Задача A. Наибольшее кратное трём

Пусть число единиц в строке — c_1, а нулей — c_0. Просто наибольшее возможное число без двух 1 подряд выглядит как `101010...101000...`. Если $c_0 < c_1 - 1$, то такого числа не существует и ответ — `NO`. Если число единиц во входной строке делится на 3, то это число тоже уже делится на 3.

Теперь если $c_0 = c_1 - 1$, то другого числа с таким количеством 0 и 1 не существует. То есть теперь $c_0 \geqslant c_1$.

Пусть $c_1 = 3k + 1$. Если $k = 0$, то может получиться только степень 2, то есть ответа нет. Если $k \geqslant 1$, то число вида `101010` $k - 1$ раз, потом `10100101`, потом `0` $c_0 - c_1$ раз делится на 3 и нетрудно видеть, что все большие числа не делятся на 3 (есть только два больших числа).

Пусть $c_1 = 3k + 2$. Тогда число вида `101010` k раз, потом `1001`, потом `0` $c_0 - c_1$ раз делится на 3, а единственное большее число не делится на 3.

Задача B. Недлинный путь максимальной стоимости

Будем в каждой вершине считать двоичные подъемы (по степеням двойки до 10^{12}) и для каждого подъема считать суммарную длину ребер пути длины 2^k (и стоимость вершин на нем), заканчивающегося в данной вершине.

Теперь для каждой вершины можно будет понять максимальную стоимость требуемого пути, заканчивающегося в ней, просто проходясь по подъемам. Будем пробовать прыгнуть на 2^k, и если этот путь еще не очень длинный, то прыгать, а иначе уменьшать степень k. Осталось выбрать максимум по всем вершинам.

Задача C. Новая настолка

Очевидно, что, чтобы трубы можно было покрасить, необходимо, чтобы любая специальная ячейка либо лежала на границе поля, либо имела свободную соседнюю с ней ячейку. Количество полей, удовлетворяющих этому условию, можно посчитать формулой включений-исключений по множеству специальных ячеек, все соседние с которыми — особые или занятые за $2^{20} \cdot 20$ (так как специальных ячеек не более 20).

Покажем, что этого условия достаточно. Ребра между специальной и занятой клетками можно покрасить в конце, для этого у нас точно останутся цвета. Осталось покрасить ребра между специальными клетками. Удалим самую верхнюю из самых левых специальных клеток (назовем ее C), покрасим все ребра без нее и вернем ее обратно. У C есть не более двух особых соседей (справа и снизу). Если сосед один, то у него точно остался цвет, чтобы покрасить ребро в него. Если соседей два и при этом неверно, что у них самих есть по 2 соседа кроме C и пары уже занятых цветов ребер у этих соседей совпадают, то несложно видеть, что ребра из C можно покрасить. Пусть теперь соседи C — A и B, и у A и B уже заняты красный и синий цвета. Попробуем перекрасить красное ребро из A в желтый. Если при этом раскраска сломалась, то из другого конца этого ребра уже есть желтое ребро. Перекрасим его в красный и пойдем дальше. Мы не можем прийти в вершину, в которой уже были (это бы означало, что в какую-то вершину приходило 2 ребра одного цвета или что в A было желтое ребро). Значит в какой-то момент мы закончим. Если в конце пути мы пришли не в B, то теперь пары цветов у A и B не совпадают и ребра в C можно покрасить. Если же мы пришли в B, то первое и последнее ребро перекрашенного пути теперь желтые, то есть путь имеет нечетную длину, то есть A и B разного цвета в шахматной раскраске доски. Но у них есть общий сосед, то есть они одного цвета. Значит всегда можно получить нужную раскраску.

Задача D. Ямы, песок и доска

Будем для каждого начала отрезка ям, которые мы заделали, искать самый дальний возможный конец. Очевидно, что если двигать начало вправо, то конец тоже будет двигаться вправо. То есть если научиться за быстро проверять, что отрезок можно покрыть, то всю задачу можно будет решить двумя указателями. Отрезок можно покрыть, если его сумма минус максимальная сумма подотрезка длины d не превосходит m. Максимальная сумма подотрезка длины d — максимум на отрезке в массиве сумм подотрезков длины d. Границы этого отрезка тоже движутся только вправо в процессе двух указателей, то есть максимум можно искать любым из линейных алгоритмов поиска максимума в окне, то есть решение работает за $\mathcal{O}(n)$.

