Открытая олимпиада школьников по программированию 2018

Разбор задач

Задача Picasso. «Зебры»

Автор задачи: Максим Ахмедов, МГУ, Яндекс
Разработчик задачи: Егор Чунаев, МГУ

Формальная постановка

- Дана бинарная строка *s*.
- Надо разбить эту строку на несколько подпоследовательностей, которые являются “зебрами”.
- Зебра – строка, которая начинается и заканчивается на 0, и в ней чередуются 0 и 1. Например 0, 010, 01010, и т. д.

Решение на 31 балл [только для фанатов ДП]

- Для каждой из 2^n подпоследовательностей определим является ли они “зеброй”.
- Посчитаем динамику dp[mask] - можно ли разбить подпоследовательность, состоящую из элементов исходной строки, закодированной в mask, на зебры.
- Пересчет можно делать перебором подмаски, соответствующей некоторой зебре из разбиения.
- Асимптотика *$O(4^n + n\,2^n)$ или $O(3^n + n\,2^n)$.*

Решение на 60 или 100 баллов

- Назовем зебра-1 строку, которая начинается на 0, заканчивается на 1 и в ней чередуются 0 и 1, например 01, 0101, и.т. д. Обычную зебру будем звать зебра-0.
- Будем идти по строке *s* слева направо и поддерживать множество зебр-0 и зебр-1.

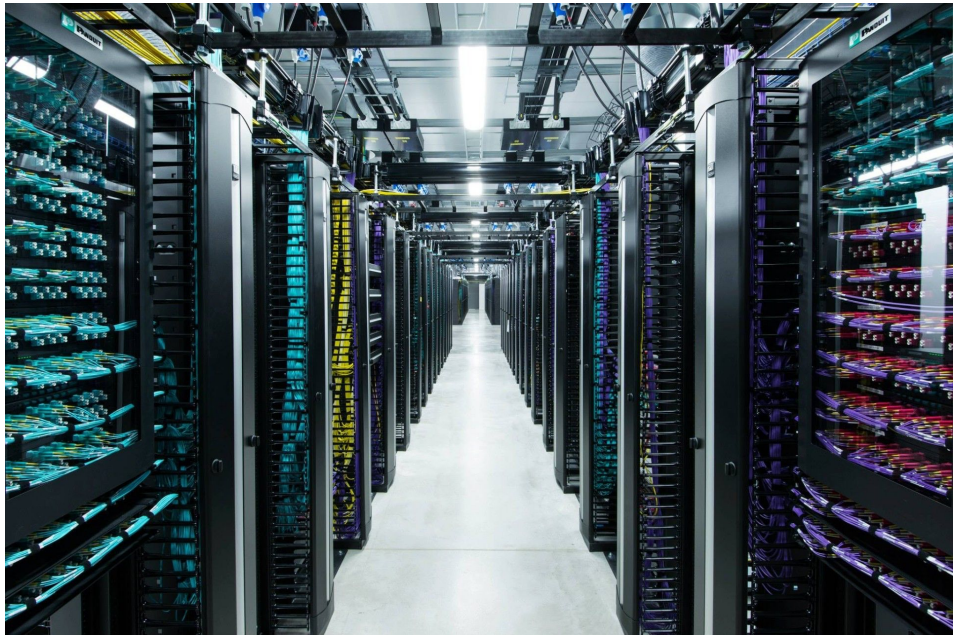
Решение на 60 или 100 баллов

- Если очередной символ 1, то мы обязаны добавить его к зебра-0, а если зебр-0 сейчас нет, то ответа не существует.
- Если очередной символ 0, то мы можем либо начать новую зебру-0, либо добавить его к зебре-1.
- Будем всегда предпочитать приписывать символ 0 к зебре-1, если хотя бы одна зебра-1 есть, и только в противном случае будем заводить новую зебру-0.

Решение на 60 или 100 баллов

- Утверждается, что если хотя бы одно разбиение существует, то мы тоже сможем его построить (то есть, в конце не останется ни одной зебры-1).
- *$O(n^2)$ или $O(n)$* в зависимости от метода поддержания текущих зебр.
- Доказательство корректности оставляется в качестве упраженения (шаг 1: в любой момент времени количество зебр-0 однозначно определено, шаг 2: наш алгоритм поддерживает минимальное возможное количество зебр-1 в любой момент времени).

