# Problem A. Accurate Movement

| | |
|---|---|
| Идея задачи: | Виталий Аксенов |
| Условие: | Павел Кунявский, Георгий Корнеев |
| Проверяющая программа: | Павел Кунявский |
| Тесты: | Павел Кунявский |

Каждый раз когда мы передвигаем блок, его имеет смысл сдвигать как можно правее. Таким образом, мы сможем двигать блоки по очереди, и каждый раз блок будет передвигаться на $b - a$. Так как общее расстояние, на которое требуется передвинуть длинный блок равно $n - b$, то на его передвижение потребуется $2\left\lceil\frac{n-b}{b-a}\right\rceil$ действий. Так как для передвижения маленького блока потребуется еще одно действие, то общее число действий будет равно $2\left\lceil\frac{n-b}{b-a}\right\rceil + 1$.

# Problem B. Bad Treap

| | |
|---|---|
| Идея задачи: | Михаил Дворкин |
| Условие: | Михаил Дворкин |
| Проверяющая программа: | Михаил Дворкин |
| Тесты: | Михаил Дворкин |

Чтобы декартово дерево имело глубину $n$ достаточно, чтобы и $x$, и $y$ были монотонными.

Так как синус имеет период $2\pi$, будем искать решение вида $sin(k(2\pi+\varepsilon))$, то есть последовательность $sin(0) < sin(2\pi + \varepsilon) < sin(2(2\pi + \varepsilon)) < \cdots < sin(k(2\pi + \varepsilon))$. Для этого переберем все натуральные числа до $(2^{31} - 1)/50\,000$ и найдем среди них число, имеющее минимальный остаток по модулю $2\pi$. Таким числом окажется $710 \approx 113 \cdot 2\pi + 0.00006$.

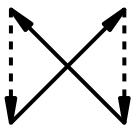
К сожалению, $0.00006 \cdot 50\,000 = 3 > \pi/2$, поэтому синусы выбранных чисел не будут возрастающей последовательностью. Однако, мы можем начать не с 0 а с $-710 \cdot 25\,000$, тогда мы получим интервал отклонений от чисел вида $2\pi \cdot k$ от $-1.5 = -0.00006 \cdot 25\,000$ до $1.5 = 0.00006 \cdot 25\,000$, что позволяет получить серию из 50 001 возрастающих значений синуса.

# Problem C. Cross-Stitch

| | |
|---|---|
| Идея задачи: | Георгий Корнеев |
| Условие: | Георгий Корнеев |
| Проверяющая программа: | Георгий Корнеев |
| Тесты: | Георгий Корнеев |

В этой задаче необходимо составить схему вышивки крестом заданного узора, требующую нить минимальной длины.

Заменим в узоре каждый крест на «бабочку» из двух лицевых и двух изнаночных стежков, как показано на схеме:



В результате замены между соседними по горизонтали крестами могут возникнуть кратные ребра, которые надо будет аккуратно обработать в решении.

Заметим, что если два креста были соседями по вертикали, горизонтали или диагонали, то у соответствующих им бабочек есть общая вершина. Так как исходное множество крестов 8-связно, то построенный граф так же будет связным.

В построенном графе, число лицевых ребер (стежков), входящих в каждую вершину равно числу изнаночных ребер выходящих из него, и наоборот. Таким образом, алгоритмом аналогичным поиску

эйлерова цикла в этом графе можно построить цикл, проходящий по всем ребрам, в котором лицевые и изнаночные ребра чередуются. Разорвем этом цикл по произвольному изнаночному ребру и получим путь из $n$ лицевых и $n-1$ изнаночного стежка. Так как меньше чем $n-1$ изнаночных стежков сделать нельзя, и каждый такой стежок должен иметь длину не меньше 1, то длина построенного пути минимальна.

Так как построенный граф имеет $O(n^2)$ вершин и ребер, то искомый путь в нем можно найти, затратив $O(n^2)$ времени и памяти.

