Задача A. Площадь

Давайте подумаем, как будет выглядеть фигура с максимальной площадью, если мы можем использовать линейку сколь угодно большое количество раз.

Нетрудно понять, что среди всех выпуклых многоугольников с фиксированным периметром наибольшую площадь имеет тот, который по форме стремится к окружности. Чем больше вершин мы можем добавить, тем ближе наш выпуклый многоугольник к окружности.

Кроме того, из естественного жадного соображения следует, что в оптимальной конфигурации все стороны должны быть равны, то есть многоугольник должен быть правильным.

Следовательно, нам достаточно посчитать площадь выпуклого правильного k-угольника.

Задача B. Почти палиндром

Давайте сначала посчитаем количество позиций, в которых символы не совпадают относительно центра строки. Иными словами, посчитаем число пар $(i, n-i-1)$, для которых $s[i] \neq s[n-i-1]$. Обозначим это количество через X. Таким образом, мы понимаем, каким X-палиндромом сейчас является наша строка.

Если $X = K$, то минимальное количество операций уже равно 0, и мы просто выводим исходную строку.

Если $X > K$, то нам необходимо сделать некоторые несовпадающие пары совпадающими. Количество таких изменений должно быть ровно $X - K$, чтобы в итоге осталось ровно K несовпадающих пар.

С минимальным количеством изменений мы разобрались. Теперь нужно понять, какие именно символы менять, чтобы итоговая строка была лексикографически минимальной.

Рассмотрим все пары $(i, n-i-1)$ и будем идти слева направо. Если $s[i] > s[n-i-1]$, то выгодно заменить $s[i]$ на $s[n-i-1]$. Такое жадное действие уменьшает строку лексикографически, так как мы уменьшаем более левый символ.

Если же таких замен недостаточно (например, во всех несовпадающих парах выполнено $s[i] \leqslant s[n-i-1]$), то после прохода слева направо можно рассмотреть пары в обратном порядке — начиная от центра к началу строки. Теперь, если $s[i] < s[n-i-1]$, мы будем заменять $s[n-i-1]$ на $s[i]$. Это уже не улучшает строку в самой левой позиции, но позволяет выполнить необходимое количество изменений, сохраняя лексикографический порядок минимальным среди допустимых вариантов.

В случае $X < K$ решение строится аналогичным образом, однако теперь нам, наоборот, нужно увеличить количество несовпадающих пар, аккуратно выбирая позиции, чтобы итоговая строка оставалась лексикографически минимальной среди всех допустимых.

Задача C. Взаимно простой зайчик

Будем считать динамику $dp[i][mask]$, где

• i — номер лужайки, до которой мы допрыгнули,

• $mask$ — битовая маска, хранящая информацию о том, какие простые числа уже использовались в наших прыжках,

• $dp[i][mask]$ — количество способов добраться из 1 в i с соответствующим набором использованных простых.

Например, если из стартовой клетки мы совершаем прыжок длины 20, то множество использованных простых изменяется так: {} -> {2, 5}, поскольку $20 = 2^2 \cdot 5$.

Однако если хранить информацию обо всех простых числах, асимптотика станет слишком большой. Заметим важное ограничение: нам неинтересно, использовали ли мы простые числа больше 50. Дело в том, что не существует двух длин прыжков A и B, таких что:

• A делится на простое число больше 50,

• B делится на простое число больше 50,

• и при этом $A + B \leqslant 100$.

Следовательно, простые числа больше 50 могут встретиться не более одного раза и не влияют на совместимость прыжков — их можно не учитывать в маске.

Простых чисел до 50 всего 15, поэтому размерность динамики будет: $n \cdot 2^{15}$.

Если реализовать переходы аккуратно, добившись суммарной сложности порядка $O(n \cdot 2^{15})$ (или $O(n)$ переходов на состояние при предварительной обработке), этого достаточно для прохождения всех тестов.

Задача D. ГЦД везде ГЦД!!

Давайте считать массив $cnt[i]$ — количество пар чисел, таких что их НОД равен i. Если мы посчитаем этот массив, то с помощью префиксных сумм и бинарного поиска сможем отвечать на каждый запрос за $O(\log n)$.

Разберёмся, как считать массив cnt. Покажем это по индукции. Так как все элементы массива не превосходят 10^6, в качестве базы возьмём максимальное возможное значение.

