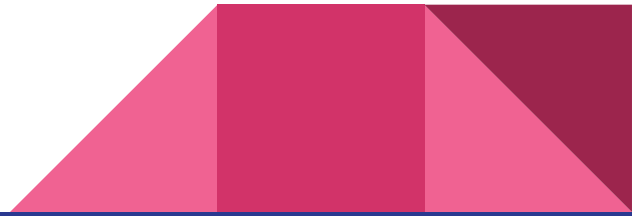
Разбор задач 3-го этапа республиканской олимпиады по информатике 2018 года

Тур 2 Задача 1

Условие

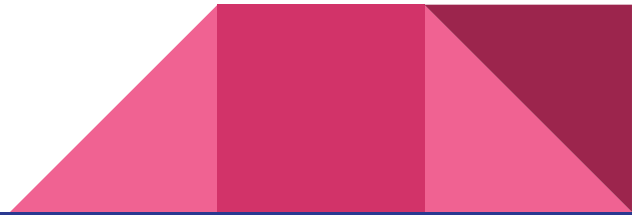
Дана квадратная матрица размера **N**, нужно поменять местами две строки, и два столбца, чтобы числа во всех строках и столбцах шли по возрастанию.



Решение на 40 баллов

Полный перебор. Можно перебрать две строки и два столбца и проверить матрицу на валидность.

Сложность $\mathbf{O(N^6)}$.



Решение на 80 баллов

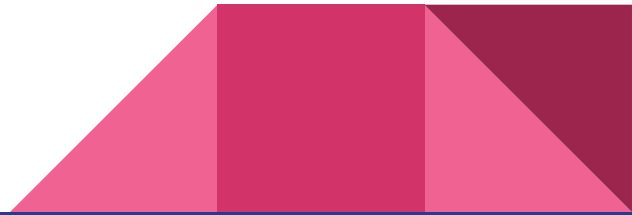
Переберем две строки, которые поменяем местами. Тогда во всех столбцах должен установиться правильный порядок. Отсортируем первую строку если в ней найдется два элемента, которые стоят не на своём месте, то мы обязаны поменять их местами и соответствующие столбцы. Проверяем матрицу на валидность.

Сложность $\mathbf{O(N^4)}$.

Решение на 100 баллов

Заметим, что если мы поменяем местами два столбца, то порядок следования элементов в столбцах не изменится, а если поменяем две строки, то порядок следования элементов в строках не изменится. Следовательно можно решать для строк и столбцов независимо. Находим пару элементов, которые нужно обменять в первой строке и в первом столбце, и получаем ответ.

Сложность $\mathbf{O(N^2)}$.



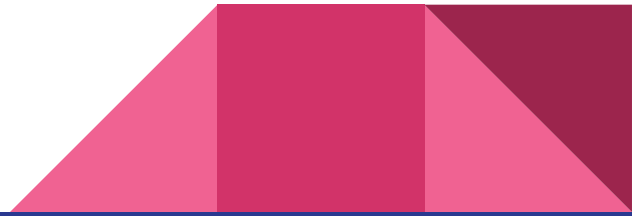
Тур 2 Задача 2

Условия. Нам задана матрица 2^k на 2^k в заданном в условии формате. **k** <= 30. Требовалось найти число единиц в этой матрице.

Решение на 50 баллов

Рекурсивно строим матрицу и затем считаем число в ней.

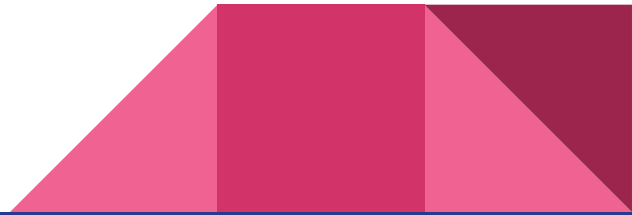
Сложность $\mathbf{O(2^{2k})}$.



Решение на 100 баллов

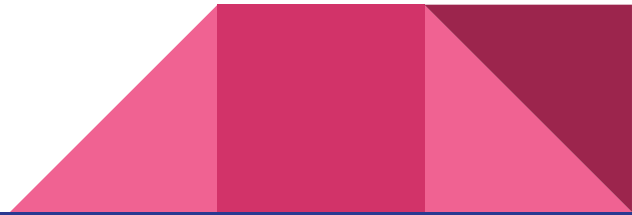
Для каждой 1 мы можем определить квадрат какого размера она заполняет. Достаточно посмотреть на количество незакрытых открывающихся скобок. Путь это число **open.** Тогда, образуется квадрат со стороной $\mathbf{2^{(k - open)}}$.

Сложность **O(N)**.



Тур 2 Задача 3

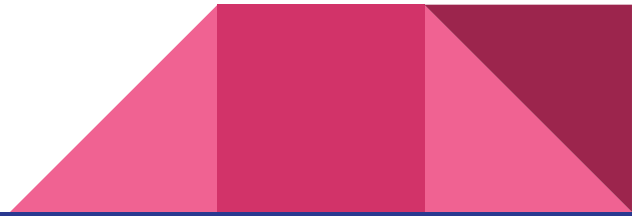
Условие. Была дана последовательность из **N** элементов, у которой было такое свойство: для любого **i** > 1 и меньше либо равного **N** модуль разности a_i с a_{i-1} не превосходит 1. Также было дано **K** запросов. Каждый запрос задавался двумя числами **l** и **r**. Нужно было найти максимальную возрастающую подпоследовательность среди элементов $a_l, a_{l+1}, \dots, a_r$.



