Задача A.

Нужно было вывести $\max(n, \max(a_i))$.

Задача B.

Построим вспомогательный ориентированный граф G, где из вершины a в вершину b ребро будет идти тогда и только тогда, когда игрок a может победить игрока b в личной встрече.

Заметим, что если в исходном графе вершины a и b не были соединены ребром, то во вспомогательном графе будет одновременно ребро из a в b и ребро из b в a. Иначе, между этой парой вершин будет ровно одно ребро – такое же, как и в исходном графе.

Нетрудно заметить, что внутри компоненты сильной связности графа G, любой игрок может победить любого. Поэтому сожмем все компоненты сильной связности в одну вершину, т.е. построим конденсацию G' графа G.

Теперь вершины в графе G' соответствуют множеству вершин (игроков) исходного графа.

Вершина графа G' может выиграть турнир тогда и только тогда, когда из нее достижимы все остальные вершины этого графа. Но т.к. граф G' – ацикличен, то можно заметить, что такая вершина будет ровно одна – первая в топологической сортировке G'.

Оценим асимптотику решения этой задачи. Легко заметить, что она будет линейна относительно количества вершин и ребер графа G. Вершин в нем n, а ребер будет $O(n^2 - m)$, что очень много для данных ограничений.

Впрочем, можно заметить, что граф G нужен только для того, чтобы построить граф G'. Посмотрим, как можно сделать это быстрее.

Возьмем вершину исходного графа, которая соединена с наименьшим числом других вершин (обозначим это число x). Т.к. она не соединена с другими $(n - x - 1)$ вершиной, то в графе G' все эти $(n - x)$ вершин будут лежать в одной компоненте. Следовательно, их можно заранее объединить.

Теперь в графе G' будет не более $(n - (n - x) + 1)^2 = (x + 1)^2$ ребер. Оценим число x.

1) Если $n < \sqrt{m}$, то т.к. $x \leq n$, то $(x + 1)^2 \leq m$.

2) Если $n > \sqrt{m}$. Пусть $x > \sqrt{m}$. Но т.к. из каждой вершин исходит хотя бы x ребер, то суммарное кол-во ребер будет хотя бы $n \cdot x > \sqrt{m} \cdot \sqrt{m} = m$. Получаем противоречие. Следовательно $x \leq \sqrt{m}$, а значит $(x+1)^2 \leq m$.

Таким образом, мы получили решение за $O(n+m)$.

Задача C.

Обозначим кол-во девочек как x, а кол-во мальчиков – y. Тогда ответом на задачу будет $\min(x, y)$.

Задача D.

Заметим, что рассмотрев какой-то последовательный набор функций, они будут либо вложенные, либо не пересекающиеся. То есть, этот набор функций можно представить в виде дерева. А значит, ответ – это количество способов построить дерево на всей строке.

Реализуется с помощью динамики с С-шками.

Задача E.

Рассмотрим метод трапеций (для нахождения площади) в данном многоугольнике. Заметим, что каждая трапеция будет либо линией, либо прямоугольником. Подсчитаем трехмерный массив *arr[c][x][y]*, в котором будет храниться количество клеток с символом *c*, на прямоугольнике *(0,0,x,y)*. С помощью этого массива мы сможем узнать количество клеток с символом *c* на прямоугольнике (x_1,y_1,x_2,y_2) = $arr[c][x_2][y_2] - arr[c][x_1][y_2] - arr[c][x_2][y_1] + arr[c][x_1][y_1]$.

Теперь, зафиксировав многоугольник на матрице, мы сможем узнать, сколько в нем каких цифр, и, естественно, одна ли там цифра.

Таким образом, получили решение за $O(n^3)$.

Задача F.

Нужно было найти следующую величину: $\sum_{a=l_1}^{r_1} \sum_{b=l_2}^{r_2} \gcd(a, b)$.

Преобразуем это выражение: $\sum_{a=l_1}^{r_1} \sum_{b=l_2}^{r_2} \gcd(a, b) = \sum_{g=1}^{r_1} g \cdot f(g)$, где *f(g)* – кол-во пар чисел a, b таких, что $l_1 \leq a \leq r_1, l_2 \leq b \leq r_2$ и $\gcd(a, b) = g$.