# Problem D. Double Palindrome

| | |
|---|---|
| Идея задачи: | Геннадий Короткевич |
| Условие: | Геннадий Короткевич |
| Проверяющая программа: | Геннадий Короткевич |
| Тесты: | Геннадий Короткевич |

Для того, чтобы посчитать число двойных палиндромов длины $n$, посчитаем вспомогательную функцию $R(n,k)$ — число способов построить строку длины $n$ над алфавитом размера $k$ в виде конкатенации двух палиндромов. Для этого переберём длину первого палиндрома от 0 до $n-1$. Так как число палиндромов длины $l$ над алфавитом размера $k$ равно $k^{\lceil l/2 \rceil}$, то получим $R(n,k) = \sum_{l=0}^{n-1} k^{\lceil l/2 \rceil} \cdot k^{\lceil (n-l)/2 \rceil}$.

Однако при таком разбиении некоторые двойные палиндромы будут посчитаны два раза, например строку "`abacabacabac`" можно представить как пару палиндромов тремя способами: "`aba`|`cabacabac`", "`abacaba`|`cabac`" и "`abacabacaba`|`c`". Таким образом, мы посчитаем эту строку три раза.

Заметим, что строка $s$ длины $n$ является двойным палиндромом тогда и только тогда, когда $s$ равна циклическому сдвигу развернутой строки $s$, т. е. существует число $k$ (которое на самом деле равно длине первого из палиндромов, на которые разбивается $s$) такое, что $\forall i : s_i = s_{k-1-i \mod n}$.

Пусть для строки $s$ нашлось два способа разбить её на два палиндрома, при этом длины первого палиндрома в этих способах равны $l_1$ и $l_2$ соответственно ($l_1 < l_2$). Тогда

$$s_i = s_{l_1-1-i \mod n} = s_{l_2-l_1+i \mod n},$$

то есть строка $s$ периодична с периодом $l_2 - l_1$, и даже с периодом $\gcd(n, l_2 - l_1)$. Пусть $p$ — минимальный период двойного палиндрома $s$, тогда $p$ — тоже двойной палиндром, и $p$ представляется в виде конкатенации не более двух палиндромов единственным способом, при этом в $R(n,k)$ мы посчитали $s$ ровно $\frac{|s|}{|p|}$ раз.

Давайте посчитаем значение $D(n,k)$ — количество двойных палиндромов длины $n$ над алфавитом размера $k$, которые разбиваются на два палиндрома единственным способом. Для $D(n,k)$ верна следующая рекуррентная формула:

$$D(n,k) = R(n,k) - \sum_{\substack{l|n \\ l<n}} \frac{n}{l} D(l,k)$$

Тогда ответом будет $\sum_n \sum_{l|n} D(l,k)$. Суммарное время работы составляет $\mathcal{O}(n \log n)$.

# Problem E. Equidistant

| | |
|---|---|
| Идея задачи: | Михаил Мирзаянов |
| Условие: | Борис Минаев |
| Проверяющая программа: | Борис Минаев |
| Тесты: | Борис Минаев |

Подвесим дерево за вершину $c_1$ и найдём самую глубокую команду $c_f$ и соответствующую ей глубину $d$. Пусть $v$ — вершина на середине пути от $c_1$ до $c_f$. Докажем, что либо она является ответом, либо ответа не существует.

Заметим, что ответ должен лежать в поддереве вершины $v$ ($T_v$), так как в противном случае расстояние до $c_1$ будет меньше расстояния до $c_f$.

Для всех команд не из $T_v$, расстояние до любой вершины из $T_v$ одинаковы. Таким образом, либо любая вершина из $T_v$ будет ответом для таких команд, либо ответа не существует.

С другой стороны, все команды из $T_v$ имеют глубину не большую $d$, иначе $c_f$ была бы не самой глубокой вершиной. Если все они имеют глубину ровно $d$, то $v$ является ответом для команд из $T_v$. В противном случае, для этих команд решение не существует.

Таким образом, всё что нам требуется — проверить, что вершина $v$ является ответом. Сделаем это, переподвесив дерево за неё и проверив, что глубины всех команд равны $d$.

Таким образом, задача решается за два обхода в глубину, а общее время работы составляет $O(n)$.

# Problem F. Foreach

| | |
|---|---|
| Идея задачи: | Павел Кунявский |
| Условие: | Павел Кунявский |
| Проверяющая программа: | Павел Кунявский |
| Тесты: | Павел Кунявский |

Если в массиве $b$ есть значения, которых не было в $a$, то ответа не существует. Кроме того, так как первая же операция сделает какие-то два элемента одинаковыми, если $b$ не равен $a$ и все его элементы различны, то ответа не существует.

