Задача A. Анализ растительности

Если граф является деревом из X вершин, то он имеет $X-1$ ребро. Это доказывается, например, методом математической индукции: для $X = 1$ утверждение тривиально, пусть у нас есть граф из $N+1$ вершин и мы доказали для $X \leqslant N$, тогда уберём какое-то ребро; так как в графе нет циклов, то он распадётся на два графа из k и $N-k+1$ вершин, являющихся деревьями, по предположению индукции, в них $k-1$ и $N-k$ ребро, складывая и учитывая удалённое ребро, получаем N рёбер. Утверждение доказано.

Соответственно, для леса из K деревьев разность между суммой вершин и суммой рёбер равна K. Таким образом, количество деревьев однозначно задаётся разностью между M и N. Если $M \leqslant N$, то граф не может быть лесом, в противном случае количество деревьев равно $M - N$ (например, можно взять одно дерево из $N+1$ вершин и N рёбер и $M-N-1$ деревьев из одной вершины).

Задача B. Битовые последовательности

Так как число не превосходить 10^{19}, то его можно представить в виде 64-битного беззнакового целого. Дописав к двоичному преобразованию один ведущий 0, получаем строку из не более, чем 65 символов.

Поэтому работает самое простое решение: для каждой позиции можно определить длину самой длинной разнообразной строки, начинающуюся в этой позиции (не более, чем за 65 проверок); далее можно заметить, что имеет смысл рассматривать только позицию, находящуюся вначале, или позиции, на которых цифра совпадает с цифрой на предыдущей позиции (на остальных позициях строка является подстрокой более длинной строки, начинающейся слева, так как если цифры различны, то строку можно продолжить влево).

Таким образом, решение работает за линейное время относительно количества бит.

Впрочем, при ограничениях задачи в лимит времени укладывается и квадратичное решение с вычислением самой длинной строки для каждой позиции.

Задача C. Выгибание многоугольников

Если у нас есть правильный K-угольник с периметром N и целой стороной, то сторона однозначно определена и равна N/K, то есть число N/K должно быть целым.

Поэтому количество разны правильных K-угольников периметра N равно количеству целых делителей числа N, больших 2 (так как минимальное количество сторон в многоугольнике равно 3).

Заметим, что если x является делителем N, то и N/x является делителем N, при этом каждое такое число, квадрат которого строго меньше N, порождает два различных делителя (так как $x * x < N$, а $x * (N/x) = N$, то $x < N/x$, при этом все делители, квадрат которых строго больше N, будут сгенерированы (аналогичным образом показывается, что делителю y, квадрат которого строго больше N, соответствует делитель N/y, квадрат которого строго меньше N). Если N — полный квадрат, то к этим делителям добавляется число, равное квадратному корню из N.

Поэтому перебираем все целые положительные числа x, не превосходящие $\sqrt{N}$. Если N делится на x, то мы учитываем многоугольник с N/x сторонами, а также учитываем многугольник с x сторонами, если $x > 2$ и $x \neq N/x$.

Сложность решения по времени — $O(\sqrt{N})$.

Задача D. Генерация паролей

Будем решать задачу методом динамического программирования.

Заведём двумерный массив `dp` размером $N \times 4$. Первый параметр задаёт длину строки N, второй параметр — тип последней буквы в слове (гласная, обычная согласная отличная от «й», согласная «й», мягкий/твёрдый знак). Значение элемента массива — количество слов заданной длины, заканчивающихся на заданную букву.

В случае гласной количество слов равно количеству слов длины $i-1$, заканчивающихся не гласной, то есть суммой `dp[i-1][j]` для j от 1 до 3. Всего гласных 10, так что сумму надо умножить на 10.

В случае обычной согласной количество слов равно количеству слов длины $i-1$, заканчивающихся гласной, то есть `dp[i-1][0]`. Всего согласных, отличных от «й», 10, так что это число надо умножить на 20.

В случае «й» количество слов равно количеству слов длины $i-1$, заканчивающихся гласной, то есть `dp[i-1][0]`.

В случае мягкого/твёрдого знака количество слов равно количеству слов длины $i-1$, заканчивающемуся обычной согласной, то есть `dp[i-1][1]`. Так как знаков два, то это число надо умножить на 2.

Значения для строк длины 1 надо подсчитать явно (они равны, соответственно, 20, 10, 1 и 0).

Ответ будет равен сумме `dp[N][j]` по всем j от 0 до 3.

Задача E. Длительность выполнения

Сначала проверим, что строка является регулярным скобочным выражением и заодно для каждой скобки найдём позицию соответствующей ей скобки (для этого заведём вспомогательный массив).

Для проверки используем стек; если приходит открывающая скобка, кладём на стек пару из этой скобки и её номера в строке; если приходит закрывающая скобка, проверяем, какая скобка лежит на вершине стека. Если скобка не соответствует (или стек пуст) — выдаём ошибку на текущей позиции, если скобка соответствует — запоминаем для каждой из двух позиций позицию парной скобки в массиве и снимаем пару со стека.

Если в конце проверки стек пуст, то строка является регулярным скобочным выражением, так что можно подсчитать время выполнения.

