Problem A. Absolutely Flat

Идея: Дмитрий Штукенберг
Разработка: Геннадий Короткевич

Нужно проверить оба случая, описанные в условии задачи.

Если $a_1 = a_2 = a_3 = a_4$, то стол уже ровный и ответ равен 1.

Иначе можно вручную проверить четыре случая — четыре варианта, под какую ножку положим подкладку. Но также можно поступить и проще: упорядочим ножки по неубыванию, чтобы выполнялось условие $a_1 \le a_2 \le a_3 \le a_4$, и поймём, что удлинять есть смысл только самую короткую ножку. Поэтому проверим, что $a_1 + b = a_2 = a_3 = a_4$, и выведем 1, если это так, и 0 иначе.

Problem B. Bricks in the Wall

Идея: Виталий Аксенов
Разработка: Николай Будин

Вам дана таблица, заполненная нулями и единицами. Вам нужно найти две непересекающихся полоски (прямоугольника с высотой или шириной равной 1) из нулей, с максимальной возможной суммарной длиной.

Вычислим значения $d_u[x][y]$, $d_d[x][y]$, $d_l[x][y]$, $d_r[x][y]$ для каждой клетки (x, y) — длину наибольшей полоски из нулей, начинающейся в клетке (x, y) в направлении вверх/вниз/влево/вправо, соответственно. Эти значения можно вычисить за время $O(n \cdot m)$. Например, можно выделить все максимальные по включению горизонтальные и вертикальные полоски за длину каждой из таких полосок. Либо можно воспользоваться динамическим программированием.

Любые два непересекающихся прямоугольника (с вертикальными и горизонтальными сторонами) можно разделить какой-то вертикальной или горизонтальной прямой. Переберем эту прямую, разделяющую два прямоугольника. Не умаляя общности, пусть мы выбрали горизонтальную прямую. Тогда нужно найти максимальную полоску сверху от этой прямой, и максимальную полоску снизу от этой прямой. Чтобы найти максимальную полоску сверху, нужно взять максимум по всем $d_u[x][y]$ и $d_r[x][y]$ для всех клеток сверху от прямой. Чтобы не перебирать все такие клетки для каждой прямой, будем двигать прямую сверху вниз, при этом множество клеток будет расширяться, и нам нужно будет в максимум добавлять новые значения. Аналогично для полоски снизу от прямой.

И аналогично для вертикальных прямых. Чтобы уменьшить число случаев, можно сначала решить задачу с горизонтальной прямой для исходной таблицы, а потом повернуть таблицу на 90 градусов и ещё раз запустить то же решение.

Problem C. Computer Network

Идея: Виталий Аксенов
Разработка: Виталий Аксенов

Пусть у нас есть какая-то конфигурация — она выглядит как k путей, которые идут из хаба (центра). Посмотрим на самые удалённые компьютеры на каждом пути от хаба в сети. В суммарных задержках их задержки учитываются один раз. Также, на каждом из k путей задержки предпоследнего компьютера учитываются два раза. И так далее.

Нам выгодно, чтобы в конфигурации было k компьютеров, у которых задержка учитывается один раз, k компьютеров, у которых задержка учитывается два раза, и так далее. Чтобы минимизировать сумму задержек, нам следует отсортировать все задержки по убыванию, и первые k компьютеров использовать с коэффициентом 1, вторые k компьютеров использовать с коэффициентом 2, и так далее.

Problem D. Dice Grid

Идея: Георгий Корнеев
Разработка: Нияз Нигматуллин

Для решения задачи будем пытаться двигать кубик, в котором изначально неизвестны цвета по полю, постепенно раскрашивая стороны кубика.

Так как нам надо учесть все возможные варианты путей и раскрасок, построим следующий граф. Вершинами в нём будут тройки $\langle cube, i, j \rangle$, где $cube$ — последовательность из шести цветов кубика, либо специального значения, которое обозначает, что цвет неизвестен, а (i, j) — координаты клетки, в которой мы находимся. Запретим вершины, в которых цвет нижней грани кубика неизвестен или не совпадает с цветом клетки (i, j). Рёбра из вершины — движения вверх или влево.

