5`.

3 Задача 3. Редакционное расстояние

3.1 Листинг кода

```python
def edit_distance(s1, s2):
    n, m = len(s1), len(s2)
    dp = [[0] * (m + 1) for _ in range(n + 1)]
    for i in range(n + 1): dp[i][0] = i
    for j in range(m + 1): dp[0][j] = j
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            cost = 0 if s1[i-1] == s2[j-1] else 1
            dp[i][j] = min(dp[i-1][j] + 1, dp[i][j-1] + 1, dp[i-1][j-1]
    + cost)
    return dp[n][m]
```

3.2 Текстовое объяснение

Используется классический алгоритм Вагнера-Фишера. Мы строим матрицу, где ячейка (i, j) содержит минимальное количество правок для превращения префикса одной строки в префикс другой. Допустимые операции: вставка, удаление и замена (с весом 1, если символы разные).

3.3 Результаты выполнения

Ввод: `editing, distance`. **Вывод:** `5`.

4 Задача 4. НОП двух последовательностей

4.1 Условие

Вычислить длину самой длинной общей подпоследовательности (НОП) для двух заданных массивов.

4.2 Листинг кода

```
def lcs2(a, b):
    n, m = len(a), len(b)
    dp = [[0] * (m + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if a[i-1] == b[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])
    return dp[n][m]
```

4.3 Текстовое объяснение

Алгоритм сравнивает элементы двух последовательностей. Если элементы совпадают, текущее значение НОП увеличивается на 1 относительно диагонального соседа. Если нет — значение выбирается как максимум из ячейки сверху или слева. Сложность — $O(n \cdot m)$.

4.4 Результаты выполнения

Ввод: `A=[2, 7, 5], B=[2, 5]`. **Вывод:** `2`.

5 Задача 5. НОП трех последовательностей

5.1 Листинг кода

```python
def lcs3(a, b, c):
    n, m, l = len(a), len(b), len(c)
    dp = [[[0]*(l+1) for _ in range(m+1)] for _ in range(n+1)]
    for i in range(1, n+1):
        for j in range(1, m+1):
            for k in range(1, l+1):
                if a[i-1] == b[j-1] == c[k-1]:
                    dp[i][j][k] = dp[i-1][j-1][k-1] + 1
                else:
                    dp[i][j][k] = max(dp[i-1][j][k], dp[i][j-1][k], dp[
   i][j][k-1])
    return dp[n][m][l]
```

5.2 Текстовое объяснение

Эта задача является расширением предыдущей на три измерения. Мы строим трехмерный массив динамики. Элемент инкрементирует результат только при полном совпадении $A[i] = B[j] = C[k]$. Временная и пространственная сложность составляют $O(n \cdot m \cdot l)$.

5.3 Результаты выполнения

Ввод: `A=[1,2,3], B=[2,1,3], C=[1,3,5]`. **Вывод:** `2`.

6 Задача 7. Шаблоны

6.1 Условие

Реализовать проверку соответствия строки заданному шаблону, содержащему спецсимволы ’?’ (любой один символ) и ’*’ (любая подстрока, включая пустую).

6.2 Листинг кода

```python
def is_match(pattern, s):
    n, m = len(s), len(pattern)
    dp = [[False] * (m + 1) for _ in range(n + 1)]
    dp[0][0] = True
    for j in range(1, m + 1):
        if pattern[j-1] == '*': dp[0][j] = dp[0][j-1]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if pattern[j-1] == '*':
                dp[i][j] = dp[i-1][j] or dp[i][j-1]
            elif pattern[j-1] == '?' or s[i-1] == pattern[j-1]:
                dp[i][j] = dp[i-1][j-1]
    return "YES" if dp[n][m] else "NO"
```

6.3 Текстовое объяснение

Для решения используется двумерная динамика. `dp[i][j]` истинно, если префикс строки i соответствует префиксу шаблона j. Обработка ’*’ требует проверки, было ли истинно состояние до появления текущего символа в строке или в шаблоне.

6.4 Результаты выполнения

Ввод: `k?t*n, kitten`. **Вывод:** `YES`.

7 Вывод

1. В данной лабораторной работе был освоен метод динамического программирования для решения задач оптимизации и сравнения последовательностей.

2. Было продемонстрировано, что хранение промежуточных результатов позволяет значительно сократить количество вычислений по сравнению с рекурсивным перебором.

3. Мы научились восстанавливать ответ не только количественно (длина НОП), но и качественно (сама последовательность шагов в калькуляторе).