Всероссийская олимпиада школьников по информатике 2024–2025

Региональный этап

Разбор задач

Условия задач, тесты, решения и разбор задач подготовили Александр Бабин, Николай Ведерников, Екатерина Ведерникова, Мария Жогова, Владимир Рябчун, Маргарита Саблина, Андрей Станкевич, Григорий Шовкопляс, Егор Юлин.

Ценные замечания по результатам тестирования задач сделали Тимур Дегтярев, Илья Кондаков, Виталий Курин, Игорь Маркелов, Григорий Солнышкин, Максим Туревский, Максим Шевкопляс, Ксения Шкулева.

Задача 1. Кузнечик 2D

Подзадача 1

Заметим, что при $k = 1$ любые два хода вверх и вправо можно заменить на один ход по диагонали. Тогда до тех пор, пока кузнечик не дойдет до n-й строки или m-го ряда, будем двигать его по диагонали, а затем вправо или вверх, сколько будет нужно. Для $n, m \leqslant 10$ можно реализовать такое решение итеративно, используя цикл.

Асимптотика такого решения $O(n + m)$.

Подзадача 2

Для решения данной подзадачи нужно модифицировать решение прошлой подзадачи для $k > 1$. Заметим, что выгодно всегда ходить на максимальное возможное число клеток (не выходя за пределы поля). Алгоритм ходов кузнечика будет аналогичным, но нужно дополнительно учесть, что последний ход по диагонали может быть меньше k. Это можно сделать, например, вычисляя координаты после следующего хода по формулам: $x' = min(n, x + k)$, $y' = min(m, y + k)$.

Асимптотика такого решения $O(n + m)$.

Подзадача 3

На самом деле для решения случая $k = 1$ можно не использовать цикл, а придумать формулу. Максимальное число ходов по диагонали будет равно $min(n, m)-1$, а оставшихся ходов нужно будет сделать $max(n, m) - min(n, m)$. Итоговый ответ будет равен $max(n, m) - 1$.

Асимптотика такого решения $O(1)$.

Подзадача 4

По условию данной подзадачи ответ всегда равен 1 или 2, поэтому в решении можно использовать следующий подход. Если можно переместить кузнечика за один ход, выведем в ответ 1, иначе выведем 2.

Как понять, что можно переместить кузнечика за один ход? Заметим, что это значит, что либо $n = 1$, либо $m = 1$, либо $n = m$, так как за один ход нам доступно только одно направление движения. Тогда остается проверить, что в рамках заданного направления расстояние не превосходит k клеток, что можно сделать по формуле: $max(n, m) - 1 \leqslant k$.

Асимптотика такого решения $O(1)$.

Подзадача 5

Совместим идеи решения второй и третьей подзадач, а именно, используя алгоритм ходов из второй подзадачи, придумаем формулу количества ходов, как в третьей.

Максимальное число ходов по диагонали будет равно $diagonal = \lceil \frac{min(n,m)-1}{k} \rceil$, а оставшихся ходов нужно будет сделать $rest = \lceil \frac{max(n,m)-min(n,m)}{k} \rceil$. Итоговый ответ будет равен $diagonal + rest$.

Асимптотика такого решения $O(1)$.

Задача 2. Простоватые числа

Подзадача 1

Переберем все числа от l до r и каждое проверим, является ли оно простоватым или нет.

Подзадача 2

Заметим, что простоватые числа имеют вид: все цифры, кроме одной – единицы, а эта цифра – 2, 3, 5 или 7.

Сгенерируем все числа такого вида длины не более 18 и для каждого проверим, входит ли он в диапазон от l до r.

Подзадача 3

Количество чисел длины d равно $4 \cdot d$, так как на каждую из d можем поставить одну из четырех цифр.

Тогда количество чисел длиной от 1 до k равно $\sum_{d=1}^{k} = 4 \cdot d = 4 \cdot d \cdot (d+1)2 = 2 \cdot d \cdot (d+1)$.

Подзадача 4

Чтобы найти количество чисел на отрезке от l до r, найдём количество чисел от 1 до r и вычтем количество чисел от 1 до $l-1$.

Пусть мы хотим подсчитать количество чисел от 1 до s.

Для начала добавим к ответу все числа длины меньше $len(s)$, что мы умеем решать в подзадаче 3.

Осталось понять, сколько чисел длины $len(s)$ будут подходить. Для этого надо перебрать позицию i и проверять, можем ли мы поставить цифры 2, 3, 5 и 7. То есть формировать строки вида $1 \dots 1p1 \dots 1$, где p — 2, 3, 5 или 7. И сравнивать с s. Сколько строк меньше, столько и добавим к ответу. Такое решение будет работать за $O(n^2)$, так как строк кандидатов у нас $4 \cdot n$ и сравнение двух строк за $O(n)$.

Полное решение

Оптимизируем последний шаг из решения подзадачи 4. Для этого можно заметить, что есть несколько случаев, в которых мы не можем поставить цифру d на позицию i.

- Первые j цифр числа единицы, а $s[j+1] = 0$, где $s[j+1]$ — цифра на $j+1$ месте, и $j+1 < i$.
- Первые $i-1$ цифр числа единицы, а дальше $s[i] < d$.
- Первые $i-1$ цифр числа единицы, а дальше $s[i] = d$, а дальше идут несколько единиц подряд и сразу после них ноль.

