Задача A. Влад A4

Просуммируем площади кусочков, получим площадь исходной фигуры. В задаче гарантируется, что бумага точно строится из кусочков, кусочки не поворачиваются, а одна из размерностей (ширина) зафиксирована и дана в инпуте. Поэтому корректно будет просуммировать и поделить полученную площадь на данную в инпуте ширину, таким образом получив длину.

Задача B. Зеркальное табло

Для того чтобы решить задачу, достаточно было сформировать новую строку, которую можно составить из входной строки, развернув ее задом наперед. Дополнительно в новой строке нужно было заменить символы 3 на E, а символ 2 — на 5. Это и будет строка, которая получится в зеркале. После этого необходимо было сравнить новую строку и входную строку на равенство. Если они не равны, то входная строка не зеркальная. Если же равны, то она зеркальная, если длина строки четная. А вот если не четная, то, для того чтобы строка была зеркальная, дополнительно нужно проверить, что центральный символ 0 или 8.

Задача D. 11 — не такое уж и простое число

Зафиксируем остаток по модулю 11, например 4. Будем искать все простые числа с этим остатком, то есть вида $p = 11 * k + 4$. Простых чисел много, такие найдутся. Так как у них у всех одинаковый остаток, то, взяв любые 11 из них и просуммировав, получим число большее 11, делящееся на 11, а значит составное. Док-во: $(11*k_1+4+11*k_2+4+\cdots+11*k_{11}+4) = 11*(k_1+k_2+\cdots+k_{11})+11*4 = 11*(k_1+k_2+\cdots+k_{11}+4)$ - делится на 11.

Задача E. Запретное слово

Построим строку без повторений вообще. Если в такой строке в качестве подстроки есть запретное слово, ответ - да, иначе - нет (в запретном слове нет подряд идущих одинаковых символов).

Поиск подстроки в строке — известная задача, можно искать за линию префикс функцией. Либо, если длина запретного слова мала, то искать подстроку в строке можно наивным алгоритмом (квадратичным).

Задача F. Бит-монеточки

Для решения задачи посчитаем сумму всех монет. У нас может быть 3 случая.

Если сумма всех монет делится на 3 нацело, тогда нужно выбрать монету с минимальным номиналом, делящимся на 3. Причем, если среди монет нет монеты с номиналом, делящимся на 3, выводим -1.

Если сумма всех монет делится на 3 с остатком 1, тогда нужно выбрать монету с минимальным номиналом, который делится на 3 с остатком 1. Причем, если среди монет нет монеты с номиналом, делящимся на 3 с остатком 1, выводим -1.

Если сумма всех монет делится на 3 с остатком 2, тогда нужно выбрать монету с минимальным номиналом, который делится на 3 с остатком 2. Причем, если среди монет нет монеты с номиналом, делящимся на 3 с остатком 2, выводим -1.

Сложность решения $O(n)$.

Сложность переборных решений $O(n^2)$, такие решения не пройдут все тесты

Задача G. Простые отрезки

Несложно доказать, что любой отрезок длины больше 2 из последовательных натуральных чисел дает составную сумму ($\sum_{i=L}^{R} i = (L+R)*(R-L+1)/2, (R-L+1) \geqslant 3, (L+R) \geqslant 6$). Значит, мы просто проверим, что наш отрезок длины не более 2, и попытаемся найти для него подходящий (за $O(1)$), проверяя простоту быстро ($O(1)$) с помощью решета Эратосфена.

Задача H. Сделай не меньше

Числа можно сравнивать по определению лексикографии, ищем максимальный префикс строки b такой, что я могу сделать его равным префиксу строки a, а следующий символ (если он есть) сделать больше.

В этой задаче много подводных камней и крайних тестов (корнер-кейсов). Но есть один важнейший идейный корнер кейс: не всегда можно сделать следующий символ больше (если в строке a находится символ '9'). Значит, придется попытаться сделать последний символ префикса больше, оставшиеся операции потратить на «зануление» суффикса.

Пример:

19456999 — строка a

19456118 — строка b

19457018 — строка c

Было разрешено 2 операции замены.

Теперь осталось грамотно реализовать описанный алгоритм, учитывая вышеизложенный корнер-кейс и ограничение на разрешенное число замен.

