Задача A. Космоплитка

Общие наблюдения

1. Удобно считать, что пол увеличивается в x раз, а не плитки и их куски уменьшаются в x раз;
2. Если $lcm(a, b) \leqslant a \cdot k$, то ответ 0.

Так как $lcm(a, b) = \dfrac{a \cdot b}{gcd(a, b)}$, неравенство $lcm(a, b) \leqslant a \cdot k$ равносильно неравенству $\dfrac{b}{gcd(a, b)} \leqslant k$.

В неравенствах выше $lcm(a, b)$ — это наименьшее общее кратное чисел a и b, а $gcd(a, b)$ — это наибольший общий делитель чисел a и b.

Подгруппы 1 и 2

В подгруппе 1 число b — нечётное. Следовательно, куски после разреза будут разных размеров. Следовательно, использовать оба куска, полученных после разреза одной плитки, не получится.

В подгруппах 1 и 2 мы можем позволить себе смоделировать укладку плитки для каждого x и каждого ряда плитки. Укладка плитки происходит до тех пор, пока суммарная длина плиток в ряду не достигнет длины пола. Асимптотика такого решения $\mathcal{O}(k^3 \cdot a)$. Псевдокод:

```
for x = 1 .. k
    sum = 0
    rem = 0
    for i = 1 .. x
        cur = 0
        while cur < a * x
            cur += b

        if (cur - a * x) * 2 != b
            sum += cur - a * x
        else if rem > 0
            sum -= cur - a * x
            rem -= 1
        else
            sum += cur - a * x
            rem += 1

    ans = min(ans, sum)
```

Подгруппа 3

Заметим, что ряды плитки абсолютно одинаковы. И поэтому достаточно смоделировать процесс укладки плитки в первом ряду и домножить полученный результат на количество рядов. Асимптотика такого решения $\mathcal{O}(k^2 \cdot a)$. Псевдокод:

```
for x = 1 .. k
    cur = 0
    while cur < a * x
        cur += b

    if (cur - a * x) * 2 != b
        sum = (cur - a * x) * x
    else
        sum = (cur - a * x) * (x % 2)

    ans = min(ans, sum)
```

Подгруппа 4

Заметим, что при фиксированном x последняя плитка в первом ряду имеет либо размеры a на a, если $a \cdot x \mod b = 0$, где $\mod$ — это операция взятия остатка от деления; либо a на $a \cdot x \mod b$, если $a \cdot x \mod b \neq 0$. В первом случае неиспользованных кусков плитки после укладки ряда не останется, а во втором случае останется кусок размером a на $b - a \cdot x \mod b$, который может быть будет использован в другом ряду.

Асимптотика такого решения $\mathcal{O}(k)$. Псевдокод:

```
for x = 1 .. k
    cur = a * x % b
    if cur != 0
        cur = b - cur

    if cur * 2 != b
        sum = cur * x
    else
        sum = cur * (x % 2)

    ans = min(ans, sum)
```

Подгруппа 5 и полное решение

Воспользуемся тем наблюдением, что ответ равен 0 при $\frac{b}{gcd(a,b)} \leqslant k$, и получим решение с асимптотикой $\mathcal{O}(b)$. Псевдокод:

```
if b / gcd(a, b) <= k
    ans = 0
else
    for x = 1 .. k
        cur = a * x % b
        if cur != 0
            cur = b - cur

        if cur * 2 != b
            sum = cur * x
        else
            sum = cur * (x % 2)

        ans = min(ans, sum)
```

Решения, в которых случается переполнение 64-битного типа данных, например, при расчёте $a \cdot k$, проходят тесты подгруппы 5.

Задача B. Степенатор-2025

Заметим, что результатом работы степенатора всегда является число вида $a_i^{w_i}$ для некоторого $1 \leqslant i \leqslant n$ и некоторого натурального w_i. Поэтому мы хотим для каждого a_i понять, для какого максимального w_i возможно такое, что результат работы степенатора равен $a_i^{w_i}$, и проверять на делимость на x только эти n чисел вида $a_i^{w_i}$.

