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

Вариант 21

Выполнила:
Савченко А.С.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №1. Обмен монет

Текст задачи:

---

1 задача. Обмен монет

Как мы уже поняли из лекции, не всегда "жадное"решение задачи на обмен монет работает корректно для разных наборов номиналов монет. Например, если доступны номиналы 1, 3 и 4, жадный алгоритм поменяет 6 центов, используя три монеты (4 + 1 + 1), в то время как его можно изменить, используя всего две монеты (3 + 3). Теперь ваша цель - применить динамическое программирование для решения задачи про обмен монет для разных номиналов.

- **Формат ввода / входного файла (input.txt).** Целое число $money$ ($1 \leq money \leq 10^3$). Набор монет: количество возможных монет k и сам набор $coins = \{coin_1, ..., coin_k\}$. $1 \leq k \leq 100$, $1 \leq coin_i \leq 10^3$. Проверку можно сделать на наборе $\{1, 3, 4\}$. Формат ввода: первая строка содержит через пробел $money$ и k; вторая - $coin_1 coin_2 ... coin_k$.
  - ***Вариация 2:*** Количество монет в кассе ограничено. Для каждой монеты из набора $coins = \{coin_1, ..., coin_k\}$ есть соответствующее целое число - количество монет в кассе данного номинала $c = \{c_1, ..., c_k\}$. Если они закончились, то выдать данную монету невозможно.
- **Формат вывода / выходного файла (output.txt).** Вывести одно число – минимальное количество необходимых монет для размена $money$ доступным набором монет $coins$.
- Ограничение по времени. 1 сек.
- Примеры:

input.txt	output.txt
2 3 1 3 4	2

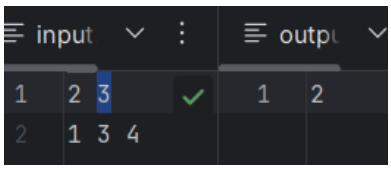
input.txt	output.txt
34 3 1 3 4	9

Листинг кода:

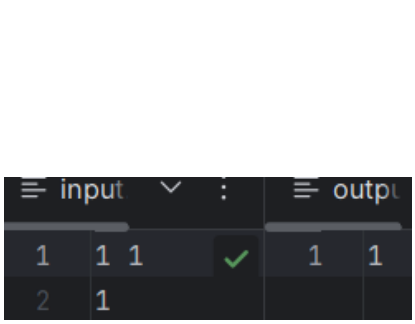
```
def min_coins(money, coins):
    dp = [float('inf')] * (money + 1)
    dp[0] = 0
```

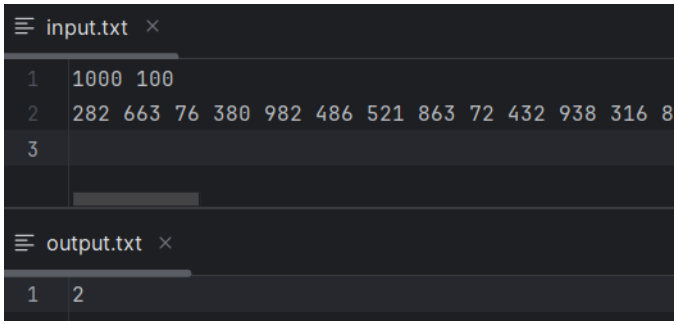
```
    for i in range(1, money + 1):
        for coin in coins:
            if i >= coin:
                dp[i] = min(dp[i], dp[i - coin] + 1)

    return dp[money]


if __name__ == "__main__":
    with open('input.txt', 'r') as input_file:
                                money,    k    =    map(int,
input_file.readline().split())
                                coins    =    list(map(int,
input_file.readline().split()))

    result = min_coins(money, coins)

    with open('output.txt', 'w') as output_file:
        output_file.write(f"{result}")
```

