Задача A. Антипалиндром

Увеличиваем исходное число на 1: $x := x+1$, чтобы искомое число было строго больше исходного. Проверяем длину числа:

- Если длина n нечётная, минимальный антипалиндром имеет длину $n+1$, так как средняя цифра всегда совпадает сама с собой.
- Если длина n чётная, продолжаем проверку симметричных пар.

Рассматриваем пары (d_i, d_{n-i-1}) начиная с центральной пары и двигаясь к краям. Для каждой пары:

- Если цифры различны, продолжаем.
- Если цифры совпадают, увеличиваем **правую цифру пары** и обнуляем все младшие разряды, чтобы получить минимальное число, большее исходного.
- При переносе (например, когда цифра = 9) корректно обрабатываем увеличение.

Пример: Пусть $x = 1220$. Увеличиваем: $x + 1 = 1221$. Длина чётная, проверяем симметричные пары с середины:

- $(d_1, d_2) = (2, 2)$ — совпадают. Увеличиваем правую цифру и обнуляем младшие: получаем 1230.
- $(d_0, d_3) = (1, 0)$ — уже различны.

Таким образом, минимальное антипалиндромное число, большее 1220, равно 1230.

Задача B. Пятнашки

Для решения задачи естественным образом подходит алгоритм поиска с ограничением на глубину, известный как IDA* (Iterative Deepening A*), либо ограничение обхода в ширину с помощью mitm. Поскольку в условии гарантируется, что минимальное число ходов меньше 25, использование IDA* обеспечивает эффективный поиск без переполнения памяти, которое могло бы возникнуть при классическом BFS. BFS можно ограничить, воспользовавшись методом встречи в середине.

Ключевой компонент алгоритма A* и его модификаций — эвристика, которая оценивает минимальное число оставшихся ходов до цели. Наиболее естественной и точной эвристикой для пятнашек является манхэттенское расстояние каждого числа до его целевого положения. Манхэттенское расстояние для числа x с текущими координатами (i, j) и целевыми координатами (i', j') вычисляется как $|i - i'| + |j - j'|$. Сумма манхэттенских расстояний для всех чисел (кроме пустой клетки) формирует допустимую эвристику, которая никогда не превышает реальное минимальное количество ходов.

Алгоритм A* работает следующим образом. Задаётся начальный порог $threshold = h(start)$, где h — эвристика для начального состояния. Затем выполняется поиск в глубину с отсечением по порогу, который рекурсивно перемещает числа в пустую клетку. Если стоимость текущего пути плюс эвристика превышает порог, рекурсия обрывается, а минимальное превышение запоминается для увеличения порога на следующей итерации. При достижении состояния цели алгоритм завершает поиск и возвращает последовательность чисел, которые перемещались в пустую клетку на каждом ходе.

Для эффективного хранения состояния можно использовать одномерный массив из 16 элементов, где числа записаны построчно, а пустая клетка обозначена нулём. Возможные перемещения ограничены соседними клетками по сторонам, то есть максимум четыре направления. Для ускорения поиска полезно не возвращаться в состояние, из которого мы только что пришли, избегая обратного хода.

Поскольку задача гарантирует достижимость цели, A* с манхэттенской эвристикой и ограничением числа ходов до 25 гарантированно найдёт оптимальное решение. Эвристика обеспечивает допустимость, а глубина поиска ограничена, поэтому алгоритм возвращает минимальное число ходов.

Если выбрать путь оптимизации BFS, то можно начать поиск с двух сторон, слой за слоем: у нас есть начальное и конечное положения. Нужно проверять для каждого слоя, не умеем ли мы уже получать состояние с другой стороны, и как только дойдем до состояния с двух сторон, определять ответ.

Задача C. Привлекательные участки

Если участок между метками i и j допустим, то любой его расширенный участок между метками $i - d$ и $j + e$ для $d \geqslant 0$ и $e \geqslant 0$ также допустим.

Пусть i — это западнейшая метка, такая что участок между метками 0 и i допустим, а участок между i и K тоже допустим. Если такой i не существует, результат равен 0 (ни один столбик поставить нельзя).

Далее можно доказать, что существует способ поставить максимальное количество столбиков так, чтобы один из них был на метке i, и при этом не было столбиков западнее i. Если это верно, то следует жадный алгоритм:

1. Найти такое i;
2. Поставить столбик на i;
3. Рекурсивно пытаться поставить больше столбиков на участке $[i, K]$.

Чтобы решить задачу за линейное время, рассмотрим список высот слева направо, фиксируя моменты, когда встречаются подряд идущие метки в строго возрастающем и строго убывающем порядке. Как только мы видим обе последовательности, мы нашли потенциальное i (проверку участка $[i, K]$ откладываем, чтобы сэкономить время).

Далее:

1. Увеличиваем результат на 1;
2. Сбрасываем флаги, фиксирующие встречу последовательностей;
3. Продолжаем проход.

После завершения сканирования восточный участок может быть недопустим (так как проверку мы не делали), но это можно проверить по значениям двух булевых переменных, которые показывают, встречались ли последовательности с момента последнего сброса.

Если западный участок недопустим, последняя найденная позиция для столбика также недопустима, поэтому уменьшаем текущий результат на 1. Обратите внимание: каждый раз, когда мы находим место для нового столбика, это означает, что все предыдущие столбики были поставлены корректно. Таким образом, алгоритм делает один линейный проход и поддерживает константное количество переменных (две булевых переменные и результат) на каждом шаге, что обеспечивает линейное время работы.