С помощью вспомогательного массива это можно сделать рекурсивно: пусть у нас есть регулярное скобочное выражение. Если первая и последняя скобка этого выражения являются парными, его значение равно значению выражения без первого и последнего элемента плюс числовое значение соответствующей скобки. Иначе его значение равно произведению значений выражения от первой скобки до парной ей и остальной части выражения (которая также является правильной скобочной последовательностью).

Задача F. Есть четыре команды...

Рассмотрим произвольное состояние процессов. В каждом состоянии, кроме состояния «все процессы выгружены» и состояния «выгружены все процессы, кроме процесса с самым большим номером» существует два перехода в другое состояние: поменять состояние процесса, следующего за активным процессом с минимальным номером, и поменять состояние процесса с номером 1. В состоянии «все процессы выгружены» и в состоянии «выгружены все процессы, кроме процесса с самым большим номером» первое действие невозможно (в первом случае нет активных процессов, во втором процесса, следующего за активным, не существует), так что есть только один переход в другое состояние.

Кроме того, заметим, что любое действие является обратимым: повторное изменение состояния первого процесса приводит к предыдущему состоянию (так как остальные процессы не менялись), после изменения состояния процесса, следующего за минимальным, минимальный процесс остался тем же, а значит, повторное применение аналогичного действия приведёт к повторному изменению состояния того же самого процесса, то есть предыдущее состояние повторится.

Начнём изменение с состояния «выгружены все процессы». Тогда в силу двух доказанных выше утверждений каждый раз существует единственное действие, меняющее состояние на состояние, отличное от предыдущего.

Представим список процессов как двоичное число x, в котором младший бит соответствует первому процессу, а старший — последнему.

Рассмотрим функцию $f(x)$, в которой i-й бит задаётся побитовым XOR всех бит числа x, не старших i-го. Очевидно, что если $f(x) = f(y)$, то $x = y$ (действительно, старший бит $f(z)$ равен старшему биту z, то есть старшие биты x и y равны; далее замечаем, что каждый предыдущий бит зависит от соответствующего бита и старших, которые уже являются равными).

Покажем, что при последовательном применении операций от состояния «выгружены все процессы» так, чтобы каждый раз получалось состояние, отличное от состояния, предшествующего текущему, значение $f(x)$ всякий раз увеличивается на единицу.

Действительно, при первом переходе это верно (значение увеличивается с 0 на 1). Пусть это верно при всех предыдущих переходах и у нас есть какое-то число X и соответствующее ему значение $f(X)$. Возможные действия: поменять первый бит (тогда действие затрагивает только последний бит $f(X)$ и число увеличится на 1, если $f(X)$ было чётным или уменьшится, если $f(X)$ было нечётным) или поменять бит, следующий за младшей единицей (тогда нетрудно заметить, что $f(X)$ увеличится на 1 в случае, если $f(X)$ было нечётным и уменьшится на 1, если $f(X)$ было чётным).

Так как функция $f(x)$ является биекцией, то из этого следует, что каждое состояние можно получить из состояния «выгружены все процессы», то есть граф состояний является бамбуком, а значит, все состояния линейно упорядочены.

Поэтому для решения задачи достаточно найти абсолютную величину разности значений $f(x)$ от начального и конечного состояний системы.

Задача G. Ё-компьютер

Пусть у нас первое число a меньше второго числа b (если это не так, поменяем числа местами).

В случае, если $b - a < 10$, сложение можно выполнить напрямую.

Иначе рассмотрим процесс сложения. В сумму входят $(3 - a_0)$ чисел (здесь a_0 — это последняя цифра числа a), отличающихся от a только последней цифрой или не отличающихся от a вообще. Также в сумму входят $b_0 + 1$ чисел вида $b - i$, отличающихся от b только последней цифрой или не отличающихся от b вообще (здесь b_0 — это последняя цифра числа b). Остальные числа разбиваются на блоки по 3 числа с совпадающими первыми $n - 1$ цифрами и последней цифрой, пробегающей значения 0, 1, 2. Легко заметить, что поразрядная сумма по модулю 3 для каждого такого блока равна 0.

Соответственно, для получения суммы достаточно провести поразрядное сложение не более чем $3 - a_0 + b_0 + 1 = 4 + b_0 - a_0 \leqslant 7$ чисел и вывести полученную строку, начиная с самого левого символа, отличного от 0.

Задача H. Жёсткий фактчекинг

Заметим, что если у нас есть набор оригинальных высот H_i и масштабированная фотография $r \cdot H_i$, то последовательность отношений i-го элемента к $i+1$-му будет одинаковой для обоих наборов. Более того, если у нас есть последовательность h'_i, для которой последовательность отношений равна этой последовательности для H_i, то, умножив все элементы этой последовательности на H_1/h'_1, получим последовательность H_i.

Тем самым задача сводится к проверке того, присутствует ли заданная подпоследовательность рациональных чисел в качестве непрерывной подпоследовательности в последовательности H, для n таких последовательностей.

Если перейти к строкам над алфавитом из рациональный чисел, то задача сводится к многократному поиску подстрок в строке. Это известная задача, решаемая алгоритмом Ахо-Корасик за линейное время от размера входа.