Текстовое объяснение решения:
Создаем массив бесконечностей длинной числа которое будем разменивать. Зануляем первый элемент массива тк 0 не требует размена. Далее для каждой денюжки от 1 до money будем считать мин. кол-во монет ля размена, во внутреннем цикле перебираем все номиналы монет из доступных, проверяем что монету можно использовать для размена суммы, обновляем значение dp[i], выбирая меньшее из кол-ва монет для текуущей суммы i и кол-ва для размена суммы i - coin те dp[i] = min(dp[i], dp[i - coin] + 1). Записываем результат в выходной файл
Результат работы кода на примерах из текста задачи:(скрины input output файлов):



Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0007705 с	14.9921875 Мб
Пример из задачи	0.0009033 с	14.69140625 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.0070412 с	14.7109375 Мб

Вывод по задаче:

Данная задача была решена на основе динамически обновляемого массива. Так как внешний цикл проходит от 1 до money, а внутренний перебирает k штук монет,то получаем что сложность алгоритма O(money*k)

Задача №3.Редакционное расстояние

Текст задачи:

3 задача. Редакционное расстояние

Редакционное расстояние между двумя строками – это минимальное количество операций (вставки, удаления и замены символов) для преобразования одной строки в другую. Это мера сходства двух строк. У редакционного расстояния есть применения, например, в вычислительной биологии, обработке текстов на естественном языке и проверке орфографии. Ваша цель в этой задаче – вычислить расстояние редактирования между двумя строками.

- **Формат ввода / входного файла (input.txt).** Каждая из двух строк ввода содержит строку, состоящую из строчных латинских букв. Длина обеих строк - от 1 до 5000.
- **Формат вывода / выходного файла (output.txt).** Выведите расстояние редактирования между заданными двумя строками.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt	output.txt
ab ab	0

input.txt	output.txt
short ports	3

input.txt	output.txt
editing distance	5

Листинг кода:

```
def levenshtein(A, B):
    n, m = len(A), len(B)
    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):
            if A[i - 1] == B[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
```

```
                    dp[i][j] = min(dp[i - 1][j] + 1,
dp[i][j - 1] + 1, dp[i - 1][j - 1] + 1)

    return dp[n][m]


if __name__ == "__main__":
    with open('input.txt', 'r') as input_file:
        A = input_file.readline().strip()
        B = input_file.readline().strip()

    distance = levenshtein(A, B)

    with open('output.txt', 'w') as output_file:
        output_file.write(f"{distance}")
```

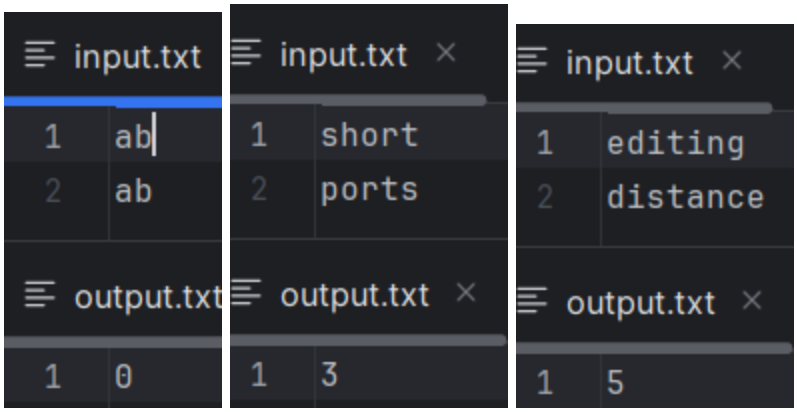
Текстовое объяснение решения:

Редакционного расстояние - расстояние Левенштейна. Реализуем одноименный алгоритм. Создаем динамическую таблицу - двумерный массив dp размером (n + 1) на (m + 1)