Доказательство ключевого утверждения.

1. Участок между метками 0 и j для $j < i$ никогда не допустим по определению i, поэтому никакое допустимое размещение столбиков не ставит столбик на метке $j < i$.

2. Так как метка i сама по себе допустима, существует хотя бы одно непустое допустимое размещение. Пусть $L = i_1, i_2, \ldots, i_m$ — список позиций столбиков в допустимом размещении максимального числа m столбиков в порядке возрастания.

Так как $i \leqslant i_1$ (по первому пункту), можно определить $L' = i, i_2, \ldots, i_m$.

Докажем, что L' также допустим:

- Участок между 0 и i допустим по определению i;
- Участок между i и i_2 является расширением участка между i_1 и i_2 (так как $i \leqslant i_1$);
- Все остальные участки в L' также присутствуют в L и поэтому допустимы.

Следовательно, L' — корректный список столбиков максимальной длины, где западнейшая позиция — i.

Задача D. Исследовательский консорциум

Обратите внимание, что также известно, что $B \leqslant \min(15, N-1)$. Как это дополнительное условие влияет на решение?

Первое, что нужно заметить: при $N = 1024$ отсутствовать будут лишь небольшая часть столбцов, но эти столбцы могут находиться в любых позициях. Рассмотрим, что можно сделать с 32 числами, которые можно представить 5 битами. Если расположить эти числа в порядке

$$0, 1, \ldots, 31,$$

заметим, что поскольку $B < 15 < 32$, этот блок из 32 чисел никогда полностью не исчезнет. Теперь посмотрим, как можно использовать это. Если у нас есть несколько таких блоков подряд:

$$0, 1, \ldots, 31,\ 0, 1, \ldots, 31,\ 0, 1, \ldots, 31, \ldots$$

ни один из блоков чисел от 0 до 31 не исчезнет полностью, и для оставшихся чисел всегда можно будет определить, из какого блока они.

Детально это выглядит так: числа внутри каждого блока возрастают. Даже после того, как некоторые числа исчезли, при переходе от одного блока к следующему числа будут уменьшаться:

$$0, 1, \ldots, 31,\ 0, 1, \ldots, 27, 5, \ldots, 31, \ldots$$

С учётом этих наблюдений мы готовы к окончательному решению. Пусть битовые столбцы строк, которые мы отправляем в базу данных, представляют повторяющиеся блоки чисел от 0 до 31:

$$0, 1, \ldots, 31,\ 0, 1, \ldots, 31,\ 0, 1, \ldots, 31, \ldots \quad \text{(всего } N \text{ чисел)}$$

После получения ответов из базы данных мы можем пройти по оставшимся числам, отмечая начало нового блока, когда текущее число меньше предыдущего, и отслеживая количество блоков, которые мы встретили.

Зная позицию числа внутри блока и количество пройденных блоков, можно однозначно определить все видимые числа: например, число 16 в пятом блоке — это $(5-1) \cdot 32 + 17 = 145$-е число в последовательности.

Зная, какие числа мы видели, можно определить, какие числа отсутствуют, и вывести их как номера пропавших элементов.

Наконец, мы использовали факт, что $B < 2^5$, чтобы подход работал. Но аналогично можно использовать $B < 2^4$, тогда для решения потребуется всего четыре строки. В этом случае два последовательных числа из разных блоков могут быть равны, так как между ними находятся $2^4 - 1 = 15$ чисел, которые могут исчезнуть. Поэтому для определения смены блока нужно использовать $\geqslant$ вместо $>$ между последовательными числами.

Задача E. Сортировка колоды

Первое важное наблюдение состоит в том, что одна операция переупорядочивания может уменьшить число соседних карт разных рангов не более чем на два. В начальной конфигурации имеется $(R \times S) - 1$ пар соседних карт разных рангов. В конечной конфигурации таких пар ровно $R - 1$. Следовательно, чтобы перейти от $(R \times S) - 1$ к $R - 1$, требуется как минимум

$$\left\lceil \frac{R \times S - R}{2} \right\rceil$$

операций.

Теперь, зная, что $\left\lceil \frac{R \times S - R}{2} \right\rceil$ является нижней оценкой ответа, достаточно построить алгоритм, который гарантированно использует не более этого количества операций — тогда он всегда будет оптимальным.

Далее опишем способ сортировки карт ровно за это число операций. Инвариант, который мы будем поддерживать, таков: для рангов X и Y любых двух соседних карт всегда выполняется либо $Y = X$, либо $Y = (X+1) \bmod R$. Очевидно, этот инвариант выполняется в начальной конфигурации колоды.

Мы многократно выполняем следующую операцию, пока число соседних пар карт одинакового ранга меньше $R - 1$ и операция не затрагивает нижнюю карту колоды. Найдём самый большой префикс колоды, содержащий ровно два различных ранга, и используем его как стопку A. По инварианту это будет одна или несколько карт ранга X, за которыми следуют одна или несколько карт ранга $(X + 1) \bmod R$.