Задача E. Все перестановки как подпоследовательности

Пусть нас есть строка, состоящая из 25 копий алфавита, каждая из которых расположена в обычном порядке. Эта строка почти является решением! Почему? Почти любая перестановка содержит две последовательные буквы, такая что первая меньше второй. В таком случае мы можем взять обе эти буквы из одной и той же копии алфавита. Например, когда мы ищем перестановку kjbd..., мы можем взять k из первого алфавита, j из второго, а обе b и d из третьего. Существует только одна перестановка, которую нельзя найти в нашей укороченной строке: перевернутый алфавит. К счастью, это легко исправить, добавив еще одну букву a к нашей строке. Это решение имеет длину $25 \times 26 + 1 = 651$, что пока недостаточно хорошо.

$$(abcdefghijklmnopqrstuvwxyz)^{25}\ a$$

Для оптимизации давайте зафиксируем для примера небольшое $n = 7$ и посмотрим на бесконечную строку $S = (abcdefg)^{\infty}$. Мы знаем, что префикс длины $n(n-1)+1$ уже содержит все возможные перестановки алфавита. Теперь обобщение: рассмотрим все строки длины k, которые не содержат ни одной буквы дважды. Назовем их разнообразными строками длины k (таких строк нет для $k > n$, а для $k = n$ это именно наши перестановки). Какой самый короткий префикс S, который содержит каждую из разнообразных строк длины k?

Ответ довольно прост: **достаточно** $k(n-1)+1$ **букв** (обозначим это утверждение за T). Например, строка $abcdefg$ содержит все однобуквенные строки, а $abcdefgabcdef$ содержит все двубуквенные разнообразные строки для алфавита от a до g. Доказательство по сути такое же, как и для перестановок. На самом деле, не только префиксы строки S обладают этим свойством. Любая подстрока строки S длины $k(n-1)+1$ также подходит (это связано с тем, что каждая подстрока S выглядит абсолютно одинаково, только порядок букв в алфавите меняется.)

Последний шаг на пути к полному решению, давайте научимся добавлять одну букву, то есть посмотрим, как выглядят ответы, добавляя букву h. Например, вот такая строка содержит все перестановки вида ??h? (каждая из двух частей слева и справа удовлетворяет T).

```
abcdefgabcdef h gabcdef
```

Аналогично, следующая строка подходит под шаблон ???h (буква f тут по факту остается не нужна, но мы оставим её, чтобы визуально показать, что символ h просто переместился на шесть $(n-1)$ позиций вправо):

```
abcdefgabcdefgabcde h f
```

Чтобы удовлетворить оба шаблона, то есть двум позициям h, объединим два подхода, то есть вставим еще одну h:

```
abcdefgabcdef h gabcde h f
```

Отсюда уже становится понятна общая конструкция ответа. Рассмотрим бесконечную строку S для первых $n-1$ букв алфавита. Возьмем ее префикс P длины $(n-1)(n-2)+2$. Для каждого i от 0 до $n-1$ включительно вставим последнюю букву алфавита после $1 + i(n-2)$ букв из P. Вот вся строка (с пробелами для ясности) для $n = 8$:

```
a h bcdefg h abcdef h gabcde h fgabcd h efgabc h defgab h cdefga h b
```

Должно быть очевидно, что эта строка содержит каждую из 8! возможных перестановок своего алфавита: например, для любой перестановки вида ???h???? мы можем использовать четвертую h в нашей строке и игнорировать остальные h. Строка из букв $a - g$ перед четвертой h достаточно длинная, чтобы содержать любую 3-буквенную разнообразную строку, и то же самое справедливо для части после выбранной h и 4-буквенных разнообразных строк.

Эта конструкция начинается со строки длины $(n-1)(n-2)+2$, а затем вставляет n дополнительных символов. Таким образом, общая длина построенной строки составляет $n^2 - 2n + 4$, что при $n = 26$ составляет 628.

Задача F. Назначения с бонусами

Пусть зафиксировано множество строк, клетки в которых взяты в первых k столбцах. Тогда нам надо знать только максимальную стоимость этих ячеек с учетом бонусов на этом префиксе столбцов

(большая стоимость всегда лучше меньшей). То есть можно написать динамику по подмножествам строк. В пересчете надо сначала посчитать, какую максимальную стоимость может иметь множество до применения бонусов, а потом применить бонусы в порядке возрастания их p_i. Итого решение работает за $\mathcal{O}(2^n(n+m))$.

