Задача A. Статистика

В задаче требовалось реализовать счётчик числа выигранных очков для каждой из команд, а также счётчик выигранных партий. После этого нужно было правильно определять, когда партия закончилась, и сбрасывать счётчики выигранных очков в партии.

Задача B. Очередная игра

Пусть у нас имеется n камней, и мы можем убирать либо a камней, либо b. Давайте сначала решим для $n < a + b$.

Давайте посчитаем величину $win[i]$ — выигрывает ли ходящий игрок, если сейчас перед ним i камней. По правилам игры кто не может сделать ход, проигрывает, значит $win[i] = false$ для $i < min(a, b)$. Для остальных i просто проверим, есть ли ход в проигрышную позицию, то есть если $win[i - a] = false$ или $win[i - b] = false$, то $win[i] = true$ (можно сделать соответсnвующий ход. Иначе куда бы мы не пошли, соперник выиграет, значит мы проиграем. Такие значения $win[i]$ можно насчитать за $O(a + b)$.

Теперь рассмотрим $n \geqslant a+b$. Если $win[n\%(a+b)] = true$, то первый игрок выиграет. Рассмотрим ход, который нужно сделать, чтобы выиграть игру $n\%(a + b)$ и сделаем его. После этого будем отвечать на ход соперника симметричным ходом, то есть если он убрал a камней, убирать b и наоборот. Так за каждую пару ходов будем убирать $a+b$ камней, пока не получим малое количество камней, где мы знаем что мы выигрываем.

Если win[n%(a+b)]=false, то и в игре с n камнями выигрывает второй игрок. Он просто всегда отвечает симметричным ходом, пока не останется $n\%(a+b)$ камней. Таким образом чтобы понимать, кто выиграет игру, нам нужно знать только остаток от деления исходного количества камней на (a+b). Если язык не поддерживает длинные числа, будем просто считывать по одной цифре числа. Пусть у нас имелось число x, и мы считали цифру d, пересчитаем наше число как $10x + d$ и сразу возьмем его по модулю $a+b$, таки образом мы всегда будем поддерживать остаток от деления числа на $a + b$.

Задача C. Архив

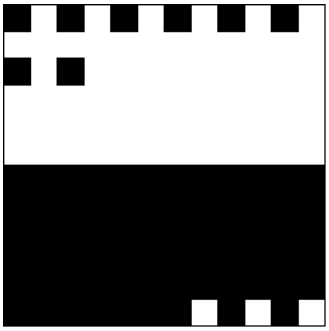
Вам потребуются две функции и структура данных для даты. Первая функция должна конвертировать данную на входе строчку длины 8 в дату, а вторая функция конвертировать дату в строчку формата выходного файла. Это простая реализационная задача.

Теперь у нас имеются промежутки дат двух типов, нужно «вычесть» из промежутков второго типа промежутки первого типа. Первым подходом было просто создать большой массив, состоящий из всех возможных дней, которые могут встретиться во вводе, затем проставить каждому дню тип (встречается он только в промежутках второго типа или нет), а затем пройтись по этому массиву и вывести ответ.

Более интересным и быстрым подходом было использование метода сканирующей прямой. У каждого промежутка есть дата начала и дата окончания, давайте рассматривать эти даты в отрыве от промежутков, но запомнив для каждой даты её тип: начало это или конец. Отсортируем все имеющиеся у нас даты, и будем обрабатывать их в этом порядке. Будем подсчитывать количество промежутков обоих типов, которые «открыты». Если у нас в текущий момент нет ни одного «открытого» промежутка первого типа, но есть «открытый» промежуток второго типа — это начало промежутка из ответа. Когда либо обнулится счетчик промежутков второго типа, либо «откроется» промежуток первого типа, промежуток из ответа закончится.

Задача D. Фантастические шахматы

Есть много разных стратегий для создания примера. Будем всегда использовать максимальное поле 100×100. Например, разобьем поле на две половины, верхняя белая, нижняя черная. Теперь будем расставлять в верхней половине черные клетки так, чтобы не нарушить связности белой половины. Нужно перекрасить $b - 1$ клетку, выбирая только клетки, у которых обе координаты, например, нечетные. Аналогично нужно перекрасить $a - 1$ клетку в нижней половине.