Затем, начиная с первой карты, не вошедшей в стопку A, формируем стопку B следующим образом: берём самый большой подряд идущий блок карт, не содержащих карт ранга X, и добавляем к нему все следующие за этим блоком подряд идущие карты ранга X. Заметим, что по крайней мере одна карта ранга X обязательно существует; иначе, по инварианту, число соседних пар карт разных рангов уже было бы равно $R - 1$.

Покажем, что эта операция уменьшает число соседних пар карт разных рангов на 2, если она не затрагивает нижнюю карту колоды. Действительно, нижняя карта стопки B имеет ранг X, а первая карта, оставшаяся в колоде, по инварианту имеет ранг $(X + 1) \bmod R$. Новые соседние пары состоят из:

- двух карт ранга X (нижняя карта стопки B и верхняя карта стопки A),
- двух карт ранга $(X + 1) \bmod R$ (нижняя карта стопки A и верхняя карта оставшейся колоды).

Разорванные соседние пары, по определению стопок A и B, всегда состоят из карт разных рангов. Следовательно, общее число соседних пар карт разных рангов уменьшается ровно на 2.

Теперь рассмотрим случай, когда выполнение операции затрагивает нижнюю карту колоды. Это означает, что все карты ранга X находятся в двух непрерывных блоках — в начале и в конце колоды. Кроме того, поскольку это первый раз, когда операция затрагивает нижнюю карту, выполняется $X = R$. По инварианту это возможно только тогда, когда все остальные ранги образуют ровно по одному непрерывному блоку. В такой конфигурации существует ровно R соседних пар карт разных рангов.

Вместо описанной выше операции мы завершаем процесс следующим образом: стопка A состоит из наибольшего подряд идущего блока карт ранга R, начинающегося с вершины колоды, а стопка B — из всех остальных карт. После выполнения операции остаётся $R - 1$ соседних пар карт разных рангов (доказательство аналогично предыдущему случаю, при этом отсутствует «оставшаяся» колода), и последней картой в колоде по-прежнему является карта ранга R.

После выполнения $\left\lfloor \frac{R \times S - R}{2} \right\rfloor$ повторений основной операции число соседних пар карт одинакового ранга становится равным $R - 1$, если $R \times S - R$ чётно, или R, если оно нечётно. Заметим, что до каждого такого шага это число строго больше R, поэтому мы ни разу не затрагиваем нижнюю карту колоды преждевременно.

В чётном случае мы уже достигли целевой конфигурации, не затронув нижнюю карту колоды. В нечётном случае последняя операция как раз затрагивает нижнюю карту, но, как было показано выше, она также переводит колоду в требуемый конечный порядок.

Задача F. Вопросы в зоопарке

Порядок ответов не имеет значения, важно только, сколько раз каждый ответ повторяется. Пусть есть массив

$$\texttt{cnt[i]}$$

— количество животных, ответ которых равен i.

Считывая условие задачи, мы понимаем, что массив должен удовлетворять следующим свойствам:

$$\texttt{cnt[0]} \leqslant 2,$$

так как животных всего два вида, и не может быть больше двух самых высоких животных.

$$\texttt{cnt[i+1]} \leqslant \texttt{cnt[i]},$$

потому что если у животного вида K есть $i+1$ более высоких животных своего вида, то обязательно существует другое животное того же вида, у которого есть i более высоких животных своего вида.

Если эти условия выполняются, то они не только необходимы, но и достаточны для существования корректной расстановки. Можно рассмотреть две последовательности: `short` и `long`. Если $\texttt{cnt[i]} = 2$, одно животное помещаем в `short`, а другое — в `long`. Если $\texttt{cnt[i]} = 1$, животное добавляем в `long`. После этого можно обозначить одну последовательность как кошек, а другую — как кроликов.

Эта конструкция позволяет также посчитать количество корректных назначений. Если длина последовательности `short` равна X, то существует 2^X способов разместить животных в `short`. Последовательность `long` формируется единственным образом. На этапе маркировки у нас есть два варианта: отметить `long` как кошек или как кроликов. В итоге общее количество корректных назначений 2^{X+1}.

Однако если длины `short` и `long` равны (то есть нет $\texttt{cnt[i]} = 1$), каждое назначение учитывается дважды, и тогда ответ равен 2^X.

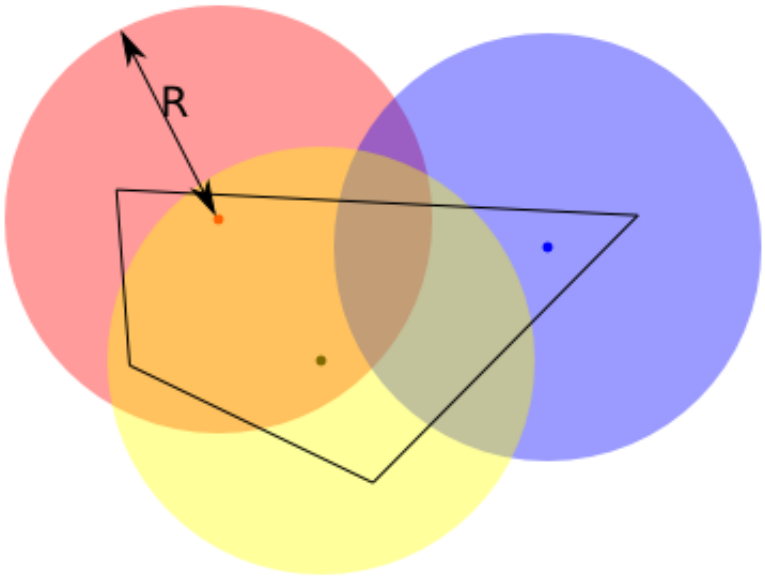
Задача G. Варвары

Первое, что важно заметить, — число 31179239! колоссально велико. Оно астрономически большое. Если разделить периметр многоугольника на 31179239! равных частей, расстояние между соседними точками будет бесконечно малым. Наша цель — распределить эти бесконечно малые точки между городами так, чтобы максимальное расстояние от точки до назначенного ей города было минимально.

