Открытая олимпиада 2020

Разбор задач

Москва, Сочи, 5-7 марта 2020 г.

«Медианный горный хребет»

- Идея задачи — Никита Сендерович
- Разработка задачи — Филипп Грибов

Постановка задачи

- Дан массив из n чисел, каждое не превышает a.
- За одну итерацию каждое число массива меняется на медиану из него и двух его окружающих.
- Требуется найти значение c — через сколько итераций числа в массиве перестанут меняться.
- Так же надо получить числа массива после c итераций.

Решение за $\mathcal{O}(n \cdot c)$ (19 баллов)

- Будем симулировать процесс, за $\mathcal{O}(n)$ производя по одной итерации.
- Так делаем пока массив меняется при очередной итерации.
- Так как гарантируется, что c не превышает 10 000, то ограничения по времени позволяют это сделать.

Числа не превышают 2 (24 балла)

- Если два одинаковых числа стоят подряд в массиве, то они меняться не будут.
- Тогда меняться могут только числа внутри отрезков из чередующихся 1 и 2.
- За одну итерацию концы таких отрезков останутся неизменными, а все числа внутри заменятся на противоположные.
- При этом числа рядом с концами отрезков станут равными концам отрезков и в будущем меняться не будут.

Числа не превышают 2 (24 балла)

- Итого за одну итерацию длины всех отрезков из чередующихся 1 и 2 уменьшаются на 2.
- Тогда общее число итераций равно длине максимального отрезка делённой на 2.
- После всех итераций все числа в левой половине отрезка станут равны левому концу отрезка, а в правой половине — правому.

1121212111	11212122
1112121111	11121222
1111211111	11112222
1111111111	

Решение за $\mathcal{O}(n \cdot a)$ (14 + 38 баллов)

- Заффиксируем некоторое число x и заменим числа, $< x$, на 1, а числа $\geqslant x$ на 2.
- Тогда если решить задачу для этого массива, то число 1 будет означать, что в исходной версии в этом месте будет число $< x$, а 2, что число $\geqslant x$.

Решение за $\mathcal{O}(n \cdot a)$ (14 + 38 баллов)

- Переберём все возможные x от 1 до a.
- Заменим на массив из 1 и 2 (по тому же принципу) и посчитаем число итераций, после которых новый массив перестанет меняться.
- Максимальное из этих количеств итераций по всем x и будет c.
- Итого асимптотика $\mathcal{O}(n \cdot a)$.
- Чтобы восстановить массив достаточно для каждого элемента массива найти самый большой x, что симуляция с этим x всё ещё приводит к “2”.

Нахождение c (+14 баллов)

- Нетрудно показать, что стоит рассматривать только те x, которые равны числам массива.
- Хотим понять сколько максимум итераций понадобится. Но число итераций при конкретном x равно 0.5 длины самого длинного чередующегося отрезка.
- Отсортируем все числа массива и будем рассматривать их как x в порядке убывания.
- Изначально сделаем x равным `max` числу массива и заведём массив из 1 и 2.
- Далее будем постепенно уменьшать x и пересчитывать $1 - 2$-отрезки.

Нахождение c (+14 баллов)

- Будем хранить все чередующиеся $1-2$-отрезки в `set`.
- Как этот `set` меняется при уменьшении x? Предположим на некоторой позиции p нужно заменить $1 \to 2$.
- В случае, если на позиции $p-1$ находится 1, нужно объединить отрезок с p с отрезком слева, содержащим $p-1$.
- В противном случае нужно наоборот разделить этот отрезок по позиции p.
- Аналогично с позицией $p+1$.

Нахождение c (+14 баллов)

- Так как все отрезки хранятся в *set*, то создание или удаление отрезков работает за $\mathcal{O}(\log n)$.
- Так как на каждой позиции изменение 1 на 2 происходит ровно один раз, то асимптотика $\mathcal{O}(n \log n)$.
- После того как пересчитали отрезки для нового x, пересчитываем ответ относительно новых длин.
- Нам осталось только получить сам массив после c итераций.

