Задача A. Зеленый чай

Проведем ряд преобразований:

$$t_1 \cdot v_1 + t_2 \cdot v_2 = t_3 \cdot (v_1 + v_2)$$

$$t_1 \cdot v_1 + t_2 \cdot v_2 = t_3 \cdot v_1 + t_3 \cdot v_2$$

$$t_1 \cdot v_1 - t_3 \cdot v_1 = t_3 \cdot v_2 - t_2 \cdot v_2$$

$$v_1 \cdot (t_1 - t_3) = v_2 \cdot (t_3 - t_2)$$

$$\frac{v_1}{v_2} = \frac{t_3 - t_2}{t_1 - t_3}$$

учитывая, что $t_3 = 80$, получим:

$$\frac{v_1}{v_2} = \frac{80 - t_2}{t_1 - 80},$$

подставив t_1 и t_2 из условия, поучим решение $v_1 = 80 - t_2$, $v_2 = t_1 - 80$. Чтобы получить минимальное, нужно эту дробь сократить, то есть разделить v_1 и v_2 на их наибольший общий делитель.

Задача B. Загадочные резисторы

Обозначим номинал загадочных резисторов как x, тогда суммарное сопротивление цепи будет

$$R' = \sum_{i=1}^{k} \frac{x \cdot r_i}{x + r_i}$$

заметим, что если x возрастает, то и общее сопротивление R' будет возрастать, а если x убывает, то и R' бужет убывать.

Найти x, такой что $R = R'$ можно любым численным методом, например дихотомией.

Задача C. Смайлики

Подсчитаем вхождения каждого символа во входной строке. Заметим, что большинство смайликов содержат уникальные символы. Например, «[:|||:]» («баян») единственный содержит квадратные скобки, поэтому «баянов» будет столько, сколько открывающих квадратных скобок. Выводим эти «баяны», и корректируем счетчики. Для решения достаточно обрабатывать смайлики в последовательности обратной к той, что приведена в условии.

Единственной сложность окажутся первые 4 смайлика: сначала нужно сгенерировать все, что содержат «;», но при этом учитывать наличие соотетствующих скобок. То есть, сначала генерируем «;-)» пока хватает закрывающих скобок и точек с запятой, затем на оставшиеся «;» генерируем «;-(». Оставшиеся скобки объединяем с двоеточиями в любом порядке.

Задача D. Power play

Для начала, найдем пределы, в которых может лежать x. Заметим, что равенство $a^x = x^b$ имеет не более двух решений на интервале $[1, +\infty)$. Эти решения — точки пересечения функций $f(x) = a^x$ и $g(x) = x^b$. Обе функции монотонно растут при $x > 0$ и $a, b > 1$. Экспонента растет быстрее степенной функции, а в точке 0 выполняется, $f(0) > g(0)$, так как при $x > 1 : a^0 = 1$, $0^b = 0$. Если экспонента очень крутая, то точек пересечения не буде вообще, если более пологая, то при большом b функция $g(x) = x^b$ может временно ее обогнать — будут 2 точки пересечения.

Возьмем, по ограничениям задачи, самую пологую экспоненту 2^x и самую крутую степенную функцию x^{10000}, обнаружим что пересечение будет находиться в районе точки $x \approx 174096$. Следовательно ограничение 10^{18} несколько избыточно ;-).

Численное решение

Если попытаться решить задачу численно, то в не преобразованном виде, нам придется работать с двумя очень быстро растущими функциями, которые будут быстро переполнять любые типы чисел с плавающей точкой.

Поэтому, прологарифмируем обе части уравнения:

$$log_a(a^x) = log_a(x^b)$$

$$x = b \cdot log_a(x)$$

$$x = b \cdot \frac{ln(x)}{ln(a)}$$

Чтобы решить данное уравнение достаточно найти, например трисекцией, минимум функции $x - b \cdot \frac{ln(x)}{ln(a)}$ на интервале $[1, +\infty)$. Пусть это будет точка x_{min}. Тогда искомые корни будут лежать на интервалах $[1, x_{min})$ и $(x_{min}, +\infty)$. Где они могут быть найдены методом половинного деления. Из полученных корней отбираем только те, что лежат достаточно близко к целым числам и округляем.

