Задача 1. Поло

Подзадача 1. Достаточно перебрать все варианты распределения игр между двумя полями. Асимптотика такого решения – $O(2^N)$.

Подзадача 2. Для решения этой и последующей подзадач уже важно иметь понимание, что ответ всегда — `YES`. Действительно, если Ане достанется игра $(a\ b)$, где команда a играет в белой форме, а команда b – в красной, тогда Кате останется только одна игра вида $(c\ b)$, где c – другая команда в белой форме. Значит, при дальнейшем распределении игр таким образом не образуются конфликты, так как за каждую такую итерацию мы распределяем оба выступления одной команды в одном цвете. Следовательно, ситуации, где мы отдали кому-то игру, а парной к ней нет, не может быть. Если две последние распределенные игры имели вид — $(a\ b)$ и $(a\ b)$, то можно продолжить распределение с любой еще не распределенной игры. В данной подзадаче находить пару к игре можно за $O(N)$, просматривая все игры. Итоговая асимптотика — $O(N^2)$.

Подзадача 3. Очевидно, что находить пару к игре можно быстрее, чем за $O(N)$, используя, например, ассоциативный массив или любую другую структуру данных, позволяющую быстро обращаться к играм по первому или второму номеру команды в паре. Асимптотика полного решения — $O(N)$ или $O(N \cdot \log(N))$, в зависимости от реализации.

Задача 2. Увольнение

Представим иерархию компании в виде графа

Тогда вершинами этого графа будут сотрудники, а рёбрами — прямые отношения начальник-подчинённый. Так как никакой сотрудник не является своим подчинённым, то в этом графе нет циклов. Так как все сотрудники являются подчинёнными гендиректора, то этот связный граф – дерево, а гендиректор – корень этого дерева.

Подзадача 1. Для каждого варианта для каждой вершины из списка тех, которые надо оставить, будем идти по направлению к корню, пытаясь найти первую вершину, которую тоже требуется оставить.

Асимптотика худшего случая – $O(N * Q * K)$.

Подзадача 2. Для каждого варианта обойдём всё дерево методом поиска в глубину (dfs), отыскивая те вершины, которые надо оставить.

Асимптотика — $O(N * Q)$.

Подзадача 3. Применим метод обхода в глубину к нашему дереву, для каждой вершины вычислим три числа:

1. Глубину вершины – длину пути от корня до вершины.
2. Количество обработанных вершин. Обработанной вершиной считается вершина, чьих потомков уже обработали. Листья считаются обработанными сразу.
3. Количество встреченных вершин, т.е. тех вершин, которые начали обрабатываться.

Для каждой вершины количество встреченных вершин вычисляется в момент, когда мы заходим в эту вершину (начинаем её обрабатывать), а количество обработанных вершин вычисляется в момент, когда мы из вершины выходим.

Тогда будет верно для любой вершины, что все её предки будут иметь меньшее значение встреченных до этого вершин, (так как после того, как был встречен предок, было встречено, как минимум, на одну вершину больше) и большее значение обработанных вершин (так как после того, как была обработана эта вершина, было обработано, как минимум, на одну вершину больше).

Тогда для каждого варианта можно перебрать все вершины из тех, что надо оставить, и для каждой вершины из её оставшихся предков найти такую, которая будет дальше всего от корня.

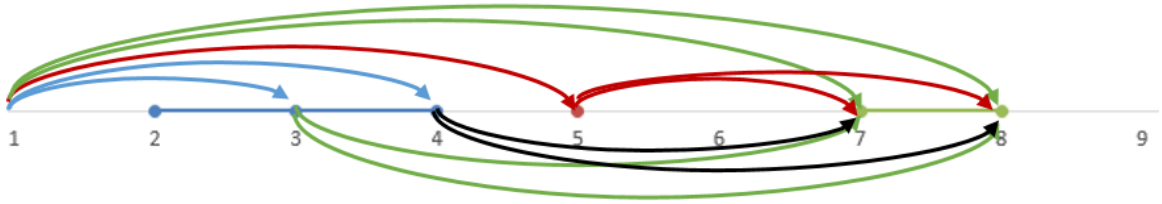
Асимптотика $O(Q * K^2)$.