Во всех остальных случаях мы сможем поставить цифру d на позицию i. Такие проверки мы можем осуществить во время прохода по строке s. Итоговое время работы будет $O(n)$.

Задача 3. Кислотные дожди

Подзадачи 1 и 2

Для решения первой и второй подзадач достаточно явно поддерживать сегменты и вычислять ответ по указанной формуле.

Структуры данных

Во всех следующих подзадачах необходимо эффективно поддерживать границы сегментов. Авторское решение использовало декартово дерево по неявному ключу. Альтернативный подход — для каждого блока i поддерживать номер сегмента s_i, к которому он принадлежит. Тогда границы сегмента k можно найти при помощи двоичного поиска, а для объединения сегментов нужно вычесть 1 на суффиксе массива s.

Подзадача 4

Для решения подзадачи 4 для вычисления ответа на сегменте $[l; r]$ нужно было найти подотрезок блоков $[l'; r']$, такой что $l \leqslant l' \leqslant r' \leqslant r$ и $\forall i \in [l'; r'] : h_i \leqslant \min(h_l, h_r)$. Тогда ответ будет равен $(r' - l' + 1) \cdot \min(h_{l'}, h_{r'}) - \sum_{i=l'}^{r'} h_i$. l' и r' можно находить при помощи двоичного поиска.

Важные формулы

Для решения остальных подзадач нужно модифицировать формулу для вычисления ответа. Воспользуемся следующим свойством: $\min(a, b) = a + b - \max(a, b)$.

$$w = d_1 + d_2 + \ldots + d_n = \sum_{i=1}^{n} \min(l_i, r_i) - h_i = \sum_{i=1}^{n} l_i + r_i - \max(l_i, r_i) - h_i$$

Важное наблюдение: $\max(l_i, r_i) = \max\{h_1, \ldots, h_n\}$. Обозначим за M максимум в сегменте. Итоговая формула будет иметь вид:

$$w = \sum_{i=1}^{n} l_i + r_i - \max(l_i, r_i) - h_i = \sum_{i=1}^{n} l_i + \sum_{i=1}^{n} r_i - \sum_{i=1}^{n} h_i - n \cdot M$$

Теперь задача состоит в поддержании этих слагаемых при объединении отрезков. Максимум и сумма обновляются тривиально. Осталось научиться вычислять $\sum l_i$ и $\sum r_i$.

Подзадача 3

В подзадаче 3 количество различных значений l_i и r_i было достаточно маленьким. Значения l_i и r_i монотонны, поэтому вместо поддержания суммы можно было хранить самые левые и правые вхождения каждого возможного числа(а их 10). Например, массиву $[2, 1, 1, 5, 2]$ будет соответствовать массив $l = [2, 2, 2, 5, 5]$, а позиции первых значений будут $[1, 0, -1, -1, 3, -1, \ldots]$. По нему легко вычислить $\sum l_i$ и $\sum r_i$.

Подзадача 5

Подзадача 5 позволяла значительно упростить объединения сегментов, поскольку блоки всегда добавлялись в конец первого сегмента. Поддержание $\sum l_i$ тривиально. Пусть наш текущий сегмент состоит из элементов $h_1, \ldots, h_n$, а мы хотим добавить элемент x в конец. Тогда какой-то суффикс значений r_i станет равен x, а остальная часть не поменяется. Эффективно поддерживать эти значения можно при помощи стека максимумов. Значения на стеке будут убывать, самое верхнее будет $r_n = h_n$. Храня на стеке значение r_i и длину отрезка этих элементов можно быстро обновлять сумму r_i. Каждый элемент будет помещён на стек ровно 1 раз, поэтому суммарная асимптотика будет $\mathcal{O}(n)$.

Полное решение

Для полного решения задачи нужно развить идею поддержания $\sum r_i$ из подзадачи 5 и научиться обновлять значения обеих сумм при добавлении одного элемента слева или справа. Авторское решение предполагало использование двух деков, в подзадаче 6 можно было использовать `std::set`. Поскольку для этого необходимо явно перемещать блоки между сегментами, требуется всегда перемещать элементы из меньшего сегмента в больший. Это гарантирует $\mathcal{O}(\log n)$ перемещений любого блока и итоговую асимптотику $\mathcal{O}(n \log n)$.

Задача 4. Поиск сокровищ

Подзадача 6

Для решения данной подзадачи можно было использовать перебор. Так как $n \cdot k \leqslant 25$, то мы можем перебрать все доступные поля и проверить их на корректность. Это будет работать за $O(2^{nk}nk)$.

Подзадача 1

Для решения данной подзадачи можно было заметить, что в соседних столбцах область сканирования пересекается ровно по одной клетке. Если же столбцы не являются соседними, то область сканирования не пересекается. Тогда количество корректных таблиц будем считать с помощью динамического программирования.Пересчет будет через предыдущую позицию. Мы знаем, какую сумму нам необходимо набрать и чему равно значение клетки в предыдущем столбце. В таком случае для текущего столбца мы можем перебрать, какое значение будет в каждой строке. Итоговое время работы будет равно $O(n)$.