Во всех остальных случаях мы можем построить ответ из следующих базовых операций:

- Заменить первое вхождение заданного значения на любой другой элемент массива;
- Заменить последний элемент на любой другой элемент массива.

Первая базовая операция может быть реализована как два цикла — сначала по ссылке до начального значения, потом без ссылка до конечного. Вторая операция может быть реализована аналогично, если в качестве значения первого цикла взять такое, которого нет в массиве.

Мы будем устанавливать элементы в правильное значение справа налево, пропустив последний элемент. Чтобы установить очередной элемент, нужно все равные значения раньше заменить на что-нибудь другое, потом сохранить значение этого элемента в последний, а это сделать правильным.

Единственная проблема которая может возникнуть — мы могли уже потерять все нужные значения. Чтобы такого не случилось, последний элемент никогда не должен быть уникальным. Для этого, если он становится уникальными, мы можем найти любое значение на необработанном префиксе, которое не уникально, или уже есть среди обработанных элементов, и заменить его на ещё одну копию значения последнего элемента.

После этого процесса, можно поставить последний элемент в нужное значение, и задача решена.

К сожалению, если несколько первых элементов массива вместе с последним уникальны, и не встречаются нигде больше, то на каком-то из шагов мы можем не найти куда скопировать значение последнего элемента.

Чтобы избежать этого, можно поменять конечный массив, заменив первый неуникальный элемент на значение первого элемента, и сначала преобразовать в такой массив. Теперь, осталось поменять один элемент на значение, которое есть где-то ещё. Так как мы выбирали первый такой элемент, ни одно из значение между первым и измененным элементов не равно ни первому, ни последнему, ни правильному значению изменённого.

Оставшаяся часть решается следующим образом. Пусть текущие значения первого, изменённого и последнего элемента это $[x, x, y]$, а целевые — $[x, z, y]$. Тогда можно сделать следующую последовательность операций:

$$[x, x, y] \to [y, x, y] \to [y, x, x] \to [y, y, x] \to [x, y, y] \to [x, z, y]$$

Заметим, что их можно делать даже если $z = y$. А $x$ не может быть равен $y$ или $z$, так как первый элемент был уникален.

Решению требуется $O(n^2)$ циклов, что, кстати, является оптимальной асимптотикой на тесте вида $a = [0, 1, 0, 1, 0, 1, \dots]$, $b = [1, 0, 1, 0, 1, 0, \dots]$. Вероятно, есть много других решений.

# Problem G. Golf Time

| | |
|---|---|
| Идея задачи: | Георгий Корнеев |
| Условие: | Геннадий Короткевич |
| Проверяющая программа: | Геннадий Короткевич |
| Тесты: | Геннадий Короткевич |

Мяч тонет в момент касания одной из сторон многоугольника. Переберем все стороны и найдем время первого касания для каждой из них. Тогда ответом на задачу, будет минимальное среди полученных времен.

Рассмотрим сторону $i$. Не уменьшая общности, будем считать что она вертикальная (случай горизонтальной стороны решается аналогично с точностью до замены координат). Введем обозначения $x_s = x_{i,1} = x_{i,2}$, $y_1 = y_{i,1}$ и $y_2 = y_{i,2}$, тогда отрезок будет иметь вид $x = x_s$ и $y_1 \le y \le y_2$.

Для удобства, вместо отражения мяча от сторон поля для гольфа, будет отражать само поле. При этом, мяч будет лететь по прямой $(\tilde{x} + t, \tilde{y} + t)$, а рассматриваемая сторона многоугольника даст бесконечную последовательность отрезков на решетке размера $2 \cdot w \cdot h$. Выпишем явным образом все виды этих отрезков:

- $x = 2wk + x_s$, $2hl + y_1 \le y \le 2hl + y_2$
- $x = 2wk + x_s$, $2hl + 2h - y_2 \le y \le 2hl + 2h - y_1$
- $x = 2wk + 2w - x_s$, $2hl + y_1 \le y \le 2hl + y_2$
- $x = 2wk + 2w - x_s$, $2hl + 2h - y_2 \le y \le 2hl + 2h - y_1$

где $k$ и $l$ — целые числа. Заметим, что каждый вид отрезков можно задать как $x = 2wk + \overline{x}$, $2hl + \overline{y}_1 \le y \le 2hl + \overline{y}_2$, для некоторых $\overline{x}$ и $\overline{y}$. Решим задачу отдельно для отрезка каждого вида и выберем среди полученных времен минимальное.