Задача Kandinsky. «Обновление дата-центров»



Автор задачи: Владислав Макеев, МГУ
Разработчик задачи: Роман Горбунов, МГУ

Формальная постановка

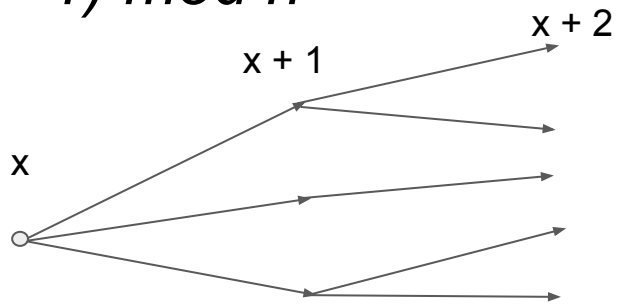
- Дан граф, покрашенный *правильным образом*.
- Правильная раскраска – такая раскраска вершин, при которой никакие две соседние вершины не покрашены в один и тот же цвет.
- Требуется выбрать минимальное непустое подмножество вершин, такое, что если прибавить к их цветам единицу по модулю h, раскраска останется правильной.

Наивное решение на 30 баллов

- Переберём подмножество вершин за 2^n.
- Увеличим цвета в подмножестве на 1, проверим получившуюся раскраску на правильность.
- Выберем подходящую маску минимального размера.
- Сложность решения: $O(2^n \cdot m)$

Ключевая идея

- Заметим, что нас интересуют только рёбра, связывающие *x* и *(x + 1) mod h*
- Если мы увеличим вершину с цветом x, то требуется увеличить и все вершины, связанные с ней, имеющие цвет *(x + 1) mod h*



Решение на 60 баллов

- Выбираем какую-нибудь одну вершину, которая войдёт в итоговое подмножество.
- Увеличиваем все вершины, зависимые от неё, затем, возможно, зависимые от них и так далее. Данную процедуру можно оформить в виде DFS.
- Как только все условия оказались удовлетворены, мы получили подмножество минимального размера, содержащее стартовую вершину.
- Сложность решения: $O(n \cdot (n + m))$

Полное решение на 100 баллов

- Построим ориентированный граф, описанный ранее.
- Добавление вершины влечёт добавление всех вершин, достижимых из неё.
- Построим конденсацию графа. Легко видеть, что нам нужен сток в получившемся ациклическом графе, являющийся КСС минмального размера.
- Сложность решения: $O(n + m)$

Задача Michelangelo. «Отбой»

Автор задачи: Никита Сендерович, МГУ, Яндекс
Разработчик задачи: Дмитрий Саютин, СПб АУ

Постановка задачи

- Один или два преподавателя укладывают детей спать.
- В каждой комнате должны находится *b* школьников, а по факту находятся *a[i]*.
- Первый преподаватель укладывает комнаты слева направо, а второй, если есть, справа налево пока не встретит первого.

Постановка задачи

- Пока преподаватели не в коридоре, школьники могут убегать не далее чем на *d* за ход преподавателей.
- Также школьники могут прятаться от преподавателей под кроватями.
- Минимизировать max(x, y), где x это соответствует количеству нарушений количества, встреченных первым преподом, а y — вторым (либо просто x, если преподаватель один).

Решение для $p = 1$, $n \leqslant 1000$, $b = 1$ (10 баллов)

- Школьников мало (ровно $n \cdot b$).
- Пронумеруем всех школьников в порядке слева направо.
- Для каждой комнаты можно вычислить минимальную правую комнату, из которой до неё можно добежать.
- Сопоставим входу двудольный граф — где вершинам одной доли соответствуют комнаты, а другой — школьники.

