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 15

Выполнил:
Скворцов Д.А.
К3140

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №4. Наибольшая общая подпоследовательность двух последовательностей

Вычислить длину самой длинной общей подпоследовательности из двух последовательностей.
Даны две последовательности A = (a1,a2,...,an) и B = (b1,b2,...,bm), найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число p такое, что существуют индексы $1 \le i1 < i2 < ... < ip \le n$ и $1 \le j1 < j2 < ... < jp \le m$ такие, что ai1 = bj1 ,..., aip = bjp.

- **Формат ввода / входного файла (input.txt).**
  - Первая строка: n- длина первой последовательности.
  - Вторая строка: a1,a2,...,an через пробел.
  - Третья строка: m- длина второй последовательности.
  - Четвертая строка: b1,b2,...,bm через пробел.
- Ограничения: $1 \le n,m \le 100; -10^9 < ai,bi < 10^9$.
- **Формат вывода/выходного файла(output.txt).** Выведите число p.
- Ограничение по времени.1 сек.

Листинг кода.

```
def solution(seq1, seq2):
    n = len(seq1)
    m = len(seq2)

    table = [[0] * (m + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if seq1[i - 1] == seq2[j - 1]:
                table[i][j] = table[i - 1][j - 1] + 1
            else:
                table[i][j] = max(table[i - 1][j], table[i][j - 1])

    return table[n][m]
```

Для каждой пары элементов последовательностей проверяется их равенство, для всех подходящих под условие клеток берется предыдущее значение из таблицы, так находится максимальная длина.

```
Lab7. Task 4

| Input   | Output   |
|---------+----------|
| 3       | 2        |
| 2 7 5   |          |
| 2       |          |
| 2 5     |          |
```

```
Lab7. Task 4

Тест 10 элементов:
Время работы: 1.3e-05 секунд
Затрачено памяти: 464 Байт

Тест 50 элементов:
Время работы: 0.000155 секунд
Затрачено памяти: 5.6 Килобайт

Тест 100 элементов:
Время работы: 0.000675 секунд
Затрачено памяти: 20.8 Килобайт
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000013 секунд	464 Байт
Медиана диапазона значений входных данных из текста задачи	0.000155 секунд	5.6 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.000675 секунд	20.8 Мегабайт

Вывод по задаче:

Алгоритм работает за квадратичное время, но из-за размера данных это не влияет на время выполнения. Последовательное, итеративное решение проверяет все необходимые подзадачи.

Задача №6. Наибольшая возрастающая подпоследовательность

Дана последовательность, требуется найти ее наибольшую возрастающую подпоследовательность.

- **Формат ввода / входного файла (input.txt)**. В первой строке входных данных задано целое число n–длина последовательности ($1 \le n \le 300000$). Во второй строке задается сама последовательность. Числа разделяются пробелом. Элементы последовательности – целые числа, не превосходящие по модулю 10^9.
  - Подзадача 1 (полегче). $n \le 5000$.
  - Общая подзадача. $n \le 300000$.
- **Формат вывода / выходного файла (output.txt).** В первой строке выведите длину наибольшей возрастающей подпоследовательности, а во второй строке выведите через пробел саму наибольшую возрастающую подпоследовательность данной последовательности. Если ответов несколько - выведите любой.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

Листинг кода.

```
def solution(lst):
    n = len(lst)

    tails = []
    indices = []
    prev = [-1]*n

    for i in range(n):
        idx = bisect_left(tails, lst[i])

        if idx == len(tails):
            indices.append(i)
            tails.append(lst[i])
        else:
            indices[idx] = i
            tails[idx] = lst[i]
```

```
        if idx > 0:
            prev[i] = indices[idx - 1]

    lis = []
    cur_idx = indices[-1]
    while cur_idx != -1:
        lis.append(lst[cur_idx])
        cur_idx = prev[cur_idx]

    lis.reverse()
    return lis
```

В начале набирается список связей между элементами: для каждого элемента находится ближайший меньший его слева. Также составляется возможная последовательность из индексов и элементов. В конце при помощи сохраненных связей восстанавливается наибольшая подпоследовательность.

```
Lab7. Task 6

| Input         | Output   |
|---------------+----------|
| 6             | 3        |
| 3 29 5 5 28 6 | 3 5 6    |
```

```
Тест 100 элементов:
Время работы: 4.3e-05 секунд
Затрачено памяти: 2.0 Килобайт

Тест 5000 элементов:
Время работы: 0.001867 секунд
Затрачено памяти: 173.2 Килобайт

Тест 3*10e5 элементов:
Время работы: 0.146748 секунд
Затрачено памяти: 10.5 Мегабайт
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.000043 секунд	2.0 Килобайт

входных данных из текста задачи		
Медиана диапазона значений входных данных из текста задачи	0.001867 секунд	173.2 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.146748 секунд	10.5 Мегабайт

Вывод по задаче:
Асимптотическая сложность алгоритма - O(n*log(n)). Так как используется n операций бинарного поиска при наполнении первого массива. Далее - всё линейно.