Полученные целочисленные корни обязательно следует проверить, так как существуют тесты, на которых вещественные корни уравнения отстоят от целой точки менее чем на 10^{-8}.

Решение из теории чисел

Разложим a на простые множители:

$$a = p_1^{t_1} \cdot p_2^{t_2} \cdot p_3^{t_3} \cdot ...$$

При возведении его в степень x получим:

$$a = p_1^{t_1 \cdot x} \cdot p_2^{t_2 \cdot x} \cdot p_3^{t_3 \cdot x} \cdot ...$$

Аналогично поступим с числом x,но тогда из равенства $a^x = x^b$ следует, что и a и x состоят из одних и тех же простых чисел, причем в одинаковом соотношении. Найдем наибольший общий делитель g для чисел $t_1, t_2, t_3, ...$ - степеней простых чисел в разложении числа a, и построим число

$$c = p_1^{t_1/g} \cdot p_2^{t_2/g} \cdot p_3^{t_3/g} \cdot ... = \sqrt[g]{a},$$

тогда можно утверждать что $x = c^k$, где k - некоторое целое число.

Будем последовательно перебирать и проверять степени c^k до тех пор, пока $c^k < 174096$.

Решение перебором x

Можно просто перебрать все x от 2 до 174096, если уметь быстро проверять равенство $a^x = x^b$. Для такой быстрой проверки воспользуемся китайской теоремой об остатках: сгенерируем массив простых чисел, и будем проверять равенство по модулю каждого числа. Если оно выполняется для некоторого x по модулю всех чисел, то оно выполняется и без модуля.

Для однозначности такой проверки необходимо, чтобы произведение всех чисел было больше 174096^{10000}, а это затруднительно, однако если удовлетворится проверкой с достаточной долей вероятности то хватит приблизительно 20 простых чисел близких к 10000 (чтобы можно было использовать integer). Такой подход можно использовать и при проверке числа x в других решениях.

Задача E. Печатная плата

В теории графов эта задача называется «Покрытие графа минимальным числом независимых путей», и имеет классическое решение:

1). определим степени всех вершин (степень вершины - количество инцидентных ей дуг)

2). найдем все вершины нечетной степени - их всегда будет четное число

3). соединим попарно нечетные вершины дополнительными дугами

4). в достроенном графе найдем Эйлеров цикл (или циклы, если граф не связный)

5). удалим добавленные ранее дуги - оставшееся множество путей и будет покрытием графа путями

На практике, вместо добавления дуг достаточно учитывать возможность дополнительного перехода из нечетной вершины. Для построения Эйлерова цикла можно использовать алгоритмы с очередью или со стеком (модифицированный обход в глубину). Алгоритм со склеиванием отдельных циклов реализовать можно, но сложно будет учесть все отдельные случаи.

Дополнительно нужно учесть и обработать следующие ситуации:

1). не связный граф

2). вложенные, но не связанные циклы

3). начальная точка «*» рядом с не связанным с ней символом, например «...*|...»

4). пути, замыкающие цикл

5). некоторые точки обозначенные как «*» могут исчезать

Некоторую сложность может представлять также перевод входных данных в граф и вывод результата в виде цепочек команд на перемещение.

При рекурсивной реализации, глубина рекурсии может достигать 10000, что может приводить к переполнению стека.

Задача F. Игра в слова

Подсчитаем количества различных букв в исходном слове, запишем их, и отсортируем, нули выкинем - назовем такой отсортированный вектор состоянием игры. В процессе игры, ходы игры будут следующими:

- удаление из состояния одного числа - аналог стирания всех букв одного вида
- уменьшение одного из чисел больших 1 (если уменьшать 1, то получим предыдущий ход) на 1 - аналог стирания одной буквы
- уменьшение одного из чисел больших 2 (если уменьшать 2, то получим первый ход, а единицу на 2 уменьшить нельзя), на 2 - аналог стирания двух одинаковых букв.

