Студент: Александр Никулин
Дата: 4 октября 2020 г.

Задача 1. *Дана строка s длины n. Для каждой пары (i, j) найти длину максимального общего префикса i -го и j -го суффиксов строки s. $\mathcal{O}\left(n^2\right)$.*

Решение. Развернем строку, тогда задача сведется к поиску максимального суффикса для каждого префикса. Это делается довольно просто. Представим, что для для двух префикосов известно, что их суффиксы совпадают. Тогда, при очередном добавлении нового символа их общий суффикс либо увеличится, либо станет равен 0. Например, для двух префиксов ab и abb добавление еще одной b увеличит общий суффикс на 1, а добавление f сделает его равным 0: abf и abb. Получаем простую формулу перехода:

$$dp[i][j] = \begin{cases} dp[i-1][j-1]+1 & \text{if } s[i] = s[j] \\ 0 & \text{otherwise} \end{cases}$$

Задача 2. *Дан набор нечестных монеток с вероятностью выпадения орла $p_1, p_2, \ldots, p_n$. Требуется посчитать вероятность выпадения ровно k орлов за $\mathcal{O}(n \cdot k)$. Операции над числами считать выполнимыми за $\mathcal{O}(1)$.*

Решение. Задача с точки зрения теорвера не совсем тривиальная и не решается с помощью полиномиальной формулы (кажется я слышал такое от однокурсников). По сути, это сумма n распределений Бернулли, но мы можем об этом забыть. Достаточно знать, что события независимы, а значит в общем вероятность выпадения любой комбинации можно считать как произведение соответствующих вероятностей успеха и неуспеха. Задачу можно довольно просто решить рекурсивно, т.к. нам нужно перебрать k-подмножества и поддерживать произведение вероятностей.

```
1    def prob(ps, k):
2        res = 0
3        def _prob_inner(i, p):
4            if i == k:
5                res += p
6            else:
7                prob_inner(i + 1, ps[i + 1] * p)
8                prob_inner(i + 1, (1 - ps[i + 1]) * p)
9
10       _prob_inner(0, 1)
11       return res
```

.

Однако, время работы будет не $\mathcal{O}(n \cdot k)$, т.к. нам надо перебрать все такие подмножества, то есть $\mathcal{O}(\binom{n}{k})$. Чтобы перейти к динамике надо заметить рекуррентное соотношение. Пусть мы уже подбросили $i-1$ монеток. Тогда, вероятность что на очередном броске мы получим s орлов можно разложить по полной вероятности так:

$$Pr_i^s = p_i * Pr_{i-1}^{s-1} + (1 - p_i) * Pr_{i-1}^s$$

Либо мы на этом броске с вероятностью p_i выбросим орла, либо мы выбросим решку с вероятностью $1 - p_i$, а значит на прошлом броске уже было s орлов. Данную формулу уже несложно преобразовать в dp:

$$dp[i][s] = p[i] * dp[i-1][s-1] + (1 - p[i]) * dp[i-1][s]$$

Осталось разобраться с инициализацией: $dp[0][0] = 1$, первый столбец - произведение обратных вероятностей, а первая строка равна 0, т.к. вероятность выбросить i орлов при 0 бросков равна 0. Теперь алгоритм представляет собой два цикла по n и k, что дает $\mathcal{O}(n \cdot k)$

Задача 3. *Пусть есть n подарков разной натуральной стоимости и три поросёнка. Нужно раздать подар- ки как можно честнее (так, чтобы минимизировать разность суммарной стоимости подарков самого везучего поросёнка и самого невезучего). Придумайте алгоритм решения данной задачи за* $\mathcal{O}\left(nW^2\right)$*, где* W *- суммарная стоимость подарков.*

Решение. Решим вначале для случая двух поросенков. Пусть W - суммарная стоимость подарков. Тогда мы можем разделить на два поросенка только двумя способами, либо W четно и мы поделим ровно пополам, либо нечетно и мы поделим на $\lceil \frac{W}{2} \rceil$ и $\lfloor \frac{W}{2} \rfloor$. Поэтому задача сводится к такой: найти, есть ли такое подмножество, сумма которого равна $\lfloor \frac{W}{2} \rfloor$. Если такое найдется, то мы сможем разделить.

Пусть у нас есть функция, которая может сказать, можно ли извлечь подмножество с суммой i из подпоследовательности $x_1, x_2 \cdots, x_j$, тогда:

$$dp[i][j] = dp[i][j-1] \vee dp[i-x_j][j-1]$$

Действительно, либо из подпоследовательности $j-1$ извлечь можно, а значит можно и из j т.к. сумма только увеличилась, либо можно извлечь с суммой $i - x_j$ из $j-1$ т.к. $(i-x_j)+x_j = i$. Других вариантов быть не может. Тогда ответ будет в $dp[\lfloor \frac{W}{2} \rfloor][n]$. Так как нам надо будет заполнять dp размера $\lfloor \frac{W}{2} \rfloor$ на N, то время работы получится $O(n \cdot W)$.

Для случая с тремя поросенками решение почти аналогично, кроме того, что нам нужно найти разбиение для $\lfloor \frac{W}{3} \rfloor$. Поступим похожим образом, но будем искать сразу для двух подмножеств. Пусть dp[i][j][k] истинно, если из подпоследовательности $x_1, x_2, \cdots, x_k$ можно извлечь два различных подмножества с суммой i и j соответственно. Тогда:

$$dp[i][j][k] = dp[i][j][k-1] \vee dp[i-x_k][j][k-1] \vee dp[i][j-x_k][k-1]$$

Здесь нам нужно заполнить табличку размера $\lfloor \frac{W}{3} \rfloor$ на $\lfloor \frac{W}{3} \rfloor$ на N. Время работы $\mathcal{O}(n \cdot W^2)$.

Признаюсь, мне не совсем понятно как при таком подходе одновременно с этим найти само такое разбиение. Но кажется, что зная ответ, можно разбить жадно.

Задача 4. *Клетки поля* $n \times 5$ *покрашены в чёрный и белый цвета. Будем называть получившийся узор красивым, если он не содержит одноцветного квадрата* 2×2*. Вычислите число красивых узоров по модулю небольшого простого числа за время* $\mathcal{O}(\log n)$*.*

Решение.

Всего способов раскрасить одну строку в два цвета $2^5 = 32$. Предподсчитаем их, а также отдельно построим матрицу $p[i][j]$, равную 1 если i и j раскраску можно приставить друг к другу и 0 в противном случае. Проверять это можно например с помощью операции XOR, в красивой раскраске не будет 0 нулей подряд.

Тогда, пусть $dp[i][j]$ - количество красивых узоров для первых i - 1 строк, которые заканчиваются на j-ую раскраску. Первую строку можно инициализировать единицами. Для перехода нам нужно просто взять сумму $dp[i-1]$ строки по всем раскраскам, которые можно приставить к раскраске j:

$$dp[i][j] = \sum_{k=0}^{32} dp[i-1][k] * p[k][j]$$

Можно заметить, что это легко записать как умножение i - 1 строки на j столбец. Также легко сразу получить всю строку $dp[i]$ домножив $dp[i-1]$ на матрицу p. Каждая следующая строка получается, как:

$$dp[i+1] = dp[i] * p = (dp[i-1] * p) * p = \cdots = dp[0] * p^i$$

Так как нас интересует только последняя строка матрицы dp, то для ее получения достаточно домножить нулевую строку из единиц на матрицу p в степени n. Ответом будет сумма последней строки (по различным раскраскам на которые может заканчивается итоговый узор). Возведение в степень можно сделать за $\mathcal{O}(\log n)$.