

Задача А. Посиделки

Пусть b — бюджет Дарьи, а p_1, p_2, p_3 — цены на товары в порядке предпочтения: Watermelon, Pomegranates, Nuts. Требуется выбрать первый по списку товар, стоимость которого не превышает бюджета; если ни один товар недоступен, вывести **Nothing**.

Алгоритм тривиален и состоит из последовательных проверок в фиксированном порядке: если $p_1 \leq b$, вывести Watermelon; иначе если $p_2 \leq b$, вывести Pomegranates; иначе если $p_3 \leq b$, вывести Nuts; иначе вывести **Nothing**.

Задача В. Выключатели света

Разбор задачи основан на нескольких ключевых наблюдениях, позволяющих упростить её до комбинаторной задачи о поиске двух путей в графе. Прежде всего заметим, что нет смысла выключать лампы раньше времени. Любое корректное решение можно преобразовать так, чтобы все выключения происходили только после того, как Лиора завершит основную часть маршрута. Для этого достаточно один раз пройти путь, игнорируя выключения, затем вернуться в комнату 1 и повторить маршрут, но теперь выключая лампу в комнате в тот момент, когда мы покидаем её в последний раз. Поскольку такая модификация не увеличивает число включённых ламп, мы можем считать, что лампы сначала включаются, а затем все, кроме нужной в конце, выключаются.

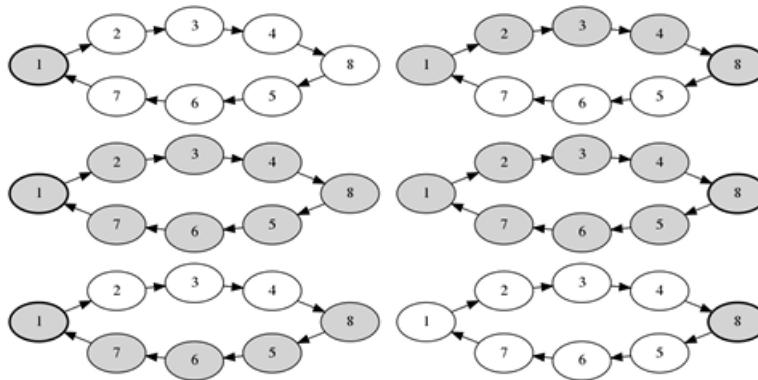
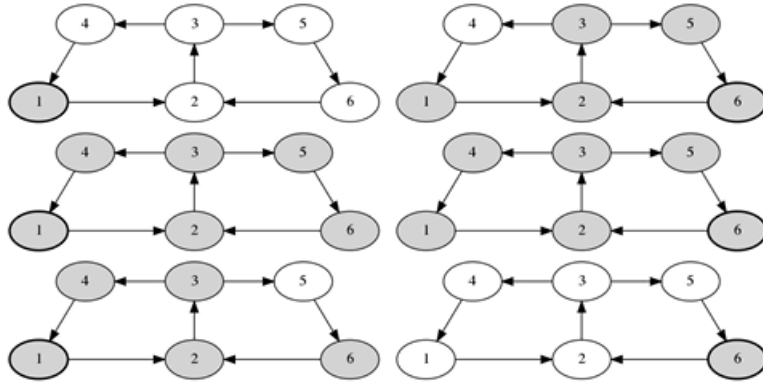


Иллюстрация того, как свет может включаться и выключаться, при заданном пути от первой до последней вершины и пути от последней до первой. В данном случае пути не пересекаются.

Заполненный прямоугольник обозначает освещённую комнату, а толстая рамка означает, что Лиора в данный момент находится в этой комнате. Шаги показаны слева направо и сверху вниз.

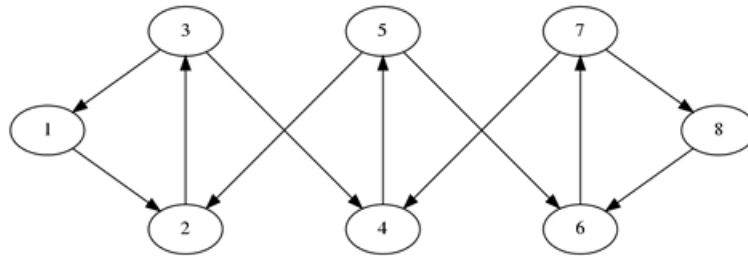
Далее важно отметить симметрию задачи. Если существует путь от комнаты 1 к комнате N , то его обращение даёт путь из N обратно в 1. Отсюда следует, что множество комнат, чьи лампы загораются, обязательно содержит путь в прямом и обратном направлениях. Другими словами, существует подграф, в котором одновременно содержится путь от 1 до N и путь от N до 1, и именно в его вершинах когда-либо горят лампы.

