Задача A. Поворот

Внимательно посмотрим на пример и заметим закономерность. Алгоритм простой. Пробегаемся по столбцам, а уже потом перебираем строки в обратном порядке. То есть, если раньше у нас был обход (i: 0..n) (j: 0..m) , то теперь (j: 0..m) (i: n - 1..0)

Задача B. Состязания-2

Для каждого спортсмена храним лучший бросок. Если таких людей несколько, то выбираем того, у кого максимальная сумма результатов. Если таких несколько, выбираем того, чей индекс меньше. Будьте внимательны, индексы начинаются с нуля!

Задача C. Выручка театра

Считываем матрицу стоимости билетов и матрицу проданных билетов. Для каждого билета проверяем, если он был продан, то считаем его цену. Выводим общую стоимость.

Задача D. Спираль

Решение на 40 баллов создаёт двумерный массив размером $N \times M$ и моделирует движение черепашки, отмечая в массиве закрашенные клетки. Такое решение довольно трудно в реализации, приведем пример реализации на языке C++. Решение имеет сложность $O(NM)$.

```
#include<iostream>
#include<vector>
using namespace std;

int n, m;
vector <vector<bool>> used;;
unsigned int ans = 0;
int di[4] = {1, 0, -1,  0};
int dj[4] = {0, 1,  0, -1};

bool check_used(int i, int j)
{
    return i >= 0 && i < n && j >= 0 && j < m && used[i][j];
}

bool check_good(int i, int j)
{
    if (i < 0 || i >= n || j < 0 || j >= m)
        return false;
    int count = check_used(i - 1, j) + check_used(i + 1, j)
              + check_used(i, j - 1) + check_used(i, j + 1);
    return count == 1;
}

int main()
{
    cin >> n >> m;
    used.resize(n, vector<bool>(m));
    int i = 0;
    int j = 0;
    int dir = 0;
    while (true)
    {
        ++ans;
        used[i][j] = true;
```

```
        int i1 = i + di[dir];
        int j1 = j + dj[dir];
        if (check_good(i1, j1))
        {
            i = i1;
            j = j1;
            continue;
        }
        dir = (dir + 1) % 4;
        i = i + di[dir];
        j = j + dj[dir];
        if (!check_good(i, j))
            break;
    }
    cout << ans << endl;
}
```

Чтобы написать решение на 60 баллов, рассмотрим движение черепашки вдоль левой стороны прямоугольника. Черепашка закрасит столбец из N клеток. Теперь пусть черепашка сдвинется на одну клетку вправо, закрасив суммарно $N + 1$ клетку. Справа от черепашки теперь располагается прямоугольник из $M - 2$ столбцов и N строк, и черепашка начинает обходить его вдоль стороны длиной $M - 2$. То есть мы свели задачу от прямоугольника размером $N \times M$ к прямоугольнику размером $(M - 2) \times N$, закрасив $N + 1$ клетку. Это можно сделать, если $N \geqslant 2$ и $M \geqslant 2$. Пока соблюдается это условие, будем в цикле прибавлять к ответу $N + 1$ и менять числа (N, M) на $(M - 2, N)$.

В результате мы получим либо вырожденный пустой прямоугольник (если мы отрежем от него всё), либо прямоугольник, одна из сторон которого равна 1. В этом случае черепашка закрасит весь оставшийся прямоугольник, в котором будет MN клеток.

Такое решение имеет сложность $O(M + N)$ и набирает 60 баллов. Пример решения на языке Python.

```
n = int(input())
m = int(input())
ans = 0
while n >= 2 and m >= 2:
    ans += n + 1
    n, m = m - 2, n
ans += m * n
print(ans)
```

Наконец, для решения на полный балл заметим, что суммируются числа вида $N + 1$, $M - 2 + 1$, $N - 2 + 1$, $M - 4 + 1$, $N - 4 + 1$, ... То есть каждое следующее горизонтальное или вертикальное перемещение черепашки окажется на 2 короче предыдущего горизонтального или вертикального перемещения, и мы суммируем арифметические прогрессии, что можно сделать без использования цикла.