Задача E. Фильмы

Если k учеников хотят посмотреть один и тот же фильм, этот фильм необходимо скачать и посмотреть ровно k раз. С начала первой передачи необходимое время для всех желающих посмотреть фильм равно $k * (t + d)$. Единственная проблема это загрузка фильма с сервера, потому что вначале все хотят скачать с сервера. И тут вопрос в том, в каком порядке скачивать фильмы с сервера?

Если два фильма посмотрят k_1 и k_2 человек ($k_1 > k_2$) и начинают загрузку с сервера с задержкой c_1 и c_2, то время для просмотра обоих фильмов желающими будет $k_1(t+d)+c_1$ и $k_2(t+d)+c_2$. Если $c_1 > c_2$, мы можем поменять местами порядок загрузки двух фильмов, и получить лучшее общее время. Следовательно, фильмы следует загружать с сервера в зависимости от того, сколько людей будут их смотреть, при этом наиболее желаемые фильмы начнут загружаться в более раннее время.

Решение требует от нас посчитать для каждого фильма, сколько людей его посмотрят. Сортируем по этому числу. Далее мы вычисляем общее время для каждого фильма, добавляя время первоначальной загрузки с сервера после каждых m фильмов (для первых m фильмов это 0, для следующих m фильмов это d и так далее. После этого находим наибольшее общее время из всех.

Задача F. Дарите девушкам цветы

Необычной относительно других задач является третья операция: проверка, отсортирован ли отрезок массива. Отрезок отсортирован, когда каждое очередное значение не больше следующего, то есть разность между соседними элементами не положительна.

Перейдем от массива a к массиву разностей r: $r_i = a_i - a_{i-1}$. Также для восстановления массива a необходимо поддерживать значение a_0.

Первый запрос L R X. Заметим, что значения внутри отрезка $[L + 1, R]$ массива разностей не меняются. Поэтому изменения будут только такие: $r_L := r_L + X$, $r_{R+1} := r_{R+1} - X$, если $L = 0, a_0 := a_0 + X$.

Второй запрос i, j. Восстановим значения a_i, a_j. $a_i = a_0+(r_1+r_2+...+r_i)$, $a_j = a_0+(r_1+r_2+...+r_j)$. Дальше этот запрос сводится к запросу первого типа на отрезках $[i, i], [j, j]$ соответственно.

Третий запрос L, R. Если $min(r_{L+1}, r_{L+2}, ..., r_R) >= 0$, то отрезок отсортирован.

Для обработки запросов можно использовать дерево отрезков, построенное на массиве r. Общая сложность алгоритма $O(n + mlog(n))$

Задача G. Женя на физике

Утверждение 1: ход луча обратим, то есть если при запуске через отверстие x луч выходит через отверстие y, то при запуске через отверстие y луч выйдет через отверстие x.

Из этого следует, что все отверстия разбиваются на пары.

Утверждение 2: в каждой паре одно отверстие с левой или нижней стороны коробочки, а второе отверстие с правой или верхней стороны коробочки.

Доказательство: если луч движется вверх, то после отражения он будет двигаться вправо. Аналогично, если луч движется вправо, то после отражения он будет двигаться вверх. Соответственно, если луч начал движение из отверстия на левой или нижней стенке, то он может выйти только через отверстие на правой или верхней стенке.

Утверждение 3: условие существования корректной расстановки зеркал эквивалентно выполнению двух следующих условий (которые мы будем обозначать *):

1. Для каждого отверстия на нижней стенке соответствующий выход находится либо на правой стенке, либо на верхней в том же столбце или правее.

2. Для каждого отверстия на левой стенке соответствующий выход находится либо на верхней стенке, либо на правой в той же строке или выше.

