Задача A. Цветы

Наименьшее количество букетов равняется одному при нечётном n и двум при чётном n (первый букет — из трёх цветов, второй — из $n-3$ цветов).

Чтобы получилось наибольшее количество букетов, нужно составить как можно больше букетов из трёх цветов. Если n делится на 3, то можно сделать $\frac{n}{3}$ букетов. Если n даёт остаток 1 при делении на 3, то можно сделать $\frac{n-4}{3}$ букетов (в одном — семь цветов, во всех остальных — по три цветка). Если n даёт остаток 2 при делении на 3, то можно сделать $\frac{n-2}{3}$ букетов (в одном — пять цветов, во всех остальных — по три цветка).

Задача B. Сломанный Shift

В этой задаче нужно просто перебрать все пары соседних букв в строке, добавляя к ответу единицу, если регистр этих букв различается. И, если первая буква в строке заглавная, нужно добавить к ответу ещё единицу.

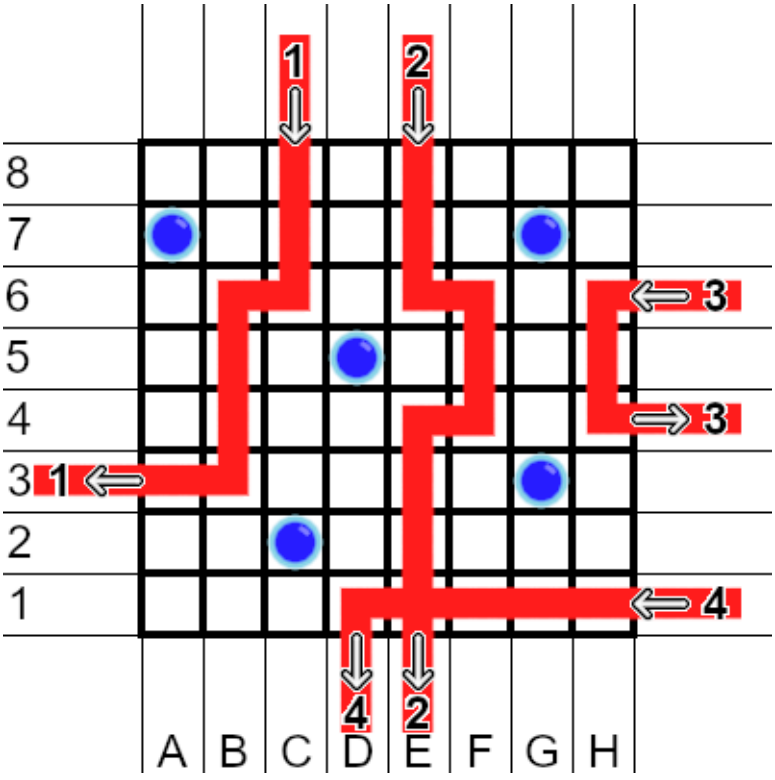
Задача C. Бедная Полина

Оказывается, если для какого-то i $a_i \neq i$, то Полина не сможет выполнить все дела.

Понятно, что Полина не сможет выполнить дело, если оно находится в цикле с другими делами. Поэтому покажем, если для какого-то i $a_i \neq i$, то i-е дело находится в цикле. Поскольку $a_i \neq i$ и в списке a присутствуют все числа от 1 до n, то есть индекс $j \neq i$, у которого $a_j = i$. Очевидно, тогда $a_j \neq j$. Тогда, по той же логике, есть индекс k, у которого $a_k = j$. Если $k = i$, то мы нашли цикл. Иначе продолжим рассуждения для k и так далее. Так как список конечен, рано или поздно мы попадём на индекс i, т.е. найдём цикл.

Задача D. Чёрный ящик

У данной головоломки ровно одно решение:



Догадаться до него можно так:

1. Очевидно, что луч 4 не может подняться вверх. Значит в клетке `C2` гарантированно есть шар.

2. Луч 1 не может идти по столбцу `A`, так как он не может повернуть в него. Значит луч 1 проходит по столбцу `B` и поворачивает в клетке `B3`. Можно заключить, что в одной из клеток `A5`, `A6`, `A7` или `A8` есть шар, который повернёт луч 1 в столбец `B`.

3. Луч 1 поворачивает нечётное число раз, так как он входит по вертикали, а выходит по горизонтали; и мы знаем, что как минимум два раза. Если луч 1 поворачивает пять или более раз, он должен касаться всех шаров. Можно проверить, что такое расположение не может быть решением. Значит, луч 1 поворачивает три раза. Для этого в одной из клеток `D3`, `D4`, `D5` или `D6` должен быть шар.

