Задача A. Магнит VK

Сначала проверяем есть ли буквы V и K в заданной строке. Если есть, то выводим строку «VK» и завершаем работу программы.

Затем проверяем есть ли буквы v и k в заданной строке. Если есть, то выводим строку «vk» и завершаем работу программы.

В противном случае выводим строку «:(».

Задача B. Робин Гуд и массив

Пусть мы выполнили одну операцию. После выполнения этой операции массив будет иметь вид $a_1, \ldots, a_{i-2}, x, x, x, a_{i+2}, \ldots, a_n$, где $x = \text{median}(a_{i-1}, a_i, a_{i+1})$. Заметим, что теперь мы можем сделать все элементы массива равными x. Для этого надо применить операцию к элементам, которые стоят на позициях $i-2, i-1, i$, чтобы элемент на позиции $i-2$ тоже стал равен x. Таким образом мы сможем расширить влево и вправо отрезок массива, который состоит из элементов равных x, до тех пор пока он не займет весь массив.

Заметим, что если мы хотим приравнять все элементы массива к какому-то числу y, то мы должны уметь получать подмассив y, y, y первой операцией. Если это не так, то не существует элементов, равных y, на расстоянии хотя бы два. Такие элементы не появятся после применения любых операций, а потому количество элементов равных y будет только уменьшаться.

Из вышеописанного факта следует, что мы можем перебрать a_{i-1}, a_i, a_{i+1} и взять в качестве ответа **максимум** по $\text{median}(a_{i-1}, a_i, a_{i+1})$.

Асимптотика решения $\mathcal{O}(n)$.

Задача C. Армагеддон

Рассмотрим три возможных случая.

1) Если Магнусу требуется строго меньше времени, чем Хикару, то он гарантированно выигрывает. Для победы Магнус должен сделать ставку k_2 минут s_2 секунд — ровно то время, которое ему нужно для успешной игры за черных. В этом случае мы выводим -1 и завершаем работу программы.

2) Если Хикару требуется строго меньше времени, чем Магнусу, то он гарантированно выигрывает. Для победы Хикару должен сделать ставку k_1 минут s_1 секунд — ровно то время, которое ему нужно для успешной игры за черных. В этом случае мы выводим k_1 s_1 и завершаем работу программы.

3) Если же Хикару и Магнусу требуется одинаковое время, то гарантировать себе победу не может ни один из игроков. А так как Магнус в этом случае гарантированно выиграть не может, то по условия мы должны вывести оптимальную ставку Хикару.

Оказывается, ставка Хикару должна быть на одну секунду меньше, чем k_1 минут s_1 секунд. То есть, если $s_1 > 0$, то ставка Хикару должна быть k_1 $s_1 - 1$, а иначе ставка Хикару должна быть $k_1 - 1$ 59. Будем эту ставку называть *предпочтительной*. Ниже будет показано, почему *предпочтительная* ставка является оптимальной.

Если Хикару сделает *предпочтительную* ставку, а Магнус поставит строго больше, то играть за черных будет Хикару и времени у него будет как минимум k_1 минут s_1 секунд (ведь ставка Магнуса строго больше *предпочтительной* ставки) и он успешно сыграет за черных.

Если Хикару сделает *предпочтительную* ставку, а Магнус поставит строго меньше, то играть за черных будет Магнус и времени у него будет менее k_1 минут s_1 секунд, поэтому он не сможет успешно сыграть за черных и победа снова будет за Хикару.

Если Хикару сделает *предпочтительную* ставку и Магнус поставит столько же, то с равной вероятностью за черных будет играть Хикару или Магнус и каждый из них станет победителем с равной вероятностью. Обратите внимание, что Хикару не может гарантированно выиграть в этом случае, но он будет выигрывать с вероятностью $\frac{1}{2}$, а в данном случае это максимум, которого можно добиться.

Если Хикару сделает не *предпочтительную* ставку, то Магнус сможет победить сделав *предпочтительную* ставку. Действительно, если Хикару поставит больше, то Магнус будет играть за черных и ему хватит времени. Если же Хикару поставит меньше, то он будет играть за черных и ему не хватит времени.

Таким образом, игрокам выгодно сделать *предпочтительную* ставку. В таком случае они побеждают с вероятностью $\frac{1}{2}$. Если же только один игрок сделает *предпочтительную* ставку, то он гарантированно победит. То есть, вне зависимости от хода соперника каждому игроку выгодней сделать *предпочтительную* ставку.

