Задача A. Не иллюзия обмана

Автор задачи: Павел Скобелин, разработчик: Степан Мишанин

Пусть изначально не было сделано ни одного разреза, тогда Всадники смогут унести с собой $\lfloor \frac{n}{2} \rfloor$ слитков золота. Очевидно, что при любом раскладе Всадники заберут себе максимальные по весу слитки золота из тех, что могут забрать.

Тогда как нужно разрезать слитки, чтобы улучшить ответ? Во-первых, нам точно не нужно резать слитки, что уже у нас, так как тогда большой слиток, который был у нас разделится на два более маленьких и какая-то из частей останется в сейфе. Следовательно, мы всегда режем только меньшие $\lceil \frac{n}{2} \rceil$ слитков. Во-вторых, мы всегда режем наименьший слиток из тех, что можем разрезать, то есть наименьший больший единицы, который находится не у нас. Почему так? Для начала заметим, что если после очередного разреза количество слитков стало нечётным, то итоговый результат никак не изменился, ведь мы разрезали слиток, что не у нас, но он все равно должен остаться в сейфе, а если количество стало чётным, то мы можем взять себе какой-то один слиток, — очевидно, что мы хотим взять себе наибольший по весу из тех, что ещё не у нас. Исходя из этого, наконец, несложно понять, что мы хотим совершить как можно больше разрезов кусков, что ещё не у нас, а значит наименьший слиток, который мы сейчас режем, будем резать на слитки массой 1 и $x - 1$, где x — масса этого слитка.

Таким образом, для решения достаточно резать Минимальный по весу слиток как описано выше до тех пор, пока можно — то есть до тех пор, пока все меньшие $\lceil \frac{n}{2} \rceil$ слитков не станут веса 1. Каждые два разреза количество слитков веса 1 увеличивается на 2, а один слиток максимального веса из меньших $\lceil \frac{n}{2} \rceil$ слитков отходит Всадникам, а значит они смогут сделать не более $\mathcal{O}(n)$ разрезов. Тогда для решения задачи отсортируем изначальный массив весов в порядке возрастания, добавим в ответ сумму весов наибольших $\lfloor \frac{n}{2} \rfloor$ слитков золота из второй половина массива, а затем будем идти двумя указателями по первой половине массива и аккуратно обрабатывать условия, описанные в предыдущем абзаце. Получится асимптотика решения $\mathcal{O}(n \log n)$.

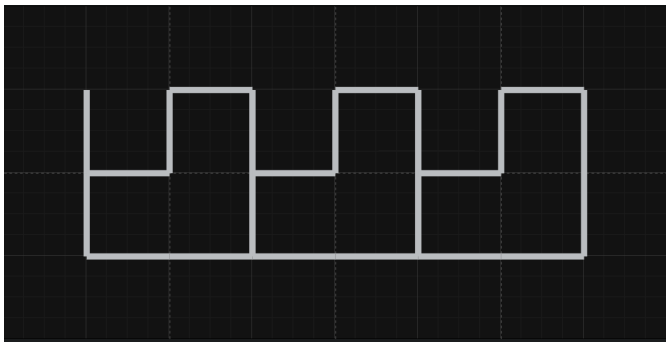
Задача B. Спичечный трюк Лулы

Автор задачи: Павел Скобелин, разработчик: Иван Пашков

Ограничения в условии позволяют найти ответ с помощью динамического программирования по профилю. С помощью него можно показать, что меньше чем с помощью $\lceil \frac{3 \cdot n}{2} \rceil$ спичек нельзя выполнить условие.

Давайте построим конкретную конструкцию для $\lceil \frac{3 \cdot n}{2} \rceil$. Будем удалять следующим образом: для каждого столбца клеток $c = 1, 2, \ldots, n$ удалим верхнюю спичку, если c четно, иначе среднюю. Также удалим нижние вертикальные спички для четных вертикальных линий.

