	Название	**Автор идеи**	**Подготовка**	**Разбор**
A	Разбиение массива	Евгений Тушканов	Павел Глушень	Павел Глушень
B	Китайские конфеты	Андрей Райский	Андрей Райский	Андрей Райский
C	Удвоение прямоугольников	Андрей Райский	Андрей Райский	Андрей Райский
D	Галактическая Служба Связи	Иван Казменко	Иван Казменко	Иван Казменко
E	Актуальная задача	Олег Евсеев	Александр Логунов	Александр Логунов
F	K-pop	Александр Марков	Александр Марков	Александр Марков
G	Шифрованный калькулятор	Александр Логунов	Александр Логунов	Александр Логунов
H	Captcha	Михаил Иванов	Михаил Иванов	Михаил Иванов
I	Эстафета	Михаил Иванов	Михаил Иванов	Михаил Иванов
J	Покер	Александр Логунов	Дмитрий Саютин	Дмитрий Саютин
K	Фотоувеличение	Артур Рязанов	Артур Рязанов	Артур Рязанов
L	Xor шаров	Вадим Володин	Вадим Володин	Вадим Володин

Разбор задачи «Разбиение массива»

Нужно заметить, что поставленная задача равносильна задаче поиска в массиве *положительного* числа k, встречающегося в этом массиве ровно k^2 раз.

Чтобы найти в массиве такой элемент, нужно, пройдя по массиву, посчитать для каждого положительного элемента количество его вхождений. Хранить информацию можно, например, в словаре: ключом словаря будет элемент, значением — количество вхождений. После этого нужно пройти по словарю, пытаясь найти ключ, которому соответствует значение, равное квадрату ключа.

При использовании неотсортированного словаря (хеш-таблицы) среднее время работы такого решения оценивается как $\mathcal{O}(n)$. Если же использовать отсортированный словарь, время работы будет оцениваться как $\mathcal{O}(n\log(n))$.

Разбор задачи «Китайские конфеты»

Говоря формально, необходимо максимизировать длину отрезка массива, где все элементы различны, возможно, обменяв некоторые два элемента. Пусть отрезок $L..R$ можно превратить в отрезок, где все элементы различны. Тогда возможна одна из двух ситуаций:

- Все элементы встречаются на этом отрезке ровно по одному разу.
- Все элементы кроме y встречаются на этом отрезке ровно по одному разу, y встречается там дважды, и существует такое число x, что оно принадлежит массиву, но не принадлежит отрезку. Тогда можно заменить одно из двух вхождений y на x.

Переформулируем условия в терминах количества различных чисел. В первом случае окажется, что количество различных на отрезке равняется длине отрезка, а во втором — отличается от него на единицу и при этом строго меньше числа уникальных элементов в массиве (иначе не существовало бы x, принадлежащего массиву и не принадлежащего отрезку).

При фиксированной длине отрезка m легко проверить выполнение условий одним проходом по массиву двух указателей — на начало и конец отрезка. Для полного решения достаточно заметить, что функция существования решения монотонна, и добавить сюда бинарный поиск по ответу.

Асимптотика: $\mathcal{O}(n\log n)$.

Разбор задачи «Удвоение прямоугольников»

Будем рассматривать состояния нашей игры как четвёрки $(l, r, step, turn)$ — самую правую клетку прямоугольника Гриши, самую левую клетку прямоугольника Димы, номер хода (для понимания, насколько сильно растянется прямоугольник) и то, чей сейчас ход.

Тогда переходы будут выглядеть одинаково: нужно проверить, можно ли увеличить прямоугольник текущего игрока на 2^{step} так, чтобы не выйти за границы доски, аккуратно пересчитать границы

прямоугольника и перейти к ходу текущего игрока, не забыв увеличить *step* после хода Димы. Стоит отметить, что из каждого состояния игры не больше двух переходов (так как вариантов для увеличения не более двух).

Попробуем смоделировать этот процесс как игру на графе переходов и определить выигрышные и проигрышные состояния. Заметим, что этот граф является деревом, и пути от корня до листьев (то есть терминальных вершин, для которых $l \geqslant r$) не пересекаются.

Какие наблюдения можно сделать?

- Если у состояния все переходы выигрышные, оно проигрышное;
- В любом другом случае состояния объявляется выигрышным.

Обойдём все достижимые состояния и честно посчитаем ответ (он будет храниться в состоянии $(x, y, 0, 0)$. Кажется, что такой подход даст асимптотику порядка $\mathcal{O}(w^2 \log w)$ по числу состояний. Однако это не так: путь из стартового состояния до листа по длине не превышает $2 \cdot \log w$ (так как происходит удвоение прямоугольников), что с учётом удвоения на каждом шаге даёт общую асимптотику порядка $\mathcal{O}(w^2)$. Это, конечно, неоправданно долго при $w = 10^5$.