Задача I. Путь к Храму

Задача решается методом динамического программирования. Пусть $dp_{i,j}$ — количество лестниц при условии, что мы пока набрали i ступенек, последняя ступенька из i ларька — j-тая по возрастанию. Значит, следующая должна быть больше. Тогда мы обновим $dp_{i,j}$ через все такие $dp_{i-1,jj}$, что $h_{i-1,jj} < h_{i,j}$. Найдем с помощью бинарного поиска максимальный такой jj и обновим $dp_{i,j}$ за $O(1)$ с помощью префиксной суммы предыдущего слоя. Асимптотика $O(n * k * log(k))$

Задача J. Упаковка окружностей

Воспользуемся динамическим программирования. Обозначим за dp_i - минимальную длину ленты, необходимой для упаковки i окружностей. Тогда оптимальный ответ для dp_i можно получить из $dp_{i-1}, dp + i - 2, dp_{i-3}$. Другими словами

$$dp_i = min(dp_{i-1} + f(r_i), dp_{i-2} + g(r_i, r_{i-1}), dp_{i-3} + h(r_i, r_{i-1}, r_{i-2}))$$

где функции f, g, h означают длину ленты, необходимой для упаковки одной, двух и трех окружностей соответственно.

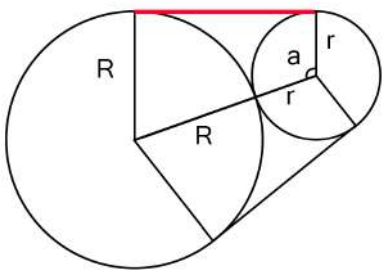
Тогда ответ на задачу будет лежать в dp_n, а $dp_0 = 0$.

Очевидно

$$f(r_i) = 2\pi \cdot r_i$$

Возьмем $r \leqslant R$, тогда $g(r, R)$ будет состоять из четырех частей

- Два отреза на касательных. Их длины равны $\sqrt{(R+r)^2 - (R-r)^2} = 2\sqrt{Rr}$;
- По дуге от каждой из окружностей. Длина дуги большей окружности - $2a \cdot R$, меньшей - $(2\pi - 2a) \cdot r$ (угол a можно найти из трапеции).



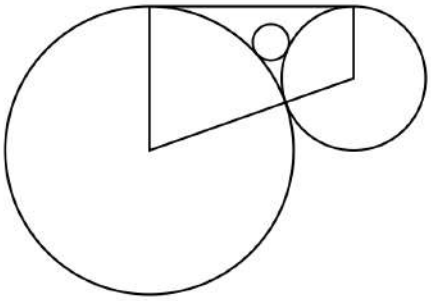
Тогда суммарная длина будет

$$g(r, R) = 2\left(\sqrt{Rr} + a \cdot R + (\pi - a) \cdot r\right)$$

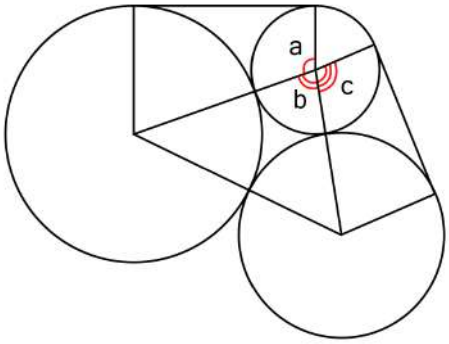
Аналогично будет считаться и для трех окружностей. $h(r_1, r_2, r_3)$ состоит из шести частей

- Три отрезка касательных;
- По дуге от каждой из окружностей.

Но, помимо этого, может получится следующее: если радиус маленькой окружности достаточно мал в сравнении с двумя другими, то случай с тремя окружностями сводится к случаю с двумя.



Чтобы отделить данный случай от остальных, можно найти углы a, b, c. Если их сумма $a+b+c > 2\pi$, то случай сводится к двум окружностям.



Углы a и c можно найти из трапеций, а угол b - из центрального треугольника, используя теорему косинусов.

Задача K. Целое посередине

Проверим сначала, что отрезок существует, для этого сравним дроби, дроби сравниваются посредством умножения числителя одной дроби и знаменателя другой дроби и сравнения этих значений. Если отрезок не существует, ответа нет.

Затем посчитаем первое целое число большее левой границы, это делается прибавлением к числителю первой дроби значения «знаменатель - 1» и целочисленным делением полученной дроби.

Если полученное целое число $\leqslant$ правой границы, то ответ есть, иначе нет.

Задача L. Пятновыводитель Ильфата

Для решения задачи используются предподсчитанные суммы (префикс суммы). Заведем массив s, где $s[i]$ — сумма всех элементов, начиная с 1-ого до i-ого.

Далее для каждого запроса проверяем, удовлетворяется ли условие: $\frac{s[r]-s[l-1]}{r-l+1} \leqslant V$, и выводим ответ на запрос. $s[r] - s[l-1]$ — сумма элементов с индексами с l по r, количество элементов в этом диапазоне $r - l + 1$, V — заданное значение.

