Задача A. Эх, было время...

Автор задачи: Михайлов Глеб.

Шаг 1: Группируем палочки по длинам — используем словарь для подсчета количества палочек каждой длины.

Шаг 2: Для каждой длины вычисляем, сколько полных наборов по 12 палочек мы можем собрать.

Шаг 3: Суммируем результаты по всем длинам.

Сложность решения: $O(n)$.

Задача B. Букашки-таракашки

Автор задачи: Кузляев Никита.

Букашек и таракашек в совокупности будем называть «зверями».

В случае, когда $n+m$ является нечетным числом, нужно сразу вывести «No» — равные команды сделать нельзя. Далее рассматривается случай, когда $n+m$ является четным числом.

Утверждение «букашка i хочет быть в команде с букашкой j» связывает букашек i и j в одну компоненту — это значит, что если мы размещаем или перемещаем букашку i в какую-либо команду, то вместе с ней перемещаются все букашки, находящиеся с ней в одной компоненте. В частности, если есть разбиение всех зверей Макара на две равные команды, то в каждой команде каждая компонента присутствует полностью. Для таракашек, что никак не связаны с букашками, ситуация аналогичная.

Таким образом, мы манипулируем не отдельными букашками и таракашками, а их компонентами. Пусть мы имеем p компонент с размерами $w_1, w_2, \ldots, w_p$. Тогда для того, чтобы сказать, можно ли распределить n букашек и m таракашек на две равные команды, нужно определить, можно ли выбрать из w некоторое подмножество с суммой $\frac{n+m}{2}$ (тогда невыбранные w_i в сумме будут тоже давать $\frac{n+m}{2}$, и у нас будет две равные команды) — это является классической задачей «о рюкзаке», решаемой методом динамического программирования.

Остается вопрос, как найти число компонент и их размеры — для этого мы можем использовать *dsu* (систему непересекающихся множеств) в самом классическом ее понимании с методами *unite* для объединения компонент (множеств) и *find* для нахождения «представителя» компоненты (более подробно с алгоритмом вы можете ознакомиться самостоятельно).

Мы находим все компоненты, зная их размеры, делаем «рюкзак» — эта задача объединяет две различные идеи в их самом каноничном виде. Ограничения на n и m допускали неточности и свободу в реализации *dsu* и *рюкзака* — были доступны неэффективные реализации *dsu* за $n^2 + m^2$, однако задача осталась *не открытой*.

Сложность решения: $O(t^2 + t \cdot \log t)$, где $t = (n + m)$.

Задача C. Ноль, Целковый, Полушка, Подушка...

Автор задачи: Кузляев Никита.

Будем решать задачу рекурсивно: пусть есть некоторая *волшебная* функция $f(n)$, которая принимает число n и возвращает ответ (сумму всех представлений), но, забегая вперед, еще возвращает, сколько различных представлений вообще есть, то есть функция возвращает пару чисел (s, c).

Есть базовые случаи — когда $n \leqslant 9$, тогда $f(n) = (n, 1)$.

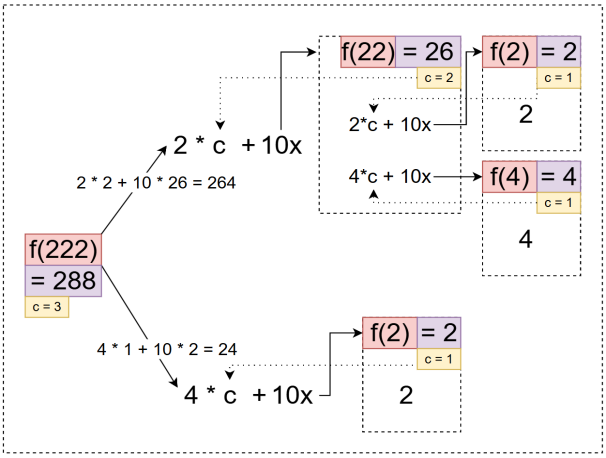
А в общем случае внутри функции будем рассматривать число n с младшего разряда: если число n **не** оканчивается на «22», то посчитаем $(s, c) = 10 \cdot f(\lfloor \frac{n}{10} \rfloor)$ — мы получили c различных представлений числа $\lfloor \frac{n}{10} \rfloor$ с общей суммой s, и для каждого такого представления мы учитываем последнюю цифру числа $(n \mod 10)$ — то есть мы должны добавить $(n \mod 10) \times c$ к s и вернуть

новую пару (s, c). Таким образом, если число n в своей десятичной записи не содержит подстроки «22» вообще, то $f(n)$ вернет то же самое число $s = n$ и единственное представление $c = 1$.

