Задача A. A-голов B-ног

Пусть у нас x людей и y АБшек.

Тогда имеет место следующая система уравнений:

$$\begin{cases} x + A \cdot y = n \\ 2 \cdot x + B \cdot y = m \end{cases}$$

Получаем:

$$y = \frac{m - 2 \cdot n}{B - 2 \cdot A}, x = n - y$$

Решение отсутствует в случае если:
$(m - 2 \cdot n) \mod (B - 2 \cdot A) \neq 0$ или $x < 0$ или $y < 0$

Отдельно следует рассмотреть случай когда $B = 2 \cdot A$. В этом случае любого АБшку можно представить как A людей.

Задача B. Циферблат

Для удобства определим: pos_i, neg_i — минимальное количество сил которые надо потратить чтобы a_i стало положительным и отрцательным соответственно.

$$pos_i = \begin{cases} 0 & a_i > 0 \\ \lceil \frac{(-a_i+1)}{c_i} \rceil \cdot b_i & a_i \leqslant 0 \end{cases} \quad neg_i = \begin{cases} 0 & a_i < 0 \\ \lceil \frac{(a_i+1)}{d_i} \rceil \cdot b_i & a_i \geqslant 0 \end{cases}$$

$\lceil * \rceil$ — округление вверх.

Пусть $dp[pos][i], dp[neg][i]$ — минимальное количество сил которые надо потратить чтобы произведение первых i чисел стало положительным и отрцательным соответственно.

Получаем следующую формулу пересчета:

$$dp[pos][0] = 0, dp[neg][0] = +\infty$$

$$dp[pos][i] = min(dp[pos][i-1] + pos_i, dp[neg][i-1] + neg_i)$$

$$dp[neg][i] = min(dp[pos][i-1] + neg_i, dp[neg][i-1] + pos_i)$$

Ответ: $dp[pos][n]$

Задача C. Привлекательное разбиение

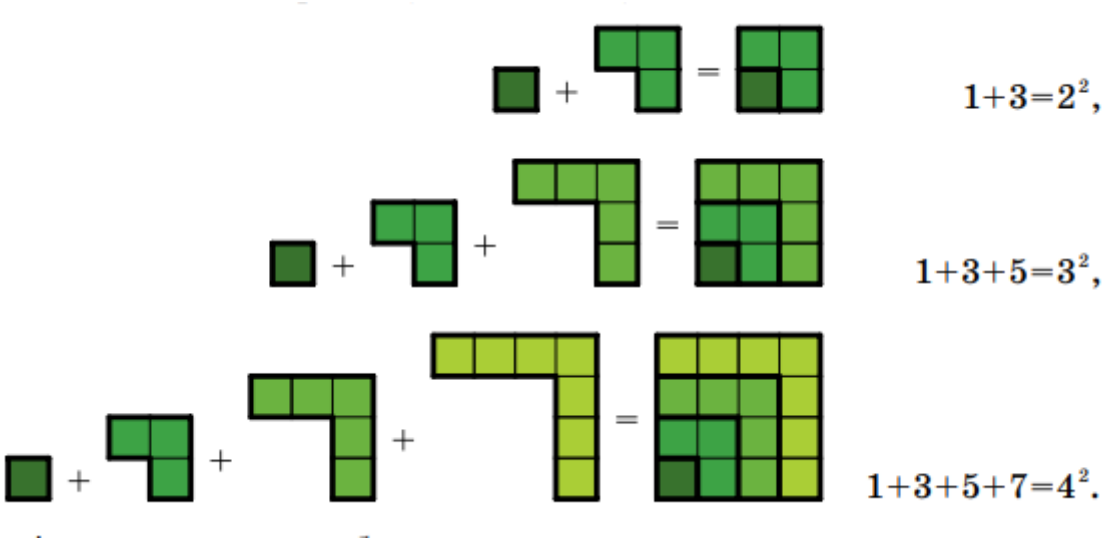
Подзадача $1 \leqslant 32, m = 10^5$
Можно сообразить что-то вручную на листочке для всех возможных n

Подзадача 2(16 баллов) $n = k^2 - 1, m = 10^5$ для некоторого натурального k, n – чётное
Очевидно следующее:

$$1 + n = 2 + (n - 1) = 3 + (n - 2) = ... = k^2$$

Подзадача 3(18 баллов) $m = 10^5$, необходимые подзадачи – 1, 2
Вероятно какое-то разбиение есть, и не одно.

Подзадача 4(46 баллов) $m = 20$, необходимые подзадачи – 1, 2, 3, 4
Заметим, что сумма нечетных чисел — это квадрат.