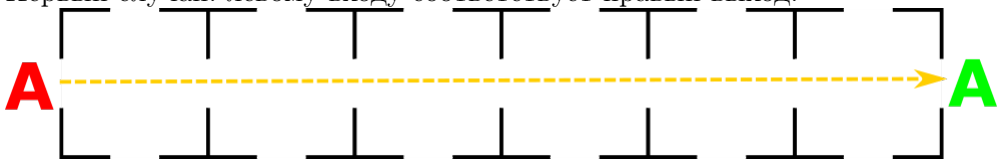
Следование вправо мы уже доказали. Осталось показать следование влево. Будем строить подходящую расстановку по индукции, поддерживая выполнение двух условий.

База: ноль строчек, только нижние входы. Понятно, что тогда в каждом столбце вход снизу будет соответствовать выходу сверху. Тогда утверждение доказано.

Переход: $k \to k + 1$.

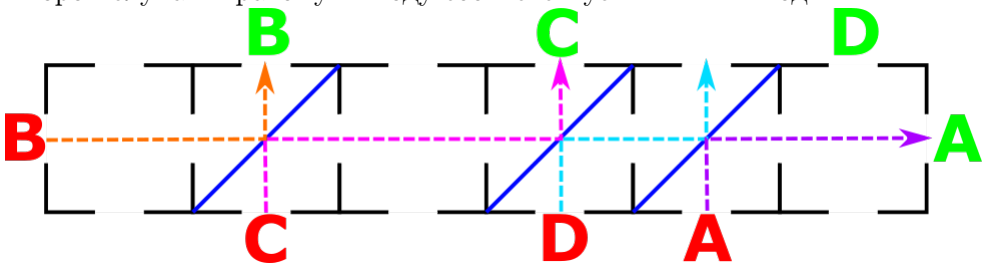
Рассмотрим последнюю строчку.

Первый случай: левому входу соответствует правый выход.



В этом случае никаких зеркал ставить не надо. Лучи из нижних входов не изменят направление после прохождения этого ряда, тогда переходим к k строчкам, при этом оба условия * выполняются.

Второй случай: правому выходу соответствует нижний вход.



В этом случае надо ставить зеркало в столбец входа A. Если бы выход B был бы в столбце входа A или правее, то этого зеркала было бы достаточно, так как для оставшихся k строк выполнялись бы условия *. Но на самом деле, мы должны поставить зеркало в столбец выхода B, посмотреть на вход C: его выход левее столбца входа A, поэтому ставим ещё одно зеркало в столбец выхода C и уже для входа D выполняется, что его выход правее столбца входа A. Это значит, что после такой расстановки зеркал у нас будут выполняться условия * для оставшихся k строк, так как те лучи, которые изменили направление в текущем ряду, не оказались правее своего выхода. Переход доказан.

Осталось посчитать, сколько зеркал мы расставили нашим методом. В первом случае мы не ставим зеркал, а во втором каждым поставленным зеркалом мы соединяем соответствующую пару входа и выхода, так как в столбце B и C больше не надо будет ставить зеркал. То есть зеркал поставим не более $n + m$.

Реализации этого алгоритма легко осуществляется с помощью массивов с итоговой ассимптотикой $\mathcal{O}(n + m)$.

Задача H. Закономерность

Заметим, что на нечётных позиция стоят чётные числа, а на чётных кубы.

Приведём решение на языке Python:

```
n = int(input())

print((n + 1) if n % 2 == 1 else (n // 2) ** 3)
```

Задача I. Хитмейкер – 2

Во всех решениях ищем ответ в виде возрастающей последовательности.

Решение 1: Динамическое программирование.

Посчитаем $dp[i][last]$ — максимальное возможное количество прослушиваний, если сумма звуков равна i, а последний звук $last$. Тогда $dp[i][last] = \max\limits_{j<last} j \cdot dp[i-j][j]$. Ответ находится в строчке $dp[n]$. Затем восстанавливаем ответ.

Кроме того, данное решение можно сократить до одномерной динамики, где будем хранить $dp[i]$ —максимальное возможное количество прослушиваний с суммой звуков равной i. В отличие от предыдущего решения мы в начале перебираем, какой последний звук берём и обновляем значение динамики. Для наглядности приведём код на C++.

```
dp[0] = 1;
for (int i = 1; i <= n; ++i) {
    for (int pos = n - i; pos >= 0; --pos) {
        if (dp[pos + i] < dp[pos] * i) {
            dp[pos + i] = dp[pos] * i;
            from[pos + i] = i;
        }
    }
}
```

Решение 2: Рекурсивный перебор.