Траектория мяча пересекает последовательность прямых вида $x = 2wk + \overline{x}$ с периодом $2w$. Таким, образом, времена пересечения имеют вид $t_k = t_0 + 2wk$ для некоторого $0 \le t_0 < 2w$ и $k \ge 0$. Найдем решение с минимальным $k$. В начале проверим $k = 0$. Если $\overline{y}_1 \le \tilde{y} + t_0 \le 2hl \le \overline{y}_2$, то ответ найден. В противном случае, задача свелась к поиску минимального решения диофантового неравества вида $L \le ak \bmod m \le R$, где $m = 2hl$, $a = 2w$, $L = (\overline{y}_1 - \tilde{y}) \bmod m$, $R = (\overline{y}_2 - \tilde{y}) \bmod m$.

Решим это неравенство, рассмотрев несколько случаев

1. Если $2a > m$, то сведем к решению неравенства $m - R \le ((m - a)k) \bmod m \le m - L$.
2. Если $k = \left\lceil \frac{L}{a} \right\rceil$ являетсе решением, но оно минимальное.
3. Иначе, если $m$ делится на $a$, то решений не существует.

4. Иначе, рассмотрим первые значения, после каждого переполнения по модулю $m$. Каждый раз оно будет изменяться на $m \bmod a$. Нам требуется найти минимальное значение, которое удовлетворяет по модулю $a$ попадет в отрезок $L \bmod a..R \bmod a$. Так как решение не было найдено во втором случае, то $k = \lceil \frac{L}{a} \rceil$ не является решением и $L \bmod a < R \bmod a$. Таким образом, мы свели задачу к решению неравенства $(L \bmod a) \le ((a - (m \bmod a)) \cdot k' \le (R \bmod a)$. Решим его тем же алгоритмом. Пусть $k'$ — минимальное решение нового неравенства, тогда $\left\lceil \frac{k'm+l}{a} \right\rceil$ — минимальное решение исходного неравества. Если новое неравенство не имеет решений, то исходного неравенство так же не имеет решение.

Временная сложность алгоритма решения неравества будет $O(\log m)$, так как после каждого применения случая 1, $a$ уменьается как минимум в два раза. Таким образом, общая сложность полученного решения $O(tn \log(wh))$.

# Problem H. High Load Database

| | |
|---|---|
| Идея задачи: | Виталий Аксенов |
| Условие: | Георгий Корнеев |
| Проверяющая программа: | Нияз Нигматуллин |
| Тесты: | Нияз Нигматуллин |

Пусть $\overline{a} = \max\limits_{i=1}^{n} a_i$. Заметим, что при $t < \overline{a}$, то выполнить скрипт до конца невозможно.

Пусть $A = \sum_{i=1}^{n} a_i$. Предподсчитаем $f[j]$ — номер транзации, в которой находится $j$-й запрос ($j = 1..A$). Будем жадно объединять транзакции в блоки, начиная я первого. Тогда в первый блок попадут транзакции с 1 по $f[t] - 1 = b_1$, во второй блок попадут транзакции $b_1..f[b_2 + t] - 1 = b_2$, в $k -$ блок — транзакции $b_{k-1}..f[b_{k-1} + t] - 1 = b_k$.

Посчитаем время, которое потребуется на симуляцию. Для этого разделим транзакции на *большие*, для которых $2t > a_i$ и *малые*, для которых $2t \le a_i$. Пусть больших транзакций $M$ ($M < 2A/t$), тогда, число блоков содержащих большие транзакции, и соседних с ними не более $3M = O(\frac{A}{t})$. Общее число запросов в блоке, содержащем только малые транзакции, за которым следует блок так же содержащий только малые транзакции не может быть меньше $t/2$, так как иначем мы бы могли взять еще одну малую транзакцию из следующего блока. Суммарная длина малых транзакций не превышает $A$, то число блоков, состоящих только из малых транзакций не превышает $\frac{2A}{t}$. Таким образом, суммарное время обработки всех блоков составит $O(\frac{A}{t})$.