Полное решение (100 баллов)

- Будем так же уменьшать x и пересчитывать отрезки из чередующихся 1 и 2.
- Будем в `set` хранить позиции, на которых числа в итоговом массиве пока не известны.
- После рассмотрения очередного x научимся понимать, на каких позициях будут стоять числа, равные x.
- Это те позиции, в которых после c итераций написано 2 и они ещё не удалены из `set`

Полное решение (100 баллов)

- После уменьшения x некоторые отрезки останутся прежними, а некоторые изменятся.
- На отрезках, которые останутся прежними, не могут появиться новые позиции, про числа на которых мы раньше не знали.
- Для каждого изменившегося отрезка поймём во что они превратятся, исходя из их длин и чисел по краям.
- Если этот изменившийся отрезок порождает какой-то подотрезок с двойками, то извлечём из `set` все ещё не удалённые элементы на этом подотрезке и запишем для них ответ.

Полное решение (100 баллов)

- Получилось решение за $\mathcal{O}(n \log n)$, можно решить задачу за $\mathcal{O}(n)$.

«Интересные конкурсы»

- Идея задачи — Григорий Резников, МГУ
- Разработка задачи — Максим Деб Натх, ВШЭ

Постановка задачи

- Дана скобочная последовательность
- За t времени можно перемешать любой её подотрезок длины t
- Нужно за минимальное время сделать последовательность правильной скобочной

Ответ отрицательный

- Поймём, когда мы не можем сделать последовательность ПСП
- Тогда и только тогда, когда число «(» и число «(» в строке отличается
- Иначе всегда можно взять всю строку и переставить в ней за n символы так, чтобы строка стала ПСП, например, «((((...))))»

Основная идея

- Заметим, что в оптимальном решении отрезки, на которых производятся перемешивания не пересекаются (иначе можно было перемешать сразу нормально)
- Значит для перемешивания нужно выбрать набор непересекающихся отрезков.

- Скобочный баланс – разность между количеством «(» и количеством «)»
- За bal_i $(0 \leqslant i \leqslant n)$ обозначим скобочный баланс на первых i символах.

Решение за $\mathcal{O}(n^2)$ (50 баллов)

- Используем динамику dp_i = сколько времени нужно, чтобы сделать префикс из i символов префиксом ПСП
- Если уж перемешивать какой-то отрезок, то его оптимально делать вида «((...))».
- $dp_0 = 0$, $dp_i = -\infty$, если $bal_i < 0$, иначе

$$dp_i = \min \begin{cases} dp_{i-1} & \text{если } bal_i \geqslant 0 \\ j + dp_{i-j} & \text{по всем } j \leqslant i \text{ таким, что } bal_i \leqslant bal_{i-j} \end{cases}$$

Полное решение за $\mathcal{O}(n)$

Изобразим с.п. графически:

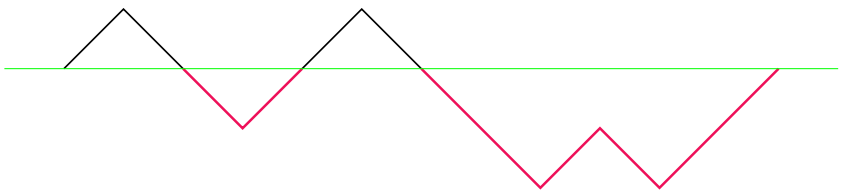


Рис.: ПСП до перемешиваний

Видно, что если взять все последовательности красных скобок и развернуть их, получим ПСП:

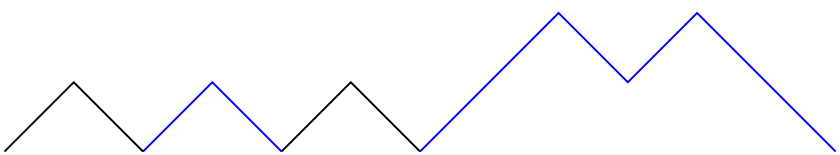
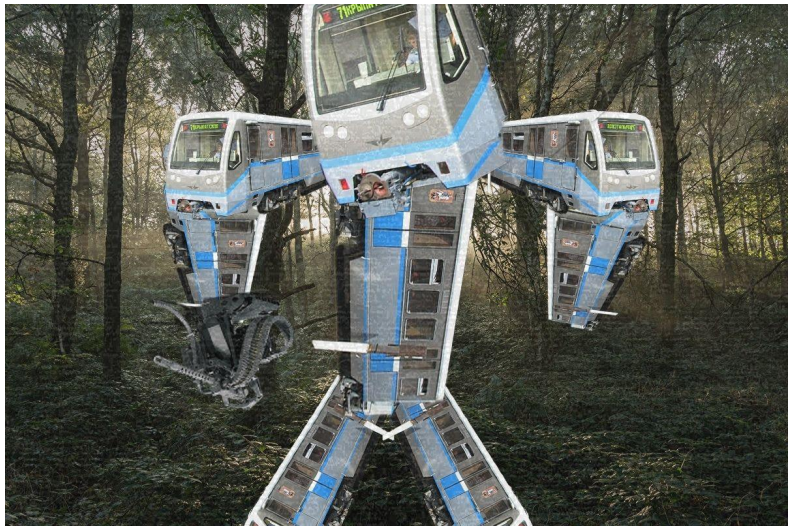


Рис.: ПСП после перемешиваний

Полное решение за $\mathcal{O}(n)$

- При этом каждая из этих скобок должна хотя бы один раз поучаствовать в перемешивании.
- Значит, достаточно найти число скобок «ниже нулевого баланса», их количество и будет ответом.

«Разработка тарифов»



- Идея задачи — Олег Мингалёв, Google
- Идея задачи — Григорий Резников, МГУ
- Разработка задачи — Константин Амеличев, ВШЭ

Постановка задачи

- Дано дерево на n вершинах и m путей
- Требуется задать каждой вершине цвет c_v
- Вдоль пути цвета должны быть расположены монотонно
- Максимальное из чисел должно быть как можно меньше

Решение за $\mathcal{O}(n^n \cdot nm)$ (6 баллов)

- Если раскраска существует, то она существует и с цветами от 1 до n
- Переберем все раскраски и проверим, что числа вдоль путей возрастают

Решение за $\mathcal{O}(2^n \cdot nm)$ (16 баллов)

- Разделим ребра на покрытые и не покрытые путями
- Ребра второго типа разбивают задачу на несколько меньших задач
- Ребра первого типа задают сравнение — либо $c_v < c_u$, либо $c_u < c_v$

Решение за $\mathcal{O}(2^n \cdot nm)$ (16 баллов)

- Переберем все возможные сравнения
- Рассмотрим вершины в порядке топологической сортировки
- Задаем вершине минимальный доступный цвет
- Проверяем, что раскраска корректна

Граф — звезда (+8 баллов)

- Длина каждого пути не больше 3
- Построим граф на путях
- Проведем между пересекающимися путями рёбра вида “эти пути ориентированы одинаково” или “эти пути ориентированы по-разному”.

Граф — звезда (+8 баллов)

- Выбор направления одного пути задает направления во всей компоненте.
- Возможно, ответа вообще нет (есть само-противоречащий цикл)
- Если же ориентировать пути удалось, то в оптимальной раскраске цвета не больше 3.

Пути не пересекаются, $\mathcal{O}(n \log^2 n)$ (+10 баллов)

- Раскраска точно существует
- Сделаем бинарный поиск по максимальному цвету
- Проверим, возможно ли раскрасить дерево в k цветов

Пути не пересекаются, $\mathcal{O}(n \log^2 n)$ (+10 баллов)

- Динамика по поддеревьям.
- Для определённости считаем, что ребро $v \to parent_v$ направлено так, что $c_v < c_{parent_v}$.
- Тогда лучше красить v в минимально возможный цвет, т.к. покраска $c_v = k$ задаёт ограничения на родителя: $c_{parent_v} \geqslant k + 1$.
- Скажем, что $dp[v]$ — минимальный возможный цвет v, согласованный с поддеревом. В силу симметрии, максимально возможный цвет (имеет смысл если $c_v > c_{parent_v}$) равен $k + 1 - dp[v]$