Введем вспомогательную функцию: *x(g)* – кол-во пар чисел a, b таких, что $l_1 \le a \le r_1, l_2 \le b \le r_2, a \vdots g, b \vdots g$.

Заметим, что количество чисел от 1 до n кратных g равно $\left\lfloor \frac{n}{g} \right\rfloor$. Используя это, можно легко выразить *x(g)*: $x(g) = (\left\lfloor \frac{r_1}{g} \right\rfloor - \left\lfloor \frac{l_1 - 1}{g} \right\rfloor) \cdot (\left\lfloor \frac{r_2}{g} \right\rfloor - \left\lfloor \frac{l_2 - 1}{g} \right\rfloor)$.

Для того, чтобы найти *f(g)*, можно воспользоваться формулой включений/исключений: $f(g) = x(g) - f(2g) - f(3g) - f(4g) - \cdots - f(\left\lfloor \frac{r1}{g} \right\rfloor g)$.

Теперь мы можем находить *f(g)* методом динамического программирования, перебирая g от r_1 до 1.

Итоговая сложность – $O(r_1 \cdot \log r_1)$.

Задача G.

Пройдемся бинарным поиском по ответу. На каждой итерации будем проверять, можно ли составить корректную матрицу, с текущей разницей k.

Посмотрим, как можно выполнить это проверку. Найдем клетку с наибольшим числом A. Обновим для неё соседей: то есть, для тех соседних клеток, где еще не поставлено число, поставим число $A - k$. Далее среди еще невыбранных клеток выберем клетку с наибольшим числом и проделываем ту же операцию. Если в какой-то момент времени окажутся 2 соседние клетки, разность которых больше, чем k, то данное k слишком большое и перейдем к следующей операции бинарного поиска.

Для того, чтобы эффективно находить еще невыбранную клетку с максимальным значением, воспользуемся методом двух указателей. А именно, заведем 2 массива, в одном из которых будут храниться отсортированные по убыванию значения – исходно отмеченные клетки. А второй (динамический) массив (или очередь) будет увеличиваться, путем добавления точек, в которые мы записываем значения.

Итоговая сложность – $O(n \cdot \log n)$.

Задача H.

Для начала заметим, что оптимальный ответ всегда можно получить, применяя указанную операцию только к одной определенной клетке. Действительно, с каждой очередной перекраской мы как бы расширяем

текущую компоненту, пока не захватим все поле, и если мы переберем все возможные «стартовые» клетки, то среди них будет та, начиная с которой захватить всё поле можно быстрей всего.

Собственно, это и сделаем – переберем все $n * m$ клеток и для каждой посчитаем, за сколько шагов можно покрасить все поле, применяя указанную операцию только к стартовой клетке. Сделать это можно с помощью BFS 0-1 на клетчатом поле, используя структуру данных «дек». Введем понятие длины пути от стартовой клетки до текущей, которая будет равняться количеству операций перекрашивания стартовой клетки, необходимого для того, чтобы добраться до текущей. Тогда, находясь в клетке цвета a, будем добавлять её соседей цвета a в начало дека с таким же расстоянием, а соседей цвета b – в конец дека с расстоянием, увеличенным на 1 (т.к. нам придется сделать 1 операцию перекрашивания, чтоб добраться до этого соседа). Ответом для фиксированной стартовой клетки будет клетка с самым большим расстоянием. А ответом на задачу будет минимум из ответов по всем стартовым клеткам.

Итого, для каждой из $n * m$ клеток мы обойдем все поле, и итоговая сложность решения составит $O(n^2 * m^2)$.

Задача I.

Будем считать ответ методом динамического программирования. Пусть $f(x)$ – кол-во последовательностей длины x, в которых никакие две единицы не стоят подряд.

База динамики: $f(0) = 1, f(1) = 2$.