Для $n = 6$ конструкция будет выглядеть следующим образом:



Задача C. Иллюзия суммы

Автор задачи: Павел Скобелин, разработчик: Вдовин Антон

Найдем серию решений, в которых $b = a$. Тогда:

$$2a^n = c^{n+1}$$

$$c = 2z$$

Пусть $z = x^n$

$$c = 2x^n$$

$$2a^n = (2x^n)^{n+1}$$

$$(2x^n)^{n+1} = 2^{n+1}x^{n(n+1)}$$

$$a^n = 2^n x^{n(n+1)}$$

$$a = 2x^{n+1}$$

Итого имеем семейство решений по x: $a = b = 2x^{n+1}$ и $c = 2x^n$. При заданных ограничениях $2k^{21} \leqslant 10^{18}$, а значит, достаточно рассмотреть x от 1 до k.

Задача D. XOR Фибоначчи

Автор задачи и разработчик: Ильдар Гайнуллин

Рассмотрим последовательность чисел Фибоначчи, не превосходящих m.

Если мы рассмотрим все числа как вектора по модулю два, то почти все числа Фибоначчи находятся в базисе, кроме порядка двадцати. Так как часто старший бит F_k отличается от старшего бита F_{k-1}.

Переберем для всех чисел Фибоначчи, которые не входят в базис, входят ли они в итоговый XOR (встречаются нечетное число раз), или нет, пусть их XOR равен x. Дальше разложим $n \oplus x$ в линейную комбинацию векторов базиса. Так мы переберем порядка 2^{20} различных векторов, которые соответствуют числам Фибоначчи, которые встречаются нечетное число раз. Если их сумма равна y, то дальше необходимо $m - y$ представить в виде суммы удвоенных чисел Фибоначчи, что можно сделать жадно, при правильной четности.

Задача E. Иллюзия XOR-а

Автор задачи и разработчик: Егор Юлин

Для решения данной задачи заметим следующий факт:

Пусть $f(u)$ будет равно красоте пути от корня до вершины u, тогда красота пути между вершинами u и v будет равна $f(u) \oplus f(v)$.

Тогда если мы для каждой вершины посчитаем красоту от корня до нее, то нам останется как-то быстро объединить попарно значения $f(u)$ и $f(v)$ для всех пар (u, v).

Это можно сделать с помощью `xor`-свертки или же преобразованием Уолшера-Адамара, которое находит для каждого $c_i = \sum_{j;k;j \oplus k=i} a_j \cdot b_k$. Тогда если мы запишем в a_i значение равное количеству путей красоты i и сделаем $b = a$, то в c_i будет лежать ответ для значения i.

Задача F. Термальные шнуры

Авторы задачи: Даниил Орешников и Ильнур Валеев, разработчик: Вдовин Антон

Заметим, что операции следует делать только в те моменты времени, о которых мы точно знаем. Условно, если сейчас время t и какая-то свеча сгорит через x секунд, то следующие операции мы сможем сделать только во время $t + x$. Будем решать полным перебором. Изначально в 0-й момент времени мы можем сделать 3 различных операций для каждого шнура: не трогать, поджечь с одной стороны, поджечь с двух сторон, затем время сдвинется на столько, за сколько догорит первая (не по порядку в массиве, а в хронологическом порядке) свеча, затем можем сделать то же самое, что и в 0-й момент. Для удобства будем использовать рекурсию, на каждом шаге поддерживая состояние свечи — со скольких сторон она горит (от 0 до 2) и обновим сколько ей осталось гореть.

Задача G. Вечный двигатель

Авторы задачи: Даниил Орешников и Ильнур Валеев, разработчик: Степан Мишанин

Исправным механизмом является механизм, у которого первой стоит шестерёнка с максимальным количеством зубчиков, а последней — с минимальным. Покажем, почему это так.