Пути не пересекаются, $\mathcal{O}(n \log^2 n)$ (+10 баллов)

- Для вершины v есть набор ограничений
- Если через v ввысь идёт вертикальный путь из поддерева u, то $dp[v] \geqslant dp[u] + 1$
- Если через v проходит путь из поддерева a в поддерево b, то:
  1. $dp[v] \geqslant dp[a] + 1$
  2. $dp[v] \leqslant k - dp[b]$
- Но мы не знаем, в какую сторону направлен путь, кто же a, а кто же b?

Пути не пересекаются, $\mathcal{O}(n \log^2 n)$ (+10 баллов)

- Попробуем и так, и так
- $dp[v] \in [dp[a] + 1, k - dp[b]]$ или $dp[v] \in [dp[b] + 1, k - dp[a]]$
- Хотим решить систему таких условий. Для каждого пути выпишем два отрезка и сделаем сканлайн.
- Найдем самую левую точку, покрытую отрезками всех цветов.
- Решение за $\mathcal{O}(n \log^2 n)$.

Пути не пересекаются, $\mathcal{O}(n \log n)$ (+16 баллов)

- Заметим, что мы взяли ограничения в два множества — L и R
- $max(L) + 1 \leqslant min(k - R) \iff$
  $max(L) + max(R) \leqslant k - 1$
- $dp[v] = max(L) + 1$
- Надо разложить пары $dp[a_i]$, $dp[b_i]$ по двум множествам
- При этом L уже содержит какие-то ограничения

Пути не пересекаются, $\mathcal{O}(n \log n)$ (+16 баллов)

- Зафиксируем, что будет больше — $max(L)$ или $max(R)$
- Тогда в каждой паре разложение определится однозначно — логично отправить максимум в то множество, где максимум больше.
- Проверим оба варианта на условие $max(L) + max(R) \leqslant k - 1$ и возьмём наилучший.

Полное решение (100 баллов)

- Как в графе-звёздочке, строим новый граф.
- Для каждого ребра становится известен класс эквивалентности.
- Внутри классов эквивалентности мы знаем ориентацию ребер, относительно «основного» ребра внутри класса.

Полное решение (100 баллов)

- Бинарный поиск по ответу, такая же $dp[v]$.
- Рёбра в том же классе эквивалентности задают конкретные L/R ограничения.
- Все остальные классы характеризуются множествами ограничений A, B (и нужно одно отправить в L, другое в R).
- Внутри этих множеств достаточно взять a, b так, что $dp[a] \geqslant dp[a']$, $dp[b] \geqslant dp[b']$.
- Дальше решаем так же, как в прошлый раз. В зависимости от подсчета динамики решили за $\mathcal{O}(n \log^2 n)$ или $\mathcal{O}(n \log n)$

«Двойной палиндром»

Me: I'm scared of palindromes

Therapist: Wow

Me:

- Идея задачи — Дмитрий Ковальков, ВШЭ
- Разработка задачи — Дмитрий Ковальков, ВШЭ

Постановка задачи

- Для заданной строки посчитать количество подстрок, являющихся двойными палиндромами
- Двойной палиндром — конкатенация двух палиндромов одинаковой длины

Решение за $\mathcal{O}(n^3)$ (19 баллов)

- Явно переберем все подстроки
- Проверим что левая и правая половина являются палиндромами за $O(n)$

Решение за $\mathcal{O}(n^2)$ (33 балла)

- Проверять что подстрока является палиндромом можно за $O(1)$ с помощью алгоритма Манакера или hash-функции

Полное решение

- Бывает случай чётной длины палиндрома и нечётной, рассмотрим их отдельно.
- Попробуем понять, когда два выбранных центра палиндромов порождают двойной палнидром.
- *LCenter*, *RCenter* — позиции центров левого и правого палиндромов
- *LRadius*, *RRadius* — радиусы максимальных палиндромов из позиций LCenter и RCenter
- Случаи когда дистанция между центрами нечётна не рассматриваем.