Вывод из сказанного состоит в том, что необходимо лишь найти два пути — из 1 в N и обратно — использующие как можно меньше различных комнат. Все остальные комнаты можно игнорировать. Более того, если такие пути известны, последовательность действий строится однозначным образом: сначала идём из 1 в N , включая лампы вдоль пути; затем идём из N в 1 и включаем лампы оставшихся комнат, после чего совершаем возвращение и выключаем лампы при финальном выходе из каждой комнаты. Коридоры при этом всегда освещены, поскольку мы никогда не покидаем комнату, не оставив за собой включённую лампу до последнего использования.



Один из примеров: общие вершины на путях от начала до конца и от конца до начала.

Чтобы понять, как устроено оптимальное решение, рассмотрим два пути: из 1 в N через вершины a_1, \dots, a_k и из N к 1 через b_1, \dots, b_k . Если какие-то вершины встречаются в них в одинаковом направлении, то участок между ними можно сделать одинаковым в обоих путях, уменьшив общее число комнат. В оптимальном же случае общие вершины появляются в обратном порядке, и пути переплетаются сегментами. Это задаёт специфическую структуру решения: два маршрута состоят из чередующихся общих и отдельных участков.



Структура решения. Каждое ребро можно заменить путём с произвольным количеством вершин

Теперь представим, что мы строим эти пути одновременно. В каждый момент состояние можно описать парой комнат (u, v) , где u — текущая комната на пути из 1 в N , а v — текущая комната на пути из N в 1. Как только оба пути достигают общей комнаты, мы можем считать, что один из них «пересекает» участок в прямом направлении, а другой — в обратном. Такое состояние позволяет перейти к следующему без запоминания всей предыдущей истории, что критически важно: иначе возникла бы экспоненциальная память.

Переходы между состояниями бывают трёх типов.

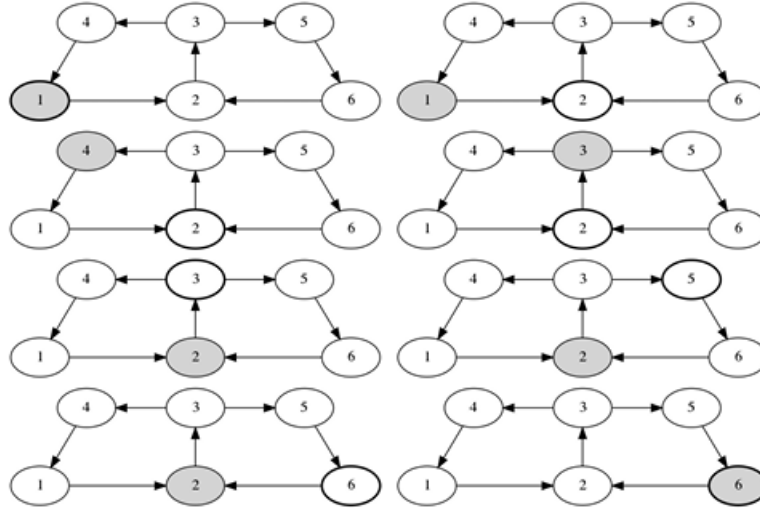
1. Можно переместиться из u в смежную с ней комнату в исходном графе;
2. можно переместиться из v в смежную с ней комнату в обращённом графе;
3. можно обменять u и v местами, что соответствует началу или окончанию прохождения общего сегмента.

Все эти действия имеют стоимость. Переход из u или v в соседнюю вершину стоит 0, если перемещающаяся вершина совпадает с другой, и 1, если нет. Стоимость обмена равна расстоянию между u и v минус 1, поскольку каждая комната на общем участке пути будет зажжена ровно один раз.

Эти операции позволяют задать новый граф состояний, где вершины — пары (u, v) , а рёбра соответствуют указанным действиям. Количество таких состояний порядка N^2 , а рёбер порядка NE . Стоимости переходов неотрицательны, поэтому можно применить алгоритм Дейкстры, запустив его из состояния $(1, 1)$ и найдя минимальную стоимость достижения состояния (N, N) . Этот минимум и есть ответ — наименьшее число ламп, которые придётся включить.