Задача G. Мороженое ожидание

Пусть мы идем слева направо и каждый студент сейчас получил какое-то количество мороженого. Можно показать, что очередное мороженое надо отдать тому студенту, у которого уже есть наибольшее число порций, и которому его можно отдать (Доказывается перебором. Предполагаем, что это не так, рассматриваем студента, которому дали мороженое в итоге и студента, которому должны были отдать по нашему алгоритму и перебираем все варианты распределить их 6 порций между собой). При этом не будет одновременно возникать двух студентов, у которых есть по одной порции, но при этом вкусы разные (более позднюю порцию надо было отдать первому студенту). То есть надо просто поддерживать количества студентов каждого типа и каждый раз домножать на соответствующее количество.

Задача H. Чудо-машина с карточками

Рассмотрим ориентированный граф, где вершины соответствуют карточкам и из карточки выходит 2 ребра в карточки, в которые ее можно преобразовать. Вершины, из которых не достижима единица, можно удалить, так как из этих карточек никогда не получится получить один. Вершины, которые не достижимы ни из какой вершины с ненулевым числом карточек тоже можно удалить, так как таких карточек никогда не будет.

Теперь если в графе без единицы есть цикл, то можно привести туда одну карточку и двигать ее по циклу до бесконечности, отправляя все побочные результаты в 1. Из цикла достижима 1, то есть мы получим сколь угодно много карточек с ней.

Пусть из 1 есть хотя бы одно ребро. Тогда есть цикл, содержащий единицу. Если в этом цикле есть вершина исходящей степени 2, то тоже можно привести в этот цикл одну карточку и двигать ее, и ответ тоже бесконечность. Если же все вершины исходящей степени 1, то ребро из 1 бесполезно использовать, так как карточка, проведенная по нему может только просто вернуться обратно, то есть это ребро можно удалить.

Теперь из 1 нет ребер, то есть весь граф ацикличен и можно просто все карточки протолкнуть по всем ребрам в порядке обратной топологической сортировки и собрать в единице.

Задача I. Максимальный и минимальный диаметры

Пусть мы выбрали ребро, которое надо удалить. Тогда если диаметры в компонентах, на которые распалось дерево — d_1 и d_2, то максимальный диаметр нового дерева — $1+d_1+d_2$, а минимальный — $max(d_1, d_2, 1 + \left\lceil \frac{d_1}{2} \right\rceil + \left\lceil \frac{d_2}{2} \right\rceil)$ (в случае минимума выгоднее всего добавить ребро между серединами диаметров, и для любого другого ребра диаметр будет хотя бы таким же). То есть для решения задачи достаточно найти все диаметры всех возможных компонент, получаемых удалением одного ребра.

Чтобы это сделать, найдем во всем дереве диаметр и подвесим дерево за один из его концов. Теперь надо найти диаметр каждого поддерева и каждого наддерева (дополнения к поддереву).

Диаметры в поддеревьях можно найти динамикой по поддеревьям, считающей самую глубокую вершину и диаметр в поддереве. Самая глубокая вершина нужна, чтобы обновляться через диаметры, проходящие через корень поддерева.

Во всех наддеревьях вершин не из диаметра (наддерево вершины это дополнение к ее поддереву) сохранился диаметр всего дерева. То есть для них диаметр уже найден. Теперь научимся искать диаметр наддерева вершины v на диаметре. Пусть корень дерева — r, а родитель v — p. Как известно, диаметр дерева можно искать так: первый конец это самая удаленная вершина от любой выбранной вершины, второй конец — самая удаленная вершина от первого конца. r — конец диаметра всего дерева, v лежит на диаметре, то есть r — одна из наиболее удаленных вершин от p в наддереве (иначе бы был строго больший диаметр). То есть диаметр наддерева — расстояние от корня до

самой глубокой вершины в наддереве. Самые глубокие вершины в наддеревьях вершин диаметра легко посчитать сверху вниз через уже посчитанные самые глубокие вершины в поддеревьях.

После того как мы нашли оптимальные ребра для удаления для минимизации и максимизации диаметра, можно найти сами диаметры в компонентах и их середины, чтобы найти ребро, которое надо добавить. Итого решение работает за $\mathcal{O}(n)$.

