Задача A. Города

Можно для каждой буквы сохранить все строки, начинающиеся с нее. Если нет ни одной строки, начинающейся с последней буквы строки предыдущего игрока, то ответ — ?. Если среди строк, начинающихся с этой буквы, есть строка, заканчивающаяся на букву, с которой ничего не начинается, или есть ровно одна строка, начинающаяся на эту букву, и она на нее и заканчивается, то ответ — эта строка и !. Иначе ответ — любая из строк, начинающихся на нужную букву.

Задача B. Дерево в кучу

Пусть $dp[v][i]$ — минимальный возможный максимум среди взятых вершин в поддереве v при условии, что всего взято i вершин. Для фиксированного v $dp[v][i]$ не убывает с ростом i. Если мы хотим, чтобы максимум в поддереве v был не более x, то можно взять столько вершин, сколько есть положительных i, таких что $dp[v][i] \leqslant x$, то есть в $dp[v]$ лежат в отсортированном порядке значения максимумов, в которых можно взять на одну вершину больше с учетом кратности (то есть если можно взять на k вершин больше, то значение встречается k раз). Если пока не рассматривать возможность взять саму v, то чтобы максимум был не более x, надо, чтобы максимум во всех поддеревьях детей v был не более x. Во всех значениях максимума, в которых количество взятых вершин увеличивается на 1 в детях, количество увеличивается на 1 и в v. Значит $dp[v]$ — объединение dp в детях. Дальше, если мы берем саму v, то до этого максимум должен был быть меньше значения в v. То есть, чтобы учесть v, надо найти в dp[v] последнее число, меньшее значения в v, и заменить следующее на это значение.

Если $dp[v]$ хранить в multiset-е и сливать multiset-ы приливанием от меньшего к большему, то все решение работает за $\mathcal{O}(n \log^2 n)$.

Задача C. Кубок Физтеха по КНБ

Можно сделать Meet in the middle. Перебираем первую половину строки, за $\mathcal{O}(nk)$ получаем вектор размера n с текущими счетами этой половины против каждого из n игроков и прибавляем 1 в map по этому вектору. Операции с map размера $3^{k/2}$ по векторам размера n занимают $\mathcal{O}(n \log(3^{k/2}))$, то есть $\mathcal{O}(nk)$. Перебираем вторую половину строки, получаем ее вектор счетов, прибавляем к ответу значение в map-е по вектору из счетов, домноженных на -1 (чтобы в сумме получились все 0). Итого $\mathcal{O}(3^{k/2}nk)$.

Задача D. Долгопрудный будущего

Можно параллельными переносами и осевыми симметриями перевести начальную точку в $(0, 0)$, а конечную в (x_0, y_0), где $x_0, y_0 \geqslant 0$.

Докажем, что единственный способ прийти в конечную точку быстрее, чем за $x_0 + y_0$ — это пройти по данной прямой, если она направлена в сторону увеличения x и y и пересекает прямоугольник с углами в начальной и конечной точке, причем выходить за пределы прямоугольника бесполезно. Пусть $f_x(x) = x_0$ при $x \leqslant 0$, $f_x(x) = 0$ при $x \geqslant x_0$ и $f_x(x) = x_0 - x$ иначе. Аналогично определим $f_y(y)$. Рассмотрим, как меняется величина $\Phi = f_x(x) + f_y(y)$, если мы перемещаемся по разрешенным линиям со скоростью 1. Если мы двигаемся вне прямоугольника, то f_x или f_y не меняется, то есть Φ уменьшается со скоростью не более 1. Если мы идем по линиям сетки, то Φ уменьшается со скоростью не более 1. Если мы идем по данной прямой, но при этом неверно, что и x, и y увеличиваются, то Φ уменьшается со скоростью не более 1. В итоге надо уменьшить Φ на $x_0 + y_0$, единственный способ уменьшить Φ со скоростью больше 1 — идти по прямой внутри прямоугольника в сторону увеличения x и y. Существует путь, который всю длину пересечения прямой с прямоугольником идет вдоль прямой, а все остальное время — по линиям сетки в сторону увеличения x или y, то есть он и есть оптимальный.