Если же в какой-то момент мы обнаружили, что в вызов функции было передано число n, оканчивающееся на «22», то мы имеем выбор:

- **Не заменять** «22» на «4». Тогда все так же вызываем и получаем $(s_1, c_1) = 10 \cdot f(\lfloor \frac{n}{10} \rfloor)$, а потом добавляем $(n \mod 10) \times c_1$ к s_1;
- **Заменить** «22» на «4». Тогда вызываем и получаем $(s_2, c_2) = 10 \cdot f(\lfloor \frac{n}{100} \rfloor)$ — обратите внимание, что деление на 100, так как мы берем сразу две цифры числа, а потом добавляем $4 \times c_2$ к s_2;

И из такого вызова мы должны вернуть $(s_1 + s_2, c_1 + c_2)$.



Небольшая визуализация вызовов функции.

Оценка сложности:

Сложность зависит от длины числа в десятичной записи. Пусть T_i — число вызовов, которые сделает вызов $f(n)$ с таким n, что оно имеет i знаков в десятичной записи. База: $T_0 = 1, T_1 = 1$ — в самих вызовах у нас нет тяжелых операций, поэтому их считаем константными. Далее, при $i > 1$ имеем: $T_i = T_{i-1} + T_{i-2}$ — мы можем вызваться дважды, когда число оканчивается на «22». Последовательность T_i образует числа Фибоначчи.

Сложность решения: $O(\phi^n)$.

Задача D. Не баг, а фича!

Автор задачи: Михайлов Глеб.

Напомним: изначально в коде B багов. За одну посылку Константин исправляет A багов, после чего (если баги остались) появляются ещё C новых. Рассмотрим процесс исправления багов.

Если $B = 0$, то багов нет, и ответ равен 0.

Если $A \geqslant B$, то все баги уйдут за одну посылку, и новых не добавится — ответ «1».

Если $A \leqslant C$ и при этом $B > A$, то количество багов не уменьшается (а при $A < C$ даже растёт), поэтому избавиться от них невозможно — ответ «-1».

Во всех остальных случаях $A > C$ и $B > A$. После каждой посылки (кроме, возможно, последней) число багов уменьшается на $(A - C)$. На последней посылке новые баги не появляются, поэтому минимальное число посылок k удовлетворяет неравенству:

$$B - (k - 1)(A - C) - A \leqslant 0,$$

откуда

$$k \geqslant \frac{B - C}{A - C}.$$

Следовательно,

$$k = \left\lceil \frac{B - C}{A - C} \right\rceil.$$

Сложность решения: $O(1)$.

Задача E. Проклятие Кассандры

Задача предложена «Центробанк России».

Используем метод двух указателей и структуру данных «словарь» (*dict* в Python и *map* в C++) для хранения количества каждого символа. Создадим два указателя: правый указатель будет двигаться по строке, увеличивая частоту символов, а левый указатель будет двигаться только тогда, когда какой-то символ превышает по частоте k — при сдвиге левого указателя будет уменьшаться частота. В процессе вычисляем максимальную длину диапазона как правый указатель минус левый плюс один.

Сложность: $O(n)$ — каждый символ обрабатывается дважды.

Задача F. Да где эта 105-я аудитория!?

Автор задачи: Кузляев Никита.

В этой задаче можно было применить рекурсивный обход с некоторыми отсечениями вариантов, так как размеры матрицы очень малы (n, m не превышают 10) — в целом, была возможность поиграться, и задача позволяла писать неоптимальные решения и многое *могло пройти*.

Но самый проверенный и напрашивающийся способ решения тут (опять же: это достаточно «классическая» задача) — обход в ширину или *bfs*: поддерживаем очередь непосещенных клеток и еще для каждой клетки матрицы запоминаем за сколько ходов мы в нее пришли. Добавляем в очередь стартовую клетку, отмечаем ее как посещенную за 0 ходов. Далее, пока очередь не пустая, берем первый элемент из нее – клетку (x, y), смотрим на соседние непосещенные с ней (какие клетки соседние определяется из типа клетки — их два вида), отмечаем их как посещенные за $p + 1$ ходов, где p — число ходов до клетки (x, y), и закидываем в очередь.

Если очередь опустела, а нижняя правая клетка не была посещена — выводим «-1», иначе, когда в процессе работы с очередью мы попадаем в правую нижнюю клетку, печатаем число ходов до нее.

Сложность решения: $O(n \times m)$.

Задача G. Простой и ровный парень

Автор задачи: Михайлов Глеб.

Шаг 1: Предвычислим все простые числа до максимального N из введенных данных с помощью решета Эратосфена.

Шаг 2: Для каждого числа n_i перебираем все возможные p от 2 до $n_i - 2$. Для каждого проверяем, является ли p простым, и является ли $q = n_i - p$ простым.

Как только найдена первая подходящая пара (p, q), выводим ее и завершаем программу и переходим к проверке следующего числа.