Рассмотрим для начала, как ведут себя две шестерёнки: пусть вращающаяся имеет k зубьев, а вращаемая — m, тогда если вращающаяся совершает q оборотов в единицу времени, то вращаемая за это же время совершит $\frac{qk}{m}$ оборотов. Например, шестерёнка с 6 зубьями вращает шестерёнку с 3 зубьями со скоростью 1 оборот в единицу времени — тогда шестерёнка с 3 зубьями совершит за единицу времени 2 оборота. Из этого уже очевидно, что первой нужно поставить шестерёнку с максимальным количеством зубьев, а последней — с минимальным. Теперь поймём, почему порядок остальных шестерёнок не имеет значения. Заметим, что во вращение приводится только первая шестерёнка, так что выбрав её, мы как бы «фиксируем» количество зубьев, на которое прокрутится весь наш механизм. Иначе говоря, первая шестеренка прокручивается на количество её зубьев, следующая за ней — на это же количество, но если в ней зубьев меньше, то она совершит больше оборотов, и так далее. Таким образом, от расстановки остальных шестерёнок ничего не изменится, ведь последняя шестерёнка прокрутится на столько же зубьев, на сколько прокрутится первая.

Следовательно, задача сводится к тому, чтобы просто найти сумму расстояния от шестерёнки с максимальным количеством зубчиков до начала механизма и расстояния от шестерёнки с минимальным количеством зубчиков до конца механизма, ведь каждым обменом шестёренка становится на единицу расстояния ближе либо к началу, либо к концу механизма. Заметим также, что если шестерёнок с максимальным количеством зубьев было несколько, то следовало выбрать ту, которая ближе к началу, аналогично из минимальных — ту, что ближе к концу. Наконец, нужно проверить, что если шестерёнка с максимальным количеством зубьев стоит после шестерёнки с минимальным, ответ нужно уменьшить на единицу, ведь в какой-то момент мы поменяем их местами и они одновременно станут ближе к нужным им позициям.

Задача H. Расширение Всадников

Автор задачи и разработчик: Павел Скобелин

Для начала поймем, что ответ на запрос — это сумма величины $(\lfloor \frac{t-1}{k} \rfloor \mod 2) \cdot x$ по всем актуальным командам.

Давайте зафиксируем константу B, которую определим позже. Тогда назовем команды с $k \leqslant B$ «маленькими», остальные — большими. Научимся отдельно работать с маленькими и большими командами.

Для маленьких команд давайте поддерживать сумму x-значений по всем командам с таким k. Таким образом, добавлять команду (k, x) мы будем за $\mathcal{O}(1)$, просто прибавляя значение x к сумме по этому k. Считать ответ для маленьких команд тоже несложно — для конкретного t пройдем по всем маленьким k и посчитаем необходимую сумму за $\mathcal{O}(B)$. Таким образом, обработка всех запросов с маленькими командами работает за $\mathcal{O}(q \cdot B)$.

Научимся обрабатывать большие команды. Положим T как максимально возможный запрос. Тогда давайте поддерживать ответ для всех t от 1 до T. Тогда для добавления новой команды (k, x), нужно добавить x на таких отрезках: $[1; k], [2k + 1; 3k], [4k + 1; 5k], \ldots$, и так далее до T. Заметим, что таких отрезков будет не более $\frac{T}{2k} \leqslant \frac{T}{2B} = \mathcal{O}(\frac{T}{B})$. Тогда, нам нужно научиться добавлять x на отрезке при добавлении команды, и считать значение в точке t при запросе в момент t, и прибавить ответ для маленьких команд, посчитанный за $\mathcal{O}(B)$.

Осталось реализовать это достаточно эффективно. Первым в голову приходит решение, использующее дерево отрезков с массовыми операциями. Оценим асимптотику такого решения. Запросы update и get в дереве отрезков будут работать за $\mathcal{O}(\log T)$, тогда итоговая асимптотика решения будет равна $\mathcal{O}(q \cdot B + q\frac{T}{B} \log T)$. Если положить $B = \sqrt{T \log T}$, то асимптотика решения будет равна $\mathcal{O}(q\sqrt{T \log T} + q\frac{T}{\sqrt{T \log T}} \log T) = \mathcal{O}(q\sqrt{T \log T})$ — что очень неплохо асимптотически, но, к сожалению, константа массового дерева отрезков не позволит этому решению отработать за нужное время при данных ограничениях.