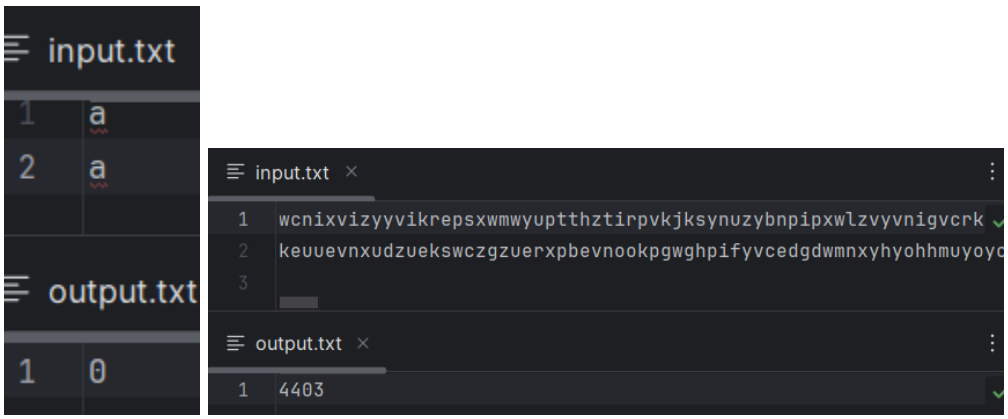
Заполняем базовые случаи: для всех i от 0 до n dp[i][0] = i тк чтобы преобразовать строку из i символов в пустую, нужно удалить i символов, аналогично j от 0 до m dp[0][j] = j чтобы преобразовать пустую строку в строку состоящую из j символов, нужно вставить j символов. Таким образом мы получили таблицу с заполненными верхним и левым ребрами.

Что ж заполняем таблицу дальше, если символы в строках A и B совпадают - менять ничего не нужно --> dp[i][j] = dp[i - 1][j - 1]. В противном случае считаем минимальное количество операций среди трех возможных:1)даление символа из строки A: dp[i - 1][j] + 1 ;2)Вставка символа в строку A: dp[i][j - 1] + 1; 3) Замена символа в строке A: dp[i - 1][j - 1] + 1, обновляем Обновление dp[i][j] на минимальное из 3х случаев.

Результат работы кода на примерах из текста задачи:(скрины input output файлов):



Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0007865 с	14.81640625 Мб
Пример из задачи	0.0007531 с	14.734375 Мб
Верхняя граница диапазона значений входных данных из текста задачи	9.5600499 с	18.015625 Мб

Вывод по задаче:

В этой задаче я реализовала алгоритм Левенштейна для нахождения минимального количества операций, необходимых для преобразования одной

строки в другую. На тесте верхней границы диапазона значений входных данных из текста задачи мы видим превышение ограничений времени в 2 секунды, из чего можно заключить вывод, что python - самым подходящим языком для решения этой задачи.

Задача №7. Шаблоны

Текст задачи:

7 задача. Шаблоны

Многие операционные системы используют шаблоны для ссылки на группы объектов: файлов, пользователей, и т. д. Ваша задача – реализовать простейший алгоритм проверки шаблонов для имен файлов.

В этой задаче алфавит состоит из маленьких букв английского алфавита и точки («.»). Шаблоны могут содержать произвольные символы алфавита, а также два специальных символа: «?» и «*». Знак вопроса («?») соответствует ровно одному произвольному символу. Звездочка «+» соответствует подстроке произвольной длины (возможно, нулевой). Символы алфавита, встречающиеся в шаблоне, отображаются на ровно один такой же символ в проверяемой строчке. Строка считается подходящей под шаблон, если символы шаблона можно последовательно отобразить на символы строки таким образом, как описано выше. Например, строчки «ab», «aab» и «beda.» подходят под шаблон «*a?», а строчки «bebe», «a» и «ba» –нет.