Найти точки, в которых прямая пересекает прямоугольник, легко, дальше надо к $x_0 + y_0$ прибавить длину диагонали прямоугольника с углами в точках пересечения и вычесть его полупериметр.

Задача E. Окружности на столе

Пусть $dp[r]$ — количество способов разместить вложенный набор окружностей, внутри фиксированной окружности радиуса r. Тогда $dp[r] = 1 + \sum_{r_1=1}^{r-1}(dp[r_1]\cdot$ количество способов разместить окружность радиуса r_1 внутри окружности радиуса r). Это количество способов равно количеству целых точек внутри или на границе окружности радиуса $r - r_1$. Количество целых точек внутри окружности каждого радиуса до R можно найти за R^2: переберем все точки с координатами по модулю не превосходящими R, найдем расстояние от этой точки до начала координат с округлением вверх до целого. Получим такое число r, что эта точка лежит во всех окружностях радиуса $\geqslant r$ и больше ни в каких. Прибавим 1 к позиции r массива подсчета, потом посчитаем префиксные суммы на этом массиве и получим нужные количества. Теперь саму динамику можно посчитать за $\mathcal{O}(\min(n,m)^2)$ и найти ответ на задачу перебрав радиус самой большой окружности от 1 до $\min(n,m)$.

Задача F. Массив и сумма степеней

Если $k = 0$, то все числа кроме 0 дают одинаковый вклад в ответ, то есть надо как можно больше 0-й превратить в 1.

Если $k = 1$, то каждая монета дает $+1$ к ответу, то есть не важно, какие числа

Теперь $k = 2$. Пусть a_{max} — максимальное число в массиве a. Пусть к a_i мы прибавили c_i. Тогда $(a_1 + c_1)^2 + (a_2 + c_2)^2 + \cdots + (a_n + c_n)^2 = (a_1^2 + \cdots + a_n^2) + (c_1^2 + \cdots + c_n^2) + (2a_1c_1 + \cdots + 2a_nc_n) \leqslant (a_1^2 + \cdots + a_n^2) + (c_1 + \cdots + c_n)^2 + 2a_{max}(c_1 + \cdots + c_n)$. То есть при фиксированном количестве потраченных монет выгоднее всего их все потратить на увеличение максимального числа. Полученная функция от количества потраченных монет — квадратичная с ветвями вверх, то есть максимум достигается на границе, то есть либо когда мы потратили 0 монет, либо когда мы потратили все монеты на увеличение максимума. Надо выбрать лучший из этих вариантов.

Задача G. Конфеты по парам

Условие, что $2k$ конфет можно разбить на пары эквивалентно тому, что всех типа конфет не более k. Сгруппируем конфеты по типам. Из каждого типа нам важны только k лучших конфет. Если k-я и $k + 1$-я конфеты имеют одинаковые значения, то нужно взять все конфеты с этим значением и заменить на предметы состоящие из нескольких конфет. Если у нас было a конфет с этим значением x, b из них попали в лучшие k, то для всех i от 0 до b можно брать i конфет этого типа, то есть i конфет суммарного значения ix с C_a^i способами их выбрать. Из этих $b + 1$ варианта мы обязана выбрать ровно 1. Теперь надо решить задачу о рюкзаке, где некоторые предметы сгруппированы в группы из которых надо взять ровно 1 элемент. Она решается так же как и обычная (в обычной тоже надо каждый элемент либо брать, либо не брать, то есть надо выбрать ровно один вариант). Суммарный размер групп — $\mathcal{O}(n)$, итоговый вес — $2k$, то есть решение работает за $\mathcal{O}(nk)$.

Задача H. Выбор сеансов

Зафиксируем правую границу. Теперь каждый тип фильмов прибавляет свой вес на полуинтервале от предпоследнего вхождения этого фильма в массив, не далее правой границы, до последнего (предпоследнее не включительно, последнее — включительно).