Применим следующую эвристику: случайно перемешаем переходы во всех вершинах и будем прерывать их перебор при нахождении первого же проигрышного потомка. Ведь как только мы нашли проигрышного потомка, мы сможем гарантировать, что текущее состояние является выигрышным.

Попробуем оценить математическое ожидание количества листьев в поддереве выигрышного и проигрышного состояний на высоте n (предполагаем, что высота корня равна примерно $2 \cdot \log w$, а высота листа равна 0). Назовём эти величины $\mathrm{win}(n)$ и $\mathrm{lose}(n)$. Рассмотрим состояние, которое оказалось выигрышным. Вероятность того, что первый переход будет проигрышным, не менее $\frac{1}{2}$ (так как мы их случайно перемешали, а кроме того, возможен случай, когда оба перехода являются проигрышными). Соответственно, вероятность того, что первый переход будет проигрышным, а второй — выигрышным, не превышает $\frac{1}{2}$. Для простоты отбросим случай, когда оба перехода оказались проигрышными, и остановимся на вероятностях порядка одной второй, то есть сознательно поставим себя в наиболее невыгодную ситуацию.

Систематизируем накопленную информацию:

- $\mathrm{lose}(0) = 1$ (поддерево из одного листа);
- $\mathrm{lose}(n) = 2 \cdot \mathrm{win}(n-1)$ при $n > 1$ (так как придётся совершить два перехода, чтобы гарантировать проигрыш);
- $\mathrm{win}(n) = \frac{1}{2} \cdot \mathrm{lose}(n-1) + \frac{1}{2} \cdot \big(\mathrm{win}(n-1) + \mathrm{lose}(n-1)\big) =$
  $= \mathrm{lose}(n-1) + \frac{1}{2} \cdot \mathrm{win}(n-1) = 2 \cdot \mathrm{win}(n-2) + \frac{1}{2} \cdot \mathrm{win}(n-1)$.

Несложно догадаться, что $\mathrm{win}(n) = 2^{na}$ для некоторого a. Решая вторую рекурренту, получаем $a \approx \frac{1}{0.6}$. То есть $\mathrm{win}(n) = 2^{\frac{5n}{3}}$, а $\mathrm{lose}(n)$ отличается от неё на константный множитель.

Если подставить высоту корня (то есть $n = 2 \cdot \log w$), получится, что $\mathrm{win}(2 \cdot \log w) = w^{1.45}$. Иными словами, от квадрата мы пришли к асимптотике, близкой к линейной.

Асимптотика: $\mathcal{O}(w^{1.45})$.

Разбор задачи «Галактическая Служба Связи»

Можно было бы просто удвоить каждый бит. Это недостаточно эффективно: из 30 битов получится 60, а мы должны уложиться в 50 битов.

Решение 1: точное (требуется 43 бита).

Можно закодировать сообщение строкой Фибоначчи и передать её. Строка Фибоначчи — это строка из нулей и единиц, в которой нет двух единиц подряд. В качестве подзадач используются две стандартные задачи:

1. Дана длина строки Фибоначчи n и её номер k среди всех строк Фибоначчи длины n, найдите саму строку.

2. Дана строка Фибоначчи, найдите её лексикографический номер k среди всех строк Фибоначчи такой же длины.

Рассмотрим сообщение как число в двоичной записи и найдём строку Фибоначчи длины 50 с таким номером.

Кодирование: если видим единицу, на следующей позиции окажется ноль.

Раскодирование: если видим единицу, просто пропускаем следующую цифру.

Из этого решения следует, что обойтись меньшим количеством битов, чем 43, не получится, поскольку количество строк Фибоначчи из 42 битов равно $701\,408\,733 < 2^{30}$.

Решение 2: два случая (требуется 47 битов).

Посмотрим, сколько среди заданных 30 битов единиц. Если 15 или меньше, кодируем так: `0` превращается в `0`, а `1` в `10`. Если 16 или больше, кодируем наоборот: `1` превращается в `0`, а `0` в `10`. В обоих случаях в конце дописываем что угодно: сам код занимает от 30 до 45 битов. Чтобы различить эти два случая, первым битом закодированного сообщения запишем номер случая. И ещё один ноль после него на случай, если этот бит окажется единицей.