Задача J. Чокопаи за турнир

Назовем игру нежелательной, если в данный момент мы выдали чокопай проигравшей команде, но не выигравшей.

Пусть $lose_v$ — количество раз, которые команда v проиграла в турнире. Пусть последовательность выдачи чокопаев, минимизирующая максимальное количество нежелательных игр в процессе это $v_1, v_2, \ldots, v_n$. Тогда обозначим за $lose_v^k$ количество раз, которое команда v проиграла одной из $v_1, v_2, \ldots, v_k$. Теперь если мы уже выдали k чокопаев, то количество нежелательных игр — $\sum_{i=1}^{k}(lose_{v_i} - lose_{v_i}^k) = \sum_{i=1}^{k} lose_{v_i} - \sum_{i=1}^{k} lose_{v_i}^k$. Поскольку игры были сыграны между каждой парой команд и ничьих не было, вторая сумма равна $\frac{k(k-1)}{2}$. То есть надо минимизировать первую сумму. Если мы будем выдавать чокопаи в порядке возрастания $lose_v$, то мы минимизируем все префиксные суммы одновременно, то есть и максимум тоже, то есть так и надо выдавать.

Задача K. Перестановка и фиксированные обмены

Нам надо найти кратчайшее расстояние от одной вершины до другой в каком-то графе (где вершины это перестановки, а ребра — разрешенные обмены). Причем количество вершин в графе — $n!$, то есть не сильно больше, чем что-то, для чего можно запустить простой `bfs`. Поэтому нам достаточно написать какой-то `bfs` с отсечениями, который не посетит все вершины.

Можно оценить близость каждого состояния к завершению (к тождественной перестановке) и написать алгоритм наподобие Дейкстры, но достающий каждый раз не перестановку с минимальным расстоянием, а перестановку с минимальной суммой расстояния и оценки на близость (а при равенстве — с минимальным расстоянием). Это будет алгоритм `A*`.

В качестве оценки близости перестановки к тождественной возьмем ($n-$ количество циклов в перестановке). Покажем, что тогда если в какой-то момент времени мы достали вершину v, то расстояние до нее уже не изменится. При любом обмене количество циклов увеличивается не более, чем на 1, то есть сумма, которую мы рассматриваем не уменьшается. А тогда если мы достали перестановку, то все, что мы достанем далее либо имеет большую сумму и не может обновить эту перестановку, либо имеет такую же сумму и не меньшее расстояние и тоже не сможет нас обновить. То есть алгоритм можно остановить, когда мы найдем тождественную перестановку, и все перестановки будут доставаться из очереди не более одного раза. Благодаря специфике графа, это работает быстро.

Задача L. Минимальная странность

Чтобы максимальная сумма была минимальной, надо переупорядочить элементы b в обратном порядке элементов a, то есть максимальный элемент b должен быть на той же позиции, что и минимальный a, второй максимальный — на той же позиции, что и второй минимальный a и так далее. Поскольку числа в массивах маленькие можно поддерживать массивы подсчета на префиксе и после каждого добавления элемента искать максимальную сумму двумя указателями по массивам подсчета. Итого, если элементы массивов не превосходят A, то получилось решение за $\mathcal{O}(nA)$.

Задача M. Уравнение камень-ножницы-бумага

Если построить дерево разбора выражения, то задачу можно решить простой динамикой по поддеревьям. Для поддерева можно для каждого результата выражения в нем считать количество способов заменить знаки вопроса так, чтобы результат был таким. В пересчете можно перебрать значение левого и правого ребенка в дереве и прибавить произведение соответствующих количеств к значению для операции в вершине от значений детей.

Поскольку в дереве разбора операции в вершине всегда будут выполняться позже, чем операции в ее детях, дерево можно построить алгоритмом, похожим на построение Декартова дерева за линейное время. Будем идти по строке слева направо и поддерживать текущий путь из корня вправо в стеке. Когда мы встречаем очередную операцию (или символы R, S, P или ?, которые мы будем считать операцией максимального приоритета), надо пройти сверху вниз по стеку до первой операции с приоритетом меньше нашего, подвесить нас как ее правого сына, а ее старого правого сына сделать нашим левым; удалить все над ней со стека и добавить в стек нас. В итоге получится дерево из всех операций и символов, где родитель всегда выполняется позже ребенка, то есть искомое дерево разбора.