Подзадачи 2-3

Для решения данных подзадач можно было использовать то, что сканер сканирует не более чем k клеток назад, тогда мы можем в динамике сохранять позицию последних k^2 клеток, а при переходе перебирать, какие клетки будут заняты в текущем столбце, тогда размер динамики будет $O(2^{k^2}n)$, переход будет работать за $O(2^k)$, а тогда итоговое время работы будет равно $O(2^{k^2+k}n)$.

Подзадачи 4-5

Для решения данной подзадачи воспользуемся решением предыдущей подзадачи, но заметим, что нам необходимо хранить не все клетки в предыдущих k столбцах. Пусть мы сейчас рассматриваем позицию i, тогда нам нужны верхняя $k-1$ ячейка в столбце $i-1$, верхние $k-2$ ячейки в столбце $i-2$ и так далее. Тогда необходимо хранить $\dfrac{k \cdot (k-1)}{2}$ ячеек. Для перехода в динамике так же будем перебирать расстановку ячеек в текущем столбце, тогда размер динамики будет равен $O(2^{\frac{k\cdot(k-1)}{2}}n)$, переход работает за $O(2^k)$, а итоговое время работы равно $O(2^{\frac{k\cdot(k-1)}{2}+k}n)$ или же $O(2^{\frac{k\cdot(k+1)}{2}}n)$.

Полное решение

Для полного решения задачи заметим следующее: для пересчёта в подзадачах 4-5 мы использовали только клетки в треугольнике, которые стоят на диагонали. В столбце i будет не более i клеток, тогда будем хранить сумму в каждом столбце, всего таких состояний будет не более $k!$. Тогда переход будет немного отличаться. Мы всё ещё можем перебирать маску для текущего столбца, но так как мы храним только сумму в столбце, то перебирать маску для текущего столбца становится бесполезным. Тогда будем перебирать маску того, какие значения находятся в массиве на диагонали треугольника. И зная сумму по всем предыдущим столбцам, сумму в маске и необходимую сумму, мы можем сказать, какая сумма останется в текущем столбце. Тогда переходом у нас будет перебор маски диагонали треугольника. Время работы будет $O(k!2^k n)$.

Задача 5. Разность квадратов

Подзадача 1

Напишем функцию `isSquare(x)`, которая будет проверять, является ли число x полным квадратом целого числа.

Двумя вложенными циклами переберем все возможные пары чисел, проверим, являются ли эти числа полными квадратами, и равна ли разность между ними d. Если все условия выполняются, увеличим ответ на один. Время работы такого алгоритма $O(r^2)$.

Подзадача 2

Оптимизируем предыдущее решение. Будем запускать второй цикл только в том случае, если первое число полный квадрат. Так как полных квадратов, которые меньше либо равны r, не более чем $\sqrt{r}$, то итоговое время работы $O(r\sqrt{r})$.

Подзадача 3

Теперь будем перебирать циклами только полные квадраты чисел. Каждый цикл будет работать за $\sqrt{r}$. Тогда итоговое время работы $O(\sqrt{r} \cdot \sqrt{r}) = O(r)$.

Подзадача 4

Применим ещё оптимизацию. Будем перебирать полные квадраты числа. Пусть a — полный квадрат, тогда нам надо проверить, правда ли число $a - d$ является полным квадратом, а так же лежит в диапазоне от l до r. Такое решение будет работать за $O(\sqrt{(r)})$.

Полное решение

Рассмотрим уравнение $x^2 - y^2 = d$. Преобразуем $(x - y)(x + y) = d$. Получается, что $x - y$ и $x + y$ делители числа d. Переберем делители числа d за $O(\sqrt{d})$ и проверим, что получившиеся x^2 и y^2 удовлетворяют условию задачи.

Задача 6. Перекошенное разбиение

Подзадача 1

В первой подзадаче были ограничения $n \leqslant 15$, поэтому задачу можно решить рекурсивным перебором. В рекурсивной функции будем передавать текущую сумму на суффиксе, количество подотрезков, а так же максимальную и минимальную сумму на подотрезках разбиения. При переходе нужно перебрать два варианта — новый элемент продолжает отрезок-суффикс или начинает новый. Получаем решение за $O(2^n)$.

Подзадача 2

Во второй подзадаче $k = 2$, поэтому достаточно перебрать префикс, который будет образовывать первый отрезок, после чего найти сумму на префиксе и на суффиксе и обновить ответ. Сумму на префиксе можно поддерживать в переменной, а сумму на суффиксе можно вычислить как сумма всего массива минус сумма префикса.

Ключевая идея решения

Попробуем понять, как выглядит оптимальный ответ. Рассмотрим оптимальное разбиение массива на k подотрезков. Не умаляя общности, будем считать, что отрезок, на котором достигается максимальная сумма, находится левее отрезка, на котором достигается минимальная сумма (иначе можно решить задачу для развернутого массива).

Заметим, что длина любого отрезка в таком разбиении не превышает $n - k + 1$. Рассмотрим следующий процесс:

1. Если длина отрезка с максимальной суммой равняется $n - k + 1$ — закончить процесс.
2. Иначе, если отрезок с максимальной суммой не является первым (префиксом), отобрать последний элемент у предыдущего подотрезка разбиения и отдать его подотрезку с максимальной суммой. В случае, если он был длины один, то разбить любой другой отрезок длины больше чем один на два отрезка.
3. Иначе, если между отрезком с максимальной суммой и отрезком с минимальной суммой есть хотя бы один другой отрезок, проделать с ним ту же операцию.
4. Иначе, если отрезок с минимальной суммой имеет длину хотя бы два, отдать его первый элемент отрезку с максимальной суммой.