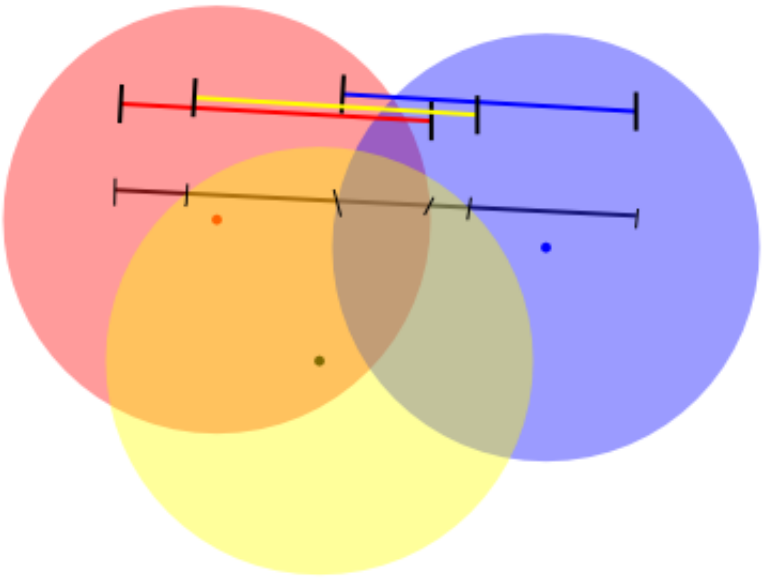
На практике можно считать, что каждая непрерывная группа варваров образует отрезок на границе многоугольника. Варвары расположены настолько плотно, что их можно рассматривать как непрерывное множество точек, покрывающих весь периметр. Тогда задачу можно переформулировать следующим образом: нужно разбить периметр многоугольника на отрезки и назначить каждый отрезок одному из городов. При этом каждому городу должна быть назначена одинаковая суммарная длина отрезков, поскольку варвары должны быть распределены равномерно. Число 31179239! гарантирует, что общее число варваров делится на количество городов, поэтому достаточно лишь обеспечить равенство длин. Иначе говоря, можно представить каждого варвара как бесконечно маленький отрезок, лежащий на границе многоугольника, тогда суммарная длина всех отрезков, назначенных городу, должна быть равна $\frac{\text{perimeter}}{N}$.

Нам нужно минимизировать максимальное расстояние между точкой на границе и городом, которому она назначена. В подобных задачах удобно рассмотреть вопрос: возможно ли решить задачу так, чтобы максимальное расстояние не превышало R? Если мы умеем отвечать на этот вопрос, то можем применить бинарный поиск по R. Если решение возможно для расстояния R, то оно также возможно для любого $R' > R$. Если невозможно для R, то невозможно и для любого $R' < R$. Таким образом, задача сводится к проверке выполнимости для фиксированного R.

Для заданного города все точки границы, находящиеся на расстоянии не более R от него, могут быть назначены этому городу. Множество таких точек образует круг радиуса R с центром в городе. Части границы могут лежать внутри круга ровно одного города, внутри пересечения нескольких кругов или вне всех кругов. Каждую точку можно назначить ровно одному городу, поэтому для областей пересечения требуется аккуратное распределение.



Рассмотрим каждый отрезок границы многоугольника отдельно. Каждый круг может пересекать отрезок в нуле, одной или двух точках. Нас интересует только случай двух пересечений, поскольку именно он даёт ненулевой участок допустимых точек. Для каждого города находим точки пересечения его круга с отрезком. После этого мы объединяем все точки пересечения и сортируем их по расстоянию от начала отрезка. Затем последовательно просматриваем эти точки: при входе в круг города добавляем город в текущее множество допустимых, при выходе — удаляем его. Так мы разбиваем исходный отрезок на несколько меньших сегментов, каждый из которых может быть назначен строго определённому подмножеству городов. Поскольку городов не более пяти, каждый отрезок может быть разбит не более чем на $2^5 = 32$ сегмента.



После разбиения всех сторон многоугольника мы получаем набор сегментов с длинами L_i, каждый из которых может быть назначен некоторому подмножеству городов. Каждому городу необходимо назначить суммарную длину, равную $\frac{\text{perimeter}}{N}$. Это эквивалентно задаче о распределении ресурса, где сегмент длины L предоставляет L единиц ресурса, город требует $\frac{\text{perimeter}}{N}$ ресурса, а сегмент может быть использован только городами из своего списка допустимых. Прямая реализация через максимальный поток возможна, но неудобна из-за вещественных пропускных способностей и необходимости выполнять алгоритм на каждом шаге бинарного поиска.

Так как количество городов очень мало, можно воспользоваться теоремой Холла. Корректное распределение существует тогда и только тогда, когда для любого множества из M городов суммарная длина всех сегментов, достижимых хотя бы одним из этих городов, не меньше $M \cdot \frac{\text{perimeter}}{N}$.