Полное решение

- Необходимые условия:

$$\begin{cases} \frac{RCenter - LCenter}{2} \leqslant LRadius \\ \frac{RCenter - LCenter}{2} \leqslant RRadius \end{cases}$$

- Тогда оба палиндрома достаточно большие, чтобы образовать двойной палиндром.

Полное решение

$$\begin{cases} RCenter \leqslant 2 \cdot LRadius + LCenter & (1) \\ RCenter - 2 \cdot RRadius \leqslant LCenter & (2) \\ LCenter < RCenter & (3) \end{cases}$$

- Перебираем $RCenter$, поддерживаем множество $LCenter$, для которых выполнены условия (1) и (3)
- Запросы на количество чисел в множестве с условием (2) можно делать с помощью Treap
- Итоговая сложность $O(n \log n)$.
- Другое решение: выполнить двумерный запрос на отрезке, можно это делать используя дерево отрезков в offline.

«Реалити-шоу»

- Идея задачи — Романов Владимир, ВШЭ
- Разработка задачи — Погодин Михаил, ВШЭ

Постановка задачи

- Даны n значений $v_i \leqslant m$, n значений s_i и $n + m$ значений c_i
- Требуется выбрать подмножество индексов $1 \leqslant i_1 < i_2 < \ldots < i_k \leqslant n, v_{i_1} \geqslant v_{i_2} \geqslant \ldots v_{i_k}$ с максимальной стоимостью
- При добавлении в множество элемента со значением v в стоимость множества добавляется c_v и если в множестве два элемента со значением v, то они оба удаляются и добавляется $v + 1$ (с увеличением стоимости множества на c_{v+1}).
- Для получения итоговой стоимости нужно вычесть $s_{i_1} + s_{i_2} + \ldots + s_{i_k}$

Решение за $\mathcal{O}(2^n n^2)$ (14 баллов)

- Переберем за $\mathcal{O}(2^n)$ подмножество элементов.
- Далее за $\mathcal{O}(n^2)$ можно просимулировать процесс и найти прибыль.

Решение за $\mathcal{O}(n^2)$ для $m = 1$ (24 балла)

- В данной группе нужно заметить, что всегда лучше брать людей, которые требуют меньшую зарплату.
- Отсортируем работников по возрастанию зарплаты.
- Переберем число людей, которых мы возьмем и честно просимулируем.

Ключевая идея

- Перефразируем происходящее:
- Представим все элементы как степени двойки.
- Будем хранить сумму взятых элементов.
- Мы платим за взятие элемента, прибавляем его к текущему числу и получаем прибыль за каждый перенос.

Решение за $\mathcal{O}(n^2 2^m m)$ (34 баллов)

- Сумма взятых элементов не больше, чем $n2^m$.
- Тогда можно применить метод динамического программирования и решить задачу за $\mathcal{O}((n2^m) \cdot nm)$.

Решение за $\mathcal{O}(n^2 m)$ (75 баллов)

- Рассмотрим сумму и минимальное число, которое мы взяли.
- Заметим, что если брать только элементы с таким же значением, то только $\mathcal{O}(\log(n))$ бит суммы могут изменится как угодно, правда ещё возможно происходит один перенос вне этих битов.
- Снова применим метод динамического программирования и попытаемся учесть предыдущее наблюдение.

Решение за $\mathcal{O}(n^2 m)$ (75 баллов)

- Пусть мы просмотрели префикс массива, тогда:
- $dp[value][carry][mask]$ = max profit, где:
  $value$ — можно брать только элементы $\leqslant$ `value`
  $carry$ — происходит ли перенос
  $mask$ — последние $\log(n)$ бит
- Переходы: взять элемент, пересчитать $dp[value + 1]$ через $dp[value]$.
- Итого мы n раз пересчитаем динамику размером $\mathcal{O}(nm)$