Можно идти по правым границам слева направо и для каждого фильма поддерживать вектор вхождений. Когда мы передвигаем правую границу на один, добавляется одно вхождение одного фильма. Надо для этого фильма вычесть его вес на старом полуинтервале и прибавить на новом и обновить ответ через текущий максимум. Это можно делать деревом отрезков на прибавление на отрезке и максимум. Итого $\mathcal{O}(n \log n)$.

Задача I. Слишком разноцветная матрица

Пусть мы выключили хотя бы один цвет. Тогда среди выключенных цветов есть цвет с самым левым пикселем и цвет с самым правым пикселем. Левую и правую границы прямоугольника можно сузить до этих пикселей. Переберем эти 2 цвета. Теперь левая и правая границы прямоугольника зафиксированы, а также известен максимальный размер прямоугольника по вертикали. Мы можем выключить только те цвета, которые целиком находятся между левой и правой границами и расстояние между верхним и нижним пикселями цвета меньше максимального размера прямоугольника. Теперь можно идти 2 указателями по отсортированным спискам верхних и нижних пикселей этих цветов, поддерживая количество цветов, которые влезут в вертикальный размер. Итого $\mathcal{O}(C^3 \log C)$, где C — количество цветов.

Задача J. Опасная лента

Рассмотрим какой-то сгиб. Во-первых, хотя бы один фрагмент ленты, соседний со сгибом свободен от химиката, так как эти два фрагмента соприкоснутся после сгиба. Во-вторых, в зависимости от того, к какой границе свободного от химиката подотрезка этот сгиб ближе, либо левый кусок, покрытый химикатом завернется в середину, либо правый.

Пусть суммарно завороты левого конца покроют c свободных от химиката фрагментов, а правого — d. Тогда надо, чтобы $a + b + c + d \leqslant n$ и задачи слева и справа независимы. То есть если для всех $p, q \leqslant n$ найти $dp[p][q]$ — количество способов заворачивать конец длины p сколько-то раз, так, чтобы в итоге он покрыл свободный отрезок длины q, То ответ на задачу — $\sum_{c+d \leqslant n-a-b} dp[a][c] \cdot dp[b][d]$.

Найдем $dp[p][q]$. Если заворот был на расстоянии i от границы химиката, то после заворота химикат покрывает длину $p + i$, а свободной осталась длина $q - p - 2i$. То есть $dp[p][q] = 1 + \sum_{i \geqslant 0} dp[p + i][q - p - 2i]$. Если параллельно с dp насчитывать $pref[p][q] = \sum_{i \geqslant 0} dp[p+i][q-2i] = dp[p][q] + pref[p+1][q-2]$, то все пересчеты работают за константное время ($dp[p][q] = 1 + pref[p][q-p]$). То есть можно посчитать dp за $\mathcal{O}(n^2)$.

Задача K. Изучаем мьютексы

Рассмотрим первый раз, когда функция обращается к какому-то мьютексу. Если команда — `release` или `access`, то мьютекс должен быть захвачен перед вызовом функции, а если `acquire`, то не захвачен. Если это не так, то происходит соответствующая ошибка.

Вызов любой функции выглядит так: есть порядок мьютексов (в порядке первого обращения), для каждого из которых известно, в каком состоянии должен быть мьютекс до вызова функции, какая ошибка случится, если он не в этом состоянии и в каком состоянии он окажется после вызова функции. Также возможно, что вне зависимости от состояния мьютексов до вызова функции, она все равно сломается. В таком случае надо также сохранить, с какой ошибкой она сломается и после поломки не записывать больше никаких изменений.

Для пустой функции порядок пустой и ошибки нет. Теперь научимся добавлять одну строку в функцию. Если функция уже сломана, то ничего не надо делать. Если строка — это операция с мьютексом, то если этого мьютекса еще нет в порядке, то надо его туда добавить и проставить необходимое начальное состояние, тип ошибки, если оно не выполнено и конечное состояние. А если мьютекс уже есть в порядке, то если его конечное состояние не совпадает с требуемым начальным, то надо записать, что функция сломана с соответствующим кодом, а если совпадает, то просто поменять конечное состояние.