Решение для $p = 1$, $n \leqslant 1000$, $b = 1$ (10 баллов)

- Наибольшее возможное количество «хороших» комнат соответствует наибольшему паросочетанию в графе.
- Заметим, что не требуется ничего делать для «лишних» школьников: они просто могут спрятаться в своих исходных комнатах.
- Искать паросочетание можно с помощью алгоритма Куна за $O(VE)$, где $V = n$, $E \leqslant n^2$.
- Сложность решения: $O(n^3)$.

Ключевая идея для одного препода

- Давайте двигаться по комнатам слева направо.
- Если в комнате слишком много школьников, то отправим излишек направо.
- Если в комнате недостаточно школьников, попробуем её заполнить, используя в том числе школьников, которые успевают добежать справа.
- Если всё же оказалось, что нельзя, то выгоним всех в следующую комнату.
- Если следующей комнаты нет, то пусть прячутся.

Где пруфы, Билли?

- Ключевое наблюдение – школьники никогда не меняются местами. Если один школьник был левее другого в начале, то можно сделать так, чтобы в конце он тоже был.

Пруфы?

- Предположим, что существует альтернативный ответ лучше жадно построенного.
- Считаем, что в альтернативном ответе все комнаты тоже состоят либо из *0*, либо из *b* школьников.
- Будем считать что все лишние школьники всегда прячутся в самой последней комнате.

Пруфы

- Пусть в альтернативном ответе насыщенными оказались комнаты *a[1], … , a[k]*, а в «жадном» — *b[1], …, b[l]*, *l* < *k*.
- Пусть *i* это первая позиция, что *a[i] != b[i]*.
- В альтернативном ответе отличающаяся комната может находится только правее *(a[i] > b[i])*.
- Но тогда мы можем просто поправить альтернативный ответ, переместив всех школьников из *a[i],* в комнату *b[i]* (из жадного решения мы знаем, что это возможно).

Решение для $p=1$ (20 баллов)

- Идём по комнатам слева направо
- Если есть нехватка школьников в комнате i, пытаемся её восполнить из комнат $[i + 1, min(n - 1, i + d (i + 1)]$.
- Сложность решения: $O(n^2)$.

Решение для *p=1* (30 баллов)

- Давайте реализуем жадность более эффективно.
- Чтобы узнать есть ли достаточное количество школьников на *[i; min(n - 1, i + d * (i + 1)]* воспользуемся частичными суммами на изначальных ***a[i]***.
- Будем дополнительно поддерживать, сколько школьников *x* пришло слева, а также сколько комнат *k* мы уже насытили.
- Насыщаем текущую комнату, если $s + x - kd \geqslant d$.
- Сложность решения: *O(n).*

Ключевая идея для двух преподавателей

- Как было замечено ранее, в оптимальном ответе школьники не меняются местами.
- Значит есть последний школьник, который будет заниматься насыщением комнат левого препода, а после него школьники будет насыщать только правого препода.

$p \leqslant 2$, $n \leqslant 100$, $b = 1$ (20 баллов)

- Место раздела задаётся одним числом от 0 до nb — количеством школьников, попадающих к первому преподу.
- Переберём количество этих школьников и решим два экземпляра задачи для одного преподавателя.
- Решая задачу для одного преподавателя также, как в первой группе, получаем сложность решения: $O(n^4)$.

Решение для $p = 2$ (55 баллов или 75 баллов)

- Используя при фиксированной границе жадное решение за $O(n^2)$ или за $O(n)$, получаем решение за $O(n^3 b)$ или $O(n^2 b)$, чего достаточно, чтобы пройти 4 или 5 группу соответственно.

Полное решение

- Чтобы получить 100 баллов, нужно искать оптимальную границу быстрее чем за *O(nb).*
- Обозначим за *f(m)* количество ненасыщенных комнат первого препода, если к нему бегут школьники от *0* до *m - 1*, а за *g(m)* количество ненасыщенных комнат второго препода, если к нему бегут школьники от *m + 1* до *nb - 1*.
- Нас интересует величина max(f(m), g(m)).

Полное решение

- Заметим, что функция *f* – неубывающая функция с ростом m, а g – невозрастающая.
- Действительно, чем больше на преподавателя приходится школьников, тем больше простора для насыщения его комнат; лишние школьники всегда могут просто спрятаться.
- Из этого следует что функция y(m) = max(f(m), g(m)) *унимодальна*, то есть до какого-то момента нестрого убывает, а затем нестрого возрастает.