Таким образом, мы можем посчитать число блоков для $t = 1..\overline{a}$ за время $O(\sum_{t=1}^{\overline{a}} \frac{A}{t}) = O(A \log \overline{a})$.

# Problem I. Ideal Pyramid

| | |
|---|---|
| Идея задачи: | Георгий Корнеев |
| Условие: | Илья Збань |
| Проверяющая программа: | Илья Збань |
| Тесты: | Илья Збань |

Минимальная пирамида, содержащая $i$-й обелиск имеет центр $(x_i, y_i)$ и высоту $h_i$. Основание такой пирамиды — квадрат с вершинами $(x_i - h_i, y_i - h_i)$ и $(x_i + h_i, y_i + h_i)$. При этом, основание любой пирамиды, содержащей этот обелиск должен содержать и указанный квадрат.

Таким образом, мы свели задачу к поиску на плоскости наименьшего квадрата, содержащего все квадраты, соответсвующие обелискам. Рассмотрим ограничичвающий прямоугольник этих квадратов: $x_l = \min(x_i - h_i)$, $x_r = \max(x_i + h_i)$, $y_l = \min(y_i - h_i)$, $y_r = \max(y_i + h_i)$. Тогда минимальная пирамида будет иметь высоту $h = \lceil \frac{\max(x_r - x_l, y_r - y_l)}{2} \rceil$, а ее центр может быть расположен в точке $x = \frac{x_l + x_r}{2}, y = \frac{y_l + y_r}{2}$.

# Problem J. Just the Last Digit

| | |
|---|---|
| Идея задачи: | Артем Васильев |
| Условие: | Артем Васильев |
| Проверяющая программа: | Артем Васильев |
| Тесты: | Артем Васильев |

Математическая постановка задачи имеет вид: дано число путей (взятое по модулю 10) между каждой парой вершин в ациклическом направленном графе, требуется востановить граф.

Начнем восстанавливать граф с вершины 1. Если число путей $1 \rightsquigarrow 2$ равно нулю, то ребра нет, иначе ребро есть, при этом, $1 \rightsquigarrow 2$ должно быть равно единице. Если мы удалим это ребро, то число путей $1 \rightsquigarrow i$ уменьшится на число путей $2 \rightsquigarrow i$. Переберем $i$ от 3 до $n$ и вычтем число путей $2 \rightsquigarrow i$ из числа путей $1 \rightsquigarrow i$ (все вычисления производятся по модулю 10).

Далее рассмотрим число путей $1 \rightsquigarrow 3$. Если это ноль, то ребра $1 \to 3$ нет, а если единица, то ребро есть и мы можем вычесть число путей $3 \rightsquigarrow i$ из $1 \rightsquigarrow i$ для $i$ от 4 до $n$. Повторим эту процедуру для $1 \rightsquigarrow 4$, $1 \rightsquigarrow 5$ и так далее. Таким образом, мы найдем все ребра из вершины 1 за $\mathcal{O}(n^2)$.

Аналогично переберем ребра из вершин 2, 3, ..., $n$. Общее время исполения будет $\mathcal{O}(n^3)$.

# Problem K. King's Children

| | |
|---|---|
| Идея задачи: | Павел Маврин |
| Условие: | Павел Маврин |
| Проверяющая программа: | Павел Маврин |
| Тесты: | Павел Маврин |

Заметим, что если не требуется максимизировать площадь прямоугольника '`A`', то задачу можно решить следующим образом. Растянем каждую букву по вертикали, пока она не упрется в другую букву или край королевства. Таким образом, получится набор полных столбцов. Растянем каждый столбец по горизонтали, пока он не упрется в соседний или край королевства. Таким образом, королевство будет разбито на прямоугольные регионы.