Раскодирование: вначале по первым двум битам узнаём, надо ли инвертировать ответ. Далее, пока не наберётся 30 битов, `0` превращается в `0`, а `1` превращается в `1` и пропускает следующую цифру кода. После этого инвертируем ответ, если требуется.

Решение 3: разбиение на кусочки (требуется 50 битов).

Например, разобьём сообщение на 10 кусочков по 3 бита, а пакет на 10 кусочков по 5 битов. Кусочки кодируем по отдельности. Последний бит кода должен быть нулём, чтобы не портить следующий кусочек. Каждый кусочек кодируем и затем раскодируем так:

`000` → `00000`	`00000` → `000`
`001` → `00010`	`0001?` → `001`
`010` → `00100`	`001?0` → `010`
`011` → `01000`	`01?00` → `011`
`100` → `01010`	`01?1?` → `100`
`101` → `10000`	`1?000` → `101`
`110` → `10010`	`1?01?` → `110`
`111` → `10100`	`1?1?0` → `111`

Разбор задачи «Актуальная задача»

Для каждого y будем хранить множество $B(y)$, состоящее из всех первых y битов ip из запросов блокировки длины y.

Тогда на запрос проверки ip можно отвечать так: переберём все y, которыми мог бы быть заблокирован этот адрес. Если хотя бы для одного y первые y битов ip встречаются в $B(y)$, то адрес заблокирован, иначе — нет.

Первые y битов ip можно получить за одну операцию, просто сделав битовый сдвиг на $(32 - y)$ бит вправо.

Множества $B(y)$ можно поддерживать в онлайне с помощью деревьев поиска (подойдут set и декартово дерево) или хеш-таблиц. Получится $\mathcal{O}(32 \cdot n \log n)$ или $\mathcal{O}(32 \cdot n)$, причём с хорошей константой. Это подразумевавшееся решение.

Ещё множество всех заблокированных адресов можно поддерживать в боре, для каждого запроса блокировки добавляя первые y битов и помечая последнюю вершину в боре. Тогда IP-адрес заблокирован, если при пропускании его через бор на пути встречается хотя бы одна помеченная вершина. Это снова $\mathcal{O}(32 \cdot n)$.

Для тех, кто пишет на C++: все числа лучше хранить в типе `unsigned int`, потому что, вообще говоря, операции с переполнением `int`-а ведут к undefined behavior.

Разбор задачи «K-pop»

Формально, нужно найти наименьшее k такое, что в подграфе, построенном из всех рёбер оригинального графа веса не больше k, сумма по всем компонентам связности минимальных c_i в компоненте не превосходит S.

У этой задачи есть несколько верных решений, в числе которых, например, бинарный поиск с проверкой обходом в глубину, но мы рассмотрим более оптимальное решение.

Отсортируем рёбра по возрастанию p_j и будем последовательно их добавлять, попутно пересчитывая сумму минимумов, изначально равную просто сумме всех c_i, пока она будет превосходить S. Для добавления ребра воспользуемся системой непересекающихся множеств, а для того, чтобы быстро пересчитывать сумму, будем поддерживать минимумы в компонентах. Если ребро соединяет вершины одной компоненты, то сумма не изменится, а если оно соединяет вершины разных компонент, то эти компоненты объединятся, в результате чего из двух минимумов в сумме останется только один (то есть в этот момент нужно вычесть из суммы максимум из минимумов в этих компонентах).

Разбор задачи «Шифрованный калькулятор»

Будем восстанавливать число от младших цифр к старшим.

Решение 1: вычитание единицы.

Восстановление каждой новой цифры будет происходить так: вычитаем из текущего числа единицу до тех пор, пока либо $S(x) \neq 0$, либо $S(x+1) \neq S(x)+1$. Пусть произошло одно из этих событий, и перед этим мы успели вычесть единицу d раз.

Утверждается, что неравенство $S(x+1) \neq S(x)+1$ равносильно тому, что последняя цифра числа x равна 9. Действительно, если последняя цифра x меньше 9, то при последнем вычитании не происходило перехода через разряд, и $S(x+1) = S(x)+1$. Если же последняя цифра x равна 9, то это означает, что число $x+1$ заканчивается на несколько нулей, и легко видеть, что $S(x+1) < S(x)$.

Отсюда видно, что $d \leqslant 10$.

Далее разберём случаи: если $S(x+1) \neq S(x)+1$, то последняя цифра изначального x равнялась $(d-1)$. Если же $S(x) = 0$, то $x = 0$, и тогда мы знаем, что изначально $x = d$.