Полное решение

- Тернарный поиск? WA.
- Так как характер монотонности нестрогий, то из $y(m_1) = y(m_2)$ нельзя сделать никакой информации — оптимум может находится где угодно, в том числе левее m_1 или правее m_2.
- Как следствие тернарный поиск не работает.

Бинарный поиск лучше тернарного!

- Найдём бинарным поиском первый момент m_0, при котором $g(m_0) > f(m_0)$ (это возможно, так как условие до некоторого момента не выполнено, а потом выполнено)
- Так как $f(m)$ невозрастает, а $g(m)$ неубывает, то начиная с m_0 максимум достигается на $g(m)$, а до m_0 на $f(m)$.
- Однако $g(m)$ неубывает, а $f(m)$ невозрастает!.
- Таким образом, ответ – $min(g(m_0), f(m_0 - 1))$.
- Сложность полученного решения – $O(n \log nb)$.

Альтернативный подход к решению

- Для $p = 1$ делаем бинарный поиск по ответу.
- Сдвигаем насыщенные комнаты не до упора влево, а до упора вправо.
- Проверку можно написать за линейное время.
- Для p = 2 замечаем, что ненасыщенных комнат у преподов должно быть поровну или почти поровну.
- Опять бинарный поиск по ответу, и опять независимо проверяем для левого и правого преподов.

Задача Munch. «Двоичные карты»

Автор задачи: Максим Ахмедов, МГУ, Яндекс
Разработчик задачи: Дмитрий Горбунов, МГУ, AimTech

Формальная постановка

- Дано n чисел и бесконечное множество карт с достоинствами $+2^k$ или -2^k.
- Нужно выбрать минимальное множество карт так, чтобы любое число могло быть представимо в виде суммы достоинств какого-то подмножества карт.

Решение на 10-20 баллов

- Заметим, что можно взять все положительные и все отрицательные степени двойки, не превосходящие по модулю *c*, это *2 log c* чисел.
- Переберём все наборы из $\leqslant$ *2 log c* степеней двойки.
- Можно пойти чуть дальше и заметить, что достаточно *log c + 1* степеней двойки – например, все положительные, не превосходящие *c,* и первая отрицательная меньше -c.

Наблюдения

- Не имеет смысла брать дважды карту с одним и тем же достоинством.
- Действительно, пусть мы дважды взяли достоинство *x*. Заменим x и x на карты 2x и x, мы сможем набрать все те же суммы.

Наблюдения

- Не имеет смысла брать карты с достоинством, модуль которого больше, чем 4 $\max|a_i|$.
- Действительно, пусть мы взяли б**о**льшую карту 2^k. Тогда, если мы используем эту карту, чтобы набрать некоторое число, мы обязаны также взять карту -2^{k-1} иначе мы не сможем набрать число меньше $2^{k-1} + 1$. Заменим карты 2^k и -2^{k-1} на 2^{k-1}.
- Для сильно отрицательной карты -2^k доказательство такое же.

Наивное решение на 36 баллов

- Переберем множество взятых положительных и отрицательных карт.
- С помощью “задачи о рюкзаке” и ДП мы можем узнать, какие числа можно набрать с помощью этих карт.
- Асимптотика решения:
  $O(2^{2\log c}\ (c + n)\ \log c) = O(c^2(c + n)\ \log c)$, где $c = max\ |a_i|$.
- Данное решение можно ускорить, если “задачу о рюкзаке” решать с помощью ДП с битовым сжатием.

Прокачиваем идею

- На самом деле аналогичным образом не имеет смысла брать карты *x* и *-x* одновременно. Действительно, допустим мы их взяли. Тогда заменим карты *x* и *-x* на *-2x* и x, мы сможем набрать те же самые суммы.

Решение на 51 балл

- Как было показано выше, не имеет смысла брать и 2^k, и -2^k. Поэтому предыдущее решение можно оптимизировать, перебирая только непересекающиеся маски положительных и отрицательных карт.
- Асимптотика решения -- $O(3^{\log_2 c} (c + n) \log c) = O(c^{1.58} (c + n) \log c)$
- Как и предыдущее решение, данное можно оптимизировать битовым сжатием.

