Студент: Тимур Хабибуллин
Группа: 1
Дата: 5 мая 2022 г.

---

1. <empty>

2. <empty>

3. **И снова подпоследовательности**
Заведем массив `dp[i]`, в который будем сохранять максимальную сумму подпоследовательности, удовлетворяющей нашим условиям и оканчивающуюся на элементе a_i. Для того, чтобы восстановить последовательность, надо в начале программы создать массив `p[i]`, в i элементе которого на каждом шаге мы будем сохранять предыдущий элемент последовательности. Изначально зададим

$$\texttt{dp[j]} = a_j,\ j < l,$$

тогда

$$\texttt{dp[i]} = \max_{l \leqslant i-j \leqslant r} \texttt{dp[j]} + a_i.$$

Пусть нашли максимум в j элементе, тогда

$$p[i] = j$$

Чтобы восстановить ответ, найдем за линию максимум среди `dp[i]`, потом в массиве `p[i]` возьмем индекс предыдущего элемента, потом еще предыдущего и таким образом восстановим последовательность. На каждом шаге делаем линейный поиск за $O(n)$, всего n шагов, получаем асимптотику $O(n^2)$. Восстановление ответа за $O(n)$.

4. **LCP**
Дана строка `s[0:n]` длины n. Пусть `dp[i][j]` - максимальная длина общего префикса i и j суффиксов.Если элементы `s[i]` и `s[j]` не совпадают, то длина общего префикса равна нулю, потому что префиксы должны совпадать с самого первого элемента. Предположим, что у `s[i:n]` и `s[j:n]` есть общий суффикс, тогда

$$\texttt{s[i+k]} = \texttt{s[j+k]}, 0 \leqslant k \leqslant m,$$

где $m-$ максимальная длина этого суффикса. Префиксы `s[i+1]` и `s[j+1]` имеют общий суффикс максимальной длины $\max(0, m-1)$ (т.е. или его нет, или он есть и ненулевой длины). Если мы узнаем `dp[i+1][j+1]` на предыдущих шагах, то сможем найти длину общего суффикса как

$$\texttt{dp[i][j]} = \texttt{dp[i+1][j+1]} + 1.$$

Поэтому, зададим начальные данные

$$\texttt{dp[i][n]} = 0,\ \texttt{dp[n][j]} = 0,\ i, j = 0...n-1.$$

Переберем все $i, j = (n-1), ..., 0$ и пересчитаем `dp[i][j]` по формуле

$$\texttt{dp[i][j]} = \begin{cases} 0, & \texttt{if s[i]} != \texttt{s[j]} \\ \texttt{dp[i+1][j+1]} + 1, & \texttt{if s[i]} == \texttt{s[j]} \end{cases} \tag{1}$$

Всего операций будет n^2, на каждую операцию тратится $O(1)$ времени, в итоге время работы алгоритма $O(n^2)$.

5. **Нечестные монетки**
Пусть dp[i][j] - это вероятность того, что мы бросили первые j монет и среди них выпало i орлов. Чтобы прийти к такому состоянию у нас два варианта - или мы бросили $j-1$ монетку и у нас выпало i орлов, а последняя монетка выдала решку, или мы бросили $j-1$ монетку и у нас выпало $i-1$ орлов, тогда последняя монетка дала орла. Тогда вероятность события можно посчитать следующим образом:

$$\texttt{dp[i][j]} = (1-p_i)\texttt{dp[i][j-1]} + p_i\texttt{dp[i-1][j-1]}$$

Зададим начальные данные как

$$\texttt{dp[i][0]} = 0, \texttt{dp[0][0]} = 1, \texttt{dp[0][j]} = (1-p_i),\ i, j > 0,$$

ответ к задаче будет находиться в dp[k][n].

6.

7. **Счастливые билетики**
Пусть dp[i][j] - число $i-$значных чисел с суммой цифр, равной j. Зададим начальные данные

$$\texttt{dp[i][0]} = 1,\ i \geqslant 0,\ \texttt{dp[0][j]} = 0,\ ,j > 0$$

Чтобы посчитать dp[i][j] через предыдущие значения, можно рассуждать так- чтобы получить $i-$значное число с суммой цифр j, можно добавить к $(i-1)-$значному числу с суммой $j-x$ в конец цифру $x \in [0, 9]$. А для подсчета dp[i][j] просуммируем все возможные dp[i][j-x]:

```
    for i in range(1, n+1):
        for j in range(1, 9n + 1):
            dp[i][j] = 0
            for x in range(0, min(9, j)):
                dp[i][j] += dp[i-1][j-x]
```

Каждая половинка $2n-$значного счастливого билетика может иметь сумму $j = 0...9n$, число счастливых билетиков можно посчитать как

```
    ans = 0
    for j in range(9n + 1):
        ans += dp[n][j]**2
```

Если нам не подходит вариант билета, где все нули, то надо вычесть единичку из ans. Время работы алгоритма $O(n^2)$.