4. Тогда луч 2 не может пройти всё поле насквозь. Он должен повернуть хотя бы четыре раза, и нам осталось поставить только два шара. Нужную траекторию можно получить, если поставить два шара в столбец `G` или `H`, тогда луч 2 коснётся шара в столбце `D` дважды.

5. У нас осталось очень мало возможных расположений шаров. Есть только одно, которое подходит лучу 3.

Задача E. Гороскоп

Если для каждого предпочтения перебирать всех парней и проверять, подходят ли они под него, такое решение будет работать за время $O(n \cdot m)$ и не уложится в заданное ограничение по времени.

Давайте посчитаем для каждого месяца (от 1 до 12) значение $M[i]$ — сколько парней родилось в i-й месяц. Аналогично, для каждого дня (от 1 до 31) посчитаем значение $D[i]$ — сколько парней родилось в i-й день месяца, и для каждой возможной пары день-месяц посчитаем значение $DM[i][j]$ — сколько парней родилось в i-й день j-го месяца (при этом неважно, бывает ли в j-м месяце столько дней — если не бывает, то мы просто получим 0 парней). Тогда ответом для предпочтения $(a,\ b)$ будет являться значение величины $D[a] + M[b] - DM[a][b]$.

Задача F. Мат. ожидание и дисперсия

Если $m = n + 1$, то единственный способ сохранить оценку мат. ожидания — добавить число, равное E. Но тогда, если $D \neq 0$, то оценка дисперсии уменьшится. Таким образом, при $m = n + 1$ решение существует, только если $D = 0$ (то есть все числа одинаковые).

Иначе давайте добавим $\left\lfloor \frac{m-n}{2} \right\rfloor$ чисел $E+c$, столько же чисел $E-c$ и, возможно, одно число E, если $(m-n)$ нечётное. Очевидно, оценка мат. ожидания от этого не поменяется. Посчитаем, каким должно быть c, чтобы и оценка дисперсии не поменялась.

Если $(m-n)$ чётное, то

$$\frac{\sum_{i=1}^{n}(x_i - E)^2}{n} \cdot \frac{n}{n-1} = \frac{c^2 \cdot (m-n) + \sum_{i=1}^{n}(x_i - E)^2}{m} \cdot \frac{m}{m-1},$$

или, если выразить это через D,

$$D = \frac{c^2 \cdot (m-n)}{m-1} + D \cdot \frac{n-1}{m-1}.$$

Выразив c, получим $c = \sqrt{D}$.

Если $(m-n)$ нечётное, то аналогично,

$$D = \frac{c^2 \cdot (m-n-1)}{m-1} + D \cdot \frac{n-1}{m-1},$$

откуда

$$c = \sqrt{D \cdot \frac{m-n}{m-n-1}}.$$

Альтернативное решение: если $m \geqslant n + 2$, то мы знаем, что ответ всегда «`YES`», независимо от чисел в выборке. Возьмём все числа, кроме двух последних, равными E. Осталось выбрать два последних числа. Для сохранения оценки мат. ожидания они должны иметь вид $E - x$ и $E + x$

для некоторого x. Значение x, при котором сохраняется оценка дисперсии, можно найти бинарным поиском.

Задача G. Трудности менеджмента

Для маленьких n и m задачу можно решать поиском в глубину. Вершинами графа будут состояния, определяемые количеством джунов и сеньоров в комнате A и текущей позицией Саши (A или B). Все возможные переходы между вершинами задаются ограничениями из условия задачи: оставляем в комнате не больше джунов, чем сеньоров (или одних джунов); берём с собой не больше джунов, чем сеньоров (или одних джунов); переводим не менее одного и не более m людей. Задача сводится к поиску пути из вершины $(n,\ n,\ A)$ в вершину $(0,\ 0,\ B)$.

Посмотрев на ответы для небольших примеров, можно увидеть следующую закономерность (можно дойти до неё и без поиска в глубину, решив несколько примеров на бумажке): при $m = 1$ решения не существует, при $m = 2$ решение существует только для $n \leqslant 3$, при $m = 3$ решение существует только для $n \leqslant 5$, а при $m \geqslant 4$ решение существует для любого n.