Решение на 30 баллов

Делаем то, что просят. Максимальная возрастающая подпоследовательность можно найти с помощью динамического программирования. $\mathbf{f_i}$ - длина максимальной возрастающей подпоследовательности, которая заканчивается в **i**-ом элементе.

Сложность $\mathbf{O(K * N^2)}$.



Решение на 60 баллов

На 60 баллов $\mathbf{a_i}$ <= 100. Запишем другое динамическое программирование. $\mathbf{f_{i,j}}$ - минимальная позиция на которую может оканчиваться возрастающая подпоследовательность длины **i** и последний элемент которой меньше либо равен **j.**

Сложность $\mathbf{O(max^2\{a_1, a_2, \dots, a_n\} * K)}$.

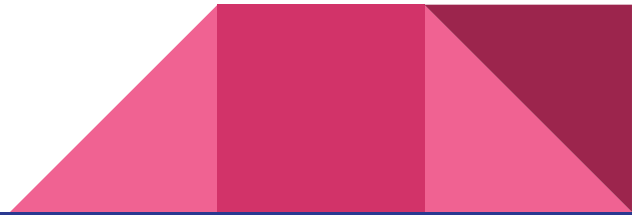
Решение на 100 баллов

Интересный факт. Если есть какие-то **i**, **j** (**i** < **j**) и $a_i < a_j$, то для любого **q** (a_i < **q** < a_j) существует такое **i** < **p** < **j**, что a_p = **q**. Из этого следует, что длина максимальной возрастающей подпоследовательности, которая начинается в **i** и заканчивается в **j** равна $a_j - a_i + 1$. Построим дерево отрезков, где в каждой вершине будем хранить минимальное, максимальное значение на отрезке, а также максимальную возрастающую на этом отрезке. Тогда можно легко обновить ответ.

Сложность **O(N log N + K log N)**.

Тур 2 Задача 4

Условие. Была дана матрица **N** на **M**, а также строка длины **L**. Нужно построить путь в матрице, чтобы буква на первой клетке в пути совпадала с первой буквой строки, вторая буква пути совпадала с второй буквой строки и т.д. Путь - какая-то последовательность не обязательно соседних клеток матрицы. Длина пути - суммарное манхэттенское расстояние между соседними клетками в пути. Требуется максимизировать длину пути.

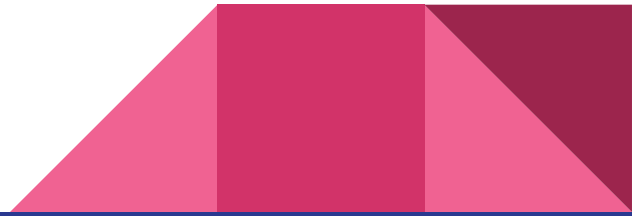


Решения

Существует много разных решений, основанных на переборе и случайных подходах. Разберём полные решения. Напишем динамическое программирование $\mathbf{f_{i,j,k}}$ **-** максимальная длина первых **k** прыжков, что в итоге мы окажемся в клетке **i, j**. Это динамическое программирование можно посчитать за **O(N * M * L)**. Чтобы восстановить порядок нужно **n * m * l** памяти, что может не поместиться в оперативную память. Существуют три подхода для решения задачи.

Первый подход

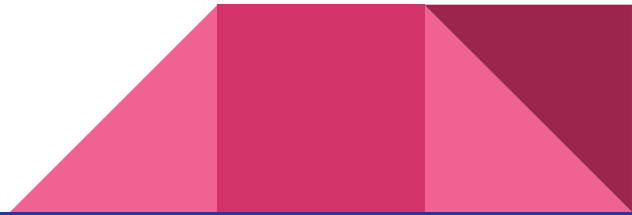
Можно доказать, что можно оставить в каждой строчке только первое и последнее вхождение каждой буквы и прыгать только между ними. Сложность решения это не изменит, но поможет значительно сократить память.



Второй подход

Пересчитывая матрицу по слоям найдем оптимальный ответ и запомним **W-**ую позицию в обходе, а также стартовую и финишную. Тогда наш путь разделится на две части от старта к **W** и от **W** к финишу. Запустимся от этих путей рекурсивно. Будем делить путь, пока не сможем полностью сохранить путь в динамике.

Сложность **O(N * M * L * ln L)**.



Третий подход

Возьмем все вхождения некоторой буквы и построим по ним выпуклую оболочку. Можно доказать, что в оптимальном ответе будут только точки, которые вошли в оболочку. Воспользуемся фактом, что математическое ожидание числа точек в выпуклой оболочке, если точки случайно будут расставлены на поле **N** на **M** это **log N + log M**. Тогда можно написать всё тоже динамическое программирование, но за $\mathbf{O(L * (\log N + \log M)^2)}$. Можно ещё использовать факт, что если у всех точек целые координаты то размер выпуклой оболочки не превосходит **O(sqrt(N * M))**.