Далее можно запустить обход в ширину или в глубину из финишной клетки (n, n), чтобы движениями влево и вверх добраться до клетки $(1, 1)$.

Если с какой-то раскраской кубика мы смогли это сделать, то полученная раскраска — это ответ.

Граф явно строить не обязательно, преобразования кубика после движения влево и вверх можно вычислять во время обхода. Единственное, что нужно хранить — это посещённые вершины (например, в хеш-таблице).

Можно доказать, что число вершин в таком графе — $O(n^4)$, так как можно показать, что число различных состояний кубика — $O(n^2)$. Если рассмотреть состояния, в которых только 4 грани из 6 с известным цветом:

1. *(2 оставшиеся грани имеют общую сторону)* Используя только эти грани, можно посетить $O(1)$ клеток, вариантов для пятого цвета — $O(1)$, варианты для шестого цвета можно ограничить $O(n^2)$;
2. *(2 оставшиеся грани противоположны)* Используя эти грани, можно добраться только до клеток одной строки или столбца. Вариантов для пятого цвета — $O(n)$, вариантов для шестого цвета в дальнейшем тоже $O(n)$.

Problem E. Easily Distinguishable Triangles

Идея: Иван Лукьянов
Разработка: Геннадий Короткевич

Для того, чтобы нарисовать треугольник в пустой клетке, нам нужно выбрать из левой и правой границ клетки, а также из верхней и нижней, причём эти выборы друг на друга не влияют. Поэтому можно решать задачу отдельно для каждой строки матрицы и отдельно для каждого столбца, а потом перемножить.

Рассмотрим некоторую строку. Она состоит из закрашенных в белый или чёрный цвет клеток, а также отрезков пустых клеток между ними. Опять же, отрезки пустых клеток друг на друга не влияют, поэтому можно перемножить ответы для них.

Наконец, для отрезка пустых клеток нам важна его длина k, цвет клетки слева от него и цвет клетки справа от него. Если слева или справа от отрезка проходит граница поля, то будем считать, что там белая клетка. Есть четыре варианта:

- Если обе клетки слева и справа от отрезка — чёрные, то заполнить отрезок корректным образом нельзя, и можно смело выводить 0.
- Если клетка слева от отрезка — чёрная, а клетка справа — белая, то в каждой клетке отрезка нужно выбрать для треугольника правую сторону. Так как для отрезка есть один вариант, ответ на задачу никак не меняется.
- Если клетка слева — белая, а справа — чёрная, то всё симметрично предыдущему случаю.
- Если обе клетки слева и справа — белые, то можно выбрать любое число x от 0 до k и в первых x клетках отрезка выбрать для треугольника левую сторону, а в остальных $k - x$ клетках — правую. Следовательно, ответ на задачу нужно домножить на $k + 1$.

Все последовательные отрезки пустых клеток в строках и столбцах можно найти простым линейным проходом. Сложность решения составит $O(n^2)$.

Problem F. Focusing on Costs

Идея: Артем Васильев
Разработка: Артем Васильев

Докажем по индукции, что можно получить все числа вида $\sqrt{\frac{a}{b}}$, $b \geq 1$. Для этого реализуем процедуру, подобную алгоритму Евклида. Базой индукции будет случай $a = b = 1$, для этого достаточно вычислить `cos(0)`. Рассмотрим переходы.

- $a < b$. Нарисуем прямоугольный треугольник с катетами $\sqrt{a}$ и $\sqrt{b-a}$. Его гипотенуза равна $\sqrt{a + (b-a)} = \sqrt{b}$. Посмотрим на угол, противолежащий катету $\sqrt{a}$. Тангенс этого угла равен $\sqrt{\frac{a}{b-a}}$, а синус — $\sqrt{\frac{a}{b}}$.

  Таким образом, чтобы от $\sqrt{\frac{a}{b-a}}$ перейти к $\sqrt{\frac{a}{b}}$, достаточно применить `atan`, а затем `sin`.