До запуска Дейкстры необходимо знать расстояния между любыми двумя комнатами, так как они используются в стоимости третьего типа операций. Их можно вычислить с помощью обходов в ширину от каждой комнаты, что занимает $O(N^2)$ времени. Итоговая асимптотика решения составляет $O(N^2)$ или $O(N^2 \log(N^2))$ в зависимости от реализации очереди при поиске кратчайших путей.

Таким образом, задача сводится к построению кратчайшего пути в расширенном пространстве состояний, что позволяет учитывать взаимодействие прямого и обратного маршрутов и минимизировать количество комнат, в которых в какой-либо момент должна гореть лампа.



Пример использования операций 1, 2 и 3 для перехода из состояния (11) в состояние (66) в графе во втором примере. u обозначено жирной рамкой, а v — закрашенным. Шаги стоят в порядке 1, 1, 1, 0, 1, 1, 0, что даёт общую стоимость 5.

Задача С. Колесо Букв

Для восстановления исходного расположения букв на колесе вводим массив из N символов, изначально заполненный вопросительными знаками. Каждый сектор массива соответствует сектору колеса. Затем последовательно обрабатываем все наблюдения вращений. Каждое вращение моделируем как движение указателя в обратную сторону по массиву, что эквивалентно вращению колеса по часовой стрелке. Когда после вращения мы знаем, какая буква оказалась под указателем, записываем эту букву в соответствующую позицию массива. Если эта позиция уже содержит другую букву, это противоречие, и восстановление невозможно — выводим “!”.

После обработки всех вращений проверяем массив на повторяющиеся буквы. Если какая-либо буква встречается более одного раза, это также противоречие, и выводим “!”. В противном случае выводим массив в порядке по часовой стрелке, начиная с позиции, оказавшейся под указателем после последнего вращения. Если какая-либо позиция так и осталась неизвестной, выводим для неё знак вопроса. Алгоритм корректно учитывает циклическую природу колеса и позволяет восстановить все однозначно определённые буквы.

Задача D. Антикризисный депозит

Решение моделирует начисление процентов по дням. Для каждого дня отслеживается число дней с последнего начисления, и при наступлении конца месяца или окончания депозита начисляются проценты на текущий баланс по формуле

$$x \rightarrow x \cdot \left(1 + \frac{p}{100} \cdot \frac{d}{365}\right).$$

После начисления счётчик дней сбрасывается, и продолжается итерация до завершения депозита. Поскольку количество дней не превышает 365, алгоритм имеет линейную сложность.

Задача Е. Разделение дорог

Заметим, что положение разделяющей прямой полностью определяется выбором точки P_0 на плоскости и направления нормального вектора. Поворот линии вокруг точки P_0 — непрерывный процесс, при котором дороги переходят с одной стороны линии на другую. Это ключевое наблюдение позволяет свести задачу к поиску направления прямой, при котором суммарный “баланс длин” равен нулю.

Для каждой дороги рассмотрим её вклад в разность длин

$$D(\alpha) = \sum_{i=1}^m L_i(\alpha),$$

где α — угол, определяющий направление линии, а $L_i(\alpha)$ — величина, равная длине дороги i с плюсом, если дорога лежит по одну сторону линии, и с минусом — если по другую. Если дорога пересекает линию, её вклад вычисляется пропорционально длине части, оказывающейся по соответствующей стороне. Таким образом, $D(\alpha)$ — непрерывная функция угла α .

Если выбрать точку P_0 , не лежащую на одной прямой с какой-либо дорогой, то при изменении угла α знак вклада каждой дороги меняется только в моменты, когда прямая проходит через направление на конец дороги. Тогда $D(\alpha)$ монотонно меняет знак: существует угол α , при котором $D(\alpha) = 0$. Найдя его, мы получаем искомую разделяющую линию.

Решение состоит из следующих шагов:

1. Случайным образом выбирается точка P_0 в допустимой области так, чтобы никакая дорога не была коллинеарна с ней. В случае неудачи выбирается новая точка.
2. Определим функцию $f(\alpha)$, возвращающую знак разности суммарных длин дорог по одну из сторон прямой, проходящей через P_0 под углом α .
3. Поскольку $f(0)$ и $f(\pi)$ имеют противоположные знаки, то по теореме о промежуточных значениях существует угол α^* , для которого $f(\alpha^*) = 0$.
4. Выполняем двоичный поиск по $\alpha \in [0, \pi]$ с высокой точностью (порядка 10^{-12}), вычисляя $f(\alpha)$ за $O(m)$ для каждого шага.
5. Выводим точки P_0 и $P_1 = P_0 + v(\alpha^*)$, где $v(\alpha^*)$ — единичный (или любой ненулевой) вектор направления линии.