Задача 3. Поймай крота

Подзадача 1. Ограничения в первой подзадаче позволяют написать перебор всех вариантов пути от начальной точки до изображений кротов. Эту подзадачу можно реализовать с помощью

перебора всех двоичных масок длины K или с помощью рекурсивного алгоритма. Асимптотика такого решения $O(2^K)$.

Подзадача 2. Рассмотрим каждого крота вместе с отрезком времени, который он проводит на поле, как несколько вершин в графе, а ребра между вершинами есть в том случае, когда игрок успевает осуществить переход из точки на поле до другого изображения в отведенное время. Пример такого графа для первых трёх точек из первого примера приведён ниже. Промежутки для всех трёх разных точек выделены разным цветом. Разный цвет рёбер использован для наглядности.



Самый длинный путь в таком графе является искомым ответом на задачу. Так как граф ориентированный и ациклический можно воспользоваться динамикой по графу и посчитать самый длинный путь для каждой вершины. Реализация динамики «вперёд» проходит подзадачу 2 и имеет асимптотику $O(K^2 \cdot T^2)$.

Подзадача 3. При реализации алгоритма из предыдущего пункта можно заметить, что переходы вперёд от одного изображения до другого на расстояние дальше, чем на $N + M - 1$ можно не использовать, так как первые $N + M - 1$ элементы отрезка будут гарантированно доступны из других вершин. Оптимизация с уменьшением правой границы каждого отрезка позволяет уменьшить длину всех отрезков и ускоряет решения при маленьких размерах поля. Такое решение имеет асимптотику $O(K^2 \cdot (N + M)^2)$.

Подзадача 4. Объединим идеи для второй и третьей групп тестов, при этом развернув динамику. Для обновления текущего значения динамики в точке i достаточно будет хранить максимальное значение на префиксе $[1, i - (n + m - 2)]$, а элементы $[i - (n + m - 2), i - 1]$ нужно обновить. Таким образом, для каждой позиции будет рассмотрено не больше $N + M$ переходов. В итоге для каждого из T моментов времени рассматриваются не более чем $N + M$ переходов, что дает асимптотику алгоритма $O((N + M) \cdot T)$.

Задача 4. Лиза и книга

Подзадача 1. Будем использовать структуру `std::vector` для хранения количества строк в каждой главе. Вставка или удаление элемента будет иметь асимптотику $O(N)$. Запросы `size` и `chapter` будем выполнять, пробегая по всему массиву. Асимптотика такого решения будет $O(Q \cdot N)$.

Подзадача 2. Применим метод корневой декомпозиции. Для каждой главы будем хранить количества строк и количество страниц, которые она занимает. Если перестраивать корневую декомпозицию при переполнении одного из блоков (например, если он увеличился в два раза относительно начального размера), то асимптотика операций вставки, удаления, нахождения суммы на отрезке будет равна $O(\sqrt{N})$.

Для выполнения запросов `chapter` будем использовать бинарный поиск по суммам на подотрезках вида $[0,\ x]$, он будет работать за $O(\log N \cdot \sqrt{N})$. Асимптотика такого решения — $O(Q \cdot \log N \cdot \sqrt{N})$.

Подзадача 3. Будем поддерживать размер книги в страницах. При изменении размера главы пересчитывается размер книги за $O(1)$. Асимптотика решения на допустимых в подзадаче запросах — $O(Q)$.

Подзадача 4. Заведем массив частичных (префиксных) сумм для количества страниц в главах. Тогда запрос `chapter` можно выполнять при помощи бинарного поиска по этому массиву за $O(\log N)$. Асимптотика решения на допустимых в подзадаче запросах — $O(Q \cdot \log N)$.

Подзадача 5. Для хранения строк и страниц в главах будем использовать структуру, которая называется декартово дерево по неявному ключу. В этой структуре обращение, вставка, удаление и изменение элемента по индексу, а также прибавление на отрезке работает за $O(\log N)$. Предлагается хранить два дерева — для глав и для частичных сумм. Обновление второго дерева осуществляется при помощи операции прибавления на отрезке.

Второе дерево удобно использовать для выполнения запросов `chapter`. Такой запрос будет работать за $O(\log^2 N)$. Все остальные запросы выполняются за $O(\log N)$. Асимптотика решения — $O(Q \cdot \log^2 N)$.

