Разбор задачи «Буквы на доске»

Будем идти по строке до того, как встретим S, после чего продолжим идти до того, как встретим букву E, и т.д. до того, как встретим букву C. Если мы этим проходом не встретили все четыре буквы, то мы не можем получить хотя бы одну строку $SESC$ и ответ 0. Иначе, мы можем получить хотя бы одну строку $SESC$. Обозначим позицию последней найденной буквы C в строке как z, и повторим описанный алгоритм, снова начиная с первой буквы $SESC$ и позиции $z+1$ данной строки.

Заведём переменную k для хранения ответа, изначально равную 0, и будем прибавлять к ней 1 каждый раз, когда мы в нашем алгоритме возвращаемся от буквы C к букве S. Итого, будем так идти до конца исходной строки. Получившееся в итоге значение k и является ответом.

Разбор задачи «Дежурство по столовой»

Эта задача практически идентична задаче «Два станка» из регионального этапа ВСОШ 2021 года, где она была первой задачей первого дня.

В первых четырех подзадачах достаточно легко найти конкретную формулу, которая позволяет найти ответ. Обозначим искомое количество столов как r. Заметим, что мы можем сначала посчитать r без учёта того, что столов ограниченное количество, после чего посчитать окончательный ответ как $min(n, r)$. Далее приводятся формулы для r без учёта n.

Подзадача 1

Если $a = 0$ и $x = 0$, то первому дежурному вообще нет смысла работать. Проинструктируем второго и получим ответ:

$$r = y \cdot max(k - b, 0).$$

Важно не забыть, что время на инструктаж может оказаться больше k, и тогда количество столов будет равно 0, для чего мы и берем max в формуле.

Подзадача 2

Если a и b равны нулю, то можно сразу проинструктировать дежурных и получить ответ:

$$r = (x + y) \cdot k.$$

Подзадача 3

Когда $a = b$, выгодно сначала проинструктировать наиболее производительного дежурного, и сразу после этого начать инструктировать второго. Ответ тогда будет равен:

$$r = max(x, y) \cdot max(ka, 0) + min(x, y) \cdot max(k2a, 0).$$

Тут снова надо не забыть о случаях, когда время на инструктаж больше доступного нам.

Подзадача 4

Если дежурные протирают столы с одинаковой скоростью, то требуется проинструктировать сначала того, на инструктаж которого уйдет меньше времени, чтобы столы раньше начали протираться. Получаем:

$$r = a \cdot max(kmin(a, b), 0) + a \cdot max(kab, 0).$$

Подзадача 5

Теперь посмотрим как решать общий случай. Переберем два варианта: какого дежурного будем инструктировать первым. Если мы сначала проинструктируем первого дежурного, ответ будет:

$$r1 = x \cdot max(ka, 0) + y \cdot max(kab, 0).$$

Наоборот, если сначала проинструктировать второго, то получим:

$$r2 = y \cdot max(kb, 0) + x \cdot max(kab, 0).$$

Чтобы получить ответ на задачу, достаточно взять максимум из двух этих значений:

$$r = max(r1, r2).$$

Заметим, что, разумеется, решение пятой подзадачи решает также и первые четыре подзадачи.

Разбор задачи «Функция Богдана»

Для того, чтобы решить задачу для $k = 1$ требовалось просто реализовать описанную в задаче функцию. Заметим, что так как оценка идёт по тестам, то решения, в которых допущены незначительные опечатки, тоже набирали баллы.

Для решения задачи для $k \leqslant 10^5$ требовалось к реализации функции добавить цикл, который будет применять к числу n функцию несколько раз. Асимптотика такого решения должна быть $O(k)$, что укладывается в ограничение по времени.