Корректность метода основана на том, что при повороте линии вклад каждой дороги меняется линейно, а суммарная функция $D(\alpha)$ является непрерывной и имеет противоположные знаки на концах интервала. Это гарантирует существование подходящего угла. Случайный выбор точки P_0 обеспечивает отсутствие вырожденных конфигураций.

Задача F. Нарезка Брауни

Нам предлагается найти наибольшее целое число K , при котором брауни можно разделить на части, содержащие не менее K шоколадных кусочков. Если бы мы могли определить, существует ли такое разделение для любого фиксированного K , то мы могли бы выполнить бинарный поиск, чтобы найти максимальное значение K , при котором такое разделение возможно.

Первым шагом является построение таблицы p , такой что $p[r][c]$ хранит количество шоколадных кусочков в каждом прямоугольном подбрауни размером $r \times c$ от левого верхнего угла. Это обычная таблица двумерных префиксных сумм. Используя этот массив, мы можем вычислить количество шоколадных кусочков в любом прямоугольном подбрауни.

Для любого фиксированного значения K мы можем использовать жадный алгоритм для разделения брауни на кусочки, содержащие не менее K шоколадной крошки. Чтобы решить, где сделать первый вертикальный разрез, мы по очереди рассматриваем все возможные места и разрезаем в первом месте, для которого полученную полоску можно разделить на B кусочков размером не менее K .

Чтобы определить, можно ли разделить вертикальную полоску на B кусочков, мы используем другой жадный алгоритм (проходим по полоске за ее длину, вычисляя сумму с помощью посчитанных ранее префиксных суммы, и откусываем кусок когда может).

После выполнения первого вертикального разреза мы решаем, где сделать второй вертикальный разрез, используя ту же процедуру. Сделав таким образом $A - 1$ вертикальных разрезов, мы определяем, можно ли разделить последнюю вертикальную полоску на B кусочков. Если это невозможно, или если нам не удалось сделать один из предыдущих разрезов из-за того, что у нас закончился брауни, то невозможно разделить брауни на AB кусочков, содержащих не менее K шоколадной крошки.

Задача G. Тур по МФТИ

При фиксированного текущего положения p и фиксированного множества доступных позиций одной буквы (упорядоченного в одну строку), оптимальнее выбрать из этих позиций либо ближайшую справа (первую не меньше p), либо ближайшую слева (последнюю строго меньше p). Дальнейшие по сторону от выбранной позиции пункты только увеличат путь, потому что расстояние от p по модулю монотонно растёт при удалении от p . Значит, для каждой стадии (выбора картинка с заданной буквой) достаточно рассмотреть не все позиции, а не более двух — ближайшую слева и ближайшую справа относительно текущей позиции.

Поскольку у нас четыре стадии, полное пространство разумных вариантов состоит из $2^4 = 16$ комбинаций: для каждой буквы заранее выбираем, будем ли брать позицию «слева» или «справа» от текущей позиции. Для каждой комбинации мы последовательно моделируем маршрут: из $p = s$ берём для M соответствующий кандидат (нижний или верхний по отношению к p), добавляем к сумме дистанцию до него, обновляем p на эту позицию и переходим к I , затем к P , T , и в конце добавляем расстояние до t . Если в процессе требуемая «сторона» не содержит позиции (например, хотим правую, но всех позиций меньше p), корректно обрабатываем это как отсутствие кандидата (в реализации берут ближайшую существующую — конец или начало массива) — такие варианты либо эквивалентно проигрышны, либо корректно учитываются при переборе.

Реализация опирается на двоичный поиск в каждом из V_M, V_I, V_P, V_T для нахождения ближайших слева/справа от текущей точки за $O(\log N)$. Для одного запроса выполняется до 16 симуляций по 4 стадиям каждая, то есть число бинарных поисков ограничено 64. Следовательно, асимптотика работы всех запросов составляет $O(Q \cdot 16 \cdot \log N) = O(Q \log N)$.

