Задача A. Дружелюбные ладьи

Автор задачи: Михаил Анопренко, разработчик: Григорий Шовкопляс

Заметим, что наибольшее число ладей, которые возможно расставить на поле $n \times m$ равно $\min(n, m)$. Соответственно, если $\min(n, m) < k$, то ответа не существует.

Иначе подойдет расстановка, в которой все ладьи стоят на одной диагонали. Например, на главной. Остается лишь аккуратно вывести ответ.

Задача B. Угадай массив

Автор задачи: Даниил Орешников, разработчики: Даниил Орешников и Григорий Хлытин

Сделаем самым первым запросом запрос «1 n». В ответ нам придет сумма на всем массиве $s_{1,n} = S$ и некоторый запрещенный отрезок.

Всеми следующими запросами попробуем выяснить суммы элементов на каждом префиксе массива. Для этого будем поддерживать следующий инвариант: в каждый момент времени для каждого еще неизвестного префикса $[1, i]$ можно сделать хотя бы один из запросов $[1, i]$ и $[i + 1, n]$.

Для этого при выдаче интерактором нового запрещенного отрезка $[l, r]$:

- При $l = 1$ сразу же сделаем запрос «$r + 1$, n». Зная сумму на таком суффиксе $s_{r+1,n}$ и сумму на всем массиве S, сумму на запрещенном $[1, r]$ можно посчитать как $S - s_{r+1,n}$;
- При $r = n$ сразу же сделаем запрос «1, $l - 1$», узнав сумму на префиксе $[1, l - 1]$;
- При любом другом запрещенном отрезке просто сделаем запрос $[1, i]$ для произвольного еще неиспользованного i. Для этого достаточно просто поддерживать очередь всех i от 1 до $n - 1$ и множество таких i, для которых эта сумма нам уже известна.

Таким образом, в каждый момент времени для всех i, кроме, возможно, одного, для которых сумма $s_{1,i}$ еще неизвестна, мы можем сделать и запрос $[1, i]$, и запрос $[i+1, n]$; а для того, для которого не можем сделать один из запросов, ганартированно можем сделать запрос второй, и сразу же его делаем. Описанный выше инвариант при этом сохраняется.

После того, как за n действий нам станут известны суммы на всех префиксах массива, i-й элемент можно будет посчитать как $s_{1,i} - s_{1,i-1}$. Найдем так все элементы массива, и выведем после «!».

Задача C. Перемещение клеток

Автор задачи: Даниил Орешников, разработчик: Григорий Хлытин

Для решения этой задачи воспользуемся методом динамического программирования.

Будем вычислять двумерный массив $dp_{i,j}$ — минимальное число действий, которое необходимо совершить с первыми i столбцами, чтобы сделать *красивой* часть картины на этих столбцах, и чтобы в i-м столбце последовательность черных клеток начиналась в j-й строке.

Пересчет будет по формуле:

$$dp_{i,j} = |s_i - j| + \min_{k=j+s_{i-1}-t_{i-1}}^{j+t_i-s_i} dp_{i-1,k}$$

где s_i и t_i — номера начальной и конечной позиции непрерывного отрезка черных клеток в i-м столбце клетчатой картины.

При наивной реализации этой формулы можно для каждого j перебирать все подходящие k и вычислять минимум среди $dp_{i-1,k}$ за $\mathcal{O}(n)$, тогда мы получим асимптотику $\mathcal{O}(m \cdot n^2)$.

Однако можно заметить, что при переходе от j к $j + 1$, среди интересующего нас множества элементов $dp_{i-1,k}$, не более двух элементов могут измениться, так как отрезок соответствующих k сдвинулся на один. Таким образом, можно поддерживать минимум в этом множестве, например при помощи `std::set` в `C++`, тогда асимптотика такого решения будет $\mathcal{O}(m \cdot n \cdot \log n)$, или при помощи структуры данных `очередь с минимумом`, тогда асимптотика будет $\mathcal{O}(m \cdot n)$.