Напишем рекурсивный перебор $gen(sum,\ last)$, где sum текущая сумма звуков, $last$ последний взятый звук. Если sum равна n, то проверяем обновился ответ или нет. Иначе перебираем звук $j > last$, который будем брать следующим, и запускаем $gen(sum + j,\ j)$. Но такое решение будет работать долго и будет получать `TL`.

Запуская перебор на различных n можно заметить, что разница между соседними числа в ответе не больше двух. Тогда можно заменить перебор на два вызова: $gen(sum + last + 1,\ last + 1)$ и $gen(sum + last + 2,\ last + 2)$. Такое решение получает `OK`.

Решение 3: Жадность.

Наберём по максимуму числа $2, 3, 4, \ldots$, чтобы их сумма была не больше n, и назовём их a_i. Обозначим за sum — сумму таких чисел, а cnt — их количество. Тогда у нас осталось ещё $n - sum$, которое мы распределим между числами a_i. Добавим к каждому числу $\lfloor \frac{n-sum}{cnt} \rfloor$, а так же к последним $(n - sum)\%cnt$ числа по единице.

Задача J. Счастливое расстояние

Будем решать задачу с помощью техники *разделяй и властвуй*. Рассмотрим решение задачи для одного полуинтервала $[l, r)$, который делим на две части – $[l, m)$ и $[m, r)$, т.е. необходимо посчитать количество отрезков, удовлетворяющих условию задачи и пересекающих границу разделения. Рассмотрим случай, когда максимум искомого отрезка находится в правой части разделения, а минимум – в левой. Остальные случаи разбираются полностью аналогично. Теперь определим и посчитаем вспомогательные функции:

1. $f(x) = x - m - \max\limits_{m \leqslant t \leqslant x} a[t]$, $m \leqslant x < r$, где a - исходный массив числел.
2. $g(x) = m - x + \min\limits_{x \leqslant t < m} a[t]$, $l \leqslant x < m$.

Теперь количество отрезков, которые мы хотим на данном этапе посчитать, можно выразить лаконичной формулой - количество пар (x, y), таких что $g(x) + f(y) = 0$. Будем итерироваться слева направо по правому полуинтервалу и поддерживать максимальный по включению отрезок $[u, v] \subset [l, m)$, такой что:

1. $\max\limits_{u \leqslant t \leqslant v} a[t] \leqslant \max\limits_{m \leqslant t \leqslant y} a[t]$, где y - текущая позиция в правом полуинтервале;
2. $\min\limits_{u \leqslant t \leqslant v} a[t] \geqslant \min\limits_{m \leqslant t \leqslant y} a[t]$.

При увеличении y концы отрезка $[u, v]$ смещаются только влево. Теперь для каждого y на отрезке $[u, v]$ нужно посчитать количество x, таких что $g(x) = -f(y)$. Это можно сделать с помощью, например, дерева отрезков.

Итоговое решение можно реализовать с временной сложностью $\mathcal{O}(n \log^2 n)$ или $\mathcal{O}(n \log n)$.

Задача K. Спиральный робот

Рассмотрим случаи:

1. $n = 1$ и $m \geqslant 1$ тогда ответ 0, потому что робот будет ходит только направо
2. $n > 1$ и $m = 1$ тогда ответ 1, робот повернет один раз вниз и обойдёт всё поле
3. $n = 2$ и $m \geqslant 2$ тогда ответ 2, робот пройдёт первую строку, повернёт вниз, сделает один шаг вниз, повернет налево и обойдёт вторую строку
4. $n \geqslant 2$ и $m = 2$ тогда ответ 3, робот пройдет первую строку, повернет вниз, обойдёт второй столбец, повернет налево, обойдёт последнюю строку, повернет вверх и обойдёт первый столбец
5. $n > 2$ и $m > 2$ тогда можно обойти границу сделав 4 поворота и свести задачу к задаче $n - 2$ и $m - 2$