Задача M. Палиндромная яма

Фиксируем позицию дна ямы и идем налево и направо одновременно, проверяя и «палиндромность», и «ямность». Набираем таким образом длину. Как только условия перестали выполняться,

останавливаемся. Несложно доказать, что мы не обойдем каждую позицию в процессе более 2 раз (разбейте массив на последовательные возрастающие и убывающие отрезки и посмотрите, сколько раз алгоритм их обходит). Итого: линейный алгоритм.

Задача N. Орнамент

Для каждой точки с помощью префикс сумм и трюка «+1 -1» можно узнать, покрывается ли она полностью какой-то фигурой, или нет. Для каждой фигуры можно пройтись по всем y-координатам, рассмотреть для фиксированной y-координаты самую левую и самую правую x-координаты, которые полностью покрываются фигурой. Прибавим на этом отрезке 1 при помощи трюка «+1 -1» (в левую прибавляем 1, в точке после правой — отнимаем 1).

После всех этих прибавлений собираем префикс суммы по всем y-координатам и теперь сможем с их помощью узнавать, покрыта ли точка полностью хотя бы одной из фигур. Если да, то она просто дает +1 к ответу.

Отдельно обработаем треугольнички от ромбов — в зависимости от конфигурации они могут давать +1, +0.75, +0.5 в клетке.

Задача O. Игра с двумя кучами

Рассмотрим матрицу выигрышных/проигрышных состояний. Каждое состояние - это пара чисел (x,y) - количество камней в первой и во второй куче. Очевидно, что состояние (0,0) - проигрышное, т.к. нельзя сделать ход. Также очевидно, что любая позиция (x,y) будет выигрышной, если выполняется одно из след условий:

- Существует такой a, что позиция (a,y) — проигрышная.
- Существует такой b, что позиция (x,b) — проигрышная.
- Существует такой $k > 0$, что позиция (x - k,y - k) — проигрышная

Заметим, что построенная матрица выигрышных и проигрышных позиций симметрична, то есть если позиция (x,y) - проигрышная, то и позиция (y,x) - проигрышная, и наоборот.

Рассмотрим расстояния между симметричными проигрышными позициями, то есть если позиция (x,y) проигрышная, то рассмотрим число abs(x-y). Заметим, что чем меньше это число, тем позиция (x,y) ближе к позиции (0,0). Также заметим, что эти числа увеличиваются ровно на 1, от каждой предыдущей проигрышной позиции. Пример проигрышных позиций и их расстояний: Для позиции (0,0), расстояние 0. Для позиций (1,2),(2,1), расстояние 1. Для позиций (3,5),(5,3), расстояние 2. Для позиций (4,7)(7,4), расстояние 3.

Заметим, что в каждой строчке, в каждом столбце, и на каждой диагонали ровно одна проигрышная позиция.

Пусть мы построили матрицу $n \times m$ выигрышных и проигрышных позиций и пусть количество проигрышных позиций в матрице было d. Тогда, для строчки $n+1$ можно вычислить проигрышную позицию. Очевидно что на этой строке еще не было проигрышной позиции. Тогда проигрышной позиции для этой строки станет позиция (n + 1, z) и позиция (z, n + 1) также проигрышная. Вспомним, что мы знаем какое расстояние между этими позициями должно быть. abs(z - (n + 1)) = (d + 1) / 2, тогда если z > (n + 1), то z = (d + 1) / 2 + n + 1, иначе z = (n + 1) - (d + 1) /2.

Отсюда следует, что проигрышных позиций не больше 2*min(n,m) и каждую проигрышную позицию легко вычислять последовательно.

В итого приходим к следующему решению. Вычислим, является ли выигрышной или проигрышной позиция (n, m). Пусть для определенности $n < m$ (иначе поменяем местами кучки в силу симметричности) и $d = 0$(начало расстояний). Будем хранить в отсортированном множестве (Treeset в Java или set в C++) все числа от 0 до n. Во множестве храним все номера строк, для которых еще не посчитали Проигрышную позицию на ней. Вытолкнем из множества минимальное число mi. Посчитаем для этой строки ее Проигрышную позицию. Этой позицией будет (mi, mi + d) и (mi + d, mi). Проверим, совпадает ли mi с n и $mi + d$ с m. Если совпадают оба числа, то это Проигрышная позиция и выиграет второй игрок, если совпадает только одно из чисел, то это Выигрышная позиция и выигрывает первый игрок. Если ни одно из чисел не совпадает, то вычеркнем mi и $mi + d$

из множества, так как для этих строк мы вычислили Проигрышную позицию. Увеличим d на 1 так как расстояние для следующей Проигрышной позиции должно быть на 1 больше предыдущей. Повторим операцию. Будем повторять до тех пор, пока $mi \leqslant n$. Сложность будет $O(n \log(n))$.

Вместо Treeset можно воспользоваться обычным массивом, что приведет к сложности $O(n)$, но в задаче это не требовалось.