Задача D. Отряд клонов

Автор задачи: Даниил Орешников, разработчик: Савелий Григорьев

Пусть d_i — максимальное количество клонов, которые могут добраться до помещения i. Тогда определим вес ребра $u \to v$ следующим образом: если $d_u > a_v$, то ребро $u \to v$ имеет вес 0, иначе 1. Заметим, что исходное неориентированное ребро $u - v$ распадается на два ориентированных ребра $u \to v$ и $v \to u$.

Длина кратчайшего пути из помещения 1 в помещение n в полученном графе будет соответствовать минимальному количеству раз, когда армия клонов теряла половину состава, то есть при движении по нему выживет наибольшее количество клонов.

Воспользуемся алгоритмом `0-1 bfs`. Однако определить вес ребра мы можем только в том случае, если до исходящего конца посчитан кратчайший путь. Так как `0-1 bfs` рассматривает только такие рёбра, наш алгоритм будет работать корректно.

Итоговая асимптотика — $\mathcal{O}(n + m)$.

Задача E. Хокку

Автор задачи: Артем Васильев, разработчик: Арсений Кириллов

Попробуем для каждого слова проверить, начинается ли в нём хокку. Для этого переберём слова, начиная с него, и найдём первое место, в котором количество слогов в перебранных словах хотя бы 5. После этого найдём первое место, в котором количество слогов хотя бы 7, а после этого первое место, в котором количество слогов хотя бы 5. Если в каждом из этих случаев получилось найти ровно столько слогов, сколько необходимо, то есть потенциальное хокку, которое начинается в стартовом слове. Так как в каждом слове есть хотя бы один слог, то для каждой проверки придётся перебрать не более 17 слов.

Задача F. Забег

Автор задачи: Даниил Орешников, разработчик: Михаил Анопренко

Из условия задачи следует, что в момент времени t участник с номером i будет находиться в точке $(s_i + t \cdot v_i, i)$.

Рассмотрим момент времени, когда на одной прямой будет находиться максимальное возможное количество участников. Зафиксируем какого-нибудь из участников, находящихся на этой прямой в этот момент времени. Переберем его номер среди всех возможных. Пусть это участник с номером i. Таким образом, теперь мы для каждого момента времени рассматриваем только те прямые, которые проходят через точку, в которой в этот момент находится участник с номером i. Для удобства при рассмотрении участника с номером i будем считать, что центр координат всегда находится в точке, где расположен этот участник (и двигается вместе с ним). Легко заметить, что теперь в момент времени t участник с номером j будет находиться в точке $(s_j - s_i + t \cdot (v_j - v_i), j - i)$. Введем новые обозначения: пусть $j' = j - i$, $s'_j = s_j - s_i$, $v'_j = v_j - v_i$. Тогда в момент времени t участник с номером j будет находиться в точке $(s'_j + t \cdot v'_j, j')$. Теперь нас будут интересовать только прямые, проходящие через точку $(0, 0)$.

Пусть в момент времени t участники с номерами j и k находятся на одной прямой, проходящей через точку $(0, 0)$. Это равносильно тому, что векторное произведение векторов из точки $(0, 0)$ в позиции этих двух участников в данный момент времени равно нулю. Запишем это условие, воспользовавшись формулой для векторного произведения: $(s'_j + tv'_j)k' - (s'_k + tv'_k)j' = 0$. Преобразуем: $s'_j k' - s'_k j' = t(v'_k j' - v'_j k')$. Заметим, что если $v'_k j' - v'_j k' \neq 0$, то существует единственный момент времени t, когда условие выполняется, и мы можем найти этот момент времени. В случае, если $v'_k j' - v'_j k' = 0$, посмотрим на значение величины $s'_j k' - s'_k j'$. Если она тоже равна 0, то условие будет выполняться в любой момент времени t, в противном же случае условие не будет выполняться никогда. Переберем все возможные пары различных участников $j, k \neq i$, и для каждой пары найдем множество моментов времени, когда условие для этой пары выполняется.