Всего существует не более $2^5 = 32$ различных подмножеств городов, поэтому эту проверку можно выполнить напрямую. Если условие выполняется для всех подмножеств, то корректное распределение существует, а значит, расстояние R допустимо. Комбинируя эту проверку с бинарным поиском и разбиением сегментов, мы получаем решение задачи.

Задача H. Локализация Квантового Исследователя

Ключевое наблюдение состоит в том, что границы между различимыми и неразличимыми областями всегда проходят по диагоналям под углом 45°. Это связано с тем, что множество маяков, обнаруживаемых из точки, может измениться только при пересечении линии, на которой для некоторого маяка выполняется равенство $|x - X_i| + |y - Y_i| = D$. Такие линии в исходных координатах являются диагоналями.

Для упрощения геометрии удобно повернуть систему координат на 45°. Вместо явного поворота используется эквивалентное преобразование координат

$$(x, y) \mapsto (u, v) = (x + y, x - y).$$

Это преобразование сохраняет площади с точностью до постоянного множителя, который в итоге сокращается в вероятности. В новых координатах манхэттенское расстояние превращается в L_∞-расстояние, то есть множество точек, из которых виден маяк, становится осево-ориентированным квадратом со стороной $2D$, центрированным в точке маяка.

Рассмотрим функцию $\mathrm{Info}(p)$ для точки p на плоскости. При малом перемещении точки множество видимых маяков обычно не меняется, а сами относительные координаты просто сдвигаются на один и тот же вектор. Существенные изменения происходят только при пересечении границы какого-либо квадрата. Поэтому разобьём всю интересующую область на прямоугольные области так, чтобы внутри каждой области множество видимых маяков было неизменно.

Для этого проведём все вертикальные прямые $u = U_i \pm D$ и все горизонтальные прямые $v = V_i \pm D$ для каждого маяка с координатами (U_i, V_i). Эти прямые разбивают плоскость на $O(N^2)$ прямоугольных областей. Точки, лежащие вне всех квадратов, можно игнорировать, так как из них не виден ни один маяк, и вероятность попасть туда равна нулю.

Для фиксированной области C множество $\mathrm{Info}(p)$ одинаково для всех внутренних точек $p \in C$ с точностью до сдвига. Если это множество пусто, вклад области в ответ равен нулю. В противном случае площадь области полностью входит в знаменатель вероятности.

Точка из области C является неразличимой тогда и только тогда, когда существует другая область R и такие точки $p \in C$, $q \in R$, что множества $\mathrm{Info}(p)$ и $\mathrm{Info}(q)$ совпадают с точностью до параллельного переноса. Это означает, что соответствующие множества маяков имеют одинаковую конфигурацию относительных положений.

Чтобы эффективно находить такие совпадения, каждой области сопоставляется нормализованный шаблон. Для этого для произвольной точки $p \in C$ берётся множество $\mathrm{Info}(p)$, сортируется в фиксированном порядке и затем сдвигается так, чтобы первый элемент оказался в начале координат. Получившийся набор относительных векторов является инвариантом области с точностью до сдвига. Области с одинаковыми шаблонами группируются вместе.

Опишем, как именно строится нормализованный шаблон для конкретной области. Рассмотрим фиксированную область C. Выберем произвольную точку p строго внутри этой области. Такой выбор возможен, поскольку границы областей имеют нулевую площадь и могут быть проигнорированы. По построению разбиения, для всех точек внутри C набор маяков, находящихся на расстоянии не более D, совпадает.

Для каждого маяка с координатами (U_i, V_i) проверяется условие

$$\max(|U_i - u_p|, |V_i - v_p|) \leqslant D.$$

Если оно выполнено, маяк виден из точки p и добавляется в множество $\mathrm{Info}(p)$ в виде относительного вектора $(U_i - u_p, V_i - v_p)$. Таким образом, получается конечное множество векторов, описывающее конфигурацию видимых маяков относительно точки p.

Полученное множество относительных координат полностью определяет наблюдаемую картину маяков в области C с точностью до параллельного переноса. Однако в таком виде оно всё ещё

зависит от конкретного выбора точки p внутри области, поскольку все векторы сдвигаются при изменении p.

Чтобы избавиться от этой зависимости, множество $\mathrm{Info}(p)$ приводится к нормализованному виду. Для этого все его элементы сортируются в фиксированном порядке, например, по возрастанию координаты u, а при равенстве — по координате v. Пусть (u_0, v_0) — первый элемент в этом порядке. После этого все элементы множества сдвигаются на вектор $(-u_0, -v_0)$.

В результате получается множество

$$S_C = \{(u - u_0, v - v_0) \mid (u, v) \in \mathrm{Info}(p)\},$$

которое называется нормализованным шаблоном области C.

При прямом подходе построение нормализованного шаблона для области потребовало бы перебора всех маяков, что привело бы к сложности $O(N)$ на одну область и $O(N^3)$ суммарно. Однако такое вычисление можно существенно оптимизировать, используя структуру разбиения плоскости на области.

После преобразования координат области образуют прямоугольную сетку. Рассмотрим фиксированную горизонтальную полосу между двумя соседними горизонтальными прямыми вида $v = V_i \pm D$. Внутри этой полосы все области упорядочены по координате u. При движении слева направо координата v точки остаётся в фиксированном интервале, а координата u монотонно возрастает.