Заметим, что на каждом шаге такого процесса значение перекоса разбиения не уменьшается, и отрезок с максимальной суммой растет по длине. Таким образом, данный процесс точно завершится, и оптимальный ответ в итоге процесса будет одним из двух:

1. Максимальный отрезок имеет длину $n - k + 1$, а остальные отрезки длину один.
2. Максимальный отрезок является префиксом массива от 1 до i, а минимальный отрезок имеет длину один и состоит из элемента на позиции $i + 1$.

Таким образом, для решения задачи нужно разобрать каждый из двух возможных случаев для массива из входных данных, а так же для развернутого массива из входных данных и выбрать лучший ответ.

Подзадача 3

В третьей подзадаче выполняется ограничение $k = 3$, поэтому легко разобрать первый случай: один отрезок будет иметь длину $n - 2$, и остается разобрать случаи возможного расположения двух единичных отрезков. Второй случай с префиксом же решается за линейное время нахождением суммы на каждом префиксе.

Подзадачи 4 – 5

В подзадачах 4 и 5 достаточно решить задачу за время $O(n^2)$. Для этого можно перебрать отрезок длины $n - k + 1$ и наивно найти сумму на нем и минимум из остальных элементов. Так же нужно разобрать второй случай: перебрать каждый префикс и обновить ответ.

Полное решение задачи

Для полного решения задачи нужно научиться быстро находить сумму на каждом отрезке длины $n-k+1$ и минимум среди остальных элементов. Для этого предподсчитаем для массива префиксные суммы, префиксные минимумы и суффиксные минимумы за линейное время. Зная это, можно за $O(1)$ найти сумму на отрезке длины $n - k + 1$, минимум на префиксе и на суффиксе и обновить ответ. Получаем решение задачи за линейное время.

Альтернативное решение с использованием динамического программирования

Так же некоторые подгруппы можно было пройти, решив задачу с использованием динамического программирования без ключевой идеи задачи. Заметим, что перекос разбиения равен максимальной сумме подотрезка минус минимальной сумме подотрезка, с другой стороны, для максимизации перекоса оптимально прибавить максимальную сумму и вычесть минимальную. Таким образом, можно про перекос думать так: мы хотим прибавить одну из сумм на подотрезке и вычесть одну из сумм на подотрезке разбиения.

Теперь можно сделать динамическое программирование, которое будет хранить в состоянии размер префикса, количество отрезков, на которое он разбит, и два флага: прибавляли мы уже максимальную сумму или нет, и вычитали мы уже минимальную сумму или нет. Тривиальная реализация такого решения работает за время $O(n^2k)$, но можно оптимизировать это решение, получив решение за $O(nk)$.

Задача 7. Главное правило личных олимпиад

Подзадача 1

В этой подзадаче требовалось набрать сумму ровно s в одной задаче. Это классическая задача о рюкзаке. Решим её за $\mathcal{O}(k_1 \cdot s)$.

Подзадача 2

Предсосчитаем $dp_1[s]$ — все возможные суммы баллов, которые можно набрать только с помощью первой задачи за $\mathcal{O}(k_1 \cdot s)$. Предсосчитаем $dp_2[s]$ — все возможные суммы баллов, которые можно набрать только с помощью второй задачи за $\mathcal{O}(k_2 \cdot s)$.

Теперь переберем $0 < x < s$ — сумма баллов по первой задаче. Тогда во второй надо набрать $s - x$ баллов. Если $dp_1[x] = True$ и $dp_2[s - x] = True$, то можно набрать заданную сумму баллов.

Итоговая сложность: $\mathcal{O}(\sum k_i \cdot s)$.

Подзадача 3

Напишем рекурсивный перебор за $2^{\sum k_i} \cdot \sum k_i$. Будем для каждой задачи поддерживать количество выбранных подзадач и набранную сумму.

Итоговая сложность: $\mathcal{O}(2^{\sum k_i} \cdot \sum k_i)$.

Подзадача 4

Если все $k_i = 1$, а по условию мы должны взять хотя бы одну подгруппу из каждой задачи, то в этой подзадаче мы обязаны взять все существующие подзадачи. А значит, достаточно проверить, что $\sum c_{i,1} = s$.

Итоговая сложность: $\mathcal{O}(n)$.

Подзадача 5

Будем поддерживать баллы, которые умеем получать, рассмотрев первые i задач.

Научимся добавлять задачу. Насчитаем для новой задачи множество баллов, доступное для набора без учета 0. Это легко сделать с помощью динамического программирования.

Теперь нужно объединить полученные данные с предыдущими задачами. Это можно сделать за m^2. Переберем доступные баллы в новой задаче s_1 и доступные баллы в предыдущих s_2. Теперь мы научились набирать $s_1 + s_2$, используя первые $i + 1$ задач.

Итоговая сложность: $n \cdot m^2$.

Подзадача 6

Давайте посчитаем $dp[i][j]$ — можно ли набрать первыми i задачами сумму ровно j.