Для полного нужно было воспользоваться техникой разностного массива. Давайте вместо ответа для каждого t, хранить массив соседних разностей. Несложно показать, что в таком случае

запрос прибавления на отрезке затронет ровно 2 позиции разностного массива, а запрос значения в точке будет равен сумме на префиксе. Эти операции уже более простые, и их можно было делать с помощью дерева отрезков, не использующее массовые операции. То есть, это работает за ту же асимптотику, что и прошлое решение — $\mathcal{O}(q\sqrt{T \log T})$, только с более приятной константой. У жюри есть решение с деревом отрезков, укладывающееся в TL более чем в 2 раза. Также, вместо дерева отрезков в данном случае можно было использовать дерево Фенвика — оно сильно более легковесное, и решение жюри с деревом Фенвика укладывается в TL более чем в 4 раза.

Но это решение можно улучшить еще сильнее, причем асимптотически. В этот раз положим $B = \sqrt{T}$. Давайте поймем, что в нашем прошлом решении запрос изменения в точке нужно было применить порядка $\mathcal{O}(q\sqrt{T})$ раз, а запрос суммы на префиксе — $O(q)$ раз. Это наталкивает на мысль — не обязательно, чтобы оба запроса имели одинаковую асимптотику. Тогда, чтобы решить задачу за $\mathcal{O}(q\sqrt{T})$, нужно придумать структуру с $\mathcal{O}(1)$ на изменение в точке и $\mathcal{O}(\sqrt{T})$ на сумму на префиксе. Несложно показать, что такая структура — это обычная корневая декомпозиция, в которой мы побьем массив на блоки размера $\sqrt{T}$ и будем поддерживать сумму в каждом блоке. Таким образом, задачу можно было решить за асимптотику $\mathcal{O}(q\sqrt{T})$, что легко укладывается в TL.

Задача I. Повороты магических меток

Автор задачи и разработчик: Вдовин Антон

Рассмотрим последовательность из n точек $(x_i;\ y_i)$ на плоскости. К каждому запросу задан отрезок индексов $[l;\ r]$, точка $(X;\ Y)$ и угол $a \in [90^o, 180^o, 270^o]$. Необходимо повернуть все точки с номерами от l до r вокруг $(X;\ Y)$ на угол a по часовой стрелке и после этого узнать минимальную площадь осевого прямоугольника, покрывающего все точки.

Поворот вокруг произвольного центра $(X;\ Y)$ представляется как композиция трёх аффинных преобразований: сдвига $(x,\ y) \to (x-X,\ y-Y)$, поворота на a градусов вокруг точки $(0,\ 0)$ и сдвига $(x,\ y) \to (x+X,\ y+Y)$.

Для поворотов на углы кратные 90 используем следующие формулы $r(x',\ y',\ 90^o) = (y,\ -x)$, $r(x',\ y',\ 180^o) = (-x,\ -y)$, $r(x',\ y',\ 270^o) = (-y,\ x)$.

Для поддержки операций по подотрезкам используем декартово дерево по неявному ключу (implicit treap), где каждая вершина соответствует одной точке. В вершине хранятся координаты точки $(x,\ y)$, минимальное и максимальное значения x по поддереву, минимальное и максимальное значения y по поддереву, размер поддерева и случайный приоритет. Для ленивых преобразований в вершине дополнительно хранятся вектор сдвига $(add_x,\ add_y)$ и параметр поворота $rot \in [0, 1, 2, 3]$ и обозначающий поворот на 90 градусов вокруг начала координат. Применение сдвига или поворота к поддереву реализуется обновлением координаты самой точки, пересчетом четырёх агрегированных экстремумов по соответствующим формулам и корректировкой ленивых тегов; при повороте вектор (add_x, add_y) также поворачивается.