Если строка — это вызов другой функции, то сначала надо пройтись по последовательности мьютексов другой функции. Если очередной мьютекс новый, то надо добавить его в порядок и записать значения из другой функции. Если его требуемое начальное состояние не совпадает с нашим конечным, то надо записать, что функция сломана с кодом мьютекса из другой функции и остановить процесс. Иначе надо просто перезаписать конечное состояние мьютекса в нашей функции. Если мы дошли до конца этого процесса и не сломались, но другая функция сломана, то надо записать, что наша — тоже сломана с кодом ошибки из другой функции.

Эти значения для функций можно считать dfs'ом из main (не заходя в уже посчитанные функции еще раз).

В конце надо вызвать main, считая, что все мьютексы уже увидены и изначально не захвачены.

Задача L. Пересекающиеся отрезки

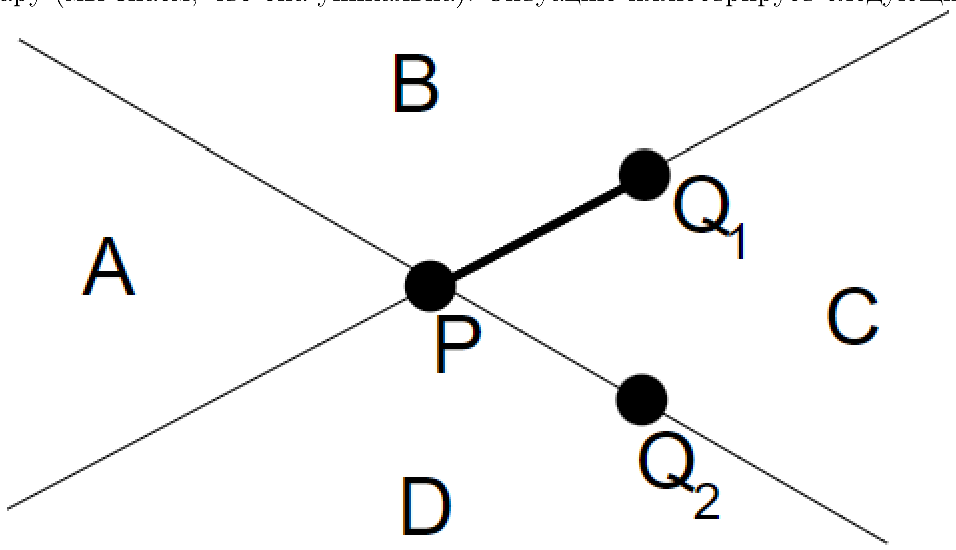
Мы можем расширить любой отрезок S до прямой L, которая делит плоскость на две части. В искомом множестве отрезков S пересекает все остальные отрезки, а значит все остальные отрезки имеют по одной конечной точке по обе стороны от L (при этом никакая точка не может находиться на L, иначе она будет на одной прямой с конечными точками S, чего не может быть по условию). Это означает, что если мы выберем точку P на входе, то она должна быть соединена с другой точкой Q так, чтобы прямая, проходящая через обе точки, оставляла с каждой стороны одинаковое количество оставшихся точек. Это намекает на алгоритм: выбираем точку P, находим Q в соответствии с приведенным выше определением, соединяем P и Q и повторяем. Однако что делать, если таких Q будет несколько?

Выберем в качестве P крайнюю левую точку, то все кандидаты на Q окажутся в правой половине плоскости, разрезанной вертикальной линией, проходящей через P (при этом на этой линии может находиться не более одной другой точки). Рассмотрим линию, вращающуюся по часовой стрелке с центром на P. В начальный момент на левой стороне линии нет точек, потому что P — самая левая. При вращении линия будет проходить через все точки, неоднократно добавляя одну точку к одной стороне и удаляя одну с другой. Поскольку коллинеарных троек точек не существует, этот сдвиг всегда равен 1. Это означает, что существует ровно одна такая прямая, на которой есть P и еще одна точка, с $n-1$ точками на каждой стороне. Более того, если отсортировать все точки, не являющиеся P, по углу, который они образуют с P (выбрав 0 для совпадения с вертикальной линией), то точка, которая окажется на линии, будет в точности медианой отсортированного списка.