Задача H. Счастливые основания

Заметим прежде всего, что если само число n является счастливым, то для любого основания $B > n$ его представление в базе B состоит из одной цифры, равной n . Эта цифра счастливая, следовательно, таких оснований бесконечно много, и ответ в этом случае равен -1 . Далее предполагается, что n не является счастливым.

Рассмотрим возможные длины представления числа n в базе B . Минимальная возможная ненулевая цифра равна 4, поэтому если длина записи не меньше трёх, то

$$n \geq 4B^2 \quad \Rightarrow \quad B^2 \leq \frac{n}{4}.$$

Следовательно, основания $B > \sqrt{n/4}$ не могут давать представление длины три или больше; для таких оснований запись имеет длину не более двух. Наоборот, все основания $2 \leq B \leq \sqrt{n/4}$ потенциально могут породить длинные записи.

Это приводит к разбиению задачи на два независимых случая. Для всех оснований $B \leq \sqrt{n/4}$ можно явно построить представление числа n в базе B путём последовательного деления на B и проверять, что каждая полученная цифра является счастливой. Если это так, основание B подходит.

Во втором случае $B > \sqrt{n/4}$, и, как показано выше, запись числа n в базе B обязательно состоит из одной или двух цифр. Одноцифровый вариант невозможен, так как n не является счастливым. Следовательно, существует запись

$$n = aB + b,$$

где a и b — счастливые числа, $a > 0$, $0 \leq b < B$ и $B = \frac{n-b}{a}$ является целым. Необходимо перебрать все возможные пары (a, b) счастливых чисел и проверить выполнимость данных условий. Поскольку $n \leq 10^{16}$, достаточно рассматривать числа a и b длиной до 18 десятичных цифр, что позволяет явным образом сгенерировать все счастливые числа, а затем рассмотреть все способы разбиения каждого из них на префикс a и суффикс b . Для каждой такой пары значение B восстанавливается по формуле выше. Условия $b < B$ и $B > \sqrt{n}/4$ гарантируют корректность двузначного представления и исключают основания, уже обработанные на предыдущем этапе. Различные подходящие значения B собираются в множество, чтобы избежать повторного подсчёта.

Обе части алгоритма вместе охватывают все возможные основания. В первой части проверяются все основания, способные породить представления длиной три и более; во второй — все основания, для которых представление состоит из двух цифр. Других случаев не существует. Таким образом, найденное количество оснований B и является ответом задачи.

Итоговый алгоритм работает за отведённое время, так как число проверяемых оснований в первой части ограничено $O(\sqrt{n})$, а количество счастливых чисел, необходимых для перебора во второй части, экспоненциально по длине, но ограничено фиксированной константой 18, что приводит к приемлемому числу проверок.

Задача I. Все, что вы можете выбрать

Ключевое наблюдение состоит в том, что момент блокировки наступает через $T - 0.5$ минут после активации первого модуля, поэтому последний модуль должен быть активирован строго до этого момента. Следовательно, все модули, кроме последнего, должны завершиться не позднее чем через $T - 1$ минут, а последний модуль может выполняться сверх этого времени. Значит, выбор последнего модуля определяется исключительно тем, что он добавит значение B_i к максимально возможной сумме значений остальных модулей, запускаемых за $T - 1$ минут.

Рассмотрим модуль i как последний. Тогда остальные модули должны уместиться в $T - 1$ минут, и мы хотим найти максимальную сумму B для любого подмножества модулей, кроме i , укладываемого в это время. Наивный перебор последнего модуля и пересчёт динамики для каждого варианта дал бы сложность $O(N^2T)$, что слишком много.

Чтобы сократить время, используют два массива динамики:

$DP1[i][t]$ = максимальная сумма B среди первых i модулей за t минут,

$DP2[i][t]$ = максимальная сумма B среди модулей с i по N за t минут.

Тогда вклад всех модулей, кроме i , которые можно выполнить за $T - 1$ минут, выражается как

$$\max_{0 \leq t \leq T-1} (DP1[i-1][t] + DP2[i+1][T-1-t]).$$

Добавляя к этому B_i , получаем лучший результат при выборе i в качестве последнего. Перебор всех i занимает $O(NT)$ времени после предварительного вычисления динамик.

Заметим также, что если среди выбранных модулей существует модуль с максимальным временем выполнения A_i , его всегда можно поставить последним без ухудшения результата. Отсюда модули удобно отсортировать по возрастанию A_i и использовать только массив $DP1$, что сохраняет ту же асимптотическую сложность $O(NT)$.