Зафиксируем одну такую полосу и будем последовательно обходить области в порядке возрастания u. Для каждой области выберем точку p в её внутренности. Для каждого маяка условие видимости имеет вид

$$|U_i - u_p| \leqslant D \quad \text{и} \quad |V_i - v_p| \leqslant D.$$

Второе условие либо выполняется для всех областей полосы, либо не выполняется ни для одной из них. Первое же условие означает, что маяк виден ровно на одном непрерывном отрезке областей данной полосы. Следовательно, при переходе от одной области к соседней множество видимых маяков либо не меняется, либо в него добавляется один маяк, либо из него удаляется один маяк.

Это позволяет поддерживать текущее множество $\mathrm{Info}(p)$ инкрементально, обновляя его за амортизированное $O(1)$ время на область. Однако само множество всё ещё зависит от выбора точки p и нуждается в нормализации.

Для нормализации множество относительных координат сортируется в фиксированном порядке, и далее используется каноническое представление, не зависящее от параллельного переноса. Вместо хранения самих координат точек хранится последовательность разностей между соседними точками в отсортированном порядке, а также размер множества. Эти разности инвариантны относительно сдвига всего множества, поэтому однозначно задают его нормализованный шаблон.

При добавлении или удалении одного маяка в $\mathrm{Info}(p)$ в отсортированном множестве изменяются только разности, связанные с его непосредственными соседями. Следовательно, обновление нормализованного шаблона также выполняется за амортизированное $O(1)$ время.

Таким образом, при обходе областей внутри одной полосы мы поддерживаем как множество видимых маяков, так и его нормализованный шаблон, выполняя только локальные изменения при переходе между соседними областями. Повторяя этот процесс для всех горизонтальных полос, мы получаем нормализованный шаблон для каждой области за амортизированное $O(1)$ время, а вся фаза построения шаблонов работает за $O(N^2)$.

Все области с одинаковым шаблоном обрабатываются совместно. Их прямоугольники уже приведены к одной системе координат, и требуется найти суммарную площадь точек, которые покрыты ровно одним из этих прямоугольников. Именно эта площадь соответствует различимым позициям для данного шаблона. Суммируя её по всем шаблонам, получаем числитель искомой вероятности.

Задача сводится к вычислению площади области, покрытой ровно одним прямоугольником, для набора осево-ориентированных прямоугольников. Это делается стандартным алгоритмом с заметающей прямой по оси u. Все события добавления и удаления прямоугольников сортируются по координате u. По оси v поддерживается дерево отрезков, в вершинах которого хранятся длины отрезков, покрытых хотя бы одним прямоугольником, и покрытых хотя бы двумя прямоугольниками. Разность этих величин даёт длину отрезков, покрытых ровно одним прямоугольником. Умножая эту длину на приращение координаты u между соседними событиями, получаем вклад в площадь.

Суммарное число областей равно $O(N^2)$, и каждая область принадлежит ровно одной группе шаблонов. Для каждой группы обработка выполняется за $O(K \log K)$, где K — число областей в группе, а суммарно по всем группам это даёт $O(N^2 \log N)$.

Задача I. Объекты в ориентированном дереве

Первое наблюдение: жадность оптимальна

Естественная стратегия — на каждом шаге перемещать как можно больше объектов ближе к корню (вершине 1). Это действительно оптимально: если объект может двигаться, то чем ближе он к корню, тем больше у него шансов добраться туда раньше. Формально это можно доказать индукцией по поддеревьям.

Однако прямая симуляция по шагам времени невозможна: t_i может достигать 10^9, а $n, k \leqslant 10^5$, поэтому алгоритм $O(n \cdot \max t_i)$ не проходит.

Второе наблюдение: поток убывает со временем

Рассмотрим поток объектов через ребро из вершины i в p_i. На начальных шагах он равен m_i (максимум), но как только в вершине i и её поддереве остаётся меньше объектов, чем нужно для поддержания потока m_i, поток начинает уменьшаться. Более того, поток через каждое ребро является **невозрастающей** функцией времени.

Это позволяет описать поведение системы не пошагово, а **кусочно-линейно**: пока поток через ребро не меняется, количество объектов в каждой вершине меняется линейно со временем.

Третье наблюдение: объединение вершин

Когда в вершине i заканчиваются объекты (и входящий поток не компенсирует исходящий), её больше нельзя рассматривать отдельно: все оставшиеся объекты из её поддерева будут проходить через неё мгновенно (в следующем шаге), и можно считать, что они напрямую поступают в p_i.

Это приводит к идее **объединения вершин**: когда вершина «высыхает», её поддерево сливается с родителем. Такие события происходят в определённые моменты времени, и их можно обрабатывать в порядке возрастания.

Алгоритм: симуляция событий с СНМ

Для каждой вершины i (кроме корня) будем поддерживать:

- $\texttt{current}[i]$ — количество объектов, изначально находящихся в i и её поддереве;
- $\texttt{incoming}[i]$ — суммарный поток, приходящий в i из её потомков (в единицу времени);
- $\texttt{M}[i] = m_i$ — пропускная способность ребра $i \to p_i$.