Задача D. Охота на монстра

Для начала заметим, что время жизни всей команды ограничено её героем с самым низким уровнем здоровья — он погибнет первым. Допустим, что у героя с самым низким уровнем здоровья m единиц здоровья. Тогда команда героев успеет атаковать монстра не больше, чем m раз перед тем, как один или более героев погибнет. Пусть s — это сумма величин атаки всех героев в команде, тогда перед смертью одного или более героев команда успеет нанести $m \cdot s$ единиц урона монстру. В таком случае, если $a \leqslant m \cdot s$, где a — здоровье монстра, то команда сможет убить монстра без потерь.

Предположим теперь, что мы можем брать в команду только тех героев, здоровье которых не меньше некоторого значения m. Тогда очевидно, что среди всех героев со здоровьем, не меньшим m, выгодно брать героев с самыми большими величинами атаки. Достаточно брать очередного героя в порядке убывания силы атаки до тех пор, пока остался еще хотя бы один не взятый герой и $a > m \cdot s$.

Осталось понять, что происходит, когда требование на минимальное здоровье героя для вступления в команду возрастает, например с m до $m+1$. В таком случае, если в команду входили герои со здоровьем m, то теперь их нужно убрать из команды. После этого суммарной атаки героев может не хватить, чтобы убить монстра за $m+1$ ход. В таком случае нужно просто добрать в команду самых сильных ещё не взятых героев.

Реализовать это можно следующим образом. Сперва отсортируем героев по возрастанию здоровья. Вначале возьмем в команду всех героев или только самых сильных героев, которых хватит для убийства монстра. Затем будем итерироваться по возрастанию минимального здоровья, необходимого для вступления в команду. Далее нам понадобится структура данных, из которой можно быстро извлекать минимальный/максимальный элемент, например, множество (std::set) в языке C++. В одном множестве будем хранить не состоящих в команде героев по убыванию силы. Во втором множестве будем хранить героев из нашей команды по возрастанию здоровья. Пусть сейчас требование на минимальное здоровье составляет m_i и минимальная по количеству команда героев собрана. Обновим ответ и перейдем к следующему по возрастанию здоровью для вступления в команду m_{i+1}, где $m_{i+1} > m_i$. Тогда до тех пор, пока минимальный элемент в нашем множестве, хранящем героев из команды, имеет здоровье m_i — будем выполнять удаление этого элемента. После этого добавляем в нашу команду из второго множества самых сильных героев, чтобы общая численность команды была такая, как до удаления. Теперь если хватает суммарной силы, то пытаемся обновить ответ. В последнюю очередь пытаемся удалить героев из команды, если их больше, чем надо. Заметим, что если в какой-то момент нам хватило x героев для победы над монстром, то в будущем нам уже не будет смысла пытаться добавить в команду больше x героев, так как такие ответы заведомо хуже. Мы добавляем героев в команду только взамен удаленных, а также пытаемся удалить лишних.

Задача E. Наибольшее общее простое

Из того, что для двух разных чисел на отрезке их $\mathrm{gcp}(a, b)$ должен равняться какому-то простому числу p следует, что оба числа должны делиться на p. Значит, это должны быть числа кратные p. Все числа кратные p имеют вид $x \cdot p$, где x — некоторое целое положительное число. Тогда, для проверки того, что на отрезке $[l; r]$ есть два числа, которые делятся на p, можно просто брать первые два числа кратные p на этом отрезке (если они есть). Первое число кратное p, назовем его a, которое больше или равно l, можно вычислить по формуле $a = \lceil \frac{l}{p} \rceil \cdot p$. Второе число кратное p, которое больше или равно l, это $b = a + p$. Разумеется, оба числа a и b при этом должны быть меньше или равны r.