Будем последовательно перебирать подзадачи задачи i. Пусть стоимость текущей подзадачи k, и мы хотим её решить. Тогда в $dp[i][j]$ можно прийти двумя способами.

1. Это первая решённая подзадача в данной задаче — сделаем переход из предыдущей задачи $dp[i-1][j-k]$.
2. Это не первая решённая подзадача в данной задаче — сделаем переход из текущей задачи $dp[i][j-k]$.

Чтобы правильно посчитать dp, необходимо перебирать j в порядке убывания.

Итоговая сложность: $\mathcal{O}(\sum k_i \cdot s)$.

Задача 8. Туристический маршрут

Введение

Ключевое значение в задаче играет следующий факт. Допустим, что цикл имеет длину $2n+2m-4$ и проходит через клетки $(1,1)$ и (n,m), тогда при удалении этих клеток из цикла образуются два пути. Один из этих путей соединяет клетки $(1,2)$ и $(n-1,m)$, другой путь соединяет клетки $(2,1)$ и $(n,m-1)$, при этом сами пути не пересекаются.

Таким образом, вместо подсчета числа циклов можно вычислять количество пар таких путей. В дальнейшем путь $(2,1) \to (n,m-1)$ будем называть *нижней дугой*, а путь $(1,2) \to (n-1,m)$ — *верхней дугой*. Также не забудем, что у цикла по условию задачи есть ориентация, так что итоговый ответ во всех случаях надо будет удвоить.

Клетки, которые содержат достопримечательности, будем называть *отмеченными*.

Подзадача 1

Заметим, что при $n = 3$ существует только $O(m^2)$ *интересных* циклов. Это так, потому что верхняя и нижняя дуга содержат по одному вертикальному переходу и $m-1$ горизонтальный переход.

Таким образом, для решения этой подгруппы достаточно перебрать m^2 пар дуг и проверить каждую дугу на корректность за время $O(m)$. Проверка на корректность включает в себя:

- Проверка на то, что дуги не пересекаются.
- Проверка на то, что дуги проходят через все отмеченные клетки (кроме, возможно, клеток $(1,1)$ и (n,m), так как по нашему определению дуги через эти клетки не проходят, но при этом понятно, что такие клетки все равно будут включены в любой рассматриваемый цикл).

Подзадачи 2–3

Для решения этих подзадач требовалось как-нибудь, необязательно оптимально, перебрать всевозможные пары дуг и проверить, что они формируют *интересный цикл*. С учетом $n, m \leqslant 8$ для каждой дуги можно было независимо перебрать *маску переходов* длины $n+m-1$ (каждый горизонтальный переход можно закодировать цифрой 0, а каждый вертикальный — цифрой 1). Требовалось проверить следующие условия:

- Количество единиц в каждой из масок равняется $n-1$.
- Дуги не пересекаются. Этот факт можно было проверить наивно, выписав все клетки верхней и нижней дуги, а затем проверив всевозможные пары на совпадение.
- Каждая отмеченная клетка содержится в одной из двух дуг.

Без дополнительных оптимизаций такое решение с запасом работает при $n, m \leqslant 5$. Для решения подзадачи 3 требовалось оптимизировать этот перебор. Например, можно было поступить следующим образом:

- Вычислить список возможных дуг, это можно сделать за $O(2^{n+m-2})$, если перебрать все маски переходов и выписать в массив только те, которые содержат верное количество единиц. Количество таких путей не будет превышать $\binom{12}{6} = 924$.
- Для каждого пути вычислить битовую маску клеток, которые посещает этот путь (поле содержит не более чем 64 клетки, так что для хранения таких масок хватает беззнакового 64-битного числа).
- Наивно перебрать всевозможные пары путей и проверить каждую из них за $O(1)$.

Ясно, что перебор, организованный таким образом, работает за время $O((n+m) \cdot 2^{n+m} + \binom{n+m-4}{n-2}^2)$.

Подзадачи 4–5

Для решения этих подзадач требовалось воспользоваться методом динамического программирования. Ясно, что требуется перебрать всевозможные пары дуг, которые посещают все отмеченные клетки. Основной проблемой является условие на то, чтобы эти дуги не пересекались. Для того, чтобы легко и просто учитывать это условие, в переходах динамического программирования будем одновременно продлевать оба пути из пары.

А именно предлагается вычислить массив $\mathrm{dp}[s][x_1][y_1][x_2][y_2]$ — количество пар непересекающихся путей, первый из которых соединяет клетки $(1, 2)$ и (x_1, y_1), а второй — клетки $(2, 1)$ и (x_2, y_2).

- Для того, чтобы корректно обрабатывать переходы будем вычислять только состояния, для которых выполняются $x_1 + y_1 = x_2 + y_2$. Тогда переходы будут иметь вид:

$$\begin{aligned}\mathrm{dp}[s][x_1][y_1][x_2][y_2] \;=\; &\mathrm{dp}[s'][x_1-1][y_1][x_2-1][y_2] + \mathrm{dp}[s'][x_1-1][y_1][x_2][y_2-1] + \\ &+\; \mathrm{dp}[s'][x_1][y_1-1][x_2-1][y_2] + \mathrm{dp}[s'][x_1][y_1-1][x_2][y_2-1]\end{aligned}$$

- Число s в состояниях динамики вычисляется в соответствии с его определением, таким образом $s - s'$ равняется количеству отмеченных клеток среди пары клеток (x_1, y_1) и (x_2, y_2).