В обоих случаях мы знаем последнюю цифру. Если $S(x) = 0$, то мы знаем и всё число. В противном случае добавим к x единицу обратно и разделим его на 10 — таким образом мы свели задачу к аналогичной задаче без последней цифры.

Этот алгоритм требует не больше, чем $12 \cdot 19$ действий, что меньше 300.

Решение 2: прибавление единицы.

Это решение аналогично предыдущему: будем прибавлять единицу к текущему числу до тех пор, пока не окажется, что $S(x+1) \neq S(x)+1$. Это произойдёт в тот момент, когда последняя цифра стала равна нулю. Очевидно, это вновь требует от одного до десяти действий, и после этого последняя цифра исходного числа также восстанавливается однозначно. Далее можно за одно действие вычесть все прибавленные единицы, вернувшись к исходному числу, и ещё одним действием поделить его нацело на 10. Этот алгоритм также требует не больше, чем $12 \cdot 19$ действий.

Решение 3: деление на 10.

Заметим, что, если число просто поделить нацело на 10, из его суммы цифр будет вычтена та цифра, которая была последней. Повторяя это действие, пока сумма цифр не стала нулём, мы можем последовательно найти все цифры. Максимальное количество действий, которое может потребоваться — это 19.

Разбор задачи «Captcha»

Решение 1. Когда мы будем писать «n m», мы будем иметь в виду, что ответом является число m с n делителями. Рассмотрим случаи; мы рассматриваем конкретный случай, только если он имеет место, а все предыдущие — нет.

1. Есть цифра 1. Тогда ответ «1 1».

2. Есть цифра 2, 3, 5 или 7. Тогда ответ «2 2», «2 3», «2 5» или «2 7» соответственно.

3. Нет цифр 4 и 9. Тогда есть только цифры 0, 6, 8, и из них можно составить только чётное число, которое либо делится на 8, либо делится на какое-то простое, кроме 2, так что в нём как минимум четыре делителя. Ответ «4 6» или «4 8» в зависимости от того, какая из цифр 6, 8 присутствует.

4. Нет цифры 9. Тогда есть только цифры 0, 4, 6, 8, и из них можно составить только чётное число, причём составное, так как двойки нет; кроме того, обязательно присутствует цифра 4, образующая число с тремя делителями. Значит, ответ «3 4». Можно понять, что в этом случае никакой другой ответ невозможен, так как три делителя имеет только квадрат простого числа, а единственным чётным квадратом простого числа является 4.

5. Нет цифр 4 и 8. Тогда есть только цифры 0, 6, 9, поэтому из них может быть составлено только число, кратное 3, причём составное, так как тройки нет; кроме того, обязательно присутствует цифра 9, образующее число с тремя делителями. Значит, ответ «3 9».

6. Есть цифра 8. Тогда ответ «2 89», поскольку 89 простое.

7. Есть цифра 0. Ясно, что есть цифры 4 и 9. Тогда ответ «2 409», поскольку 409 простое.

8. Есть хотя бы две цифры 9. Ясно, что есть цифра 4. Тогда ответ «2 499», поскольку 499 простое.

9. Есть хотя бы две цифры 4. Ясно, что есть цифра 9. Тогда ответ «2 449», поскольку 449 простое.

10. Есть хотя бы две цифры 6. Ясно, что есть цифры 4 и 9. Тогда ответ «2 6469», поскольку 6469 простое.

11. Есть ровно по одной цифре 4, 6, 9. Тогда ответ «3 4», «3 9» или «3 49», поскольку другие чётные числа имеют не менее 4 делителей, а нечётные числа $69 = 3 \cdot 23$, $469 = 7 \cdot 67$, $649 = 11 \cdot 59$ имеют ровно по четыре делителя.

Решение 2. Для достаточно большого N перебрать все числа от 1 до N, для каждого проверить, что его цифры образуют поднабор капчи, если это верно, то посчитать количество делителей и обновить ответ. Как видно из решения выше, при $N \geqslant 6469$ решение гарантированно работает. Программа жюри, написанная в соответствии с этим решением, проходит ограничение по времени при $N = 2 \cdot 10^5$.

Разбор задачи «Эстафета»

В этой задаче рассматриваются функции, действующие на множестве пунктов (каждому пункту сопоставляется его цель). По условию функция должна быть *взаимно однозначной* — каждый элемент ровно один раз бывает значением этой функции; другими словами, для такой функции f любой элемент x равен $f(y)$ для некоторого y, и если $f(a = f(b)$ для некоторых элементов a и b, то $a = b$. Взаимно однозначные функции из множества в это же множество называются *подстановками*. Здесь и далее мы будем рассматривать подстановки на n-элементном множестве $\{1, 2, \ldots, n\}$. Подстановок на таком множестве ровно $n! = 1 \cdot 2 \cdots n$: есть n способов выбрать $f(1)$, потом $n - 1$ способ выбрать $f(2)$ (поскольку повторно взять элемент, равный $f(1)$, запрещено), и так далее, и останется ровно один способ выбрать $f(n)$.