При этом заметим, что так как x в запросах до 10^6, каждый простой множитель входит в факторизацию x в не более чем 20-й степени. Поэтому на самом деле если $w_i \geqslant 20$ нас не интересует конкретное значение, так как уже 20-й степени достаточно чтобы покрыть всевозможные простые множители из факторизации x. Поэтому нам нужно знать точное значение w_i только если оно меньше 20.

Также если $a_i = 1$, значение w_i нас не интересует, так как единица в любой степени всё ещё единица. Поэтому мы считаем w_i только для интересных $a_i > 1$.

Полное решение разбивается на несколько случаев:

- $n = 2, n = 3$, в данном случае необходимо для каждого запроса написать явный перебор всех результатов работы степенатора посчитанных по модулю x. Всего вариантов работы степенатора будет $n! \cdot (n-1)! \cdot \ldots \cdot 1!$, что для $n = 3$ равняется всего лишь 12, и легко помещается в лимит времени. Для $n = 3$ важно не брать оба числа по модулю x после первой итерации степенатора, так как одно из чисел далее будет показателем степени. При этом чтобы не вычислять огромный показатель полностью, необходимо написать функцию `bounded_pow(a, b)` возвращающую $min(20, a^b)$, так как опять же числа большие 20 передавать в показатель не имеет смысла в рамках этой задачи.

- Если все числа в массиве равны 1, ответ `Yes` только для $x = 1$.

- Если все числа кроме одного a_i равны 1, то результатом работы степенатора будет либо 1, либо a_i.

- $n \geqslant 4$, в массиве есть хотя бы две **не** единицы...

И оказывается, если $n \geqslant 4$ и при этом в массиве **хотя бы** два числа больше 1, в качестве w_i всегда и для всех i можно получить число большее 20. Продемонстрируем это на примере $n = 4$ и массива $[x, y, 1, 1]$. Последовательность операций: $[x, y, 1, 1] \to [y^x, x, 1] \to [x^{(y^x)}, y^x] \to [y^{(x \cdot x^{(y^x)})}]$. Подставляя $x = y = 2$ в показателе степени мы получим число 32. Ясно, что при $x, y \geqslant 2$ показатель степени может только увеличиться, аналогично и при увеличении количества чисел как равных 1, так и превышающих 1.

Таким образом в этом случае задача сводится к «есть ли в массиве a такое число a_i, что $B \in A$, где A — множество простых делителей числа a_i, а B — множество простых делителей числа x». Написав решето эратосфена можно для всех чисел от 1 до 10^6 вычислить их набор простых делителей. После чего каждое x можно преобразовать в x' = «произведение уникальных простых делителей x», и уже для этого числа требуется просто проверить есть ли в массиве a делящееся на него. Что можно сделать за $O(\frac{A}{x'})$ простым перебором всех возможных чисел $\leqslant A$ делящихся на x'. Запоминая и переиспользуя ответы для одинаковых x', асимптотика такого решения получится $O(\frac{A}{1} + \frac{A}{2} + \ldots + \frac{A}{A}) = O(A \log A)$, где $A = 10^6$, что спокойно помещается в лимит.

Задача C. Зайчик 3.1

Рабочее название «Фрактальный зайчик»

Общая идея решения

Будем насчитывать $dp[i]$ — число способов попасть на ступеньку с номером i.

Пересчёт такого $dp[i]$ можно делать, рассматривая последний шаг перед попаданием зайчика на ступеньку i. Тогда $dp[i] = \sum_{step \in STEP} dp[i - step]$, где $STEP$ — множество возможных последних ходов зайчика. Иными словами, количество способов попасть на ступеньку i равно сумме количеств способов попасть на ступеньки за шаг до этого.