Дополнительные задачи

Задача №2. Примитивный калькулятор

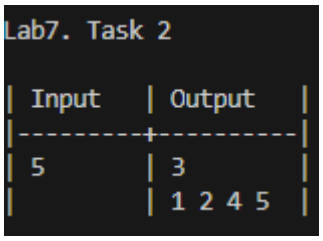
Дан примитивный калькулятор, который может выполнять следующие три операции с текущим числом x: умножить x на 2, умножить x на 3 или прибавить 1 к x. Дано положительное целое число n, найдите минимальное количество операций, необходимых для получения числа n, начиная с числа 1.

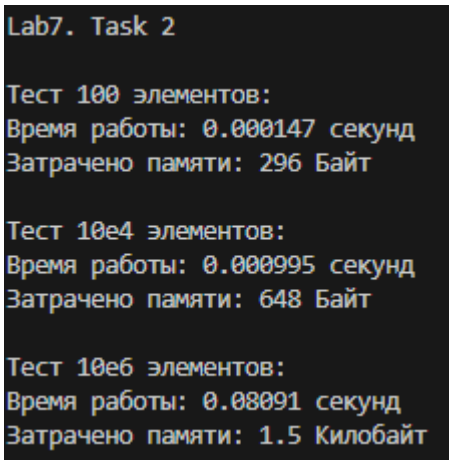
- **Формат ввода / входного файла (input.txt).** Дано одно целое число n, $1 \leq n \leq 10^6$. Посчитать минимальное количество операций, необходимых для получения n из числа 1.
- **Формат вывода / выходного файла (output.txt).** В первой строке вывести минимальное число k операций. Во второй– последовательность проме жуточных чисел a0,a1,...,ak−1 таких, что a0 = 1,ak−1 = n и для всех $0 \leq i < k$–ai+1 равно или ai +1, 2· ai, или 3 · ai. Если есть несколько вариантов, выведите любой из них. 2
- Ограничение по времени. 1 сек.

Листинг кода.

```python
def solution(n):
    stack = [(1, n, 1), (2, n, 1), (3, n, 1)]
    best_res = []
    cur_res = []
    prev_depth = 0

    if n == 1: return [1]

    while stack:
        operation, num, depth = stack.pop()
        if num % operation != 0:
            continue

        new_num = num - 1 if operation == 1 else num // operation

        if depth <= prev_depth:
            # moving from layers of solution
            for _ in range(prev_depth - depth + 1):
                cur_res.pop()

        cur_res.append(new_num)
        prev_depth = depth

        if new_num == 1:
            if len(cur_res) < len(best_res) or len(best_res) == 0:
                best_res = cur_res.copy()
            continue
        elif len(cur_res) >= len(best_res) and len(best_res) != 0:
            continue

        for op in (1, 2, 3):
            stack.append((op, new_num, depth + 1))

    ans = best_res[::-1] + [n]

    return ans
```

Операции раскладываются в дерево решений при помощи стека, чтобы избежать вызова рекурсии. Запоминается наибольшая глубина, а остальное отбрасывается за ненадобностью.





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000147 секунд	296 Байт
Медиана диапазона значений входных данных из текста задачи	0.000995 секунд	648 Байт
Верхняя граница диапазона значений входных данных из текста задачи	0.08091 секунд	1.5 Килобайт

Вывод по задаче:
Время работы алгоритма - O(n), но он расходует память на создание стека под все пути. Чтобы повысить эффективность, программа не проходит все возможные пути, а постоянно сравнивает текущий путь с лучшим и отбрасывает его, как только находит его не превосходящим идеал.

Задача №7. 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, встречаются 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

Листинг кода.

```
def solution(pattern, st):
    n, m = len(st), 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] == '?' or pattern[j-1] == st[i-1]:
```

```
                dp[i][j] = dp[i-1][j-1]
            elif pattern[j-1] == '*':
                dp[i][j] = dp[i-1][j] or dp[i][j-1]

    return "YES" if dp[n][m] else "NO"
```

Создается динамическая таблица, в которой столбцы отведены под шаблон, а строки под входную строку. Далее функция сравнивает предыдущие клетки и записывает в себя булевы значения в зависимости от корректности.

```
Lab7. Task 7

| Input    | Output    |
|----------+-----------|
| k?t*n    | YES       |
| kitten   |           |
```

```
Тест 100 символов:
Время работы: 0.0018 секунд
Затрачено памяти: 83.6 Килобайт

Тест 1000 символов:
Время работы: 0.274221 секунд
Затрачено памяти: 7.7 Мегабайт

Тест 2500 символов:
Время работы: 1.857396 секунд
Затрачено памяти: 47.9 Мегабайт
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0018 секунд	83.6 Килобайт
Медиана диапазона значений входных данных из текста задачи	0.274221 секунд	7.7 Мегабайт
Верхняя граница диапазона значений	1.857396 секунд	47.9 Мегабайт

входных данных из текста задачи		

Вывод по задаче:
Сложность алгоритма - O(n*m) или же O(n^2). Таблица заполняется только на половину, если решение находится сразу.

Вывод

Использование методов динамического программирования позволяет сводить задачи к решению меньших подзадач, последовательно улучшая результат и приближаясь к ответу. Но зачастую решения представляют собой алгоритмы с квадратичной сложностей и такими же затратами по памяти.