Такое решение работает за $O(kn^2m^2)$ и не является решением ни для одной из этих двух подгрупп. Для того, чтобы решить эти две подгруппы требовалось заметить следующие оптимизации:

- Заметить, что число y_2 в состоянии лишнее, так как ненулевыми являются только те состояния динамики, где выполнено $x_1 + y_1 = x_2 + y_2$, таким образом всегда можно восстановить число y_2.
- Число s также является лишним в состоянии динамики, так как требуется посетить все отмеченные клетки. Для того, чтобы избавиться от этого измерения заметим, что все отмеченные клетки можно распределить по диагоналям вида $x + y = \mathrm{const}$. Теперь достаточно полагать значения динамики $\mathrm{dp}[x_1][y_1][x_2]$ полагать равными нулю, если клетки (x_1, y_1) и (x_2, y_2) не покрывают все отмеченные клетки на диагонали $x_1 + y_1$.

После озвученных оптимизаций мы получили решение, которое работает за время $O(n^2m)$.

Подзадача 6

Это первая подзадача, для решения которой требовалось придумать комбинаторную идею, которая предполагает линейное время работы программы в зависимости от n и m. Рассмотрим всевозможные пары дуг (может быть, пересекающиеся), их количество равно:

$$\binom{n+m-4}{n-2} \cdot \binom{n+m-4}{n-2}$$

Отметим, что биномиальные коэффициенты легко вычислять за $O(1)$, если предпосчитать массив факториалов и обратных факториалов по модулю $10^9 + 7$. Но среди подсчитанных пар путей могли оказаться пары пересекающихся путей, поэтому давайте вычислим их количество, а затем вычтем их из ответа.

Утверждается, что количество таких путей равно:

$$\binom{n+m-2}{n-4} \cdot \binom{n+m-4}{n-1}$$

.

Примечание. Биномиальные коэффициенты вида $\binom{n}{k}$ здесь полагаются равными 0, при $k < 0$ или $k > n$.

Действительно, рассмотрим любую пару пересекающихся путей $P = P_1, \ldots, P_{n+m-3}$ и $Q = Q_1, \ldots, Q_{n+m-3}$. Пусть k — минимальное число k, такое что $C = P_k = Q_k$. Тогда такой паре путей можно сопоставить другую пару путей, которая выглядит следующим образом:

$$X = P_1, \ldots, P_{k-1}, C, Q_{k+1}, \ldots, Q_{n+m-3}$$
$$Y = Q_1, \ldots, Q_{k-1}, C, P_{k+1}, \ldots, P_{n+m-3}$$

Заметим, что путь X соединяет клетки $(1, 2)$ и $(n, m-1)$, а путь Y — клетки $(2, 1)$ и $(n-1, m)$. Любая такая пара путей гарантированно пересекается, то есть имеет общую клетку C. А это значит, что для каждой такой пары можно точно также найти минимальный индекс k: $X_k = Y_k$ и, выполнив обратное преобразование, восстановить пару пересекающихся путей P и Q.

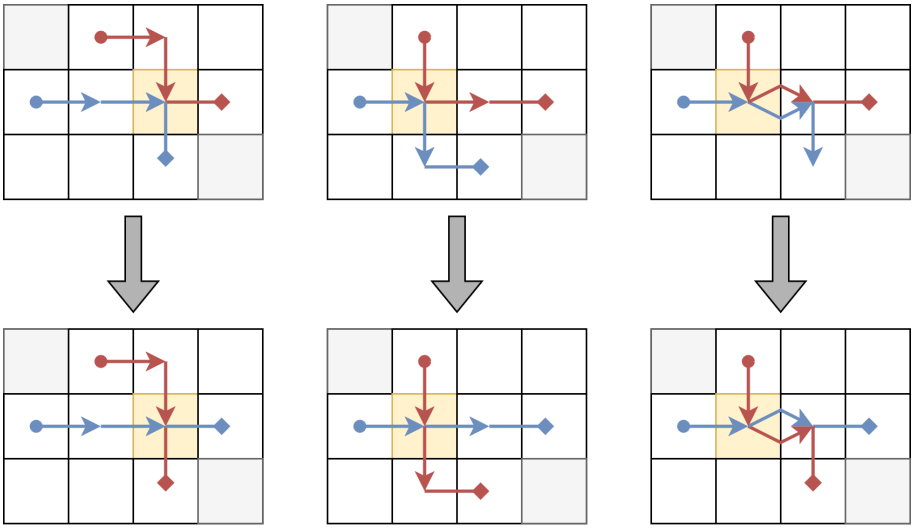

Иллюстрация описанной биекции в случае $n = 3$ и $m = 4$.

Только что мы построили биекцию между парами путей (X, Y) и парами пересекающихся путей (P, Q). Таким образом, размеры этих множеств совпадают, что доказывает предложенную ранее формулу.

Примечание от автора задачи. Та же самая идея применяется в доказательстве соотношения $C_n = \binom{2n}{n} - \binom{2n}{n-1}$, где C_n — n-е число Каталана, которое по определению равно количеству правильных скобочных последовательностей длины $2n$. Эта рассматриваемая подзадача также выступала в роли самостоятельной задачи на олимпиаде ВКОШП в 2022-м году.