- $a > b$. Научимся переворачивать дробь $\sqrt{\frac{a}{b}}$, таким образом, сводиться к предыдущему случаю. Рассмотрим прямоугольный треугольник с катетами $\sqrt{a}$ и $\sqrt{b}$. Тогда тангенсы двух острых углов равны $\sqrt{\frac{a}{b}}$ и $\sqrt{\frac{b}{a}}$. Чтобы заменить угол α на угол $\frac{\pi}{2} - \alpha$, можно, к примеру, вычислить `acos(sin(`α`))`.

  В итоге, чтобы перевернуть дробь $\sqrt{\frac{a}{b}}$, можно применить четыре операции: `atan`, `sin`, `acos`, `tan`.

Применяя этот алгоритм к паре (a^2, b^2), мы получим искомое выражение. Его длина не превосходит $4 \max(a, b)^2$.

Problem G. Greatest Common Divisor

Идея: Геннадий Короткевич
Разработка: Геннадий Короткевич

Написав простое решение, перебирающее все пары, и примерно оценив рост числа подходящих пар в зависимости от n, можно понять, что для $n \leq 2 \cdot 10^5$ хватит времени и памяти выписать все подходящие пары в один массив. Действительно, для $n = 2 \cdot 10^5$ таких пар 4 480 708.

Заметим, что после первой итерации алгоритма пара (x, y) заменится на пару (y, w), где $w = x$ `div` y. При этом произведение элементов в новой паре $y \cdot w$ не превышает x, который в свою очередь не превышает n, а значит, возможных пар (y, w) после первой итерации алгоритма всего $O(n \log n)$, и их все можно перебрать.

Зафиксируем некоторую такую пару вида (y, w). Запустим алгоритм из условия до конца, пусть он вернёт некоторое число g. Если y не делится на g, такая пара нам неинтересна. Иначе нам интересны все такие значения x, что x `div` $y = w$ и $\gcd(x, y) = g$. Первое условие эквивалентно неравенствам $y \cdot w \leq x < y \cdot (w + 1)$, а из второго условия следует, что x должно делиться на g. Значит, переберём все делящиеся на g числа x из диапазона $[y \cdot w; y \cdot (w + 1))$ и для каждого из них проверим условие $\gcd(x, y) = g$. Таким образом мы и найдём все подходящие пары (x, y).

Problem H. Hidden Digits

Идея: Михаил Дворкин
Разработка: Николай Будин

Давайте реализуем рекурсивную функцию `solve(a, f)`, которая принимает массив двоичных масок $a_0, a_1, \ldots a_{k-1}$ и находит минимальное x такое, что в x встречаются все цифры из a_0, в $(x+1)$ встречаются все цифры из a_1, и т.д. Дополнительно эта функция будет принимать флаг f, обозначающий, может ли x равняться 0, или оно должно быть строго положительным.

Теперь рассмотрим три случая для функции `solve`:

- Если $|a| = 1$. Тогда нам нужно просто собрать минимальное число, которое содержит все цифры из a_0 и удовлетворяет флагу f. В большинстве случаев достаточно взять цифры из a_0 в возрастающем порядке, и если на первом месте оказался 0, то поменять местами первую и вторую цифры. Однако, есть крайние случаи, которые нужно аккуратно разобрать: если a_0 пусто; если x не может быть 0; если a_0 содержит только 0.
- Если $|a| = 2$. Тогда есть два интересных варианта, как могут выглядеть x и $x + 1$:
  1. $x = \overline{\ldots d}$, $x + 1 = \overline{\ldots (d + 1)}$, $d < 9$. То есть, x и $x + 1$ имеют одинаковый префикс, и разные цифры на конце. Переберем d, вычеркнем d из a_0, вычеркнем $d + 1$ из a_1, сделаем $b = a_0|a_1$, и найдем минимальное число, содержащее все цифры из b.
  2. $x = \overline{\ldots d9}$, $x + 1 = \overline{\ldots (d + 1)0}$, $d < 9$. Снова переберем d.
- Если $|a| > 2$. Тогда переберем $x \bmod 10$, то есть последнюю цифру первого числа. В результате, мы узнаем последние цифры всех чисел. Вычеркнем их из масок. Теперь отбросим у каждого числа последнюю цифру, то есть нацело поделим все числа на 10. В результате у нас будут отрезки подряд идущих одинаковых чисел. Объединим числа внутри такого отрезка в одно, при этом множества необходимых цифр объединятся. Рекурсивно вызовем `solve` от нового множества масок, чтобы найти минимальное подходящее значение $\lfloor \frac{x}{10} \rfloor$.