Рассмотрим подробнее ограничение $b = a+p = \lceil \frac{l}{p} \rceil \cdot p+p \leqslant r$. Вынесем p за скобки: $p \cdot (\lceil \frac{l}{p} \rceil + 1) \leqslant r$. Известно, что частное $\lceil \frac{l}{p} \rceil$ может принимать порядка $\sqrt{l}$ различных значений. Действительно, мы имеем $\sqrt{l}$ значений при $p \leqslant \sqrt{l}$, а при $p > \sqrt{l}$ само выражение $\lceil \frac{l}{p} \rceil$ принимает значения от 1 до $\sqrt{l}$. Таким образом, если в неравенстве $p \cdot (\lceil \frac{l}{p} \rceil + 1) \leqslant r$ зафиксировать значение частного $\lceil \frac{l}{p} \rceil = x$,

то останется найти самое большое простое число p_x такое, что $\lceil \frac{l}{p_x} \rceil = x$ и $p_x \cdot (x+1) \leqslant r$. Для этого можно просто взять самый большой y такой, что $\lceil \frac{l}{y} \rceil = x$ и $y \cdot (x+1) \leqslant r$, а затем поискать подходящее простое число рядом с y. Это будет работать быстро потому, что простые числа плотно и достаточно равномерно распределены на отрезках в заданных ограничениях. При аккуратной реализации это решение позволяло сдать задачу.

Рассмотрим теперь другое решение. Из условия $b \leqslant r$ следует, что $a+p \leqslant r$ или $a \leqslant r-p$. Таким образом, для фиксированного числа x, не обязательно простого, определена безопасная зона $[l; r-x]$ такая, что если первое число кратное x на отрезке $[l; r]$ попадает в безопасную зону, то второе число кратное x гарантированно попадает в отрезок $[l; r]$.

Пусть надо проверить число p. Найдем его первое кратное на отрезке $[l; r]$. Если первое кратное попадает в безопасную зону, то проверяем p на простоту. Иначе заметим, что первое число кратное p представимо в виде $fm = p \cdot k$, для некоторого целого k. Тогда, следующее по убыванию число, которое надо проверить, это $p-1$. Рассмотрим число $(p-1) \cdot k = p \cdot k - k$. Тогда, либо оно является первым кратным на отрезке для $p-1$, либо выходит за левую границу отрезка. Во втором случае просто пересчитаем первое кратное, а в первом случае можно поступить так: при уменьшении p на 1 увеличим безопасную зону на 1, а первое кратное (если не выходит за границу отрезка) уменьшим на k. Тогда вычислим первый момент, когда какое-то число вида $p-i$ попадет в безопасную зону. Обозначим правую границу безопасной зоны за $last$. Тогда $fm - k \cdot i \leqslant last + i$, откуда $i \geqslant \frac{fm-last}{k+1}$. Это позволяет нам быстро пропускать все числа, которые не попадают в безопасную зону и заведомо не имеют хотя бы двух кратных на отрезке.

Пусть теперь дан отрезок длины хотя бы 10^6 (для меньших длин можно использовать, например, предыдущий метод). Тогда можно считать, что все интересующие нас простые также порядка 10^6, потому что всегда существует ответ порядка $\frac{10^6}{2}$. Первое число кратное любому числу x такого порядка будет иметь вид $x \cdot k$, где $k \leqslant 10^3$. Но тогда:

1. При каждом выходе за левую границу отрезка при пересчете кратного имеем число порядка $l - 1000$, так как максимальный множитель у кратного равен 10^3. Тогда для пересчета первого числа кратного p достаточно прибавить один раз само p, так как оно порядка 10^6.

2. Если ответ не будет найден в самом начале, то при подсчете первого момента попадания в безопасную зону будет за $\mathcal{O}(1)$ пропущено $\frac{10^6}{10^3} = 10^3$ заведомо неподходящих чисел. После десяти таких итераций безопасная зона будет порядка 10^4, поэтому в нее попадет порядка десяти чисел-кандидатов, среди которых найдется простое.

Задача F. Расстояние между точками

Данная задача является модификацией классической задачи о поиске двух ближайших точек среди n точек заданных на плоскости.

У классического варианта задачи есть множество решений. Одно из самых популярных решений основано на принципе разделяй и властвуй и работает за $\mathcal{O}(n \log n)$. Еще есть рандомизированное решение с ожидаемым временем работы $\mathcal{O}(n)$. Эти решения уже стали классикой и их можно найти на различных ресурсах по запросу «Задача о паре ближайших точек».

Решения классической задачи работают и в данной модификации. Нужно только следить за тем, чтобы две точки, расстояние между которыми мы рассматриваем в качестве кандидата на ответ, были из разных множеств.

При этом, не обязательно было знать классическую задачу для решения данной. Можно было придумать идею решения, в котором для каждой точки просматриваются в каком-то смысле ближайшие к ней точки из другого множества. Ближайшие точки можно было подбирать исходя из различных эвристик, например, брать точки, которые близки по одной из координат.