Для полного решения задачи для $k \leqslant 10^9$ требовалось сделать несколько наблюдений.
Во-первых, для некоторых небольших чисел функция входит в цикл:

$$3 \to 3 \to 3 \to ...$$
$$4 \to 6 \to 5 \to 4 \to ...$$

Во-вторых, все небольшие числа через небольшое количество итераций попадают в цикл:

$$1 \to 4 \to ...$$
$$2 \to 3 \to ...$$
$$12 \to 10 \to 6 \to ...$$

В-третьих, все большие числа даже уже после одной итерации значительно уменьшаются:

$$100 \to 3$$
$$999\,999 \to 53$$
$$123\,456\,789 \to 74$$

На основании этих наблюдений можно сказать, что все числа (до 10^9) через небольшое количество итераций попадают в цикл. Это число итераций точно меньше 10^5, поэтому мы можем честно считать функцию, пока не поймём, что попали в цикл. Понять, что мы попали в цикл, можно, если хранить `set` посещённых чисел, и на каждой итерации смотреть, посещали ли мы эту вершину раньше. Также, будем поддерживать количество оставшихся шагов, уменьшая его на 1 с каждой итерацией.

Пусть v - число, в котором мы зациклилсь. Если мы пришли в v менее, чем через k шагов, то искомое число точно находится в этом цикле. Для удобства, пусть k_1 - количество оставшихся шагов после того, как мы вошли в цикл в числе v.

Пусть l - длина цикла. Мы можем её найти, если пойдём от числа v пока не встретим его снова. Тогда l - это количество итераций в таком цикле.

Если мы знаем длину цикла l, то, так как если пройти l шагов от v, то мы снова окажемся в v, то числа, находящиеся в d шагах и в $l + d$ шагах равны. Тогда, искомое число находится в $k_1 \,\%\, l$ шагах от v. Заметим, что $k_1 \,\%\, l < l \leqslant 10^5$, поэтому мы можем просто пройти столько шагов и получить ответ (% - операция взятия остатка от деления).

Разбор задачи «Реалистичная задача»

Подзадача 1

В этой подзадаче предлагалось найти решение за $O(n^3)$. Для этого, можно написать перебор всех возможных мест расположения катапульты, для каждого возможного положения перебирать все возможные точки приземления и для каждой точки приземления x проходом по массиву считать сумму на отрезке $[1, x]$ или $[x, n]$.

Подзадача 2

Оптимизируем решение первой подзадачи. Заметим, что префиксные суммы $[1, x]$ и суффиксные суммы $[x, n]$ можно предпосчитать для всех возможных x. Так как $1 \leqslant x \leqslant n$, мы получим решение за $O(n^2)$ с предпосчётом за $O(n)$.

Подзадача 3

В этой подзадаче можно доказать, что для любого расположения катапульты x самой худшей точкой приземления являются либо $max(1, x - d)$, либо $min(n, x + d)$. Переберём все x и проверим оба варианта за $O(1)$, как описано во второй подзадаче.

Подзадача 4

Оптимизируем решение второй подзадачи. Пусть p - массив префиксных сумм, s - массив суффиксных сумм. Заметим, что когда мы перебираем точки приземления, мы, по сути, берём минимум на каком-то подотрезке p и подотрезке s (эти подотрезки могут быть пустыми). Мы можем воспользоваться структурами данных, например, деревом отрезков или разреженной таблицей, чтобы быстро находить минимум на подотрезке.

Благодаря ограничению на v_i, все структуры могут использовать 32-битные переменные не переполняясь, что влезает в ограничение по памяти. Для использования 64-битных переменных и полного решения задачи нужно использовать меньше памяти.

Подзадача 5

Оптимизируем решение четвёртой подзадачи. Так как остальные структуры занимают слишком много памяти, можно попробовать написать дерево отрезков в реализации снизу или другие похожие структуры. Так как длина отрезка, на котором необходимо найти минимум, фиксирована, то более простым решением будет реализовать структуру для поддержки минимума в скользящем окне. Пример реализации: `https://e-maxx.ru/algo/stacks_for_minima#2`

Разбор задачи «Найди слово»

Пусть s - строка, записанная в полоске, p_i - слово в словаре под номером i $(1 \leqslant i \leqslant m)$.