Если $\texttt{M}[i] > \texttt{incoming}[i]$, то вершина i будет постепенно «высыхать». Время, через которое в ней закончатся объекты, равно:

$$\texttt{when}[i] = \left\lfloor \frac{\texttt{current}[i]}{\texttt{M}[i] - \texttt{incoming}[i]} \right\rfloor.$$

Эти моменты времени добавляются в приоритетную очередь. При наступлении события «вершина i высохла»:

1. Объединяем i с её родителем p_i: переносим $\texttt{current}[i]$ и $\texttt{incoming}[i]$ в родителя.
2. Обновляем $\texttt{incoming}[p_i] := \texttt{incoming}[p_i] + (\texttt{incoming}[i] - \texttt{M}[i])$ — разница между входящим и исходящим потоком.
3. Если у нового родителя теперь $\texttt{M}[p] > \texttt{incoming}[p]$, вычисляем новое время высыхания и добавляем в очередь.

Для эффективного объединения используется структура СНМ со сжатием путей: $\texttt{find}(i)$ возвращает актуального представителя компоненты, в которую входит i.

Ответ на запросы

Запросы t_i сортируются по возрастанию. Во время обработки событий мы поддерживаем текущее состояние корня:

$$\text{ответ}(t) = \texttt{current}[1] + t \cdot \texttt{incoming}[1],$$

поскольку в корне накапливаются все объекты, и новых ограничений нет.

Перед каждым событием (в момент времени T) мы отвечаем на все запросы с $t_i \leqslant T$, используя текущую линейную формулу для корня.

Сложность

Сортировка запросов: $O(k \log k)$. Каждая вершина удаляется не более одного раза, каждая операция Union-Find — $O(\alpha(n)) \approx O(1)$. Приоритетная очередь содержит не более n событий, каждая операция — $O(\log n)$. Итоговая сложность: $O(n \log n + k \log k)$.

Задача J. Без воды в условии

Задача состоит в том, чтобы найти все целые числа x, такие что

$$x^k \equiv 1 \pmod{p},$$

где p — простое число.

Из теории чисел известно, что по модулю простого p множество ненулевых остатков образует циклическую мультипликативную группу порядка $p-1$. Это означает, что существует примитивный корень g по модулю p, такой что каждое число $x \not\equiv 0 \pmod{p}$ можно представить в виде

$$x \equiv g^t \pmod{p}$$

для некоторого $0 \leqslant t < p-1$.

Подставим это представление в исходное уравнение:

$$(g^t)^k \equiv 1 \pmod{p}.$$

Это эквивалентно

$$g^{tk} \equiv 1 \pmod{p}.$$

Поскольку порядок g равен $p-1$, это равенство выполняется тогда и только тогда, когда

$$(p-1) \mid tk.$$

Обозначим

$$d = \gcd(k, p-1).$$

Тогда количество решений этого сравнения равно d, и все решения для t имеют вид

$$t = 0, \frac{p-1}{d}, 2\frac{p-1}{d}, \ldots, (d-1)\frac{p-1}{d}.$$

Следовательно, все решения исходного уравнения имеют вид

$$x = g^{\frac{p-1}{d} \cdot i} \bmod p, \quad i = 0, 1, \ldots, d-1.$$

Таким образом, алгоритм решения задачи следующий. Сначала вычисляется $d = \gcd(k, p-1)$, которое и есть количество решений. Затем находится примитивный корень g по модулю p. После этого вычисляется значение

$$t = g^{\frac{p-1}{d}} \bmod p.$$

Все решения представляют собой степени этого числа:

$$1, t, t^2, \ldots, t^{d-1} \pmod{p}.$$

Полученные значения сортируются по возрастанию и выводятся.

Корректность алгоритма следует из свойств циклической группы по простому модулю и стандартного описания решений сравнения $x^k \equiv 1 \pmod{p}$. Ограничение на количество ответов гарантирует, что вывод всех решений возможен напрямую.

Задача K. Игра с диаметром

Диаметр дерева — это длина пути между двумя самыми удалёнными вершинами (расстояние считается как длина кратчайшего пути, то есть число рёбер на кратчайшем пути). Обозначим через D диаметр графа. Заметим, что если существуют хотя бы четыре вершины, такие что расстояние между каждой парой равно диаметру D, тогда второй игрок может играть так, чтобы первый игрок был вынужден выбрать хотя бы две из этих четырёх вершин, и, следовательно, разброс вершин первого игрока будет равен D. Однако разброс будет равен D и если можно выделить две непересекающиеся группы вершин (обозначим их A и B), каждая из которых содержит хотя бы два элемента, так что расстояние между любой вершиной из первой группы и любой вершиной из второй равно диаметру D. В этом случае второй игрок может сыграть так, что первый игрок выберет по одной вершине из каждой группы A и B, и тогда разброс также будет равен D.

Если дерево не удовлетворяет вышеописанным условиям, пусть u и v — вершины на расстоянии D. Предположим, что существует хотя бы ещё одна вершина w, находящаяся на расстоянии D от хотя бы одной из вершин (например, от u). Тогда можно выделить две непересекающиеся группы A и B, одна из которых может содержать только один элемент, так что расстояние между любыми двумя вершинами из этих групп равно D. В этом случае результат зависит от чётности числа вершин. Если число вершин нечётное, первый игрок совершает последний ход, и второй игрок может заставить его выбрать по одной вершине из каждой группы, так что разброс равен D. Если число вершин чётное, первый игрок может сыграть так, чтобы не выбирать вершину из группы, содержащей только одну вершину, и тогда разброс будет равен $D - 1$.