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла определяет шаблон . Вторая строка S состоит только из символов алфавита. Ее необходимо проверить на соответствие шаблону. Длины обеих строк не превосходят 10 000. Строки могут быть пустыми – будьте внимательны!
- **Формат вывода / выходного файла (output.txt).** Если данная строка подходит под шаблон, выведите YES. Иначе выведите NO.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
k?t*n kitten	YES

input.txt	output.txt
k?t?n kitten	NO

Листинг кода:

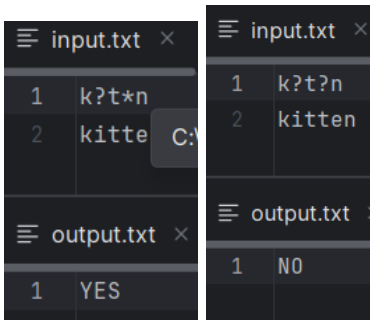
```python
def is_pattern(S, P):
    n, m = len(S), len(P)
    dp = [[False] * (m + 1) for _ in range(n + 1)]

    dp[0][0] = True
    for j in range(1, m + 1):
        if P[j - 1] == '*':
            dp[0][j] = dp[0][j - 1]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if P[j - 1] == '?' or P[j - 1] == S[i - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            elif P[j - 1] == '*':
                dp[i][j] = dp[i - 1][j] or dp[i][j - 1]

    return dp[n][m]

if __name__ == "__main__":
    with open('input.txt', 'r') as input_file:
        P = input_file.readline().strip()
        S = input_file.readline().strip()

    pattern = is_pattern(S, P)

    with open('output.txt', 'w') as output_file:
        if pattern:
            output_file.write("YES")
        else:
            output_file.write("NO")
```

Текстовое объяснение решения:

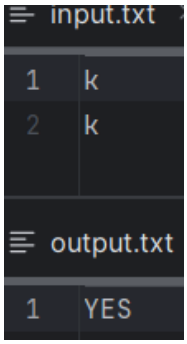
Считываем из файла шаблон P (паттерн) и саму проверяемую строку S. Задачам их длины как m , n соответственно. Будем разделять задачу на подзадачи, для хранения результатов которых создаем динамическую таблицу размером (n+1) * (m+1). Первый элемент равен тру тк пустая строка подходит

под любой шаблон. Идем в цикле по каждому символу паттерна от 1 до m, тк dp[0][j] - пустая строка, проверяем шаблон * и устанавливаем значения True. После заполняем таблицу дальше проверяем если в шаблоне встретился знак ? или текущий символ шаблона совпадает с символом строки,то обновляем значение dp[i][j] на основании значения предыдущих символов. Делаем проверку для символа шаблона *: обновляем dp[i][j] = dp[i - 1][j] если соотв >= 1 символу, dp[i][j] = dp[i][j - 1] - если соотв пустой строке. В конце функция вернет булевое значение соотв тому подходит ли строка под щаблон или нет. Выводим ответ в файл.

Результат работы кода на примерах из текста задачи:(скрины input output файлов):



Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0006745 с	14.92578125 Мб
Пример из задачи	0.0005716 с	15.00390625 Мб

Верхняя граница диапазона значений входных данных из текста задачи	17.9497852 с	17.96875 Мб

Вывод по задаче:

Так как временная сложность O(n×m) вместо python лучше было бы использовать компилируемый ЯП например C++ или Java, чтобы уложиться во временные ограничения 2 секунды. Но для изучения различных алгоритмов и структур данных python тоже подходит так что простим ему это время)

Вывод (по всей лабораторной)

В данной лабораторной мы приступили к тем,е которая более подробно изучается во 2 семестре - Динамическое программирова. Мы создавали динамические массивы таблицы которые позволяли хранить промежуточные результаты и приводить к эффективному решению. К сожалению не все задачи удалось решить уложившись в ограничения по времени в связи с использованием ЯП Python, но тема была изучена, а алгоритмы для подсчета монет и обработки строк, были созданы и протестированы.