Выделим все нечетные числа в группу — останутся четные.
Заметим следующее:

$$2+4+6+8+...=2\cdot(1+2+3+4+...)=2\cdot(1+3+...)+2\cdot(2+4+...)$$

Также заметим, что 2 — простое. Выделим $2\cdot(1+3+...)$ как группу, останется $2\cdot(2+4+...)$.
В оставшейся части повторим наш приём внутри скобок.
Получим:

$$2\cdot(2+4+6+8+...)=2\cdot(2\cdot(1+2+3+4+...))=2\cdot2\cdot(1+3+...)+2\cdot2\cdot(2+4+...)$$

Заметим, что произведение двух квадратов также является квадратом.
Продолжая данную процедуру мы выделяем в группу либо квадрат, либо квадрат ·2.
Каждый раз, после такой процедуры, размер отавшегося множества чисел уменьшается в два раза, поэтому количество групп не превысит $\log_2 n$.

Задача D. Крош и пары соседних элементов

Для первой подзадачи можно перебрать все подмножества пар, которые будут хорошими. Тогда все хорошие пары разобьются на подотрезки подряд идущих хороших пар. Если мы научимся решать задачу определения минимального количества операций, необходимых для того, чтобы сделать подотрезок хорошим, то мы решим задачу.

Очевидно, что невыгодно делать операции увеличения на граничных элементах подотрезка(только если он не длины 2), так как можно сделать эти прибавления на внутренних элементах и ситуация будет не хуже. Тогда будем идти по подотрезку слева направо, начиная со второго элемента, поддерживая значение предыдущего элемента. При рассмотрении текущего элемента выгодно сделать такое количество прибавлений к текущему, чтобы его сумма с предыдущим была больше нуля, и продолжить алгоритм, переходя к следующему.

Именно, пусть $prev$ – значение предыдущего элемента, а cur – значение текущего элемента. Тогда нужно сделать $add = max(0, -(cur + prev - 1))$ прибавлений к текущему элементу, присвоить $prev = cur + add$ и перейти к следующему элементу. Такой жадный алгоритм пригодится нам для следующих подзадач.

Для решения второй подзадачи заметим, что каждый элемент в итоговом массиве лежит в диапазоне $[-100, 101]$. Тогда можно рассмотреть динамику – $dp(prefix, bad, last)$ – наименьшее количество операций, чтобы среди первых $prefix$ было не более bad плохих пар, а последний элемент был равен $last$. Для перехода переберем предыдущий элемент $prev$, если $prev + last > 0$, то пара будет хорошей, иначе плохой. Таким образом, получаем решение за $O(n^2 max^2)$, которое может быть ускорено до $O(n^2 max)$ с помощью частичных минимумов.

Рассмотрим следующую динамику для решения следующих подзадач:

$dp(prefix, bad)$ – рассмотрели префикс длины $prefix$, в котором bad плохих пар. Для обновления переберем индекс j такой, что все пары на подотрезке $[j, prefix]$ будут хорошими(отрезок может быть длины 1, то есть не иметь пар вообще). Тогда $dp(prefix, bad)$ обновляем значением $dp(j-1, bad) + cost(j, prefix)$, где $cost(l, r)$ – сколько нужно сделать прибавлений для того, чтобы отрезок (l, r) стал хорошим.

Третью подзадачу можно пройти, если считать $cost(l, r)$ за линию.

Для решения четвертой подзадачи будем перебирать j от большего к меньшему, добавляя элемент слева к подотрезку хороших пар элементов и пересчитывая $cost$ так, как описано в первой подзадаче.

Итоговая сложность – $O(n^3)$, что позволяет пройти все подзадачи.

Задача E. Цветная строка

В первой подзадаче все элементы различны, а все недиагональные значения в cost равны. Тогда задача заключается в определении минимального количества операций, необходимого для приведения последовательности символов в отсортированный по неубыванию вид(за одну операцию один элемент меняется на любой другой). Это можно сформулировать по-другому: вычислить максимальное количество элементов, которые останутся на своих местах. Это должна быть неубывающая подпоследовательность. Так как мы максимизируем количество остающихся на своих местах элементов, то получаем, что нам нужно найти наибольшую неубывающую подпоследовательность. Кстати, неявное ограничение, которое выполняется в этой подзадаче: $n \leqslant m \leqslant 1000$, поэтому её можно искать за квадратичное время.

Во второй подзадаче элементы не обязательно различны. Давайте найдем $left(x), right(x)$ – самое левое и правое соответственно вхождения цвета x в последовательность.

Утверждение: для каждого x все цвета в итоговом массиве в подотрезке $[left(x), right(x)]$ будут равны.

В самом деле, равны края подотрезка, и они будут равны после того, как мы применим к x замену. Так как последовательность должна быть неубывающей, то получим, что все элементы внутри должны быть равны между собой.