Подзадача 7

Для решения этой подзадачи надо было осознать, как идея предыдущей подзадачи обобщается, если есть ограничение на то, что нужно посетить какую-то клетку. Для удобства введем функцию $K(A, B)$, которая принимает на вход пару точек (A, B), а на выходе выдает количество путей, которые начинаются в A и заканчиваются в B.

Для этой и последующих подзадач обозначим ключевые клетки: $S_1 = (2, 1)$, $S_2 = (1, 2)$, $T_1 = (n, m-1)$, $T_2 = (n-1, m)$.

Пусть отмечена клетка L, тогда, если L совпадает с $(1, 1)$ или (n, m), то она будет посещена автоматически, так что ответ на задачу можно вычислить, решив предыдущую подзадачу.

В ином случае, возможны два варианта, либо путь P: $S_1 \to T_1$ содержит клетку L, либо путь Q: $S_2 \to T_2$ содержит L. Суммарное количество способов выбрать, на каком из путей будет лежать клетка L, и какие именно это будут пути будет равно:

$$K(S_1, L) \cdot K(L, T_1) \cdot K(S_2, T_2) + K(S_2, L) \cdot K(L, T_2) \cdot K(S_1, T_1)$$

Проблемы у формулы в отличие от предыдущего случая уже две:

- Учитываются пересекающиеся пары путей P и Q.
- Пути, которые пересекаются в клетке L, учитываются дважды.

Примечание В разборе подзадачи 10 будет указано, что никакого «нового» решения не требуется. Тем не менее, если вы дочитали разбор до этого момента, то, должно быть, заметили, что слова и страницы в данном разборе не экономятся. Поэтому разбор этой подзадачи будет содержать в том числе те рассуждения, которые не используются в основном решении.

Чтобы пути, которые пересекаются в клетке L, не учитывались дважды, вычтем все такие пары путей, их количество равно:

$$K(S_1, L) \cdot K(L, T_1) \cdot K(S_2, L) \cdot K(L, T_2)$$

Только что мы научились отсекать среди всех пар путей только те пары, в которых ровно один из двух путей проходит через клетку L. Заметим, что рассуждения для предыдущей подзадачи продолжают работать и при таком условии. Построенная биекция будет иметь вид:

- Пусть (P, Q) — пара путей, в которой один из путей содержит клетку L.
- Пусть k — минимальный индекс $P_k = Q_k = C$.
- Поменяем суффиксы путей, идущие после C, местами и получим пару путей (X, Y).
- Заметим, что (X, Y) это пара путей между клетками (S_1, T_2) и (S_2, T_1), в которой ровно один из двух путей содержит клетку L.

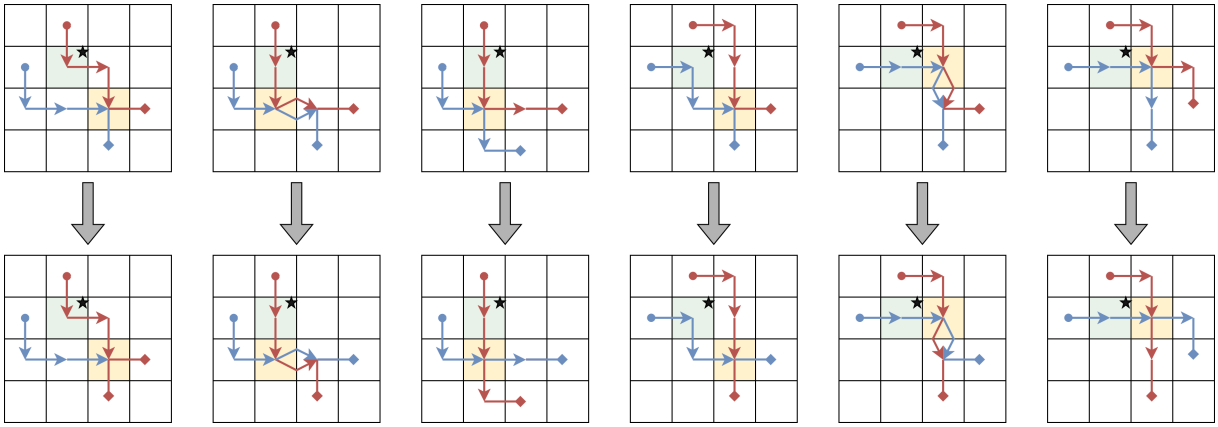

Иллюстрация описанной биекции в случае $n = m = 4$ и $L = (2, 2)$.

Количество путей (X, Y) вычисляется аналогично предыдущей подзадаче. Для того, чтобы в коротком виде записать формулу для решения этой подзадачи введем обозначение $[P_1, \ldots, P_k] = K(P_1, P_2) \cdot \ldots \cdot K(P_{k-1}, P_k)$.

Таким образом, ответ на задачу равен:

$$\begin{aligned}\text{ans} = [S_1, L, T_1] \cdot [S_2, T_2] + [S_1, T_1] \cdot [S_2, L, T_2] - [S_1, K, T_1] \cdot [S_2, K, T_2] - \\ - [S_1, L, T_2] \cdot [S_2, T_1] - [S_1, T_2] \cdot [S_2, L, T_1] + [S_1, K, T_2] \cdot [S_2, K, T_1]\end{aligned}$$

Подзадача 8

Обобщим идею из решения предыдущей подзадачи. Пусть отмечены клетки $L = \{L_1, \ldots, L_k\}$, заранее из этого списка вырежем клетки $(1, 1)$, (n, m). Теперь за $O(k \cdot 2^k)$ для каждого B, подмножества клеток L, вычислим $K(S_i, B, T_j)$ — количество путей из S_i в T_j, которые проходят через все клетки B.