Определение допустимых прыжков

Шаг $step$ является красивым, если в числе $step - 1$ нет цифр 1 в троичной записи. Научимся проверять шаг $step$ на красоту. Для этого напишем функцию $check$, в которой проверим запись числа $step - 1$ (через сколько ступеней зайчик перепрыгнул) в троичной системе счисления.

```
bool check(int step) {
    step--;

    while (step) {
        if (step % 3 == 1) {
            return false;
        }
        step /= 3;
    }
    return true;
}
```

Функция $check$ работает за $\mathcal{O}(\log n)$

Подгруппа 1. $n \leqslant 2000$

Будем пересчитывать $dp[i]$ в цикле по i. Пройдём по всем возможным $step$. Пересчитываемся только из допустимых. При пересчёте обновляем значение по модулю: $dp[i] = dp[i] \ \% \ MOD$.

```
#define ll long long
const ll MOD = 998244353;
```

```
for (int i = 1; i <= n; i++) {
    for (int step = 1; step <= i; step++) {
        if (check(step)) {
            dp[i] += dp[i - step];
            dp[i] %= MOD;
        }
    }
}
```

Получаем решение за $\mathcal{O}(n^2 \log n)$. Такое решение проходит первую подгруппу.

Подгруппа 2. $n \leqslant 2 \cdot 10^4$

Заметим, что каждый раз проверять, является ли шаг красивым — долго. Вместо этого можем заранее сохранить для каждого шага, является ли он красивым.

```
vector <bool> is_nice(n + 1, false);

for (int step = 1; step <= n; step++) {
    is_nice[step] = check(step);
}

for (int i = 1; i <= n; i++) {
    for (int step = 1; step <= i; step++) {
        if (is_nice[step]) {
            dp[i] += dp[i - step];
            dp[i] %= MOD;
        }
    }
}
```

Такое решение работает за $\mathcal{O}(n \log n + n^2)$, проходит вторую подгруппу тестов и получает 45 баллов.

Подгруппа 3. $n \leqslant 2 \cdot 10^5$

Заметим, что из всех шагов от 1 до n довольно мало красивых. А именно — их $2^{\log_3 n} = n^{log_2 3} \approx n^{0.63}$, что при $n = 2 \cdot 10^5$ примерно 2000.

Воспользовавшись этим фактом изменим решение. Сначала найдём список всех допустимых шагов, после чего будем пересчитывать dp только через них.

```
vector <int> nice_steps;

for (int step = 1; step <= n; step++) {
    if (check(step)) {
        nice_steps.push_back(step);
    }
}

for (int i = 1; i <= n; i++) {
    for (int step: nice_steps) {
        if (i - step < 0) {
            break;
        }
        dp[i] += dp[i - step];
        dp[i] %= MOD;
    }
}
```

Такое решение уже работает быстрее, но может не проходить подгруппу 3 — в таком случае можно, например, заметить, что мы часто выполняем тяжёлую операцию деления по модулю, в то время как выполнять её достаточно однажды после суммирования.

```
for (int i = 1; i <= n; i++) {
    for (int step: nice_steps) {
        if (i - step < 0) {
            break;
        }
        dp[i] += dp[i - step];
    }
    dp[i] %= MOD;
}
```

Такой код работает за $\mathcal{O}(n \cdot n^{log_3 2})$, проходит подгруппу 3 и получает 60 баллов.

Подгруппа 4. $n \leqslant 10^6$

Рассмотрим множество ступенек, из которых мы могли попасть на i-ю ступеньку.

```
last_j = "#*#"
digit0 = "012"

last_j = "#*******#"
digit0 = "012012012"
digit1 = "000111222"

last_j = "#*#####*#**********#*****#*#"
digit0 = "012012012012012012012012012"
digit1 = "000111222000111222000111222"
digit2 = "000000000111111111222222222"
```

Заметим, что такое множество точек при $n = 3^k$ не содержит разрешённых шагов в средней трети ступенек. Хотим посчитать для каждого i сумму в такой структуре, завершающейся в точке $i - 1$.

Видно, что сумма по всем # на отрезке длины 3^k, заканчивающемся в точке i, пересчитывается таким образом: $sum_k[i] = sum_{k-1}[i - 2 \cdot 3^{k-1}] + sum_{k-1}[i]$.

Тогда мы можем в каждой точке хранить $sum_k[i]$ для всех k таких, что $0 \leqslant k \leqslant \log_3 n$. Будем пересчитывать $sum_k[i]$ в порядке возрастания k. Тогда следующая сумма будет пересчитываться через предыдущую.

Такое решение работает за $\mathcal{O}(n \log n)$, проходит подгруппу 4 и получает 85 баллов.