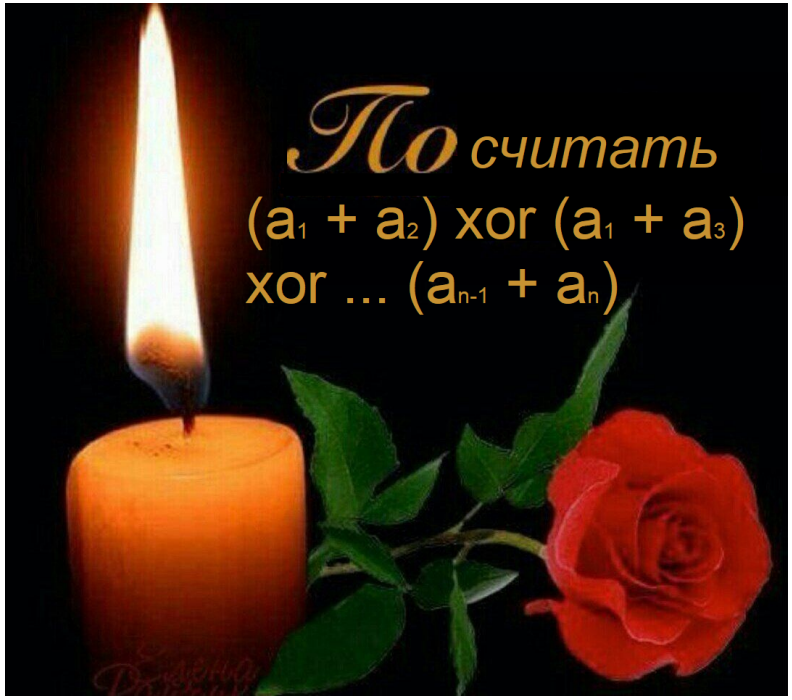
Более простое решение за $\mathcal{O}(n^2m)$ (75 баллов)

- Попробуем набирать элементы в ответ справа налево. Тогда состояние процесса:
- $\langle index, curvalue, cntvalue \rangle$, уже прошли до `index`, текущее рассматриваемое значение `curvalue` и их таких набрано `cntvalue`.
- Тогда перехода всего два: сдвинуть `index` и возможно взять элемент,
- или укрупнить элементы:$(val, cnt) \to (val + 1, cnt/2)$.
- Формально, поддерживаем $dp[curval][cntval]$ и обновляем её новыми элементами.

Полное решение $\mathcal{O}(n(n+m))$

- Осталось заметить, что динамика меньше, чем кажется.
- Если у нас есть элемент со значением x, то при $value = x$ он вкладывается в кратность как 1, при $value = x + 1$ как 0.5, и так далее.
- Сделаем dp неравномерного размера, оценим для каждого `value` максимальное `cntvalue` из соображений выше.
- Сумма таких `max cntvalue` равна $\mathcal{O}(n+m)$, т.к. можно свернуть геометрическую прогрессию.

«Подарок»



- Идея задачи — Миша Пядёркин, Google
- Разработка задачи — Александр Курилкин, ВШЭ

Постановка задачи

- Дан массив чисел, нужно посчитать побитовый xor сумм всех пар чисел в нем

Решение за $\mathcal{O}(n^2)$

- Двойным for явно переберем все пары и посчитаем ответ.

Решение за $\mathcal{O}(n + C^2)$

- Посчитаем количество каждого числа;
- После этого двойным for переберем все возможные пары значений чисел;
- Пусть значения равны i и j;
- Если $i \neq j$, то если $cnt_i \cdot cnt_j \bmod 2 = 1$, то $i + j$ войдет в считаемое выражение нечетное кол-во раз, и его нужно учесть (проксорить ans с $i + j$), иначе нет;
- Если $i = j$, то делаем все то же, только теперь кол-во раз, которое $i + i$ войдет в выражение, считается по формуле $\frac{cnt_i \cdot (cnt_i - 1)}{2}$.