После этого останется научиться при $m \geqslant 4$ быстро строить некоторый путь длины не более $2 \cdot 10^5$. Это можно делать, например, так. На каждом нечётном шаге будем переводить двух джунов и двух сеньоров из A и B, а на каждом чётном шаге переводить одного джуна и одного сеньора обратно. Тогда в обеих комнатах всегда будет поровну джунов и сеньоров, и мы закончим за $2n - 3$ шага.

Задача H. Золотая Рыбка

Допустим, мы знаем, что i-е и j-е желания должны идти друг за другом. Давайте определим, какое из них идёт первым, а какое — вторым.

Пусть до выполнения этих двух желаний у нас уже накопилось x рублей. Если сначала идёт i-е желание, потом j-е, то после них у нас будет

$$(a_i \cdot x + b_i) \cdot a_j + b_j$$

рублей. А если идёт сначала j-е желание, потом i-е, то после них у нас будет

$$(a_j \cdot x + b_j) \cdot a_i + b_i$$

рублей. Так как все последующие желания увеличат количество рублей тем больше, чем больше рублей мы получим сейчас, то надо сравнить эти два значения и жадно выбрать тот вариант, который нам даёт больше денег.

Таким образом, i-е желание в этом случае идёт после j-го, если выполняется неравенство

$$(a_i \cdot x + b_i) \cdot a_j + b_j \leqslant (a_j \cdot x + b_j) \cdot a_i + b_i.$$

Раскрыв скобки, заметим, что в обеих частях есть слагаемое $a_i \cdot a_j \cdot x$. Избавимся от него и получим неравенство

$$b_i \cdot a_j + b_j \leqslant b_j \cdot a_i + b_i.$$

Это неравенство не зависит от x, значит порядок двух соседних желаний не зависит от текущего количества денег.

Более того, это неравенство можно упростить еще больше. Перенесём слагаемые b_j и b_i в противоположные стороны и получим

$$b_i \cdot (a_j - 1) \leqslant b_j \cdot (a_i - 1).$$

Наконец, перенесём множители b_i и b_j в противоположные стороны, сделав их знаменателями. Получим

$$\frac{a_j - 1}{b_j} \leqslant \frac{a_i - 1}{b_i}.$$

Из этого можно сделать вывод: раньше должно идти то желание, у которого значение $\dfrac{a_i - 1}{b_i}$ меньше. Таким образом, задача свелась к сортировке по этому значению. Сравнение необходимо

производить в целых числах, потому что при $a_i, b_i \leqslant 10^9$ две различные дроби могут отличаться на число порядка 10^{-18}, что превышает точность стандартных типов `float` и `double`. В Python можно использовать модуль `decimal` или `functools.cmp_to_key`.

Задача I. Охотники за привидениями

Для каждого луча найдём точку его пересечения со стеной, если такая есть. После этого переберём все пары лучей. Если какая-то пара лучей пересекается и для одного из них точка их пересечения находится ближе к охотнику, чем точка пересечения этого луча со стеной (или у этого луча нет пересечения со стеной), то ответ задачи: `YES`.

Будем обозначать векторное произведение векторов v_1 и v_2 как $v_1 \times v_2$. Векторное произведение — это произведение длин векторов на синус угла между ними. Для векторов $v_1 = (x_1, y_1)$, $v_2 = (x_2, y_2)$ выполняется $v_1 \times v_2 = x_1 \cdot y_2 - y_1 \cdot x_2$.

Жюри предлагает проверять, что лучи пересекаются, и находить точку их пересечения следующим способом. Пусть лучи заданы точками начала p_1 и p_2 и направляющими векторами v_1 и v_2 соответственно. Сначала проверим, что прямые, содержащие эти лучи, не параллельны — это выполняется тогда и только тогда, когда $v_1 \times v_2 \neq 0$. Если прямые не параллельны, то у них есть точка пересечения p.

Точка p лежит на первой прямой, поэтому её можно представить в параметрическом виде:

$$p = p_1 + v_1 \cdot t,$$

где t — пока неизвестный нам скаляр. Точка p также лежит на второй прямой, поэтому её можно представить в виде

$$p = p_2 + v_2 \cdot k,$$

где k — другой неизвестный нам скаляр. Получаем равенство

$$p_1 + v_1 \cdot t = p_2 + v_2 \cdot k.$$

Немного преобразуем его:

$$v_1 \cdot t = (p_2 - p_1) + v_2 \cdot k.$$

Теперь векторно умножим обе части равенства на v_2. Получим

$$(v_1 \times v_2) \cdot t = (p_2 - p_1) \times v_2 + (v_2 \times v_2) \cdot k.$$