Удобней суммировать не две прогрессии, а одну. Рассмотрим передвижения черепашки вдоль сторон прямоугольника: вниз на N клеток, вправо на $M - 1$ клеток, затем вверх на 1 клетку. Так мы перейдём к прямоугольнику размером $(N-2)\times(M-2)$, прибавив к ответу $N+M$. На следующем шаге мы поибавим к ответу $(N-2)+(M-2)$, затем опять уменьшим стороны прямоугольника на 2. Мы получим арифметическую прогрессию с начальным членом $N + M$ и разностью -4.

Посчитаем количество членов этой прогрессии — число возможных сокращений сторон прямоугольника на 2, пока не останется прямоугольник, наименьшая сторона которого будет не более 2. Обозначим это число за k. Добавим к ответу сумму арифметической прогрессии из k членов с начальным значением $n + m$ и разностью -4, и разберём случаи разного количества закрашенных клеток в оставшемся прямоугольнике.

Такое решение имеет сложность $O(1)$, пример решения на языке Python.

```
n = int(input())
m = int(input())
k = min(n - 1, m - 1) // 2
ans = ((n + m) + (n + m) - 4 * (k - 1)) * k // 2
n -= 2 * k
m -= 2 * k
if n == 1:
    ans += m
elif m == 1:
    ans += n
elif m == 2:
    ans += n + 1
else: # n == 2, m > 2
    ans += m + 2
print(ans)
```

Задача E. Любовь и двери

Далее под «правильным набором дверей» будем понимать такой набор, при котором изменять их уже не нужно, то есть в начале набора стоят двери одного типа, а в конце двери другого типа, например набор 2, 2, 2, 2, 1, 1, 1 будет правильным. Такую проверку можно сделать за одно прохождение по набору путем сравнения рядом стоящих дверей. Если не более одной пары рядом стоящих дверей различны, то такой набор является правильным.

Решение для тестов первой группы, в которых не больше трех золотых дверей и длина которых не больше 10. Понятно, что в этом случае ответ не превосходит 3. Попробуем перебрать все варианты изменения 0, 1 или 2 дверей и для каждого из них проверим, что полученный после текущего изменения набор дверей является правильным. После каждого изменения и проверки на правильность отменяем все изменения и проверяем следующий вариант. При этом вариант с 0 изменений (то есть без изменений) проверяется сразу, вариант с одним изменением проверяется одним циклом, а вариант с двумя изменениями проверяется двойным вложенным циклом. Максимальная сложность будет кубической с учетом проверки каждого варианта на правильность.

Решение для тестов второй группы, в которых длина набора не превосходит 100. В этом случае можно реализовать алгоритмы с тройным вложенным циклом. Например такой: переберем место встречи героев i (до позиции i невключительно мы хотим чтобы были двери одного типа, а после этой позиции включительно — двери другого типа). Далее для выбранной границы бежим по всем дверям и каждую дверь, которая не соответствует зафиксированному выше порядку дверей для позиции i изменяем, при этом увеличивая счетчик. Непосредственно после каждого изменения проверяем является ли набор после этого изменения правильным и если да — то сравниваем полученное до этого момента количество изменений с минимальным. Не забудем проверить два варианта расположения дверей (сначала двери типа 1, потом двери типа 2 и наоборот).

Для прохождения тестов третьей группы, в которых длина набора не больше 1000 нужно избавиться от одного вложенного цикла, то есть написать квадратичное решение. Можно заметить, что в предыдущем решении не обязательно проверять на правильность набор дверей после каждого изменения, достаточно это делать только после полного приведения дверей к порядку, зафиксированному при помощи позиции встречи i. Другой вариант с такой же сложностью (наиболее близкий к основному решению всей задачи) такой: мы для позиции встречи i и заданного этой позицией расположения дверей просто переберём все двери и подсчитаем количество дверей, которые нужно изменить слева, и количество дверей, которые нужно изменить справа, чтобы получить заданную расстановку дверей. Среди всех таких вариантов изменения найдем минимальный.