Выбор P в качестве крайней левой точки дает уникальный выбор точки Q в пару к P. Мы можем удалить обе точки и продолжить. Алгоритм требует n итераций, и в каждой из них мы должны найти крайнюю левую точку и отсортировать $O(n)$ точек по углу. Выбор медианы и удаление пары точек можно сделать за время O(n) или лучше. Таким образом, общий алгоритм требует $O(n^2 \log n)$ времени (если вместо сортировки использовать линейный алгоритм для нахождения медианы, то сложность уменьшится до $O(n^2)$).

Чтобы решить задачу дальше, нужно заметить, что если множество из $2n$ точек можно разбить требуемым образом на отрезки, то для каждой точки P в этом множестве найдется ровно одна Q такая, что прямая через P и Q оставляет половину точек по обе стороны. То, что существует хотя бы одна такая точка, следует из существования решения, а аргумент о том, что их не может быть больше одного, требует более глубокого осмысления.

Предположим, что существует точка P и две другие точки Q_1 и Q_2, такие, что обе прямые, проходящие через P и каждую Q_i, имеют по половине всех остальных точек с каждой стороны. Более того, не теряя общности, предположим, что Q_1 — это та точка, которая действительно составляет с P пару (мы знаем, что она уникальна). Ситуацию иллюстрирует следующий рисунок.



На рисунке мы обозначили четыре области, на которые две прямые делят плоскость. Пусть A, B, C и D — это наборы исходных точек, содержащихся в каждой области (не включая P, Q_1 и Q_2). Обратите внимание, что любая точка в A должна быть соединена с точкой в C, чтобы отрезок пересекал PQ_1. Аналогично, любая точка в D должна быть соединена с точкой в B. Все точки под прямой, проходящей через P и Q_2, находятся либо в A, либо в D, то есть их $|A| + |D|$. Число точек выше, с другой стороны, равно $|B| + |C| + 1$. Поскольку мы показали, что $|A| \leqslant |C|$ и $|B| \leqslant |D|$, $|B| + |C| + 1 \geqslant |A| + |D| + 1 > |A| + |D|$. Это противоречит предположению, что прямая, проходящая через P и Q_2, имеет одинаковое количество точек входа с каждой стороны.

Это наблюдение само по себе позволяет лишь избежать шага «найти крайнюю левую точку» предыдущего алгоритма, что не меняет его общей временной сложности. Определенное улучшение заключается в более быстром сокращении набора точек, которые мы должны рассмотреть, чтобы все шаги в рамках итерации занимали меньше времени. Для этого рассмотрим, что после того, как мы сопоставили M пар точек, прямые делят плоскость на $2M$ областей, причем только внешние области (те, что имеют неограниченную площадь) могут содержать оставшиеся точки. Более того, каждая точка должна быть соединена с точкой в области, противоположной ей, чтобы пересечь все прямые, что является необходимым условием для пересечения всех отрезков.

Когда мы создаем новую пару, ровно две такие области делятся пополам. Идея заключается в следующем: вместо того чтобы удалять новую пару и продолжать работу с полным набором, будем рекурсивно работать с двумя отдельными наборами. Если мы рассматриваем наборы X и Y из противоположных областей, разбитых после создания пары на X_1 и X_2 и Y_1 и Y_2 соответственно, то нам нужно решать рекурсивно задачу для X_1 и Y_1 отдельно, и для X_2 и Y_2 отдельно (концепция разделяйки). Тут предполагается, что области обозначены так, что X_1, X_2, Y_1, Y_2 идут подряд в порядке по часовой стрелке.

Последнее, осталось убедиться, что мы разделили X и Y более или менее равномерно с помощью новой линии. Обратите внимание, что если мы ограничимся тем, что начнем с точки P, которая входит в выпуклую оболочку X или Y или $X \cup Y$, то это может оказаться невозможным (то есть все эти пары могут разделить X и Y, оставив большинство оставшихся точек на одной стороне).