Состояния, для которых однозначно можно опередить, что ходящий побеждает (выигрышное) или проигрывает (проигрышное):

- состояние из одной цифры (много одинаковых букв) - выигрышное
- состояние из одних единиц (все буквы разные) - зависит от четности количества единиц - нечетное - выигрышное, четное - проигрышное
- состояние, когда цифры состояния можно разбить на пары равных цифр, (например (3, 3, 2, 2, 1, 1) разбивается на пары (3, 3), (2, 2) и (1, 1)) - проигрышное, так как для противника существует симметричный ход

По общим правилам теории игр, состояние считается выигрышным, если из него есть хотя бы один ход в проигрышное. Если такого хода нет - состояние проигрышное.

Обратим внимание, что в процессе ходов сумма чисел в состоянии всегда уменьшается, значит можно перебрать все возможные ходы, и рекурсивно вычислить для заданного состояния, является ли оно выигрышным для Алисы или проигрышным.

Возможно также использование динамического программирования, так как количество возможных состояний не велико - каждое состояние отвечает некоторому разбиению числа на слагаемые. Таких разбиений не так и много, например, для 50 их всего 204 226.

Задача G. Песочные часы

Рассмотрим более простую задачу для двух песочных часов с номиналами t_1 и t_2.

Сразу отсечем случай, когда решения точно не будет - если номиналы песочных часов имеют наибольшиий общий делитель $g = GCD(t_1, t_2)$, и число K не кратно ему, то решения нет.

Если же решение есть, то разделим t_1, t_2 и K на g - получим задачу со взаимно простыми t_1 и t_2 решение которой даст нам решение исходной с точностью до последовательности переворачивания часов.

Теперь, рассмотрим случай взаимно простых t_1 и t_2, если $K > t_1 \cdot t_2$, то решение есть всегда(!), причем, оно состоит из нескольких одиночных переворачиваний сначала первых часов, а затем вторых.

Рассмотрим, остатки от деления на t_1 чисел $0, t_2, 2t_2, 3t_2, ..., (t_1-1)t_2$: так как числа t_1 и t_2 взаимно простые, то все остатки будут разными, и один из остатков $(r \cdot t_2) \bmod t_1$ совпадет с остатком $K \bmod t_1$, тогда $K - r \cdot t_2$ будет нацело делится на t_1. Таким образом, чтобы получить K нам потребуется сначала $(K - r \cdot t_2)/t_1$ раз перевернуть первые часы зетем r раз вторые. Так как $1 \leqslant t_1, t_2 \leqslant 20$, то найти r можно последовательным перебором чисел 0, 1, 2...

В случае, если $K < t_1 \cdot t_2$, решение может быть таким же как раньше, либо потребовать одновременного переворачивания часов. Так как K ограничено сверху, то найти (или не найти) нужную последовательность переворачиваний можно полным перебором.

В случае нескольких часов, перебор ограничивается сверху еще сильнее, так как можно с помощью динамического программирования найти все такие моменты времени, из которых можно достигнуть K, переворачивая разные часы поодиночке. Если при переборе мы достигли такого момента, то можно останавливаться - мы нашли решение.

Итак, решение задачи:

1). Ищем общий делитель g для номиналов часов, и проверяем, кратно ли K этому g - если не кратно, то сообщаем, что решения нет

2). С помощью динамического программирования строим множество моментов времени $\mathbf{T} = T_1, T_2, ...,$ из которых можно достигнуть K одиночными переворотами. Принцип построения прост: если мы можем достигнуть K из момента T_x, то сможем и из моментов $T_x - t_i$, где $i = 1..n$.

3). Если момент 0 принадлежит $\mathbf{T}$, то формируем решение с помощью одиночных переворотов часов

4). Перебираем все возможные комбинации переворотов часов, пока либо не дойдем до момента T_x, либо не закончим перебор.

5). Если в процессе перебора мы дошли до некоторого момента T_x, то строим решение, причем сначала используем одиночные перевороты от T_x до K, только потом, те сложные перевороты, которые найдены перебором. Такая последовательность обуславливается тем, что после сложных переворотов не все часы могут быть пусты.