На основе случаев можно написать следующий код:

```
while (n > 2 && m > 2) {
    n -= 2;
    m -= 2;
    ans += 4;
}
if (n == 1 && m >= 1) ans += 0;
if (n > 1  && m == 1) ans += 1;
if (n == 2 && m >= 2) ans += 2;
if (n > 2  && m == 2) ans += 3;
```

Но он будет работать долго из-за цикла. Давайте посчитаем сколько раз выполнится цикл. Каждый раз мы вычитаем 2, пока минимальное число не станет 1 или 2. Можно подсчитать, что цикл выполнится $\min((n-1)/2,\ (m-1)/2)$ раз. И получаем следующий код:

```
long long cnt = min((n - 1) / 2, (m - 1) / 2);
long long ans = cnt * 4;
n -= cnt * 2;
m -= cnt * 2;

if (n == 1 && m >= 1) ans += 0;
if (n > 1  && m == 1) ans += 1;
if (n == 2 && m >= 2) ans += 2;
if (n > 2  && m == 2) ans += 3;
```

Кроме того, можно заметить, что случаи можно объединить и итоговый код будет таким:

```
long long ans = 2 * min(n, m) - 2;
if (n > m) ans++;
```

Задача L. Энтропия осведомленности

Рассмотрим значения a' и b' (новые значения каждый день) $a' = b + c$, $b' = a + c$. Как мы видим, к ним обоим прибавляется число c, поэтому оно никогда не сказывается на разности $a' - b' = (b + c) - (a + c) = b - a$. При этом, как мы видим, модуль разности никогда не меняется, а каждый день меняется только знак.

Покажем, как изменяется искомая разность $a - b$ для некоторых значений k:

k	a	b	c	Ответ
1	b+c	a+c	a+b	$b - a$
2	2a+b+c	a+2b+c	a+b+2c	$a - b$
3	2a+3b+3c	3a+2b+3c	3a+3b+2c	$b - a$
4	6a+5b+5c	5a+6b+5c	5a+5b+6c	$a - b$

Очевидно, что искомая разность равна $a - b$ для четных k, и $b - a$ для нечетных k.

Задача M. Тримино

Данная таблица задаёт правильное покрытие прямоугольного поля цифрами тримино тогда и только тогда, когда выполняются следующие два условия:

1. Число 0 встречается в таблице ровно $nm\%3$ раз ($\%$ — операция остатка от деления), а каждое из чисел от 1 до $\frac{nm}{3}$ ровно три раза.
2. Для каждого числа от 1 до $\frac{nm}{3}$ клетки, в которых встречается это число, образуют тримино.

Пусть даны три клетки с координатами (i_1, j_1), (i_2, j_2) и (i_3, j_3). Имеет место следующий факт: необходимым и достаточным условием образования тримино из этих трех квадратов является выполнение равенства $(|i_1 - i_2| + |j_1 - j_2|) + (|i_1 - i_3| + |j_1 - j_3|) + (|i_2 - i_3| + |j_2 - j_3|) = 4$.

Доказательство тривиально. Нужно заметит, что 4 можно представить как сумму трех натуральных чисел одним из следующих способов: $4 = 1 + 1 + 2$, $4 = 1 + 2 + 1$, $4 = 2 + 1 + 1$. Легко заметит, что для каждой из двух фигур тримино условие выполняется. Наоборот, пусть условие выполняется для фигуры, состоящей из трех квадратов, которые не состовляют тримино. Выражения в скобках указывают манхэттенское расстояние между каждыми двумя квадратами на поле. Это расстояние равно 1 тогда и только тогда, когда два квадрата смежны. Из представления числа 4 как суммы трех положительных целых чисел видно, что для каждых трех отдельных квадратов, удовлетворяющих условию утверждения, выполняется: один из квадратов имеет общие стороны с двумя другими, у которых нет общей стороны друг с другом. Это как раз две фигурки тримино.