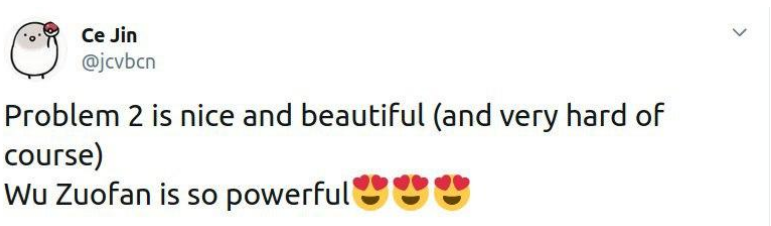
Полное решение

- Будем считать каждый бит в ответе отдельно. Пусть мы хотим посчитать, чему равен k-й (в 0-индексации) бит;
- Заметим, что тогда от чисел нас интересуют только их биты от 0-го до k-го, то есть можно взять все числа по модулю 2^{k+1};
- Теперь сумма двух чисел не может превышать $2^{k+2} - 2$, при этом k-й бит равен 1, если эта сумма попадает в отрезок от $[2^k; 2^{k+1})$ или $[2^{k+1} + 2^k; 2^{k+2} - 2]$.

Полное решение

- Осталось посчитать кол-во пар чисел, дающих такую сумму. Для этого отсортируем все взятые по модулю 2^k числа и пройдемся двумя указателями или сделаем бинпоиски для каждого числа;
- Итоговое время работы: $O(n \log n \log C)$;
- Бонус: можете ли вы убрать $\log n$ из асимптотики?

«Лапша быстрого приготовления»



- Идея задачи — Григорий Резников, МГУ
- Разработка задачи — Егор Чунаев, Яндекс

Постановка задачи

- Дан двудольный граф
- $f(S)$ — сумма чисел в соседях множества вершин левой доли
- Необходимо найти gcd всех чисел $f(S)$

Решение за $\mathcal{O}(2^n \cdot m)$ или $\mathcal{O}(2^n \cdot n)$ (21 балл)

- Переберем все множества вершин левой доли, и посчитаем $f(S)$
- Необходима аккуратная реализация
- Чтобы получить константно более эффективное решение можно сохранить граф в битовых масках, и заранее для каждого подмножества вершин справа предподсчитать сумму по все подмножествам.

Полное решение

- Если у вершины нет соседей в левой доле, ее можно удалить
- Если у вершин одинаковое множество соседей в левой доле, их можно заменить на вершину с суммой
- Ответ — gcd чисел в оставшихся вершинах. В зависимости от эффективности реализации получится 54 или 100 баллов.

Полное решение. Доказательство

- Ответ делится на этот gcd. Поделим все числа в вершинах справа на него, и докажем, что для этой задачи ответ будет 1.
- Выберем произвольное k, и докажем, что есть S, такое что $f(S)$ не делится на k
- Если сумма всех чисел не делится на k, то вся левая доля подходит.

Полное решение. Доказательство

- Выберем правую вершину t которая по весу не делится на k и с самым маленьким количеством соседей.
- Пусть S множество левых вершин не смежных с ней.
- $f(S) = \sum_{u \in \mathtt{all}} c_u - \sum_{\substack{u, \text{ что} \\ N(u) \subset N(t)}} c_u - c_t$
- Первое слагаемое делится на k, иначе уже нашли ответ. Второе делится на k, так как у v минимальное количество соседей (у u получается строго меньше). Последнее слагаемое не делится на k, значит и сумма не делится на k.

«Латинский квадрат»

- Идея задачи — Михаил Тихомиров, МФТИ
- Разработка задачи — Николай Будин, ИТМО

Постановка задачи

- Дана матрица $n \times m$.
- Нужно найти количество подматриц, которые являются латинскими квадратами.
- Латинский квадрат — матрица $k \times k$, в которой ровно k различных элементов и в каждой строке и каждом столбце элементы не повторяются.

Решение за $\mathcal{O}(n^5)$ (9 баллов)

- Переберем подматрицу, являющуюся квадратом.
- Проверим, что подматрица является латинским квадратом за её размер.
- В зависимости от используемых для проверки структур данных, асимптотика может дополнительно умножиться на логарифм.