Ключевая идея

- Посмотрим на чётность всех чисел a_i.
- Если все числа чётные, то брать карточки 1 и -1 не выгодно, так как мы не сможем ни одной из них воспользоваться (они ломают чётность, которую нечем починить). Поделим все числа на 2 и повторим рассуждение.

Ключевая идея

- Если какое-то число не делится на два, то мы обязаны взять либо 1, либо -1, и эта карта точно войдёт в представление всех нечётных чисел.
- Переберём, какую из карт 1 или -1 мы возьмём, после чего вычтем её из всех нечётных чисел, все числа станут четными, и мы снова можем поделить их на два и перейти к задаче меньшей размерности.

Решение на 71 балл

- Оформим идеи выше в виде рекурсивного перебора.
- Перебор на каждом шаге пробует два варианта, за не более, чем $O(\log c)$ шагов все числа обратятся в ноль.
- На каждом шаге перебора мы модифицируем все числа за $O(n)$.
- Итоговая асимптотика решения – $O(2^{\log c} n) = O(cn)$.

Полное решение

- Заметим, что различных чисел после k-го шага перебора не больше, чем $O(c/2^k)$ (так как все абсолютные величины не превосходят $c/2^k$).
- Будем оставлять только уникальные числа на каждом шаге. Тогда суммарное количество обрабатываемых чисел на всех ветках глубины k нашего перебора составит $O(2^k \cdot c/2^k) = O(c)$.
- Асимптотика решения – $O(c \log c)$.

Задача Aivazovsky. «Чехарда в массиве»

Автор и разработчик задачи: Никита Сендерович, МГУ, Яндекс

Формальная постановка

- В массиве *n* элементов, *i*-й в ячейке *2i - 1*, остальные ячейки пустые
- На каждом шаге последний с конца массива элемент перемещается в ближайшую к нему слева пустую ячейку.
- Необходимо ответить на *q* запросов вида: какой элемент окажется на позиции *x* после окончания прыжков.

Решение на 31 балл

- Наивное моделирование прыжков в массиве.
- Линейным поиском с конца ищем пустую ячейку и переставляем туда последний элемент.
- После моделирования отвечаем на все запросы.
- Итоговая сложность: $O(n^2 + q)$

Решение на 60 баллов

- Линейное моделирование прыжков в массиве.
- Пусть для массива a
  - last – индекс последней заполненной ячейки
  - empty – индекс последней пустой ячейки

```
last = 2 * n - 1; empty = 2 * n - 2
Пока empty >= 2:
   a[empty] := a[last]
   last = last - 1; empty = empty - 2
```

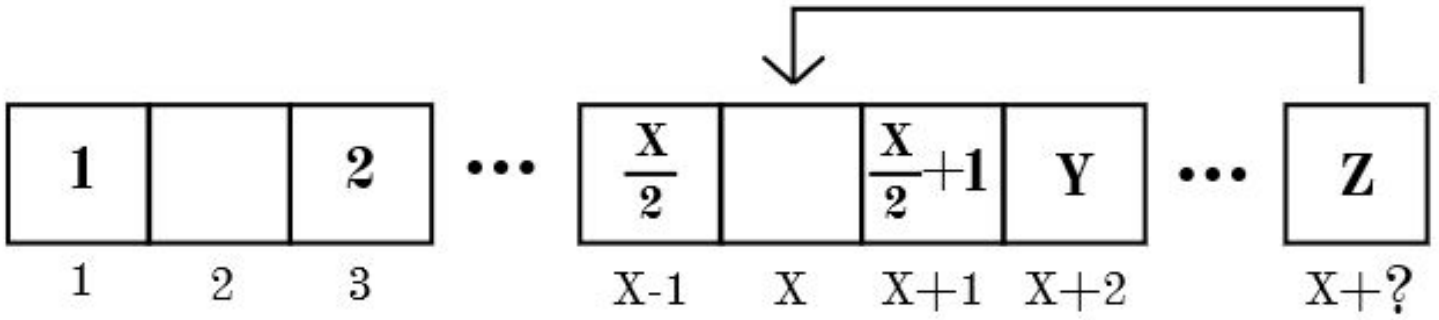
- После моделирования отвечаем на все запросы.
- Итоговая сложность: $O(n + q)$