Задача H. Мужская разборка

Формальное условие: Дано N палочек. Играют двое игроков. Каждый по очереди может вытянуть 1, 5, или 13 палочек. Тот, после чьего хода не останется палочек, проиграл. Оба играют оптимально. Кто выиграет?

В данной задаче есть 2 способа решения.

1 способ: метод выигрышных позиций (динамическое программирование). Вариант для ленивых.

2 способ: заметить, что первый игрок может выигрывать при четном количестве палочек, а второй – при нечетном.

Задача I. Андрюша и Python

Возьмем квадрат со стороной $m = 2^k \geqslant n - 1$. (Квадрат $(1, 1)$, (m, m))

Делим квадрат на 2 треугольника диагональю. Делаем запрос в треугольник, если точка в нём, то второй треугольник отбрасываем и продолжаем поиск в нужном треугольнике, в свою очередь

делим его пополам, повторяем запрос и так далее. Треугольники мы делим медианой к гипотенузе. На каждой итерации у нас прямоугольный равнобедренный треугольник. В конце мы получим треугольник со сторонами 1, 1, $\sqrt{2}$. Спросим про каждую точку треугольника. Таким образом, задачу можно решить не более чем за 60(если быть точнее за 57) запросов. Так как $m = 2^k$ мы сможем удобно делить квадрат пополам в целых точках.

Задача J. Что-то похожее на проблему Варинга

Пусть у нас есть многочлен $f(x)$ степени $k \geqslant 2$. Заметим, что $f(x+1) - f(x)$ имеет степень $k-1$. Пусть $f_i(x) = f_{i-1}(x+1) - f_{i-1}(x)$. Таким образом, мы можем из многочлена $f_1(x) = x^3$ получить многочлен $f_3(x) = ax + b$. Найдем вид $f_3(x)$.

$f_2(x) = f_1(x+1) - f_1(x) = (x+1)^3 - x^3 = 3x^2 + 3x + 1$.

$f_3(x) = f_2(x+1) - f_2(x) = 3((x+1)^2 - x^2) + 3(x+1-x) = 3(2x+1) + 3 = 6(x+1)$

$6(x+1) = f_3(x) = f_2(x+1) - f_2(x) = f_1(x+2) - f_1(x+1) - (f_1(x+1) - f_1(x)) = f_1(x+2) - 2f_1(x+1) + f_1(x) = = (x+2)^3 - 2(x+1)^3 + x^3$

Мы получаем формулу $6x = (x+1)^3 - 2x^3 + (x-1)^3$

Значит мы можем получить числа кратные 6 за 4 куба. Числа вида 6x + 1 и 6x + 5 можно получить прибавлением +- 1^3. Числа вида 6x + 2 и 6x - 2 можно получить прибавлением +- 2^3. Числа вида 6x + 3 можно получить прибавлением 3^3. Значит у нас есть представление любого числа за 5 кубов. Остается написать длинку с делением на 6, остаток по модулю 6 и +- маленькое число.

Задача K. Параболическая сортировка

Заметим следующий факт: в массиве, полученном с помощью параболической сортировки, максимальный элемент массива будет находиться либо на первом, либо на последнем месте. Сдвинем его в ту часть массива, которая потребует меньшее количество обменов. Таким образом, задача сводится к аналогичной меньшего размера, так как порядок остальных элементов массива не меняется. Быстро обмены можно проводить с помощью дерева отрезков и дерева Фенвика, поддерживающих операции нахождения значения элемента и обновления на отрезке. На позиции i будем хранить номер новой позиции элемента, который изначально был на позиции i. Пусть изначальная позиция нашего максимального элемента = pos. Если элемент сдвигается в левую часть массива, обновляем элементы дерева с 1 по pos на +1, в противном случае - с pos по n на -1. Заметим, что все элементы которые мы еще не рассмотрели будут иметь корректную новую позицию в нашей структуре данных. Итоговая сложность O(n log n)