Задача J. Теперь ты меня видишь

Автор задачи: Дмитрий Ивченко, разработчик: Тимур Кузнецов

Переформулируем задачу: найдём максимальную длину, при которой любая строка является подпоследовательностью s. Обозначим её за l. Тогда ответ на нашу задачу — $l + 1$.

Заметим, что если для некоторого l любая строка длины l достижима, и в оставшейся части строки (после некоторой позиции) встречаются все n букв, то любая строка длины $l + 1$ также достижима.

Это приводит к жадному алгоритму. Идём по строке слева направо, поддерживая l и множество букв, которые мы можем дописать к любой строке длины l (изначально $l = 0$). Будем постепенно добавлять буквы в множество. Как только в множестве есть все n букв, мы можем дописать к любой строке длины l любой символ, получив подпоследовательность в s, поэтому увеличиваем l на 1 и сбрасываем множество.

Множество можно эффективно поддерживать, например, при помощи битовых маск.

Задача K. Иллюзия размена

Автор задачи и разработчик: Павел Скобелин

Попытаемся понять, как считается ответ при разделении числа S на множество чисел $x_1, x_2, \dots x_k$. Утверждается, что стоимость этого действия **никак не зависит** от порядка действий. Значит, посчитать её можно любым удобным способом (выбрав самый простой порядок). Давайте покажем, почему это так.

Для доказательства можно представить S как полный граф на S вершинах. Что же происходит при разбиении $S = a + b$? Мы разбиваем S на 2 полных графа размеров a и b, при этом мы платим столько монет, сколько рёбер мы удаляем. Таким образом, что происходит, когда за несколько действий мы превратили S в какой-то набор чисел? Изначально был полный граф — в нём было $\frac{S(S-1)}{2}$ рёбер. После этого действия, осталось k полных графов, в каждом из которых $\frac{x_i(x_i-1)}{2}$ рёбер. Значит, вне зависимости от порядка действий, мы удалили $\frac{S(S-1)}{2} - \sum_{i=1}^{k} \frac{x_i(x_i-1)}{2}$ рёбер, значит потратили столько монет.

Далее поймём, что если S меньше суммы на отрезке $[a_l, \dots a_r]$, то ответа не существует. При этом, если S больше суммы на отрезке, то выгодно первым действием «откусить» эту разницу в сумме (её не выгодно делить на большие части и переплачивать за это). Значит, далее можно считать, что $S = \sum_{i=l}^{r} a_i$. Получается, что осталось научиться считать сумму величины $\frac{a_i(a_i-1)}{2}$ на отрезке, что можно сделать с помощью префиксных сумм. Таким образом, решение работает за $\mathcal{O}(n+q)$.

Задача L. Хакерская задача

Авторы задачи: Даниил Орешников и Ильнур Валеев, разработчик: Вдовин Антон

Для каждого слова $w \in [s_1, s_2, \dots, s_n]$ посчитаем внутреннюю стоимость $in_w = \sum_{i=1}^{|w|-1} dist(w_i, w_{i+1})$, где $dist(p, q)$ — манхэттен-расстояние между кнопками букв. Если перед словом палец стоит на букве p, то стоимость напечатать слово w и закончить в букве $q = w_{|w|}$ равна $C(p \to^w q) = dist(p, w_1) + in_w$.

Построим матрицу переходов из буквы p в q, т.е $M_{p,q} = \min_{w\ (w_{|w|}=q)} C(p \to^w q)$. Также для выбора первого слова построим массив $base_q = \min_{w\ (w_{|w|}=q)} in_w$. Пусть dp_t — минимальная стоимость напечатать t слов по всем возможным последним буквам. Тогда $dp_1 = base$, а переход будет $dp_{t+1,q} = \min_p(dp_{t,p} + M_{p,q})$. Иными словами $dp_k = base \otimes M^{k-1}$, где $\otimes$ — операция выше. На каждом шаге бинарного возведения перемножаем матрицы по правилу: $C_{i,j} = \min_t(A_{i,t} + B_{t,j})$.