Пусть функция `solve` при $|a| = n$ работает за $T(n)$. Тогда $T(n) = 10 \cdot T(n/10) + O(n)$. Поэтому, $T(n) = O(n \log n)$.

Чтобы решить задачу, нужно запустить `solve` с $a_i = \{d_i\}$.

Problem I. IQ Game

Идея:	Артем Васильев
Разработка:	Артем Васильев

Воспользуемся формулой подсчета математического ожидания неотрицательной целочисленной случайной величины X:

$$\mathbb{E}X = \mathbb{P}(X > 0) + \mathbb{P}(X > 1) + \ldots$$

Интуитивно ее можно интерпретировать следующим образом: чтобы найти матожидание числа оставшихся раундов, посчитаем вероятность, что мы сыграем первый раунд ($\mathbb{P}(X > 0) = 1$), прибавим вероятность, что мы сыграем второй раунд ($\mathbb{P}(X > 1)$) и так до бесконечности (но в нашей задаче $\mathbb{P}(X > k) = 0$, и дальше суммировать смысла нет).

Повернем стол так, чтобы гиперблиц оказался в ячейке с номером 1, и пронумеруем оставшиеся вопросы в порядке **против часовой стрелки**: $1 < q_2 < q_3 < \ldots$. В каких ситуациях после m вопросов ни разу не выпал гиперблиц?

- В полуинтервал $[1, q_2)$ мы не должны попасть ни разу,
- В полуинтервал $[1, q_3)$ мы можем попасть не более одного раза,
- В полуинтервал $[1, q_4)$ мы можем попасть не более двух раз,
- ...
- В полуинтервал $[1, q_i)$ мы можем попасть не более $i - 2$ раз.

Посчитать вероятность таких исходов можно, используя динамическое программирование. Обозначим за $P_{i,j}$ вероятность события «После j раундов не выпал гиперблиц, а все выпавшие сектора принадлежали первым i полуинтервалам». Чтобы сделать переход из такого состояния, переберем,

сколько раз волчок указывал на сектор внутри $i+1$-го полуинтервала (обозначим это за x), и перейдем в $P_{i+1,j+x}$, при условии, что $j+x$ не превосходит $i-1$. Вероятность перехода при этом нужно домножить на $\binom{j+x}{x}(\frac{q_{i+1}-q_i}{n})^x$: выбрать, какие x из $j+x$ попали в текущий полуинтервал, и вероятность каждого отдельного выпадания (x раз).

После обработки всех полуинтервалов слева направо, на основе значений $P_{k,*}$, используя самую первую формулу из этого разбора, можно найти матожидание. Всего в решении $O(k^2)$ состояний, а каждый переход занимает $O(k)$ времени. Общее время работы — $O(k^3)$.

Problem J. Joking?

Идея: Павел Кунявский
Разработка: Борис Минаев

Вместо n массивов длины k будем генерировать один массив длины $n \cdot k$. i-й элемент в этом массиве будет обозначать, на каком кубике расположено i-е по возрастанию число.

Результаты получаются довольно хорошими, если генерировать массив следующим образом. Разобьем его на k частей. В каждой части запишем случайную перестановку чисел от 1 до n. Теоретически существует хороший ответ, который соответствует такой конструкции, но чтобы его сгенерировать, должно очень сильно повезти с выбором перестановок.

Рассмотрим два способа улучшить такой ответ. Первый — написать алгоритм имитации отжига, который меняет два случайных элемента внутри одной перестановки. Чаще всего он сможет найти удовлетворяющий ответ за несколько секунд.