Для каждого значения j посчитаем количество значений k, для которых условие выполняется всегда, а также для каждого значения k, если для пары (j, k) условие выполняется ровно в один

момент времени, сохраним значение этого момента. Посчитаем максимальное количество одинаковых моментов времени среди сохраненных, прибавим количество участников, для которых условие выполняется всегда, и получим кандидатуру для ответа на задачу. Чтобы получить ответ, выберем максимум среди всех кандидатур.

Асимптотика времени работы данного решения составляет $O(n^3 \log n)$, если для поиска одинаковых значений t использовать сортировку. При использовании хеш-таблицы асимптотика может быть уменьшена до $O(n^3)$. Также стоит отметить, что чтобы избежать проблем с точностью вещественных чисел, в данной задаче следует хранить нецелые числа как рациональные дроби.

Задача G. Максимизировать сумму XOR

Автор задачи: Даниил Орешников, разработчик: Арсений Кириллов

Заметим, что когда мы меняем местами элементы A_i и B_i, значения $X(A)$ и $X(B)$ заменяются на $X(A) \oplus A_i \oplus B_i$ и $X(B) \oplus A_i \oplus B_i$. Обозначим за C_i значение $A_i \oplus B_i$.

Если какой-то бит у числа $X(A)$ и $X(B)$ отличается, то при любых операциях он останется различным, и войдёт в итоговую сумму ровно один раз. Если же бит одинаковый, то он всегда будет одинаковым.

Значит, нужно так выбрать значения C_i, чтобы на позициях, где в числах $X(A)$ и $X(B)$ находится бит 0, в xor-сумме выбранных элементов был бит 1, и наоборот. Если все такие условия выполнить не получается, то необходимо удовлетворять эти условия начиная со старших битов, так как каждая степень двойки больше, чем сумма всех меньших.

Для этого выпишем числа C_i как битовые строки, и запустим на этих строчках алгоритм Гаусса. Это позволит понять, какие биты можно изменить, и тогда их необходимо будет изменять начиная со старших битов к младшим. Так как максимальное возможное число во входных данных это 10^{18}, то понадобится всего 60 бит, а значит всего 60 итераций алгоритма Гаусса.

Задача H. Игра в осьминога

Автор задачи: Артем Васильев, разработчик: Хетаг Дзестелов

Видим, что оптимально чередовать операции, меняя на одном ходу одну карточку, а на следующем другую. Используем алгоритм Евклида по нахождение НОД a и b. В качестве ответов на каждый ход k_i достаточно брать такое число, которое приведет состояние игры к следующему шагу алгоритма Евклида. Так как в конце алгоритма одно число станет нулем, мы найдем нужную последовательность k_i. Проблема такого подхода - количество действий. Худший случай, на котором кол-во шагов в алгоритме Евклида максимально — два соседних числа Фибоначчи, для F_i и F_{i+1} требуется i - 1 шагов (последовательность остатков получится: $F_{i-1}, F_{i-2} \dots F_2, 0$), таким образом, исходят из ограничений, при использовании стандартного алгоритма в худшем случае решение будет найдено за 85 шагов.

Воспользуемся техникой Least Absolute Remainder: допустим существование отрицательных остатков от деления, теперь на каждом шагу существует два возможных остатка в алгоритме Евклида, к которым можно перейти на следующем шаге: достаточно брать наименьший по модулю. То есть из $a\%b$ и $a\%b - b$ выбирать меньшее по абсолютному значению. Заметим, что данное действие корректно с точки зрения операции с карточками (берем $k+1$, вместо k, которое взяли бы в случае обычного алгоритма Евклида).

Можно показать, что в этом случае число шагов будет $\log_{1+\sqrt{2}} \max(a, b)$.

Задача I. Зачет в третьей параллели

Автор и разработчик задачи: Даниил Орешников

Посмотрим на задачу с такой стороны: пусть Илья выбрал сдавать все блоки, отвечая на теоретические вопросы, тогда его оценка была бы равна $\sum_{i=1}^{n} x_i$, если бы не было ограничения на количество блоков, сданных практикой.