Заметим, что мы вынесли скаляры t и k за скобки, и у нас получился множитель $v_2 \times v_2$. Он равен 0, так как вектор v_2 параллелен сам себе. Получаем

$$(v_1 \times v_2) \cdot t = (p_2 - p_1) \times v_2,$$

откуда легко выражается

$$t = \frac{(p_2 - p_1) \times v_2}{v_1 \times v_2}$$

Чтобы точка p лежала на первом луче, должно выполняться $t \geqslant 0$. Аналогично вычислим k по похожей формуле

$$k = \frac{(p_1 - p_2) \times v_1}{v_2 \times v_1}$$

и проверим, что $k \geqslant 0$. Так мы убедимся, что точка p является точкой пересечения не только прямых, но и лучей.

Также нужно будет найти пересечение лучей бластеров с отрезком AB. Эта задача сводится к поиску пересечения двух лучей, только нужно проверить, что луч бластера пересекается как с лучом AB, так и с лучом BA. После чего можно проверить, что либо точек пересечения нет вообще, либо самая ближняя — точка пересечения со стеной. На самом деле, сами точки в задаче можно не искать — достаточно найти соответствующие им параметры t и выполнить для них аналогичную проверку.

Задача J. Секретная служба Санта-Клауса

Сначала выделим в графе компоненты сильной связности. Если есть компонента, не достижимая из других, и в неё нельзя никого телепортировать, то ответ 0.

Иначе найдём для каждой компоненты количество способов телепортировать эльфов в неё. Ответом задачи будет являться произведение всех этих чисел, посчитанное по модулю $10^9 + 7$.

Пусть в i-й компоненте сильной связности k_i домов, к которым можно телепортировать отряды эльфов. Если эта компонента достижима из какой-то другой, то ответ для неё равен 2^{k_i} (подходит любое подмножество отрядов, включая пустое). Если же i-я компонента не достижима из других, то ответ для неё равен $2^{k_i} - 1$ (нужно выбрать хотя бы один отряд, иначе дома в этой компоненте никто не обойдёт).

Задача K. Градиент

Давайте задавать каждую прогрессию парой чисел: шагом d_i и значением в первом пикселе a_i, даже если прогрессия не содержит первый пиксел. Тогда значение в j-м пикселе, если он попал в i-й градиент, равно $a_i \cdot (j-1) + d_i$. Найти d_i и a_i можно по формулам

$$d_i = \frac{y_i - x_i}{r_i - l_i}$$

и

$$a_i = x_i - (l_i - 1) \cdot d_i.$$

Вместо того чтобы присваивать различные значения на отрезке согласно арифметической прогрессии, будем просто присваивать пару чисел (d_i, a_i) на отрезке.

Получаем упрощённую задачу. Есть массив длины n из пар вещественных чисел, изначально заполненный значениями $(0, 0)$. Есть m запросов вида «присвоить на отрезке с l_i по r_i пару чисел (a_i, d_i)». Причём новые запросы перезаписывают старые. Требуется найти, как будет выглядеть наш массив пар после выполнения всех запросов.

Это типичная задача на структуры данных, её можно решить несколькими способами. Самый простой — метод сканирующей прямой. Будем идти по пикселам слева направо, для каждого пиксела будем поддерживать множество всех запросов, которые его покрыли. Поскольку все запросы — отрезки, можно для каждого из них запомнить первый и последний пиксел. Когда мы встречаем первый пиксел из запроса, добавляем этот запрос во множество, а после последнего пиксела из запроса удаляем этот запрос из множества.

Осталось только восстановить ответ: для каждого пиксела узнать самый последний запрос, покрывающий этот пиксел. Для этого будем хранить запросы в множестве, упорядочивая их по номеру. Подойдёт любая структура данных, поддерживающее упорядоченное множество: например, `set` в C++. Можно обойтись даже структурой данных «куча», так как нам нужно запрашивать только наибольший элемент. Это `priority_queue` в C++ или модуль `heapq` в Python.

Альтернативное решение: будем выполнять запросы в обратном порядке. Тогда каждый запрос будет закрашивать только пикселы, которые еще не закрашены. Для каждого запроса будем закрашивать пикселы явно слева направо, каждый будет закрашен не более одного раза. Нужно только уметь находить ближайший справа незакрашенный пиксел. Это можно сделать структурой данных «система непересекающихся множеств» за $O(n \cdot \alpha(n))$.