Решение на 100 баллов

- В нечётной ячейке *p* находится значение *(p + 1) / 2*.
- В каждую чётную ячейку прыгнет ровно *1* элемент.
- Научимся по чётному индексу массива x определять индекс x_{prev}, с которого произойдёт прыжок.
- Тогда можем для любого запроса “раскручивать” цепочку прыжков, пока не окажемся в нечётной ячейке, для которой знаем ответ.

Решение на 100 баллов: анализ прыжка

- В момент прыжка в чётную ячейку x массив устроен следующим образом:



- Слева от ячейки x находится *x / 2* элементов.
- Справа - остальные *(n - x / 2)*, без промежутков.
- Значит, $x_{prev} = x + (n - x / 2)$

Решение на 100 баллов: сложность

- Для фиксированного элемента длина каждого прыжка каждый раз растёт не менее чем в 2 раза.
- Любой элемент сделает не более $\log n$ прыжков.
- Итоговая сложность: $O(q \log n)$

Задача Van Gogh. «Астрономия»

Автор и разработчик задачи: Григорий Резников, МГУ

Формальная постановка

- Дано $2n$ точек
- Найти такую точку, что заданные точки разбиваются на пары, задающие n различных прямых, проходящих через неё.

Решение при $n = 2$

- Переберем две точки, которые лежат на одной прямой
- Оставшиеся две точки также лежат на одной прямой
- Найдем точку пересечения прямых, проверим, что она целая и попадает в bounding box
- Не забыть проверить, что прямые различны
- 10 баллов!

Решение за $O(n^6)$

- Искомая точка является точкой пересечения каких-то двух прямых, каждая из которых проведена через какие-то две точки, есть $O(n^4)$ вариантов. Назовём такие точки кандидатами.
- Проверить кандидата можно за $O(n^2)$: перебираем точку *i* и проверяем за линейное время, что для неё есть пара – точка *j*, лежащая на прямой через i и кандидата.
- Получили решение за $O(n^6)$

Решение за $O(n^4)$ – между 30 и 50 баллами

- Кандидатов на самом деле гораздо меньше – переберём одну прямую, как проходящую через первую точку и какую-то, а вторую – через первую из оставшихся и какую-то – $O(n^2)$ вариантов.
- Решение за $O(n^4)$.

Решение за $O(n^3(\log c + \log n))$

- Научимся проверять кандидата за $O(n \log n)$
- Необходимо быстро для прямой говорить точки на ней
- Канонизируем все прямые, разделив коэффициенты на gcd
- Положим в какую-нибудь структуру, поддерживающую поиск (map, hashtable, отсортированный массив)
- $n^3(\log C + \log n)$ или $n^3 \log C$ в зависимости от структуры

Решение за $O(n^3 \log n)$ и $O(n^3)$

- Не нужно канонизировать прямые
- Для каждой точки можно отсортировать остальные по полярному углу и искать требуемую прямую бинарным поиском
- Можно хранить величины полярных углов в hashtable и получить решение за $O(n^3)$

Решение за $O(n^2 \log n)$ и $O(n^2)$

- Используем идеи из предыдущего решения
- random_shuffle
- break

(доказательство во время тура необязательно для получения полного балла)

Решение за $O(n^2 \log n)$ и $O(n^2)$

- Зафиксируем кандидата
- Будем перебирать точки и проверять относительно него в случайном порядке
- Если в какой-то момент для точки не нашлась пара, то кандидат на ответ не подходит, можно прекратить проверку
- Данное решение работает в среднем за $O(n^2 \log n)$ или $O(n^2)$ в зависимости от реализации