Далее предлагается вычислить $U(S_i, B, T_j)$ — количество путей $S_i \to T_j$, которые проходят через клетки из множества B, но не проходят через клетки из множества $L \backslash B$. Это можно сделать с помощью динамического программирования за время $O(3^k)$ или за время $O(k \cdot 2^k)$ с помощью преобразования мебиуса (обратное преобразование `SOS-DP`).

Тогда количество пар путей $S_1 \to T_1$ и $S_2 \to T_2$, которые посещают все клетки $L_1, \ldots, L_k$, и при этом не пересекаются в этих клетках равно $f(S_1, T_1, S_2, T_2) = \sum_B U(S_1, B, T_1) \cdot U(S_2, L \backslash B, T_2)$.

Учитывая опыт решения предыдущей подзадачи, легко видеть, что ответ равен:

$$f(S_1, T_1, S_2, T_2) - f(S_1, T_2, S_2, T_1)$$

Подзадача 9

Честно говоря, жюри не знает ни одного решения, которое осмысленно и при этом проходит эту группу, но не проходит 10-ю. Эта подгруппа — некоторая гарантия того, что если участник придумал полное решение, но его решение работает слишком медленно, то это продвижение будет засчитано.

Подзадача 10

Пусть $K(S, B, T)$ — количество путей из S в T, которые проходят через множества клеток B. Обратите внимание, что не накладывается дополнительных ограничений на то, чтобы эти пути не проходили через какие-либо другие клетки. Рассмотрим выражение:

$$\left(\sum_B K(S_1, B, T_1) \cdot K(S_2, L \backslash B, T_2)\right) - \left(\sum_B K(S_1, B, T_2) \cdot K(S_2, L \backslash B, T_1)\right)$$

Исходя из определения, легко видеть, что здесь вычисляется количество пар путей, где каждая пара путей может быть учтена какое-то количество раз. Заметим следующее:

- Каждой паре путей $S_1 \to T_2$ и $S_2 \to T_1$, как и ранее, мы сопоставляем пару гарантированно пересекающихся путей $S_1 \to T_1$ и $S_2 \to T_2$. Таким образом, в первом и втором слагаемом считаются пары путей $S_1 \to T_1$ и $S_2 \to T_2$.
- Непересекающиеся пары путей $S_1 \to T_1$ и $S_2 \to T_2$, которые проходят через все клетки из множества L, учтены ровно 1 раз в первом слагаемом.
- Пары путей, которые пересекаются, в том числе пересекаются в k клетках из L в первом слагаемом учтены 2^k раз, а во втором слагаемом — тоже 2^k раз, поэтому они не вносят ни какого вклада в сумму.

То есть значение этого выражения — ответ на задачу. Давайте научимся вычислять это значение за полиномиальное время.

Рассмотрим множество клеток $E = \{S_1, S_2, T_1, T_2\} \cup L$ и отсортируем клетки этого множества (x, y) по возрастанию величины $x + y$, в итоге получив массив $E_1, E_2, \ldots, E_l$.

Пусть S_1, S_2, T_1, T_2 имеют индексы в массиве E, равные s_1, s_2, t_1, t_2, соответственно.

Положим, что $\mathrm{dp}[i][j]$ — равно количеству пар путей, которые начинаются в паре клеток (S_1, S_2), и при этом проходят через все клетки $E_1, E_2, \ldots, E[\max\{i, j\}]$. При этом пара путей (P_1, P_2) будет учитываться в этой сумме столько раз, сколько есть способов разбить клетки $E_1, \ldots, E[max\{i, j\}]$ на пару множеств $Y_1 \sqcup Y_2$, такую что путь P_1 посещает все клетки Y_1, а путь P_2 посещает все клетки

Y_2. Тогда $\mathrm{dp}[t_1][t_2] - \mathrm{dp}[t_2][t_1]$ и будет равно искомому выражению, и, по совместительству, ответом на задачу.

Легко видеть, что $\max\{s_1, s_2\} = 2$ (так как клетки S_1 и S_2 имеют минимальную сумму координат), поэтому в качестве начального состояния динамики следует положить $\mathrm{dp}[s_1][s_2] = 1$. Переходы в этой ДП не менее очевидные.

$$\mathrm{dp}[i][j] = \begin{cases} \mathrm{dp}[i][j-1] \cdot K(E_{j-1}, E_j), & i < j-1 \\ \sum_{z=0}^{j-2} \mathrm{dp}[z][j-1] \cdot K(E_z, E_j), & i = j-1 \\ \mathrm{dp}[i-1][j] \cdot K(E_{i-1}, E_i), & j < i-1 \\ \sum_{z=0}^{i-2} \mathrm{dp}[i-1][z] \cdot K(E_z, E_i), & j = i-1 \end{cases}$$

Таким образом, значения dp легко вычисляются за время $O(k^2)$.