Задача J. Парковочная теория

Рассмотрим машину X , стоящую в клетке (i, j) с номером $A[i][j]$.

В той же строке найдём ближайшие слева и справа клетки, содержащие меньшие числа. Обозначим их столбцы y_1 и y_2 . Аналогично, в столбце найдём ближайшие меньшие числа сверху и снизу, обозначив их строки x_1 и x_2 .

Заметим следующее:

- если внутри подпрямоугольника есть клетка (i, y_1) или (i, y_2) с меньшим номером, машина X не может заехать по строке;

- если внутри подпрямоугольника есть клетка (x_1, j) или (x_2, j) с меньшим номером, машина X не может заехать по столбцу.

Следовательно, если в подпрямоугольник одновременно входят обе “блокирующие” точки по строке и по столбцу, машина X не имеет никакого пути въезда. Такой подпрямоугольник является **недопустимым**.

Для каждой клетки (i, j) , у которой определены значения x_1, x_2, y_1, y_2 , построим прямоугольник

$$R_{i,j} = [x_1 + 1, x_2 - 1] \times [y_1 + 1, y_2 - 1],$$

внутри которого включать машину (i, j) запрещено: тогда она будет заблокирована по обеим осям.

Таким образом, задача сводится к подсчёту подпрямоугольников, которые не пересекают ни один прямоугольник $R_{i,j}$, содержащий клетку (i, j) .

Переберём пары строк $l \leq r$. Рассмотрим только подпрямоугольники, имеющие вертикальные границы на строках l и r .

Каждый запрещённый прямоугольник $R_{i,j}$, для которого $x_1 < l \leq i \leq r < x_2$, накладывает ограничение на возможные столбцы: если левый столбец подпрямоугольника равен y_1 , то его правый столбец должен быть строго меньше y_2 .

Для каждого фиксированного l мы будем двигаться по r сверху вниз и поддерживать массив

$R[c]$ = минимально допустимый правый конец подпрямоугольника при фиксированном левом конце c .

Изначально $R[c] = m$ для всех c . Каждое новое ограничение выполняет операцию $R[y_1] = \min(R[y_1], y_2)$.

Пусть $R[c]$ определено для текущих строк $[l, r]$. Тогда число допустимых подпрямоугольников с левым столбцом c равно:

$$R[c] - c.$$

С учётом того, что правый конец не может уменьшаться при движении влево, мы берём префиксные минимумы:

$$\text{ans}_{l,r} = \sum_{c=0}^{m-1} (\min(R[c], R[c+1], \dots, R[m-1]) - c).$$

$$\text{Ответ} = \sum_{l=0}^{n-1} \sum_{r=l}^{n-1} \text{ans}_{l,r}.$$

Для каждой пары строк мы обновляем ограничения и пересчитываем сумму за $O(m)$, всего $O(n^2m)$ операций.

Задача К. Несмежные чёрные прямоугольники

Перебирать все чёрные прямоугольники напрямую невозможно: их количество может достигать порядка $O(N^3)$. Вместо этого используется идея локального подсчёта количества чёрных прямоугольников с фиксированным углом. Рассмотрим, как найти число чёрных прямоугольников, для которых данная клетка является правым нижним углом. Для каждой строки R определим для каждого столбца высоту — количество подряд идущих чёрных клеток, заканчивающееся на строке R . Эти высоты образуют гистограмму. Проходя по ней слева направо и поддерживая «лестницу» ненулевых высот, можно посчитать количество чёрных прямоугольников, заканчивающихся в текущей клетке: оно равно числу элементов лестницы минус один (так как прямоугольники 1×1 не учитываются). Аналогичным образом можно посчитать количество чёрных прямоугольников, для которых клетка является нижним левым, верхним правым или верхним левым углом. Таким образом, для каждой клетки мы знаем, сколько чёрных прямоугольников расположено в каждом из четырёх направлений относительно неё.

Теперь рассмотрим пары непересекающихся прямоугольников. Для любой такой пары существует хотя бы одно из следующих:

- вертикальная прямая между ними,

- горизонтальная прямая между ними,
- и вертикальная, и горизонтальная прямые.