Другой способ заключается в том, чтобы подобрать еще более лучшую конструкцию. Все блоки по n чисел можно разбить на пары соседних. В каждой паре вторую перестановку сделаем такой же, как первую, только зеркально отраженную. Это уже показывает хороший результат. Чтобы еще сильнее его улучшить, можно воспользоваться алгоритмом локальных оптимизаций, который меняет случайные числа из перестановок, а также обновляет соседнюю, чтобы сохранялась зеркальность.

Поскольку всего в задаче только четыре теста, если ваше решение работает долго, можно локально сгенерировать ответы и сохранить их в программу.

Много интересных фактов про эту задачу можно прочитать на сайте `http://www.ericharshbarger.org/dice/go_first_dice.html`.

Problem K. K-Shaped Figures

Идея: Геннадий Короткевич
Разработка: Павел Кунявский

Тривиальное решение состоит в переборе всех троек отрезков и проверки условия задачи для каждого такой тройки. Такое решение будет требовать $O(n^3)$ проверок и является слишком долгим.

Для ускорения заметим, что одно из условий достаточно сильно ограничивает возможные хорошие тройки отрезков. А именно, если перебрать отрезок, являющийся «основанием» буквы К, то все остальные отрезки можно разбить на группы по общему концу, лежащему на этом отрезке, за линейное время с помощью хеш-таблицы, или за $O(n \log n)$ с помощью сортировки. В качестве двух других отрезков в тройке с отрезком-«основанием» имеет смысл выбирать только отрезки из одной группы. Однако, если почти все отрезки будут сходиться в одной точке, такое решение все еще будет работать за $O(n^3)$, хоть и с заметно более хорошей скрытой константой.

Для достаточно быстрого решения необходимо научиться обрабатывать каждую группу за ололинейное время. В группе нужно найти отрезки, которые будут направлены в одну сторону относительно основания, и при этом не сонаправлены друг с другом. Для этого нужно найти количество пар, направленных в одну сторону относительно отрезка (что легко сделать за линейное время, посчитав количество направленных в каждую из двух сторон), и вычесть количество пар сонаправленных. Для подсчёта последнего количества нужно отнормировать каждый вектор, соответствующий отрезку, чтобы сонаправленные вектора стали одинаковыми, и посчитать количество

пар одинаковых. Для нормировки вектора можно поделить на их длину или, чтобы обойтись без вещественных чисел, на наибольших общий делитель их координат. Количество пар одинаковых можно найти, сгруппировав одинаковые отрезки с помощью хеш-таблицы или сортировки. Такое решение потребует $O(n)$ или $O(n \log n)$ времени в зависимости от реализации.

Такие образом, суммарное время работы решения составит $O(n^2)$ или $O(n^2 \log n)$.

Problem L. Limited Swaps

Идея: Артем Васильев
Разработка: Артем Васильев

Математическая формулировка условия звучит следующим образом: дано две перестановки чисел от 1 до n, нужно обменами соседних элементов превратить первую во вторую, однако, менять местами элементы, отличающиеся значениями на один, запрещено.

Для начала поймем, в каком случае решение существует. Заметим, что относительный порядок чисел i и $i+1$ (для всех i от 1 до $n-1$) не может измениться, потому что им в какой-то момент нужно будет оказаться рядом и поменяться местами. Поэтому, если какая-то пара соседних значений находится в разном порядке в двух входных перестановках, то ответ `-1`.

Иначе, всегда можно превратить одну перестановку в другую. Опишем один из возможных подходов. Рассмотрим число n. Сдвинем его либо в начало, либо в конец перестановки (в зависимости от взаимного расположения с $n-1$). Забудем про число n, решим задачу для пар перестановок размера на один меньше. В конце аналогичным образом поставим n на нужное место.

Это решение использует не больше $2(n-1)+2(n-2)+\ldots \le n^2$ обменов, что комфортно укладывается в поставленные ограничения.