Заметим, что в некоторых блоках $x_i > y_i$, а в некоторых — $x_i < y_i$. При замене теории на практику в блоках первого типа сумма оценок за блоки уменьшается, а в блоках второго типа — увеличивается. Для блоков, в которых $x_i = y_i$, суммарная оценка не зависит от выбра теории или практики.

Вернемся к ситуации, в которой Илья выбрал сдавать все блоки теорией. Ему надо заменить в нескольких (хотя бы b, но не более $n - a$) блоках теорию на практику, чтобы максимизировать оценку. При замене теории на практику в i-м блоке сумма изменяется на $y_i - x_i$. Обозначим за t количество блоков с $y_i > x_i$, тогда решением будет сдать практикой $\max(b, \min(n - a, t))$ блоков с **максимальным** значением $y_i - x_i$.

Действительно, если $t > n - a$, то максимальная сумма получится при сдаче $n - a$ блоков с максимальным $y_i - x_i$ практикой, а оставшихся a — теорией, так как блоков с теорией должно быть не меньше a. Если же $b \leqslant t \leqslant n - a$, можно сдать теорией только те блоки, в которых больше баллов набирается теорией, а практикой — в которых больше баллов набирается практикой, добившись максимальной возможной $C = \sum_{i=1}^{n} \max(x_i, y_i)$. При $t < b$ же, нет смысла сдавать больше b блоков практикой, так как это уменьшит оценку.

Таким образом, один из оптимальных ответов достигается следующим образом: отсортировать все блоки по величине $y_i - x_i$ и сдавать практикой $\max(b, \min(n-a, t))$ последних, а все оставшиеся — теорией. Время работы — $\mathcal{O}(n \log n)$ на сортировку (хотя можно и за линейное время с поиском порядковой статистики).

Задача J. Вычислительная этнография

Автор задачи: Николай Ведерников, разработчик: Михаил Анопренко

Пусть s — интересное число. Оно является квадратом какого-то целого числа, обозначим это число за r. Так как $s = r^2 \leqslant B$, то $r \leqslant \sqrt{B}$.

Переберем все возможные значения r от 1 до $\lfloor\sqrt{B}\rfloor$. Для каждого значения $s = r^2$ проверим, является ли число s интересным. Для этого нужно убедиться, что перевернутая десятичная запись числа s является полным квадратом. Переберем все цифры числа s, запишем их в развернутом порядке, и убедимся, что из получившегося числа можно извлечь целочисленный квадратный корень. Посчитаем количество значений r среди рассмотренных, для которых перевернутая запись s оказалось полным квадратом, и получим ответ на задачу.

Асимптотика времени работы данного решения составляет $O(\sqrt{B} \log B)$, так как для каждого из $\sqrt{B}$ значений r мы проходим по всем цифрам числа s.

Задача K. Работай или Спи!

Автор задачи: Сергей Копелиович, разработчик: Григорий Шовкопляс

Для решения данной задачи необходимо вывести формулу зависимости эффективности от времени сна. Можно, например, применить формулу уравнения прямой по двум точкам получить, что

$$f(t_{sleep}) = \begin{cases} \frac{X \cdot t_{sleep}}{\frac{T}{6}} & 0 \leqslant t_{sleep} < \frac{T}{6} \\ 100 - \frac{(100-X) \cdot (\frac{T}{3} - t_{sleep})}{\frac{T}{6}} & \frac{T}{6} \leqslant t_{sleep} \leqslant \frac{T}{3} \\ 100 & t_{sleep} > \frac{T}{3} \end{cases}$$

Из этого получаем формулу зависимости объема работы от времени сна:

$$Work(t_{sleep}) = \begin{cases} (T - t_{sleep}) \cdot \frac{X \cdot t_{sleep}}{\frac{T}{6}} & 0 \leqslant t_{sleep} < \frac{T}{6} \\ (T - t_{sleep}) \cdot (100 - \frac{(100-X) \cdot (\frac{T}{3} - t_{sleep})}{\frac{T}{6}}) & \frac{T}{6} \leqslant t_{sleep} \leqslant \frac{T}{3} \\ (T - t_{sleep}) \cdot 100 & t_{sleep} > \frac{T}{3} \end{cases}$$

Далее задача сводится к нахождению максимума этой функции.