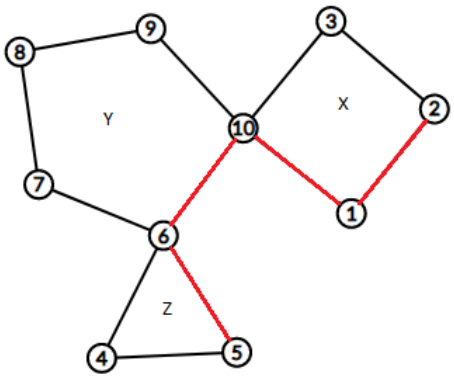
Теперь достаточно просто проверить общее количество чисел для выполнения первого условия, а для каждого числа от 1 до $\frac{nm}{3}$ проверить упомянутое выше равенство.

Задача N. Экскурсия

Задача найти самый длинный простой путь в произвольном графе, конечно, NP-полная. Но не тогда, когда на граф накладываются ограничения. В этом задаче данный граф обладает тем свойством, что каждое ребро участвует ровно в одном цикле – такой граф называется гладким кактусом (в общем случае у кактуса могут быть древовидные наросты, состоящие из ребер, не участвующих в каком-либо цикле, что сделало бы задачу сложнее). Решение данной задачи основано на следующей теореме.

Самый длинный простой путь в гладком кактусе от вершины A до вершины B проходит через те же циклы, что и самый короткий путь между A и B.

Доказательство теоремы основано на том, что простой путь не может войти в произвольный цикл (не лежащий на кратчайшем пути), потому что для выхода из него ему придется второй раз пройти через вершину, через которую он вошёл. На рисунке красным цветом показан кратчайший путь между вершинами 2 и 5 в графе из примера — $2, 3, 10, 6, 5$. Он проходит последовательно через циклы X, Y и Z. Самый длинный путь и самый короткий путь, согласно теореме, должны проходить через одни и те же циклы. Разница в том, что заходя на цикл, вместо самого быстрого пути до выхода из этого цикла, мы должны искать самый длинный путь (он идет в противоположном направлении).



Технически, вам нужно в решении проделать следующие шаги:

- Найдите циклы гладкого кактуса. Для этого достаточно модифицированного обхода графа в глубину, который мы выполняем итеративно с использованием нашего собственного стека, чтобы при нахождении цикла мы могли удалить его вершины из стека. Сложность этого шага $O(E) = O(V)$ (здесь и далее E — количество ребер в графе, V — количество вершин в графе).
- Находим кратчайший путь в графе от вершины A к вершине B с помощью обхода в ширину. Сжимаем этот путь так, чтобы в полученном пути присутствовали только A, B и вершины, общие для двух соседних циклов пути (вершины входа и выхода из циклов). Сложность этого шага также $O(E)$.

  В примере кратчайший путь сокращается до $2, 10, 6, 5$ где 2 и 10 находятся в цикле X, 10 и 6 в цикле Y, а 6 и 5 в цикле Z. Для того, чтобы это делать просто, удобно для каждого ребра запомнить на первом этапе, какому циклу оно принадлежит.
- Последний шаг — для каждых двух соседних вершин в сокращенном пути найти длину более длинного из двух путей от одной до другой в их общем цикле. Тут снова может помочь сохраненное значение номера цикла для каждого ребра, чтобы быстро понимать какому циклу принадлежат соседние вершины в сокращенном пути. Ответом будет сумма этих длин. Сложность снова $O(E)$.

Задача O. Математический фокус

Если k — нечётное, тройка чисел (a, b, c) подходит только если все числа кратны k. Пусть a некратно k и имеет остаток a', тогда чтобы пара с b подходила по условию, остаток числа b должен быть равен $k - a' \neq a'$ (из-за нечётности k), но тогда число c не сможет образовать пару с одним из этих чисел.

Посчитать количество таких троек просто — натуральных чисел, кратных k и не превосходящих n всего $\lfloor \frac{n}{k} \rfloor$, то есть троек $\lfloor \frac{n}{k} \rfloor^3$.

Если k — чётное, то тройки, в которых все числа кратны k тоже подходят. Но еще подходят тройки, в которых все числа имеют остаток при делении на k равный $\frac{k}{2}$. Доказательство того, что остальные тройки не подходят, аналогично случаю с нечётным k.

Посчитать количество таких троек тоже несложно, оставим это в качестве упражнения.