У нас остается проблема, что такие подотрезки для разных x могут иметь разное взаимное расположение – могут не иметь общих точек, могут пересекаться и вкладываться. Если два подотрезка имеют общие точки, то они должны образовывать один подотрезок из равных элементов. Нам нужно получить компоненты из подряд идущих элементов, в которых все элементы должны быть равны(это можно реализовать проходом слева направо, поддерживая самую правую позицию, до которой все элементы должны быть равны). Также, для каждой такой компоненты можно посчитать её стоимость перевода в каждый другой цвет. Заметим, что получилось не более m компонент. Далее применим динамику $dp(prefix, last)$ – сколько элементов рассмотрели, какой последний цвет. Для перехода нужно перебрать предыдущий цвет, для оптимизации – брать минимум на префиксе. Это решение работает за $O(n + m^2)$.

В третьей подзадаче нужно заметить, что $n \leqslant m \leqslant 1000$, а далее применить динамику $dp(prefix, last)$(рассмотренный префикс, последний элемент) с частичными минимумами.

Для решения четвертой подзадачи пройдет некоторое менее оптимальное решение, чем решение пятой, поэтому рассмотрим сразу решение пятой подзадачи.

Для решения последней подзадачи объединим сжатие в компоненты из второй подзадачи и динамику из третьей подзадачи. Таким образом, получим решение за $O(n+m^2)$, которое проходит все подзадачи.

Задача F. Все дороги ведут в Рим!

Для решения первой подзадачи достаточно для каждого запроса проходится за линию по всему дереву. Сначала поставить в каждую стартовую вершину каравана его стоимость, и потом найти с помощью поиска в глубину суммы во всех поддеревьях выбрать вершину с наибольшей выгодой.

Для второй подзадачи заранее считаем все запросы и сделаем один проход для подсчета всех ответов. Для этого в каждой вершине будем хранить сет пар $(query, profit)$, где $query$ это номер запроса, а $profit$ — сумма в поддереве этой вершины всех стоимостей караванов из данного запроса.

Запустим поиск в глубину из корня, и в каждой веришне после выхода из детей будем приливать сет меньших в сету наибольшего по следующим правилам :
если в месте, куда мы переливаем пару, еще не было пар с таким запросом, то просто перекинем нашу пару, иначе мы удалим ту пару и добавим пару с таким же запросом, но с суммой $profit$'ов обеих пар и прорелаксируем ответ на тот запрос этим значением, умноженным на высоту той вершины. Таким образом после выхода из корня мы будем знать ответы на все запросы.

Для последующих подзадач нужно использовать факт, что если перенумеровать вершинки в порядке эйлерова обхода, то поддерево каждой вершины — это подотрезок номеров вершин. Тогда если у нас есть караваны, стартующие из $v_1 < v_2 < ... < v_k$, то самая глубокая вершина, покрывающая их все — это $LCA(v_1, v_k) = argmin_{v_1 \leqslant x \leqslant v_k} h(x)$, где $h(x)$ — высота вершины x. Тогда, выбирая вершину, мы по сути выбираем подотрезок номеров караванов. Таким образом, если номера стартовых вершин караванов это $v_1 < v_2 < ... < v_k$ для ответа на запрос нам надо выбрать подотрезок $[l, r]$ с максимальным значением $(v_l + v_{l+1} + ... + v_r) * LCA(v_l, v_r)$. Для решения третьей подзадачи можно разбить запросы на имеющие размер больше либо равно $\sqrt{K_{sum}}$ и меньше. Первых запросов всего не более $\sqrt{K_{sum}}$, и для каждого из них можно пройтись по дереву как в первой подзадаче. А для вторых запросов искать такой подотрезок перебором за $O(k_i^2)$, если использовать подсчет LCA с помощью разреженных таблиц. Итоговое время работы в таком случае будет $O(n\sqrt{K_{sum}} + K_{sum}\sqrt{K_{sum}})$(можно доказать, что $\sum k_i^2 \leqslant K_{sum}\sqrt{K_{sum}}$.

Для решения последней подзадачи нужно искать такой подотрезок быстрее. Составим массив g следующего вида: $v_1, LCA(v_1, v_2), v_2, LCA(v_2, v_3), v_3, ..., LCA(v_{k-1}, v_k), v_k$, а вершинам $LCA(v_1, v_2), LCA(v_2, v_3), ..., LCA(v_{k-1}, v_k)$ поставим стоимость равную 0. Наша задача сводится к поиску наибольшего значения произведения суммы стоимостей элементов на наименьшую высоту на подотрезке среди подотрезков массива g. Для поиска этой величины будем перебирать элемент и построим самый лучший отрезок, в котором он самый низкий. Для этого заранее предпосчитаем ближайший с меньшей высотой слева и справа с помощью прохода со стеком и тогда такой отрезок — это отрезок между этими самыми ближайшими слева и справа.