Подгруппа 5. $n \leqslant 2 \cdot 10^6$

Заметим, что решение в прошлой подгруппе получает Memory Limit Exceeded. Значит, необходимо сократить потребление памяти.

Будем хранить наши $sum_k[i]$ как $int32\_t$ вместо $int64\_t$ ($long\ long$). Также стоит учесть, что $vector < vector < int >>$ может занять больше памяти, чем мы рассчитываем. Можно использовать двумерный массив, $vector < array < int, MAXK >>$ или хранить всё в одномерном векторе, получая значения таким образом: $sum_k[i] = dp\_sum[MAXK \cdot i + k]$.

Это решение получает полный балл.

Задача D. Ремонт метро

Будем решать задачу сразу для случая $\alpha = 1.5$

Пусть $Rsum$ — это сумма r_i по выполненным работам. $Lsum$ — сумма l_i по выполненным работам.

Тогда, если оба условия выполняются, можем записать следующую цепочку неравенств:

$m \geqslant Rsum \geqslant \alpha \cdot Lsum \to \frac{m}{\alpha} \geqslant Lsum \to \frac{2m}{3} \geqslant Lsum$

А значит $m - Lsum \geqslant m - \frac{2m}{3} = \frac{m}{3}$.

То есть, чтобы выполнить второе условие необходимо, чтобы все работы с $r_j \leqslant \frac{m}{3}$ были включены в план, так как $m - Lsum \geqslant \frac{m}{3}$ гарантированно.

При этом работ с $r_j > \frac{m}{3}$ получится выполнить не больше двух, так как $Rsum$ не может превышать m.

После данного замечания, несложно посчитать все планы работ которые включают 0 или 1 работу с $r_j > \frac{m}{3}$, просто перебрав все такие планы.

А подсчёт планов с 2 работами с $r_j > \frac{m}{3}$, после некоторых простых преобразований, сводится к подобной задаче:

«Даны два массива $x_1 \dots x_n$ и $y_1 \dots y_n$, а также две константы X и Y. Посчитайте количество $1 \leqslant i < j \leqslant n$ таких, что $x_i + x_j \geqslant X; y_i + y_j \leqslant Y$ одновременно»

После сортировки массива x по неубыванию и перебора i, такая задача сводится к запросам вида «сколько чисел на префиксе массива $\leqslant k$», на которые можно отвечать например при помощи структуры данных MergeSortTree, с суммарной асимптотикой $O(n \log^2 n)$. Но также можно решать с сканлайном и деревом Фенвика, с асимптотикой $O(n \log n)$. Оба решения должны были заходить на полный балл, но решение с MergeSortTree возможно надо было попихать, в зависимости от эффективности вашей реализации.

Задача E. MAX MEX MEX

Подгруппа 2. $c_i = 0$

В этой подгруппе мы не можем менять наборы чисел в корзинках. Значит, можно насчитать MEX_i для всех корзин, после чего насчитать MEX полученного набора.

Такое решение проходит вторую подгруппу и получает 10 баллов.

Получение множества допустимых MEX_i

Давайте для каждой корзинки научимся получать множество допустимых значений MEX_i, которое можно получить, выполнив несколько (возможно, ноль) действий в i-й корзинке.

Чтобы получить $MEX_i = m$ для некоторого m нам нужно, чтобы все числа $[0, 1, 2, \ldots, m-1]$ встретились в корзинке. А также чтобы само число m в корзинке не встречалось.

Следовательно, если некоторое число $0 \leqslant x \leqslant m-1$ в корзинке не встречается — нужно получить его из меньшего числа несколькими действиями увеличения.

Пройдём по всем числам от 0 до n_i и проверим, можно ли получить их в качестве MEX_i. Будем поддерживать множество «свободных» меньших чисел и при необходимости ставить наибольшее из них на пустующее место (можно доказать, что это оптимально). Также будем поддерживать сколько действий у нас осталось и для каждого значения проверять, возможно ли передвинуть все числа, равные m, хотя бы на один вправо (чтобы MEX_i был равен m необходим, чтобы число m не встречалось в наборе).