Решение за $\mathcal{O}(n^4)$ (19 баллов)

Сделаем наблюдение:

- Квадратная подматрица является латинским квадратом, если:
  - В каждой строке и каждом столбце все элементы различны,
  - Количество различных элементов в подматрице равно длине её стороны.

Решение за $\mathcal{O}(n^4)$ (19 баллов)

- Зафиксируем верхний левый угол латинского квадрата.
- Будем перебирать длину стороны квадрата в порядке возрастания.
- При увеличении стороны на 1, добавляем новые элементы и проверяем, что не появилось дубликатов в какой-то строке или столбце. Также поддерживаем число различных в текущем квадрате.
- Для этого для каждой строки, столбца и всего квадрата храним `set` или `hashset`.
- Для фиксированного верхнего левого угла решение работает за $\mathcal{O}(n \cdot m)$

Решение за $\mathcal{O}(n^3)$ (44 балла)

- Научимся за $\mathcal{O}(1)$ проверять что повторений в строках и столбцах нет.
- Для каждой клетки найдём ближайшую снизу и ближайшую справа клетки с таким же значением.
- Сделаем двумерные разреженные таблицы (для каждого k сохраним максимум во всех подквадратах $2^k \cdot 2^k$).

Решение за $\mathcal{O}(n^3)$ (44 балла)

- Осталось проверить, что в квадрате (пусть размера s) ровно s различных чисел.
- Проверку можно сделать с помощью хешей.
- Каждому значению назначим случайное число от 0 до $2^{64} - 1$.
- Хочется проверить правда ли, что суммы (по модулю 2^{64}) всех строк равны и всех столбцов равны.

Решение за $\mathcal{O}(n^3)$ (44 балла)

- Сделаем проще: вычислим сумму в квадрате и сравним с суммой в первой строке квадрата, умноженной на s.
- Если не равны, квадрат точно не латинский.
- Если равны, то будем считать, что латинский.
- Можно показать, что вероятность ошибиться достаточно мала.
- Для вычисления сумм на прямоугольниках, можно предпосчитать суммы в “углах”.

Полное решение за $\mathcal{O}(n^2 \cdot \log^2(n))$

- Пусть есть латинский квадрат $k \times k$.
- Рассмотрим его подквадрат $l \times l$, где $\frac{k}{2} \leqslant l \leqslant k$
- Заметим, что число различных элементов в этом подквадрате равно k, то есть встречаются все значения, которые встречаются в исходном квадрате.
- Следствие: если один латинский квадрат вложен в другой, их размеры отличаются хотя бы в 2 раза.

Полное решение за $\mathcal{O}(n^2 \cdot \log^2(n))$

- Зафиксируем q.
- Для каждого верхнего левого угла есть максимум один латинский квадрат со стороной, лежащей в полуинтервале $[2^q, 2^{q+1})$.
- Чтобы узнать сторону этого квадрата можно узнать количество различных значений внутри квадрата со стороной 2^q.
- После этого можно за $\mathcal{O}(1)$ хешами проверить квадрат с получившейся стороной.

Полное решение за $\mathcal{O}(n^2 \cdot \log^2(n))$

- Осталось для всех возможных позиций квадрата со стороной 2^q вычислить количество различных значений в нём.
- Для этого будем перебирать полосу из 2^q строк сверху вниз.
- Для каждого значения будем поддерживать множество столбцов на которых оно встречается в текущей полосе строк.

Полное решение за $\mathcal{O}(n^2 \cdot \log^2(n))$

- Сдвиг полосы приводит к линейному числу добавлений/удалений элемента из столбцов.
- Будем поддерживать массив a_y — количество различных элементов в квадрате с левой границей y.
- Когда множество столбцов для некоторого значения меняется, на некотором отрезке массива a нужно прибавить или вычесть 1.

- Всего есть $\log(n)$ различных q.
- Для каждого q решаем за $\mathcal{O}(n \cdot m \cdot \log(n))$.

Решение за $\mathcal{O}(n^2 \cdot \log(n))$

- Существует решение за $\mathcal{O}(n^2 \cdot \log(n))$, но эти поля слишком узки, чтобы его вместить.
- К тому же этого не требовалось.