База индукции. Количество пар с НОД, равным 10^6, равно количеству пар элементов, где каждый из них делится на 10^6. Это значение можно напрямую посчитать.

Переход. Предположим, что для всех чисел $j > i$ мы уже правильно посчитали $cnt[j]$. Теперь хотим найти $cnt[i]$.

Сначала посчитаем количество пар чисел, которые делятся на i. Но среди этих пар есть такие, у которых НОД на самом деле больше i. Это как раз те пары, которые уже учтены в $cnt[x]$ для всех $x > i$, где x делится на i.

Значит, чтобы получить $cnt[i]$, нужно из количества пар, делящихся на i, вычесть все такие $cnt[x]$.

Количество чисел, делящихся на d, можно найти простым циклом: идти от d до максимального значения с шагом d и накапливать частоты.

Суммарная сложность таких циклов для всех d от 1 до n равна $O(n \log n)$, так как это по сути гармонический ряд.

После подсчёта массива cnt нам остаётся построить по нему массив префиксных сумм и отвечать на каждый запрос бинарным поиском за $O(\log n)$.

Задача E. Неужели задача не на ТЧ?!

Это сложная задача, поэтому разбор будет в формате ключевых идей.

Случай без изменений

Сначала представим, что мы не можем изменять рёбра. Нам нужно найти кратчайший путь от стартовой вершины до конечной так, чтобы красивости рёбер на пути не убывали.

Это можно переформулировать следующим образом. Будем запускать Дейкстру, но рассматривать рёбра в порядке возрастания красивости.

Идея такая:

- сначала рассматриваем только рёбра с красивостью 1 и делаем релаксации;
- затем добавляем рёбра с красивостью 2;
- затем с красивостью 3;
- и так далее.

На каждом шаге мы поддерживаем массив dist[i] — кратчайшее расстояние от стартовой вершины до вершины i, при условии, что мы использовали только уже рассмотренные красивости.

Когда мы обрабатываем очередную группу рёбер с одинаковой красивостью, в запуске Дейкстры участвуют только вершины — концы этих рёбер. Таким образом суммарная сложность остаётся аналогичной базовому случаю Дейкстры.

В конце, после обработки всех рёбер, найденные расстояния гарантированно удовлетворяют ограничению на неубывание красивостей: мы никогда не могли пройти по ребру с меньшей красивостью после ребра с большей, потому что обрабатывали их строго по порядку.

Расстояния в обратную сторону

Теперь дополнительно посчитаем аналогичные расстояния до финиша.

Для этого:

1. Развернём все рёбра графа.

2. Заменим красивость каждого ребра на «бесконечность минус старая красивость».

После такой замены порядок рёбер инвертируется: если раньше у ребра A красивость была меньше, чем у ребра B, то теперь наоборот.

Это позволяет тем же самым кодом снова запустить алгоритм — теперь из конечной вершины — и получить расстояния до неё, при условии монотонности (которая в исходном графе соответствует невозрастанию красивостей).

Теперь разрешим одно изменение

Пусть теперь мы можем изменить одно ребро.

Переберём все рёбра — всего их m.

Для каждого ребра считаем, что именно его мы хотим изменить, и посмотрим, как это повлияет на ответ.

К этому моменту у нас уже есть:

• расстояния от старта до всех вершин при неубывающих красивостях;

• расстояния от финиша до всех вершин (в перевёрнутом графе).

Дальше действуем так:

Для текущего ребра смотрим на его концы. Выбираем ту вершину, у которой меньше степень инцидентности, и перебираем все рёбра, инцидентные именно ей.

Почему так? Чтобы суммарная сложность была меньше — мы всегда итерируемся по меньшему списку.

Текущее перебираемое ребро — это то, которое мы хотим изменить. Перебирая рёбра из вершины меньшей степени, мы фактически понимаем, в какую красивость выгодно изменить текущее ребро.

После этого остаётся:

• взять второй конец исходного ребра,

• в его списке (где уже отсортированы рёбра по красивости) бинарным поиском найти подходящую позицию,

• пересчитать возможный новый ответ, используя уже посчитанные расстояния.

Таким образом мы перебираем все варианты изменения одного ребра и выбираем оптимальный ответ.