Если ни одно из предыдущих условий не выполняется, существуют ровно две вершины на расстоянии D (пусть это u и v). В этом случае необходимо определить количество вершин, находящихся на расстоянии $D - 1$ от одной или обеих этих вершин, и провести аналогичное рассуждение, как в предыдущих случаях, с единственной разницей, что теперь разброс будет равен $D - 1$ или $D - 2$ в зависимости от числа вершин на расстоянии $D - 1$ от u и v, а также от чётности числа N, если существует ровно одна вершина, находящаяся на расстоянии $D - 1$ от обеих вершин u и v.

Наконец, как определить диаметр графа? Один из способов — выполнить два обхода в глубину, определяя расстояние каждой вершины от начальной вершины. Первый обход начинается из произвольной вершины w, второй — из вершины u, которая находится на максимальном расстоянии от w. Пусть v — вершина, находящаяся на максимальном расстоянии от u. Тогда расстояние между u и v равно диаметру дерева.

Сложность описанного алгоритма составляет $O(N)$.

Задача L. Соревнования по робототехнике

Максимально возможное количество кубиков достигается, если в каждой ячейке (i, j) поставить как можно много кубиков, но не превышая ограничения строки и столбца. Следовательно, для каждой ячейки берём $a_{i,j} = \min(r_i, c_j)$ и суммируем: $\text{maxSum} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \min(r_i, c_j)$.

Минимальная сумма кубиков достигается при минимизации количества кубиков в каждой ячейке, сохранив максимальные значения по строкам и столбцам. Для этого вычисляем, сколько строк и сколько столбцов имеют каждое возможное значение v. Каждый кубик, который не является максимумом в строке или столбце, может быть равен $v - 1$. Таким образом, минимальная сумма вычисляется по формуле $\text{minSum} = N \cdot N + \sum_{v=1}^{5000} \max(\text{freqR}[v], \text{freqC}[v]) \cdot (v - 1)$, где $\text{freqR}[v]$ — количество строк с максимумом v, а $\text{freqC}[v]$ — количество столбцов с максимумом v. Начальная сумма $N \cdot N$ учитывает, что каждая ячейка содержит хотя бы один кубик.

В итоге максимальная сумма равна $\sum_{i,j} \min(r_i, c_j)$, а минимальная сумма равна $N \cdot N + \sum_{v=1}^{5000} \max(\text{freqR}[v], \text{freqC}[v]) \cdot (v - 1)$. Такое распределение гарантирует корректное соблюдение всех ограничений по строкам и столбцам и даёт минимально и максимально возможное количество кубиков в сетке.

Задача M. Ксорцистка

Рассмотрим условие, при котором последовательность

$$a_1 \oplus X, a_2 \oplus X, \ldots, a_N \oplus X$$

является неубывающей. Это эквивалентно выполнению неравенств

$$a_i \oplus X \leqslant a_{i+1} \oplus X$$

для всех i от 1 до $N-1$.

Зафиксируем одну пару соседних элементов a_i и a_{i+1}. Если $a_i = a_{i+1}$, то соответствующее неравенство выполняется при любом X и не накладывает никаких ограничений. Пусть теперь $a_i \neq a_{i+1}$. Обозначим через j номер старшего бита, в котором числа a_i и a_{i+1} различаются. Все биты старше j у этих чисел совпадают, а значит, и после применения XOR с X они останутся равными. Следовательно, результат сравнения полностью определяется битом j.

Если $(a_i)_j = 0$ и $(a_{i+1})_j = 1$, то для выполнения неравенства необходимо, чтобы $X_j = 0$. Если же $(a_i)_j = 1$ и $(a_{i+1})_j = 0$, то требуется $X_j = 1$. Таким образом, каждая пара соседних элементов с разными значениями задаёт ровно одно ограничение на один конкретный бит числа X.

Для каждого бита j собираются требования двух типов: установить $X_j = 0$ или установить $X_j = 1$. Если для одного и того же бита появляются оба требования одновременно, то такого числа X не существует. В противном случае минимальное подходящее значение X определяется однозначно: каждый бит X_j равен 1 тогда и только тогда, когда существует хотя бы одно требование $X_j = 1$.

В решении для каждого бита j поддерживаются два счётчика: количество ограничений вида $X_j = 0$ и количество ограничений вида $X_j = 1$. Также поддерживается число битов, для которых одновременно присутствуют оба типа ограничений. Если это число ненулевое, ответ равен -1. В противном случае текущее минимальное значение X получается установкой всех битов, для которых есть хотя бы одно требование равенства единице.

Изначально ограничения строятся по всем парам (a_i, a_{i+1}). При изменении одного элемента массива могут измениться только пары $(p-1, p)$ и $(p, p+1)$. Поэтому перед обновлением значения соответствующие пары удаляются из структуры ограничений, затем значение элемента изменяется, и новые пары добавляются обратно. Все операции над одной парой выполняются за константное время, поскольку требуется лишь найти старший различающийся бит и обновить счётчики для него.

Так как каждый запрос обрабатывает постоянное число пар, а количество битов ограничено 30, суммарная сложность алгоритма составляет $O(N+Q)$, что удовлетворяет ограничениям задачи по времени и памяти.