1. `Способ 1: для любителей математики`. Можно найти точки, где производные равны нулю (вершины парабол) этой функции на каждом участке и получить, что на первом участке максимум достигается в точке $t_{sleep} = \frac{T}{6}$, на третьем участке в точке $t_{sleep} = \frac{T}{3}$, что ожидаемо, а на втором участке максимум достигается в точке $t_{sleep} = \frac{T \cdot (2(100-X)-25)}{3(100-X)}$. Далее остается подставить значения и взять максимум, либо заметить, что максимум всегда будет на втором участке (возможно на границе).

2. `Способ 2: для любителей алгоритмов`. Для тех участников, кто не очень хочет выводить много формул, есть замечательный алгоритм троичного поиска, который можно запустить на каждом участке функции и взять максимум из максимумов.

3. `Способ 3: для универсальных любителей`. Можно совместить идеи из описанных выше способов. Например, запустить троичный поиск только на втором участке функции, так как догадаться до факта нахождения максимума именно на этом участке функции возможно и без сложных формул.

Задача L. Трансформация перестановки

Автор задачи: Николай Ведерников, разработчик: Сергей Харгелия

Если $n = k$, то из p можно получить q только в случае $p = q$.

Если $n = k + 1$, то из p можно получить q, только если q является циклическим сдвигом p. В таком случае, q может быть получена последовательным применением k-переносов с параметрами $a = 1$, $b = 2$. Очевидно, что потребуется не более, чем $n - 1$ перенос.

Если $n \geqslant k+2$ и k нечётное, то получить q из p всегда возможно. Научимся менять два соседних элемента перестановки местами. Если $k = 1$, то это делается за один k-перенос, далее будем считать, что $k \geqslant 2$. Заметим, что достаточно научиться менять местами два первых элемента перестановки, а также любые два соседних элемента переносить в начало перестановки, потому что тогда можно будет переместить элементы в начало, поменять их, а потом вернуть на прежние места (это можно сделать, поменяв местами параметры a и b у операций переноса в начало и применив их в обратном порядке).

Первые два меняем элемента перестановки можно поменять так: сначала $\left\lfloor \frac{k+1}{2} \right\rfloor$ раз применяем k-перенос с параметрами $a = 1$, $b = 3$, а потом применяем k-перенос с параметрами $a = 2$, $b = 3$. Например, если $n = 6$, $k = 3$ и перестановка равна $1, 2, 3, 4, 5, 6$, она будет меняться следующим образом: $1, 2, 3, 4, 5, 6 \to 4, 5, 1, 2, 3, 6 \to 2, 3, 4, 5, 1, 6 \to 2, 1, 3, 4, 5, 6$.

Перенести соседние элементы в начало перестановки можно так: обозначим индекс первого из них за pos. Если $pos + k - 1 \leqslant n$, то после k-переноса с параметрами $a = pos$, $b = 1$, эти два элемента переместятся в начало. Иначе применим k-перенос с параметрами $a = n - k + 1$, $b = 1$. Тогда pos уменьшится хотя бы на $n - k \geqslant 2$, а рассматриваемые элементы останутся соседними. Заметим, что данный алгоритм требует не более, чем $\left\lfloor \frac{n-1}{2} \right\rfloor + 1$ действий.

Мы научились менять местами соседние элементы, поэтому теперь легко можем получить q из p: найдём первый элемент q в p и перенесём его на первую позицию, потом найдём второй элемент q в p и перенесём его на вторую позицию, и так далее. Итого решение использует не более, чем $(n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}$ обменов соседних элементов, то есть не более, чем $\frac{n(n-1)}{2} \cdot \left(\left\lfloor \frac{k+1}{2} \right\rfloor + 1 + 2 \left(\left\lfloor \frac{n-1}{2} \right\rfloor + 1 \right) \right) \leqslant \frac{n(n-1)}{2} \cdot \left(\left\lfloor \frac{n}{2} \right\rfloor + n + 2 \right) \leqslant n^3$ k-переносов.