Для получения полного решения нужно написать линейное решение. После чтения последовательности подсчитаем сколько в ней цифр 1 и сколько цифр 2. Далее за один проход по набору получим ответ следующим образом: для каждой позиции i от 0 до длины набора минус один мы

храним $left_1$ — сколько цифр 1 и $left_2$ — сколько цифр 2 было встречено на отрезке от начала строки до позиции $i-1$ включительно (для $i=0$ эти величины полагаем равными 0). Зная общее количество цифр по отдельности и величины $left_1$ и $left_2$, можно вычислить сколько раз встречается каждая цифра на отрезке от i до конца строки ($right_1$ и $right_2$ соответственно). Тогда для позиции i вычислим минимум из $left_1 + right_2$ и $left_2 + right_1$ — столько минимум нужно сделать замен, чтобы именно в этой позиции произошла встреча. Осталось выбрать самую маленькую по этому показателю позицию и вывести минимум для неё.

```
n = int(input())
a = [int(input()) for i in range(n)]
left_1 = 0
left_2 = 0
right_1 = a.count(1)
right_2 = a.count(2)
ans = min(right_1, right_2)
for c in a:
    if c == 1:
        right_1 -= 1
        left_1 += 1
    else:
        right_2 -= 1
        left_2 += 1
    ans = min(ans, left_1 + right_2, left_2 + right_1)
print(ans)
```

При написании и тестировании программы полезно проверить особые частные случаи.

Первый случай — когда двери изначально уже расположены так, что их можно открыть без изменений (тогда ответом будет число 0). Например, задана последовательность 2, 1, 1, 1. Очевидно, что если выдать тому кто у первой двери серебряный ключ, а тому кто у последней двери золотой, то можно открыть сразу все двери без их изменения. Важным подслучаем здесь является вариант, когда все двери уже изначально имеют один тип, например 2, 2, 2, 2, 2 или 1, 1, 1. Эти примеры тоже обязательно нужно протестировать.

Второй частный случай связан с вариантом, когда оптимально изменять только двери одного типа, например серебряные на золотые, и окончательный набор дверей так же является последовательностью дверей одного типа (в данном случае золотых). Например, последовательность 1, 1, 1, 1, 2, 1, 1, 1, 1. Для неё лучшим вариантом будет изменить за одну минуту дверь номер 5.

Задача F. Орешки для белочки

Необходимо превратить наш массив в массив $k_i \% d_i$. После этого посчитать для каждого числа $1, 2, \ldots, 100$ (так как все остатки не превосходят 100) сколько раз оно встречается в массиве и выбрать максимальное число.

Задача G. Обороноспособность

Нам надо посчитать сумму на границе. Можем сложить каждую сторону отдельно, но в таком случае мы посчитаем уголки по два раза. Поэтому мы можем просто отнять уголки от этой суммы. Исключение $n = 1$.

Задача H. Симметричная ли матрица?

Считываем массив. Мы можем пройтись по верней диагонали и проверить симметричный ей элемент, то есть для элемента [i][j] симметричный элемент [j][i]. Если они не равны, то матрицы не симметричны относительно главной диагонали.

Задача I. Побочная диагональ

Нам надо заполнить побочную диагональ единицами. То что выше - нулями, то что ниже - двойками. Если считать индексы с нуля, то: Побочная диагональ обладает свойством, что i + j =

n - 1, для всех элементов массива. То, что выше этой диагонали i + j < n - 1, то, что ниже этой диагонали i + j > n - 1. Просто проверяем, к какому из этого относится наш текущий элемент и выводим соответствующую цифру.

Задача J. Голосования на собрании факультета

Решение задачи сводится к подсчёту голосов. Для этого создадим массив $score$ такой, что $score[i]$ — это текущий результат голосования за кандидата i.

Каждый раз, когда кто-то голосует ЗА кандидата i, $score[i]$ требуется увеличивать на 1, а при голосовании ПРОТИВ — уменьшать.

Дополнительно требуется не учитывать голоса, при которых сотрудник голосует за себя, т.е. $w_i = d_i$.

Сложность составляет необходимость проверять, не голосует ли данные сотрудник за текущего кандидата повторно, что запрещено правилами. Проще всего сделать это, если завести двумерный логический массив $voted$, где $voted[i][j]$ истинно, если сотрудник номер i проголосовал за кандидата j. Тогда для каждой строчки протокола можно проверять, была ли уже такая такая пара.

Без использования двумерного массива решить задачу можно, если сохнарить весь протокол в пару одномерных массивов, и каждый раз для каждой новой строчки протокола пробегать по всем предыдущим, чтобы проверить, была ли уже такая пара.

Ответ на задачу — индекс массива $score$ с наибольшим значением. Требуется не забыть, что ответ должен быть положительным, иначе вывести -1.