Научимся максимизировать прямоугольник '`A`'. Для этого найдем прямоугольник максимальной площади, содержащий букву '`A`', и не содержащий других букв. Это стандартная задача, которая решается за время $O(nm)$. Докажем, что оставшуюся часть королевства можно разбить на прямоугольники требуемым образом. Разобьем её на четыре больших прямоугольника: выше прямоугольника '`A`', ниже его, слева и справа. Каждый из больших прямоугольников либо пуст, либо содержит хотя бы одну букву (в противном случае, мы бы могли увеличить прямоугольника '`A`' в соответствующую сторону). Решив задачу без максимизации для каждого большого прямоугольника, мы разобьем все королевство на прямоугольники. Таким образом, общее время решения задачи $O(nm)$.

# Problem L. Lengths and Periods

| | |
|---|---|
| Идея задачи: | Антон Гардер |
| Условие: | Антон Гардер |
| Проверяющая программа: | Антон Гардер |
| Тесты: | Антон Гардер |

В задаче требовалость найти подстроку, у которой длина, делённая на минимальный период, максимальна. Назовём такую подстроку оптимальной.

Если ответ не равен 1/1, то у $w$ есть подстрока с периодом, не равным её длине. Рассмотрим $w[i..j)$, пусть её наименьший период $p$, тогда $LCP(i, j-p) \geq p$, так как наибольший общий префикс у $s[i..)$ и $s[j-p..n)$ не меньше $p$. Заметим, что если $LCP(i, j-p) > p$, то наименьший период строки $w[i..j+1)$ также не больше $p$, в то время как её длина больше длины $w[i..j)$, то есть $w[i..j)$ не может быть оптимальной подстрокой.

Теперь нас интересуют только подстроки вида $w[i : j + LCP(i, j))$. Каждая из них обновляет нам ответ значением $j - i + LCP(i, j)/(j - i)$, что равно $1 + LCP(i, j)/(j - i)$. Таким образом, из всех

пар $i < j$ с равными $LCP(i, j)$ оптимальной является та, для которой $j - i$ минимально.

Построим суффиксное дерево на строке ($w$+#). Пусть $len(v)$ – длина пути от корня до $v$. LCP любых двух суффиксов, проходящих через $v$ не меньше $len(v)$. Требуется найти среди этих суффиксов два самых близких — таких, у которых разница начальных позиций ($d$) минимальна, и попробовать обновить ответ значением $1 + len(v)/d$.

Для этого построим для каждой вершины $v$ упорядоченное множество $S_v$, содержащее начальные позиции всех суффиксов, проходящих через $v$. Для этого рекурсивно построим такие множества для детей $v$, обновим для них ответ, а затем сольём их. Во время слияния, когда добавляем начало суффикса в множество $S_v$, найдём его ближайших соседей и обновим наименьшую разницу позиций. После слияния всех множеств обновим ответ. Для того, чтобы слияние работало быстро, будем всегда сливать меньшее к большему.

Таким образом, общее время работы алгоритма будет $O(|w| \log^2 |w|)$.

# Problem M. Managing Difficulties

| | |
|---|---|
| Идея задачи: | Михаил Мирзаянов |
| Условие: | Михаил Мирзаянов |
| Проверяющая программа: | Михаил Мирзаянов |
| Тесты: | Михаил Мирзаянов |

В задаче требовалось найти количество троек $(i, j, k)$, таких что $1 \le i < j < k \le n$ и $a_k - a_j = a_j - a_i$. Заметим, что зная $a_i$ и $a_j$, можно найти требуемое значение $a_k = 2a_j - a_i$.

Будем перебирать $j = n - 1..2$ (по убыванию) и $i = 1..j - 1$, тогда задача сведется к тому, что бы посчитать количество таких $k$, что $k > j$ и $a_k = 2a_j - a_i$.

Будем поддерживать ассоциативный массив $C[v]$ — количество таких $k$, что $k > j$ и $a_k = v$. Тогда при фиксированных $i$ и $j$, ответом будет $C[2a_j - a_i]$. При переходе от $j$ к $j - 1$ в $C$ изменится только одно значение: $C[a_j] := C[a_j] + 1$.

Таким образом, используя реализацию ассоциативного массива на основе хеш-таблицы, мы сможем решить задачу за $O(1)$ при фиксированных $i$ и $j$, и $O(n^2)$ для всех пар $i$ и $j$. Если же вместо хеш-таблицы используем упорядоченный ассоциативный массив (например, `std::map` в C++), то задача будет решена за $O(n^2 \log n)$.