Теперь рассмотрим случай, когда $n \geqslant k + 2$ и k чётное. Тогда q возможно получить из p, только если перестановки имеют одинаковую чётность.

Давайте докажем, что если k чётное, то k-перенос не меняет чётность перестановки. Заметим, что для фиксированной пары элементов k-перенос меняет их относительный порядок, только когда ровно один из этих элементов попал в подотрезок, к которому применяется k-перенос. Кроме того, относительный порядок поменялся либо со всеми элементами подотрезка сразу, либо ни с одним. Поэтому чётность количества инверсий поменялась $k \cdot c$ раз для некоторого $0 \leqslant c \leqslant n - k$. Поскольку k чётное, $k \cdot c$ тоже чётное, значит k-перенос не поменял чётность количества инверсий.

Если же чётности p и q совпадают, то получить q из p всегда возможно. Научимся брать три соседних элемента и циклически их сдвигать, оставляя все остальные элементы на своих местах. Если

$k = 2$, то это делается за один k-перенос, далее будем считать, что $k \geqslant 3$. Заметим, что достаточно научиться циклически сдвигать три первых элемента перестановки, а также любые три соседних элемента переносить в начало перестановки. Более того, несложно убедиться, что рассмотренный выше алгоритм переноса двух соседних элементов в начало перестановки подходит и для текущего случая.

Первые три элемента перестановки можно циклически сдвинуть так: сначала применяем k-перенос с параметрами $a = 3$, $b = 1$, а потом два k-переноса с параметрами $a = 2$, $b = 3$. Например, если $n = 7$, $k = 4$ и перестановка равна $1, 2, 3, 4, 5, 6, 7$, она будет меняться следующим образом: $1, 2, 3, 4, 5, 6, 7 \to 3, 4, 5, 6, 1, 2, 7 \to 3, 2, 4, 5, 6, 1, 7 \to 3, 1, 2, 4, 5, 6, 7$.

Заметим, что циклически сдвинуть три элемента влево — это то же самое, что и два раза циклически сдвинуть их вправо, так что мы научились циклически сдвигать три элемента в любом направлении.

Теперь мы должны научиться получать q из p с помощью таких циклических сдвигов: найдём первый элемент q в p, обозначим его индекс за j. Если $j > 2$, циклически сдвинем вправо элементы с индексами $j-2, j-1, j$. Последовательным применением таких операций мы сможем переместить рассматриваемый элемент на одну из первых двух позиций. Если оказалось, что в конце элемент находится на второй позиции, то циклически сдвинем влево элементы с индексами $1, 2, 3$. Потом проделаем аналогичную последовательность действий для второго элемента q, и так далее до $(n-2)$-го элемента. В конце мы получим либо $q_1, \ldots, q_{n-2}, q_{n-1}, q_n$, либо $q_1, \ldots, q_{n-2}, q_n, q_{n-1}$. Заметим, что вторую перестановку мы получить не могли, так как её чётность отличается от чётности q, а значит и от чётности p. Поэтому мы получим именно q.

Итого решение использует не более, чем $\sum\limits_{i=1}^{n} \left(\left\lfloor \frac{n-i}{2} \right\rfloor + 2\right) \leqslant 2n + \frac{1}{2} \sum\limits_{i=1}^{n} (n-i) = \frac{n(n-1)+8n}{4} = \frac{n(n+7)}{4}$ циклических сдвигов вправо трёх соседних элементов. Тогда, если $k = 2$, то $n \geqslant 4$, а также циклический сдвиг трёх соседних элементов выполняется за один k-перенос, поэтому решение использует не более, чем $\frac{n(n+7)}{4} \leqslant n^3$ k-переносов. Иначе $k \geqslant 4$, а значит $n \geqslant 6$, и решение использует не более, чем $\frac{n(n+7)}{4} \cdot \left(3 + 2\left(\left\lfloor \frac{n-1}{2} \right\rfloor + 1\right)\right) \leqslant \frac{n(n+7)}{4} \cdot (n+4) \leqslant n^3$ k-переносов.