Доказательство

- Так как доказательство довольно длинное, приведем его основные идеи
- Рассмотрим планарный граф, в котором вершины -- это точки пересечения каких-то прямых, проведенных через исходные точки и сами точки
- Пусть некоторая вершина v имеет степень deg(v), тогда в среднем проверка этой точки с отсечением займёт n / (n - deg(v)) шагов
- Необходимо оценить сумму 1/n^2 * sum_q1,q2,p1,p2 n / (n - deg(C(q1,q2,p1,p2)), где p1, p2, q1, q2 -- точки из исходного множества, а C -- точка пересечения прямых, проведённых через них
- Эта сумма равна 1/n^2 sum_c (n * deg(c)^2) / (n - deg(c)) по всем вершинам планарного графа

Доказательство (продолжение)

- Разобьём вершины на два класса: тяжёлые и лёгкие.
- Тогда требуемая сумма разобьётся на два слагаемых, в первом из которых суммирование производится по тяжёлым вершинам, а во втором по лёгким
- Для лёгкой вершины n / (n - deg(v)) < 2, поэтому сумма 1/n^2 * sum_c (n * deg(C)^2) / (n - deg(C)) <= 1/(2n^2) * (sum_c deg(C)^2) <= n^2 / 2
- Для оценки количества тяжёлых вершин необходимо воспользоваться теоремой Семереди-Троттера
- Изи!

Задача Raphael. «ООШП»

Автор задачи: Владислав Макеев, МГУ
Разработчик задачи: Максим Ахмедов, МГУ, Яндекс

Формальная постановка

- Дано корневое дерево, вершинам которого сопоставлены веса
- Нужно выбрать какое-то множество пар вершин дерева, находящихся в общем положении, так, чтобы для любой вершины *v* и для любой вершины *u* в поддереве вершины *v*, существовала хотя бы одна вершина *w*, такая, что пара *(u, w)* выбрана, и что *lca(u, w) = v*.
- Минимизировать суммарный вес выбранных пар.

Решение для $c_i = 1$

- Будем решать независимо по всем вершинам *v*.
- Мы хотим выбрать минимальное количество пар вершин из разных поддеревьев *v*, которые покрывают всех потомков *v*.
- Обозначим размеры поддеревьев *v* за $s_1, s_2, \ldots, s_k$ в порядке убывания.

Решение для $c_i = 1$

- Первый случай – есть **тяжелое** поддерево – поддерево, размер которого больше, чем размеры всех остальных поддеревьев ($s_1 > s_2 + \ldots + s_k$).
- Мы вынуждены воспользоваться минимум s_1 парами по одной на каждую вершину в тяжелом поддереве. Покроем этими парами все элементы остальных поддеревьев минимум по разу каждый.

Решение для $c_i = 1$

- Второй случай – тяжелого поддерева нет ($s_1 \leqslant s_2 + \ldots + s_k$).
- Нам нужно минимум $(s_1 + \ldots + s_k) / 2$ пар.
- Покажем, что ровно таким количеством можно воспользоваться индукцией по суммарному числу вершин.
- В качестве перехода возьмём по вершине из двух наибольших поддеревьев. Легко проверить, что мы снова окажемся во втором случае.

Решение для $c_i = 1$ (25 или 50 баллов)

- Независимо воспользуемся идеями выше в каждой вершине *v*.
- В зависимости от способа определения размеров поддеревьев (обход в глубину на каждую вершину *v* или ДП по поддеревьям) получается либо решение за $O(n^2)$ и 25 баллов, либо $O(n)$ и 50 баллов.

Решение для общего случая

- Слегка модифицируем наши рассуждения для минимизации суммарного веса пар.
- Пусть есть тяжёлое поддерево размера s_1 и все остальные поддеревья суммарного веса s_2 (мысленно схлопнем их в одно).
- Тогда $s_1 > s_2$, все вершины обязаны войти минимум по разу, и мы можем распределить ещё $s_1 - s_2$ "лишних" элементов по второму поддереву.
- Возьмём столько раз минимальный элемент.

Решение для общего случая

- Пусть тяжёлого поддерева нет.
- Если суммарное количество вершин чётное, мы уже показали, что мы просто покрыть каждый элемент единожды, очевидно, это оптимум.
- Если суммарное количество вершин нечётное, мысленно раздвоим вершину минимального веса.
- Заметим, что тяжёлого поддерева ещё нет – если $s_1 \leqslant s_2 + \ldots + s_k$ и суммарное число вершин нечётное, то равенства быть не может.