Каждый из этих случаев можно посчитать отдельно. Пусть сначала рассматривается случай, когда два прямоугольника разделены вертикальной прямой. Выберем столбец, который является правой границей левого прямоугольника. Тогда второй прямоугольник обязан находиться строго правее, и количество таких пар выражается через ранее подсчитанные количества прямоугольников, имеющих углы в соответствующих позициях. Аналогично считается количество пар, разделённых горизонтальной прямой.

Остаётся случай, когда прямоугольники разделены одновременно и горизонтальной, и вертикальной линиями. Тогда одна клетка таблицы может рассматриваться как точка разбиения, и один прямоугольник находится в одном из её квадрантов, а второй — в противоположном. Для каждой клетки эта величина снова вычисляется через уже известные значения количества прямоугольников, привязанных к углам.

Заметим, что пары, разделённые обеими линиями, были посчитаны дважды — один раз среди вертикально разделённых и один раз среди горизонтально разделённых. Поэтому итоговый ответ равен

$$A + B - C,$$

где A — число пар, разделённых вертикальной прямой, B — число пар, разделённых горизонтальной прямой, C — число пар, разделённых и вертикальной, и горизонтальной прямыми.

Вся необходимая информация о количестве прямоугольников, соответствующих углам клеток, вычисляется за $O(N^2)$. Итоговое вычисление величин A , B и C также занимает $O(N^2)$, что даёт полную асимптотику

$$O(N^2).$$

Задача L. Шестигранная спираль

Для решения задачи полезно рассматривать поле в виде концентрических рамок, где каждая рамка состоит из клеток, имеющих одинаковое минимальное расстояние до границ поля. Вся структура поля образована $n - 1$ такими рамками. Количество клеток в каждой рамке уменьшается на 6 при переходе к внутренней рамке, что образует арифметическую прогрессию. Чтобы определить номер клетки (p, q) , вычисляем, сколько полных рамок находится между этой клеткой и границами поля: это минимальное расстояние до края ряда, до первого и последнего ряда. Суммируя количество клеток всех внешних рамок, получаем общее число пройденных клеток до рамки, где лежит (p, q) .

Далее внутри рамки определяется порядковый номер клетки с учётом длины стороны рамки и её формы спирали. Если общее количество клеток до данной клетки больше m , она пустая и выводим 0, иначе выводим соответствующее число. Для поиска позиции наибольшего числа в поле применяем бинарный поиск по рамкам, определяем рамку, содержащую m -е число, затем внутри рамки вычисляем его порядковый номер и преобразуем его в координаты $(row, position)$ с учётом всех внешних рамок. Так можно вычислить номер любой заданной клетки и координаты максимального числа, используя свойства арифметической прогрессии для подсчёта клеток и спиральный порядок обхода рамок.

Вычисления нужно делать аккуратно, потому что ограничения таковы, что нужно использовать `unsigned long long`. Могут быть ситуации при неаккуратном подсчёте суммы арифметической прогрессии (например перемножить, а потом поделить на два, а не наоборот), приводящие к переполнению. Но если приложить аккуратность или использовать `int128`, проблем не возникает.

Задача M. Переключение фигур на магической доске

Главная идея заключается в том, чтобы рассмотреть не просто клетки доски, а пары вида «клетка + текущая фигура». Для каждой клетки существует пять возможных состояний, по одному для каждого типа фигуры. Таким образом, мы строим ориентированный граф, состоящий из

$$5 \cdot h \cdot w$$

вершин. Каждая вершина описывает: «фигура X стоит на клетке (i, j) ».

Рёбра графа соответствуют двум видам действий:

- **смена фигуры** в текущей клетке — ребро ведёт в вершину с той же позицией, но другой фигурой, и вес ребра равен стоимости этой замены;
- **шахматные ходы текущей фигуры** — рёбра ведут в вершины, соответствующие достижимым клеткам, вес каждого такого ребра равен 1.

При обработке ходов скользящих фигур (слона, ладьи, ферзя) в выбранном направлении мы продолжаем движение до тех пор, пока не наткнёмся на заблокированную клетку. Однако важная оптимизация позволяет существенно сократить количество проверяемых рёбер: если мы достигли клетки, расстояние до которой уже не превышает расстояние текущей вершины, дальнейшее движение в этом направлении можно немедленно прекратить.

После построения графа достаточно запустить алгоритм Дейкстры из стартовой вершины, соответствующей королю в клетке $(1, 1)$. Ответом будет минимальное расстояние среди всех пяти вершин, соответствующих клетке (h, w) , поскольку финальной фигурой может быть любая.