Сложность решения: $O(N \cdot loglogN + \frac{m \cdot N}{2})$. Может показаться, что использование двух указателей при поиске числа n_i в разложение простых будет занимать до $\frac{n_i}{2}$, но на практике меньшее из двух слагаемых находится сильно быстрее.

Задача H. Тэк, ну тут единички посчитать, да?

Автор задачи: Кузляев Никита.

При заданных ограничениях на длины строк, а они очень небольшие, мы можем перебрать все возможные сдвиги каждого из колец и проверить, что хотя бы в одной комбинации сдвигов замок закрыт. В качестве небольшой оптимизации мы можем не перебирать сдвиги одного из колец, например, внешнего.

Заметим, что идея о том, что если суммарное число закрытых сегментов 8 или более, то замок можно закрыть, неверна (можете придумать пример).

Сложность решения: $O(n^3)$, где n — длина строк (в задаче $n = 8$).

Задача I. Манипуляция: искусство массового обмана.

Автор задачи: Кузляев Никита.

В задаче не требуется ничего кроме как повторить все перемещения колпачков. Простите, это весь разбор: задача слишком простая.

Сложность решения: $O(n)$.

Задача J. ПОСВЯЯЯТ

Автор задачи: Михайлов Глеб.

Задача сводится к распределению p рюкзаков (время t_p) и b сумок (время t_b) между двумя охранниками так, чтобы минимизировать максимальное время работы любого из них.

Применим бинарный поиск по ответу T. В функции проверки `can_finish`(T) перебираем, сколько рюкзаков p_1 проверяет первый охранник ($0 \leqslant p_1 \leqslant \min(p, \lfloor T/t_p \rfloor)$). За оставшееся время он может проверить до $b_1 = \min(b, \lfloor (T - p_1 \cdot t_p)/t_b \rfloor)$ сумок. Тогда второму охраннику остаётся проверить $p - p_1$ рюкзаков и $b - b_1$ сумок, что требует времени $(p - p_1) \cdot t_p + (b - b_1) \cdot t_b$. Если оно $\leqslant T$, то время T достижимо.

Сложность решения: $O\left(m \cdot p \cdot \log(p \cdot t_p + b \cdot t_b)\right)$.

Задача K. Круг подозрений

Задача предложена «Центробанк России».

Идея состоит в том, чтобы перебрать все тройки i, j, k и проверить по матрице, что существуют ребра из i в j, из j в k и из k в i — но в таком виде это будет работать за $O(n^3)$, что не уложится по времени.

Будем использовать структуру данных *битсет* (к сожалению, на Python эта задача не сдается совсем) — операции с битсетом оптимизированы на низком уровне и, например, вычисление поразрядной конъюнкции всех битов (пересечения) двух битсетов длины n будет работать за $O(n/w)$, где w — размер машинного слова (зависит от процессора, в тестирующей системе $w = 64$).

Зачем нам битсеты? — Создадим для каждого из n людей два битсета: out и in. Тогда бит $out_{i,j}$ будет установлен, если человек i следил за j (ребро от i к j), а бит в $in_{i,j}$ будет установлен, если за человеком i следил человек j (ребро в i из j). Пройдем по матрице и заполним out и in.

Продолжим изначальную идею: но переберем i и j, предположим есть ребро из i в j — это легко проверить по матрице или битсетам (бит в $out_{i,j}$ должен быть установлен). Тогда если есть какой-то человек k, что образуется цикл, пересечение in_i и out_j должно иметь хотя бы один установленный бит. Как раз вот тут работает оптимизация, описанная ранее: пересечение битсетов и проверка, что установлен хотя бы один бит за $O(n/64)$ для каждой пары i и j.

Сложность решения: $O(n^3/64)$, что нормально заходит на *cpp* с установленными ограничениями.

Задача L. Мы следим!

Автор задачи: Михайлов Глеб.

Сразу заметим, что задача сводится к геометрическому вопросу: какое максимальное число точек можно покрыть сектором угла R с вершиной в начале координат. Наблюдение за углами позволяет свести задачу к одномерной — на окружности. Существует эффективный способ решить ее за линейное время после сортировки.

Шаг 1: Для каждой точки (x, y) вычислим ее полярный угол θ относительно оси Ox с помощью функции $\texttt{atan2}(y, x)$. Переведём угол из радиан в градусы и нормализуем к диапазону $[0, 360)$.

Шаг 2: Отсортируем все полученные углы по возрастанию. Это позволяет рассматривать точки в порядке их расположения на окружности.

Шаг 3: Чтобы корректно обрабатывать сектора, пересекающие границу $0°$ (например, от $350°$ до $20°$), продублируем все углы, добавив к каждому $+360°$. Получим расширенный массив длины $2n$.

Шаг 4: Применим метод двух указателей (sliding window) к расширенному массиву: будем поддерживать окно, в котором разность между крайними углами не превышает R. Максимальная длина такого окна и будет ответом.

Сложность решения: $O(n \cdot logn)$.