Problem M. Mex and Cards

Идея: Михаил Мирзаянов
Разработка: Геннадий Короткевич

Поймём, как посчитать максимальную сумму mex-ов.

Во-первых, нет смысла делать больше a_0 кучек карт — для того, чтобы у кучки был положительный mex, в ней должна быть хотя бы одна карта со значением 0. В то же время, нет смысла класть две карты со значением 0 в одну кучку — всегда можно выделить одну из этих карт в новую кучку с $\text{mex} = 1$, улучшив ответ. Значит, мы будем делать ровно a_0 кучек.

С картами со значением 1 ситуация похожая. Если $a_1 < a_0$, положим все карты со значением 1 в разные кучки. Тогда у $a_0 - a_1$ кучек значение mex зафиксируется на 1, и его больше нельзя будет улучшить. Остальные же a_1 кучек можно будет продолжать. Если же $a_1 \ge a_0$, то положим по одной карте со значением 1 в каждую кучку, а остальные — куда угодно. Тогда все a_0 кучек можно будет использовать дальше.

Легко понять, что такая жадная стратегия будет оптимальной. При этом для любого $k = 0, 1, \ldots, n-1$ значение $mex > k$ будут иметь $\min(a_0, a_1, \ldots, a_k)$ кучек. Следовательно, ответ на задачу равен сумме префиксных минимумов в массиве a:

$$a_0 + \min(a_0, a_1) + \min(a_0, a_1, a_2) + \ldots + \min(a_0, a_1, \ldots, a_{n-1}).$$

Как обрабатывать изменения? Пусть мы добавили новую карту со значением v, и таким образом число карт a_v со значением v увеличилось с x до $x+1$. Есть два случая:

- Если $\min(a_0, a_1, \ldots, a_{v-1}) \le x$, то ни одного префиксного минимума от увеличения a_v не изменится.
- Если $\min(a_0, a_1, \ldots, a_{v-1}) > x$, то префиксный минимум на отрезке $[a_0, a_1, \ldots, a_v]$ увеличится на 1, со значения x до значения $x+1$. Но также могут увеличиться и несколько следующих

префиксных минимумов. Найдём ближайшую позицию $i > v$ такую, что $a_i \le x$. Тогда минимум на префиксе $[a_0, a_1, \ldots, a_{i-1}]$ ещё увеличится, а на префиксе $[a_0, a_1, \ldots, a_i]$ — уже нет. Так как ответ на задачу равен сумме префиксных минимумов, то всего ответ на задачу увеличится ровно на $i - v$.

При удалении карты со значением v и уменьшением a_v с $x + 1$ до x все действия симметричны, только вместо слова "увеличение" в предыдущем абзаце следует читать "уменьшение".

Чтобы быстро искать первую позицию со значением не более x в изменяющемся массиве, можно использовать спуск по дереву отрезков на минимум. Сложность решения составит $O((n + q) \log n)$.

Problem N. New Time

Идея: Николай Ведерников
Разработка: Геннадий Короткевич

Представим себе окружность, на которой отмечено $24 \cdot 60 = 1440$ точек, и каждая точка соответствует некоторой минуте в сутках. Пронумеруем точки по часовой стрелке от 0 до 1439, где точка 0 — это время `00:00`, а точка 1439 — это время `23:59`. Пусть время на часах соответствует точке x, а правильное время — точке y.

Заметим, что кнопка A увеличивает номер текущей точки на 1, а кнопка B — на 60.

Найдём d — расстояние между точками по часовой стрелке. Если $x \le y$, то $d = y - x$, иначе $d = (y + 1440) - x$.

Мы хотим как можно быстрее продвинуться на расстояние d, используя прыжки вперёд на расстояния 1 и 60. Легко понять, что нам нужно использовать d `mod` 60 прыжков первого типа и d `div` 60 прыжков второго типа. Значит, ответ на задачу равен d `mod` 60 + d `div` 60.

Можно было и не использовать при решении задачи конструкцию с окружностью, описанную выше, а просто промоделировать процесс, если догадаться, что сначала нужно увеличивать минуты, пока они не станут правильными, а потом — увеличивать часы.