Подгруппа 1. $n_i \leqslant 15$, $k \leqslant 1000$

Определим множество допустимых MEX_i. Ответ зависит от того, какие именно значения MEX_i мы выберем для каждой корзинки. По большому счёту нас интересует то, какие множества значений мы можем получить перед последним действием подсчёта $MEX(MEX_1, MEX_2, \ldots, MEX_k)$.

Тогда давайте просто переберём всевозможные подмножества значений. В этой подгруппе $n_i \leqslant 15$, следовательно $MEX_i \leqslant 15$. Тогда всех возможных подмножеств всего 2^{16}.

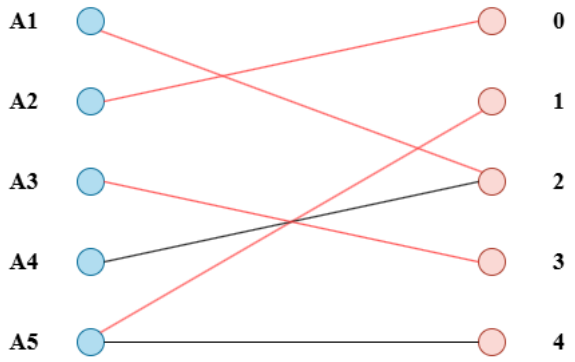
Давайте получим все достижимые подмножества. Сделать это можно, поддерживая $dp[i][mask]$ равным 1, если мы можем получить набор $mask$, и 0 иначе. Пересчёт допустимых масок через i-е множество MEX_i делаем так:

```
for (int i = 0; i < k; i++) {
    for (int mask = 0; mask < (1 << 16); mask++) {
        for (int m: possible[i]) {
            dp[i + 1][mask | (1 << m)] |= dp[i][mask];
        }
    }
}
```

Подгруппы 3 и 4

Пусть для каждой корзинки получили множество допустимых значений MEX_i в ней. Тогда получаем двудольный граф между корзинками и MEX-ми. В левой доле — множество корзинок, в правой доле — множество значений MEX_i.

Хотим, чтобы справа первые m элементов были «покрыты» корзинками. По сути ищем паросочетание, покрывающее первые m вершин в правой доле.



Синие корзинки и красные MEX'ы

Искать такое паросочетание можно следующими способами:

- Алгоритм Куна при запуске последовательно от вершин правой доли.
- Бинпоиск по ответу вместе с поиском максимального потока или алгоритмом Куна также решает задачу и проходит все тесты.

Почему такое решение быстро работает:

Во-первых, из корзинки размера n_i можно получить максимум $MEX_i = n_i$, но не больше. Следовательно суммарное число рёбер в графе не превышает s, равное 10^6.

Во-вторых ответ на задачу не может быть слишком велик.

Пусть ответ на задачу равен M. Тогда в корзинках были достигнуты MEX, равные $0, 1, 2, \ldots, M-1$. Чтобы получить $MEX_i = j$ необходимо, чтобы в i-й корзинке было хотя бы j чисел. Тогда чтобы получить ответ, равный M, необходимо, чтобы в корзинках суммарно было хотя бы $\sum_{m=0}^{M-1} m = \frac{(M-1)\cdot M}{2}$. То есть $s \geqslant \frac{(M-1)\cdot M}{2}$, следовательно $M \leqslant \sqrt{2\cdot s}+1$. Следовательно ответ не превышает 1000, из чего следует скорость работы алгоритма.

Подгруппа 5. b_i **— любые,** $k \leqslant 1000, s \leqslant 3 \cdot 10^4$

В последней подгруппе добавляется усложнение — теперь каждое число a_j встречается с кратностью b_j. Это меняет подход к поиску допустимых значений MEX_i — теперь при проходе слева направо будем поддерживать множество «свободных» чисел вместе с их кратностью.

Из-за b_i ограничение на MEX_i уже не работает как в подгруппе 4.

Заметим, что ответ не превышает k, следовательно нам достаточно рассматривать значения $MEX_i \leqslant k$. Тогда в графе будет не больше чем 10^6 рёбер, в то время как ответ не превышает k. Значит алгоритм Куна или алгоритм поиска потока с бинпоиском решит задачу и в этой подгруппе.