Задача G. Игра на числах

В первой подзадаче можно предпросчитать массив $win(x)$ – кто выигрывает, если на доске записано единственное число x.
Предпосчитать его можно с помощью динамики:

$$win(x) = \begin{cases} !win[x-1] & x \ mod \ 2 = 0 \\ !win[x-1] \mid !win[(x-1)/2] & x \ mod \ 2 \neq 0 \end{cases}$$

Во второй подзадаче записано уже несколько чисел на доске. Нам потребуются значения Гранди. Их также можно предпросчитать для всех значений до 2^{18}, а затем обрабатывать запросы наивным алгоритмом. Для определения того, кто выиграет, нужно будет посчитать ксор имеющихся значений Гранди.

В третьей подзадаче дано одно большое число. Заметим, что по формуле выигрышных позиций из первой подзадачи получается, что нечетные числа всегда выигрышны, а четные проигрышны, т.к. 0 проигрышен, то 1 – выигрышно, тогда 2 - проигрышно и так далее. Поэтому для решения третьей подзадачи надо лишь поддерживать последнюю цифру числа в двоичной записи.

Для последующих подзадач потребуется понять, как будут устроены числа Гранди для нашей игры. Рассмотрим следующее утверждение:

Пусть число x в двоичной системе счисления заканчивается на k единиц, $G(x)$ – значения Шпрага-Гранди для числа x. Тогда $G(x) = 0$, если x – чётно, то есть заканчивается на 0, иначе $G(x) = 1$, если k – нечетно, и $G(x) = 2$, если k – четно. Докажем это утверждение. Очевидно, $G(0) = 0$. Покажем, что $G(2 \cdot s) = 0$, а $G(2 \cdot s + 1) > 0$. Это следует из того, что у любой проигрышной позиции число Гранди 0, а у любой выигрышной оно больше 0. Рассмотрим теперь $x = 2 \cdot s + 1$ – нечетное число. Мы можем вычесть один и перейти к четному, а можем убрать последнюю единицу из его двоичной записи. Очевидно, что $G(x) \leqslant 2$, так как из каждого числа есть не более двух переходов. Также, из $2 \cdot s + 1$ есть переход в s, то есть $G(2 \cdot s + 1) \neq G(s)$. Получим, если к любому четному числу допишем справа единицу, то G нового числа будет равно 1. Теперь, если к любому

числу с одной единицей в конце допишем справа 1, получим G нового числа(с двумя единицами в конце) равно 2. Тогда G всех чисел с тремя единицами в конце будет равно снова 1, и продолжая такое рассуждение, мы докажем наше утверждение.

Таким образом, нам необходимо поддерживать последнюю цифру числа, а также четность количества последней цифры. Для четвертой подзадачи можно делать это линейным пересчетом после каждого запроса. Для последующих подзадач необходимо делать это быстрее. Удобно реализовать это можно с помощью структуры данных set. Будем для каждого числа поддерживать свой set таких позиций i, в которых $a_i \neq a_{i+1}$. Тогда при запросе изменения (l, r) нам нужно обновить set в позициях $l-1$ и r следующим образом — если позиции не было в set, мы добавим её, иначе удалим. Для того, чтобы найти количество последних цифр, нам нужно будет найти разность длины числа и максимального элемента set(или нуля, если сет пуст). Также не забываем поддерживать последнюю цифру числа и при необходимости менять её.

В пятой подзадаче у нас мало чисел, но они могут быть большими. Для каждого числа будем поддерживать set. Также будем поддерживать $flip(i)$ – нужно ли целиком флипнуть i-е число. Если запрос целиком попадает в одно число, либо задевает некоторое число неполностью, то мы обновляем такие числа с помощью set-ов в них. По числам, лежащим в запросе полностью, мы проходимся(их мало) и меняем $flip$.

В шестой подзадаче, наоборот, длины небольшие, но чисел может быть много. Аналогично пятой подзадаче, меняем вручную числа, которые попали в запрос частично, но можно не использовать set. Будем использовать дерево отрезков для быстрого изменения на отрезке чисел, которые целиком попадают в блок. В вершине дерева отрезков будем хранить $cntOdd(digit), cntEven(digit)$ – количество чисел на отрезке, которые заканчиваются на цифру digit, которая встречается нечетное или четное количество раз на конце этих чисел, соответственно. С помощью модификаций на отрезке можно пересчитывать эту информацию, а зная её в корне дерева отрезков, можно отвечать на запрос о том, кто выиграет.

В седьмой подзадаче объединим решение из пятой и шестой подзадачи.