Отсюда возникает необходимость в свойстве уникальности разбиения множества точек на требуемые пары. Заметьте, что если мы отсортируем все пары между точками X и Y по наклону произведенных ими отрезков, то первая и последняя разделят X и Y на пустое множество и множество с $|X|-1$ точками (заметьте, что $|X| = |Y|$). Второй и предпоследний разделят их на множество с 1 точкой и множество с $|X|-2$ точками. i-й разделит их на множество с $i-1$ точкой и множество с $|X|-i$ точками. Таким образом, при случайном выборе мы получаем рекурсию, аналогичную рекурсии quicksort, в которой уменьшение размера множеств ожидается достаточно близким к тому, чтобы всегда разбивать их равномерно. Поскольку время, необходимое для нерекурсивной части решения для множества размера m, составляет $O(m \log m)$ (вычисление разбиений является лишь дополнительным линейным шагом по времени), то общая ожидаемая временная сложность этого рекурсивного алгоритма составляет $O(n \log^2 n)$.

Задача M. Рейтинг стартовой точки

Для каждой стартовой точки мы можем выполнить двоичный поиск по минимальному значению d, так что мы можем достичь как минимум t точек (это работает, потому что увеличение d только увеличивает число доступных точек). Однако этот подход даёт сложность $O((nm)^2 \log 10^9)$, что слишком медленно, учитывая наши ограничения.

Заметим, что рейтинги близлежащих точек часто зависят друг от друга. Например, если мы знаем, что рейтинг точки P равен d, то каждая точка, которую мы достигнем из P также будет иметь рейтинг не более d (мы знаем, что можем добраться из любой из этих t точек в любую другую). Возможно, что эти другие точки на самом деле имеют более низкий рейтинг. Если бы мы могли каким-то образом сначала найти точки с наименьшим рейтингом, то мы могли бы использовать эту информацию для эффективного поиска точек с более высоким рейтингом.

Мы можем сделать это, думая о клетках таблицы как о вершинах графе. Возможные ребра в этом графе находятся между соседними парами точек, а их веса ребер равны разнице между значениями этих точек. Мы добавляем ребра в порядке возрастания веса в наш граф и отслеживаем каждую компоненту связности. Это означает, что если мы добавили ребра до веса d, то каждая компонента будет просто представлять набор точек, которые мы можем посетить, используя разницу высот не более d. Таким образом, мы можем назначить рейтинг всем точкам в компоненте, когда размер компоненты составляет не менее t.

Единственные операции графа, которые нам нужно поддерживать — это добавление ребра, проверка того, соединены ли две вершины, и запрос размера компоненты. Все это очень быстро делается с помощью СНМ.

Задача N. Сортировка матрицы

У каждого числа есть цвет — то, в какой строке это число должно оказаться в итоге. Перед третьей операцией, чтобы отсортировать матрицу получилось, все числа должны стоять в строках, соответствующих их цвету. То есть перед второй операцией надо, чтобы в каждом столбце все числа были разных цветов, тогда второй операцией надо будет просто отсортировать в каждом столбце числа по цвету.

Будем получать результат первой операции по столбцам слева направо. Пусть осталось k столбцов из m исходных. Построим двудольный граф, где в левой доле — строки, в правой — цвета, и вершины соединены ребром, если в соответствующей строке есть соответствующий цвет. Каждого цвета осталось k штук, то есть для любых t строк в них осталось tk чисел, то есть хотя бы t различных цветов. Значит по лемме Холла в этом графе есть совершенное паросочетание. Найдем его и в каждой строке поставим парный ей в паросочетании цвет на первый из оставшихся столбцов. Теперь в этом столбце все цвета различны и можно идти дальше. Итого $m\times$ поиск максимального паросочетания в графе с $2n$ вершинами и не более nm ребрами. Если искать паросочетание алгоритмом Куна, то будет $\mathcal{O}(n^2m^2)$.