Подзадача 1

В этой подзадаче требуется найти все вхождения строки p_1 в строке s. Так как длина строки $n \leqslant 1\,000$, то это можно запускаясь от каждой позиции в s и за $O(|p_1|)$ проверять вхождение p_1. После этого, из всех вхождений нужно выбрать такие, что они не пересекаются и их суммарная длина максимальна. Это можно сделать жадным алгоритмом: сначала возьмём самое первое вхождение, потом будем брать самое первое не пересекающееся с уже набранными.

Подзадача 2

В подзадачах с ограничениями $n, m \leqslant 1\,000$ предлагается найти решение за $O(nm)$.

Найдём все позиции в s, в которых начинается какое-либо вхождение любого p_i. Для этого будем перебирать p_i и с помощью алгоритма Кнута-Морисса-Пратта искать все вхождения p_i в s. В результате мы получили некоторое количество вхождений одинаковой длины - мы можем применить жадный алгоритм, описанный в подзадаче 1, чтобы найти ответ.

Подзадача 3

Формально, в этой подзадаче требуется каждый из максимум $\frac{n}{2}$ отрезков разбить на непересекающиеся подотрезки максимальной длины. Допустим, d - длина исходного отрезка, l_i - длина i-го данного слова. Тогда, надо найти максимальное число x, что x - сумма некоторых l_i, а также $x \leqslant d$. Так как $d \leqslant 1000$, то мы можем предпосчитать все возможные суммы l_i с помощью динамического программирования: dp_i - возможно ли набрать сумму i, dp_i - логическое «или» по dp_{i-l_j} для всех l_j. Тогда, для каждого отрезка найдём x - максимальное i, что dp_i верно, с помощью цикла по dp.

Подзадача 4

Найдём все вхождения всех слов из словаря таким же образом, как и в подзадаче 2. Получим не более n^2 отрезков, среди которых надо выбрать некоторые непересекающиеся с максимальной суммарной длиной. Жадный алгоритм из подзадачи 1 в случае произвольных длин отрезков не всегда даёт правильный ответ. Решить задачу за подходящее время в этом случае можно с помощью динамического программирования.

Пусть у нас есть массив отрезков, l_i, r_i - левая и правая границы этих отрезков. Отсортируем отрезки по правой границе. Пусть dp_i - максимальная длина выбранных отрезков, если самый правый выбранный отрезок - i-й. Найдём такой самый правый отрезок j, что $r_j < l_i$ с помощью бинарного поиска. Будем поддерживать массив префиксных максимумов $pmax$ по dp. Тогда, $dp_i = pmax_j + r_i - l_i + 1$. Максимум по dp и будет являться ответом.

Описанный алгоритм имеет асимптотику $O(n^2 \log n^2)$.

В задаче также требуется реализовать восстановление ответа, для этого можно вместе с значением префиксного максимума $pmax_i$ хранить его индекс $pind_i$ и для каждого dp_i хранить $pred_i$ - предыдущий взятый отрезок. Тогда, $pred_i = pind_j$.

Подзадачи 5 и 6

Найдём все вхождения всех слов из словаря в строке s быстрее, чем $O(nm)$. Для этого можно построить бор из словаря и запускаться от каждой позиции i в s, ищя вхождения, начинающиеся в i. Такой алгоритм имеет асимптотику $O(n^2)$. Также, можно сложить хеши всех строк из словаря в `set` и перебрать все подотрезки s, ищя их хеши в `set`, получая асимптотику $O(n^2 \log m)$. Найти все вхождения также можно используя алгоритм Ахо-Корасик за $O(n^2)$ (n^2 - общая длина всех совпадений).

Получим не более n^2 отрезков, среди которых надо выбрать некоторые непересекающиеся с максимальной суммарной длиной. Сделать это можно жадным алгоритмом из подзадачи 1 (для решения подзадачи 5) или с помощью динамики из подзадачи 4 (для решения подзадачи 6).