Задача 5. Иннокентий и шифрование

Первые две подзадачи можно решить, явно производя операции, описанные в условии задачи. Время работы $O(Q^2)$.

Для того, чтобы решить остальные подзадачи, научимся быстро отвечать на запрос вычисления ключа. Для того, чтобы (`xor`) x и числа на отрезке принимал максимальное значение, нужно брать те числа, у которых (`xor`) старших битов были равны 1, т.к. $2^p > \sum_{i=0}^{p-1} 2^i$. Таким образом, чтобы максимизировать сумму, нужно жадно набирать числа, старшие биты которых отличны от соответствующих битов x.

Заметим, что количество чисел с заданным p-ым битом на отрезке $[l, r]$ равно количеству чисел на отрезке $[1, r]$ минус количество чисел $[1, l-1]$. Отсюда получаем следующую идею: использовать структуру данных, которая позволяет хранить числа в некотором порядке, поддерживающую их количество на отрезке $[1, k]$.

Для третьей подзадачи можно использовать, например, персистентное дерево отрезков по значению. Для решения четвертой подзадачи дерево отрезков из-за ограничений по памяти не пройдет, поэтому вместо него нужно использовать персистентный битовый бор.

Научимся теперь считать ответ на запрос. Пусть во время обхода структуры мы попали в некоторую вершину v с количеством чисел T на глубине d и нам необходимо набрать ещё K' чисел.

Рассмотрим два случая: $T \leqslant K'$ и $T > K'$.

В первом случае необходимо набрать все числа в поддереве вершины v. Во втором случае нужно посчитать ответ для поддерева с противоположным битом. Если мы еще не набрали необходимое количество чисел, посчитаем ответ во втором поддереве. Для того, чтобы посчитать ответ для первого случая, запишем в вершине количество единичных бит в каждом разряде до d.

Итоговое время работы: $O(Q \log Q)$.

Задача 6. Дорога в парке

Любая прямая на плоскости представима в виде равенства $ax + by + c = 0$.

Подзадача 1. Для решения первой подзадачи можно перебрать угол наклона прямой φ. Тогда уравнение прямой для данного угла будет иметь вид: $x \cos\varphi + y \sin\varphi + c = 0$. Чтобы однозначно определить прямую, найдем такой коэффициент c, что расстояние от точек до нее наименьшее. Для этого можно положить $c = 0$ и для каждой точки посчитать величину $x \cos\varphi + y \sin\varphi$. Посчитав эти значения, можно определить прямую следующим образом: если отрезок проекций первого множества не пересекается с отрезком проекции второго множества, то середина расстояния между отрезками — ответ для заданного угла. Итоговое время работы $O(acc * (n + m))$, где acc — частота, с которой необходимо перебирать угол $acc \approx 10^6$

Для нахождения полного решения будем использовать тот факт, что ответом всегда будет серединный перпендикуляр к отрезку кратчайшего расстояния между двумя множествами, т.к. любая разделяющая прямая проходит через этот отрезок. Следовательно, для того чтобы расстояние от прямой было наибольшим, нужно чтобы она проходила через середину отрезка и была перпендикулярна ему.

Подзадача 2. В данном случае достаточно посчитать кратчайшее расстояние между точками одного множества и точками из другого и вывести серединный перпендикуляр для кратчайшей прямой. Асимптотика такого решения $O(nm(n + m))$.

Подзадача 3. Для решения третьей подзадачи достаточно проверять только точки на выпуклой оболочке. Поэтому нужно построить выпуклые оболочки для обоих множеств и выполнить действия для предыдущей подзадачи. Асимптотика: $O(n \log n + m \log m + n * m)$.

Подзадача 4. Для полного решения задачи нужно заметить, что расстояние от вершины из одной из оболочек до второй — выпуклая вниз функция, поэтому можно искать кратчайшее расстояние с помощью тернарного поиска. В результате решение находится за время $O((n + m)(\log n + \log m))$.

Также для решения задачи можно использовать другие методы поиска расстояния между множествами точек, например, спроецировать точки на прямую, проходящую через центры масс множеств и найти пару кратчайших и перебрать их соседей или использовать метод опорных векторов.