Пересчет динамики. Рассмотрим цифру, которую мы поставили на последнюю позицию:

1) Цифра 0. Тогда на первые $(x - 1)$ позиций можно поставить любую корректную последовательность длины $(x - 1)$. Всего их $f(x - 1)$.
2) Цифра 1. Тогда предпоследняя цифра обязательно должна быть 0, а на первые $(x - 2)$ позиций можно поставить любую корректную последовательность длины $(x - 2)$. Всего их $f(x - 2)$.

Складывая эти два варианта, получаем, что $f(x) = f(x - 1) + f(x - 2)$.

Итоговая сложность – $O(n)$.

Задача J.

Чтобы любое ребро (u, v), не входящее в остов, не входило в него и после добавления ребер до полного графа, необходимо и достаточно того, чтобы оно было больше любого ребра в минимальном остове среди всех ребер на пути от u до v. А чтобы вес конечного графа был минимален, это ребро должно быть как можно меньше, то есть равно весу максимального ребра на пути от u до v, увеличенного на 1.

Воспользуемся системой непересекающегося множества (СНМ). Будем рассматривать ребра остова в порядке возрастания веса. Рассмотрим множества, которые будем соединять текущим ребром. Заметим, что все ребра (кроме текущего), которые можно добавить между этими множествами будут веса текущего ребра, увеличенного на 1, так как все ребра остова в этих множествах не более текущего. То есть на пути из любой вершины одного множества к любой вершине другого множества, максимальное ребро будет не менее текущего, а значит, что вес ребра, концы которого находятся в этих двух вершинах будет как раз вес ребра, объединяющего эти множества, увеличенный на 1. Таким образом, можно посчитать, сколько ребер мы можем добавить между этими двумя множествами (произведение размеров множеств, уменьшенное на 1). А к ответу прибавим это количество, умноженное на вес каждого из данных ребер.

Таким образом, мы за $O(n \cdot \log n)$ (предварительная сортировка ребер) найдем ответ. СНМ в данном случае может работать за константу.

Задача K.

Будем решать задачу методом динамического программирования.

Пусть $f(len, x)$ – это минимальное кол-во операций, за которое можно получить лексикографически отсортированный префикс длины len такой, что последнее число равняется x (для удобства переведем бинарные строки в числа $0..2^m - 1$).

Также будем поддерживать вспомогательную величину $mn(len, x) = min_{i=0..x}(f(len, i))$. Эту величину можно легко поддерживать, следующим образом: $mn(len, x) = \min(mn(len, x-1), f(len, x))$.

База динамики: *f(0, 0) = 0*.

Пересчет динамики: $f(len, x) = mn(len - 1, x) + cost(a_{len}, x)$, где *cost(a, b)* – это количество операций, за которое можно из числа a получить b. Нетрудно заметить, что $cost(a, b) = bits\_count(a \; xor \; b)$, где $bits\_count(x)$

– количество единичных бит в числе x. Количество бит можно заранее предподсчитать для каждого числа x от $0..2^m - 1$.

Оценим скорость работы данного алгоритма: $O(m \cdot 2^m)$ уйдет на предподсчет, и $O(n \cdot 2^m)$ на вычисление всех состояний динамики.

Итоговая сложность – $O((m+n) \cdot 2^m)$.

Задача L.

Для начала найдем максимально возможный НОД всех чисел. Рассмотрим простое число p, которое входит q раз в произведение всех чисел S ($\prod_{n=1}^{N} \vdots p^q$). В итоговом НОД-е всех чисел этот простой множитель p будет встречаться $\left\lfloor \frac{q}{N} \right\rfloor$ раз. Теперь, факторизовав каждое число и сложив степени их простых множителей, мы найдем итоговый НОД.

Для того, чтобы посчитать минимальное количество операций, пройдемся по всем числам, факторизация которых у нас уже есть и определим, на какое количество простых чисел нужно домножить очередное число, чтобы оно стало кратно НОД-у. Суммируя все найденный количества, получим итоговое количество операций.

Оценим время работы алгоритма. $O(U \log\log U)$ необходимо для нахождения всех простых чисел (здесь U - максимально возможное исходное число). $O(N \log U)$ необходимо для факторизации всех чисел.

Итоговая асимптотика – $O(N \log U + U \log\log U)$.