Решение для общего случая (50 или 100 баллов)

- Дополнительно к размерам поддеревьев надо насчитать минимумы в поддеревьях. Это, опять же, можно делать либо обходами дерева, либо с помощью ДП по поддеревьям.
- Сложность решения – $O(n^2)$ и 50 баллов, либо $O(n)$ и 100 баллов.
- Не забываем про 64-битный тип данных.

Задача Leonardo. «Культурный контакт»

Автор и разработчик задачи: Олег Мингалёв, МАИ

Формальная постановка

- Дана строка *s*, состоящая из строчных латинских символов
- Найдите такое максимальное число *k*, что при разбиении s на *k* подстрок одинаковой длины во всех этих подстроках одинаковые символы встречаются одинаковое число раз

`acab|aabc|baac`

a:2, b:1, c:1

t=3

Решение на 30 (15) баллов

- *n* ≤ *1000 (n ≤ 100)*
- Переберём *w* – длину подстрок, на которые будет разбита *s*
- Посчитаем количество каждого из символов от `a` до `z` среди первых *w* символов строки *s*
- Затем посчитаем количество каждого из символов от `a` до `z` среди следующих *w* символов строки *s*, и так далее
- В зависимости от эффективности реализации такое решение проходит от одной до двух первых групп задачи.

Решение на 51 балл

- $n \leqslant 5 \times 10^5$
- Очевидно, что имеет смысл рассматривать только такие длины подстрок, которые делят длину исходной строки *s*.
- У чисел до 5×10^5 не более 200 делителей.
- Фиксируем *w* – делитель *n*, для каждой подстроки считаем вектор длины $a = 26$ – сколько раз встретился каждый символ
- Сравниваем все вектора
- Получили решение за $O(n \times d(N) \times a)$ – долго
- На самом деле нет!

Решение на 51 балл

- Получили решение за $O(n \times d(n) \times a)$ – долго
- На самом деле нет!
- Посчитаем, сколько раз мы сравнивали вектора на равенство:
  - n/w_1 для делителя w_1
  - n/w_2 для делителя w_2
  - …
- Суммарно сделано $\sigma(n)$ сравнений, где $\sigma(n)$ – сумма делителей числа n
- если $x \leqslant 5 \times 10^5$, то $\sigma(x) \leqslant 2160576 \approx 2 \times 10^6$
- То есть решение работает за $O(n \times d(n) + a \times \sigma(n))$

Решение на 20 баллов

- $n \le 5 \times 10^6$, только символы `a` и `b`.
- Научимся быстро считать количество символов `a` и `b` на подстроках вместо того, чтобы каждый раз пересчитывать их проходясь по строке
- Префиксные суммы: для каждой позиции в строке s посчитаем количество символов `a` не правее этой позиции.
- если $n \le 5 \times 10^6$, то $\sigma(n) \le 22686048 \approx 2 \times 10^8$
- Получили решение за $O(\sigma(n))$

Решение на 71 балл

$$50 + 21 = 71$$

Решение на 100 баллов

- $n \le 5 \times 10^6$
- $a = 26$ символов в алфавите
- Предыдущее решение работает за $O(a \times \sigma(n))$ -- долго
- Нам нужно не считать количества символов, а сравнивать их на равенство
- Хэши!
- Для каждого префикса длины `i` посчитаем хэш следующего вида:

```
hash(i) = количество_символов_a * base^0 +
          количество_символов_b * base^1 +
          количество_символов_c * base^2 +
          ...
```

Решение на 100 баллов

- Для каждого префикса длины `i` посчитаем хэш следующего вида:

```
hash(i) = количество_символов_a * base^0 +
          количество_символов_b * base^1 +
          количество_символов_c * base^2 +
          ...
```

- Тогда хэш подстроки с позиции `i` длины w равен

```
hash(i + w) - hash(i)
```

- Если хэши совпали, то почти наверняка и наборы символов в подстроках совпали
- Хэши для префиксов можно предподсчитать за *O(n)*
- Решение за *O(σ(n))*