Произведение двух подстановок, действующих на одном и том же множестве — это просто композиция двух соответствующих функций. То есть, если у нас есть подстановка f и подстановка g, то их произведение — функция, обозначаемая $f \circ g$ и действующая так: каждый элемент подставляется в g, а то, что получилось, подставляется в f. На математическом языке это записывается так: $f \circ g(x) = f(g(x))$.

k-я степень подстановки f — это подстановка $f^k = f \circ f \circ \ldots \circ f$, где f встречается k раз после знака равенства. Другими словами, k раз подряд применённая подстановка f. *Тождественная подстановка* — это функция, которая переводит каждый элемент в себя, то есть $f(x) = x$. *Порядок* подстановки f — наименьшее натуральное число k, для которого k-я степень подстановки — это тождественная подстановка. Можно доказать, что множество таких натуральных k, что k-я степень подстановки — это тождественная подстановка, совпадает с множеством всех натуральных чисел, делящихся на порядок подстановки.

У подстановок на конечных множествах всегда существует порядок. Действительно, пусть конечное множество состоит из n элементов, будем рассматривать первую, вторую, $\ldots$, N-ю степень нашей подстановки, где $N > n!$. Поскольку всего существует $n!$ подстановок на n элементах, все

эти N степеней не могут быть различны. Значит, для каких-то k и $\ell > k$ соответствующие степени совпадают; но тогда $(\ell - k)$-я степень нашей подстановки тождественна.

В данной задаче требуется для данных чисел n и m определить, существует ли подстановка n-элементного множества порядка m, и если есть, то вывести её для множества $\{1, 2, \ldots, n\}$. Как же определить порядок подстановки?

Назовём *орбитой* элемента v при данной функции множество элементов, получающихся из v применением какой-либо степени этой подстановки. Например, если мы рассмотрим функцию $f(x) = 2x$, то орбита единицы при функции f — это множество всех степеней двойки. Орбиты есть и у элементов подстановок; при этом, если подстановка имеет порядок (в частности, если она действует на конечном множестве), то любые две орбиты либо совпадают, либо не пересекаются, и всё множество разбивается на эти орбиты. Оказывается, что k-я степень подстановки является тождественной ровно в том случае, если k делится на размеры каждой орбиты в этой подстановке. Таким образом, порядок подстановки равен НОКу (наименьшему общему кратному) размеров её орбит.

Нетрудно понять, что, если нам предоставлено разбиение множества A на несколько частей, то мы сможем подобрать такую подстановку на A, что для неё каждая часть будет орбитой: нужно просто на каждой части $\{a_1, a_2, \ldots, a_t\}$ задать подстановку как $f(a_1) = a_2$, $f(a_2) = a_3$, $\ldots$, $f(a_n) = a_1$.

Значит, если перефразировать всё это математическое условие, то нам требуется проверить, можно ли число n представить в виде суммы нескольких натуральных чисел $t_1, t_2, \ldots, t_n$, НОК которых равен m. Мы будем действовать так: найдём наименьшее $n_{\min}$, представимое в таком виде для данного m, если оно не больше n, то приведём соответствующую подстановку, а потом на элементах $a_{n_{\min}+1}, a_{n_{\min}+2}, \ldots, a_n$ продолжим подстановку как тождественную.

Итак, какова же для данного m наименьшая сумма чисел, НОК которых равен m? (Случай $m = 1$ мы не рассматриваем.) Ясно, что каждое простое входит не более чем в одно из чисел: ведь можно оставить одно число с наибольшим вхождением данного простого p и выкинуть его вхождения из остальных чисел, и НОК всех чисел от этого не изменится. Кроме того, p должно хоть в какое-то из чисел входить в степени, равной степени вхождения в m. Далее, если какое-то из чисел равно 1, его можно убрать без ущерба. Наконец, легко видеть, что $ab \geqslant a + b$, если $a, b \geqslant 2$, поэтому если в одно и то же число входят две степени различных простых, то выгоднее разбить его на отдельные степени. Таким образом, если $m = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_s^{\alpha_s}$, то оптимальным набором чисел является $p_1^{\alpha_1}, p_2^{\alpha_2}, \ldots, p_s^{\alpha_s}$.

Мы приходим к такому решению: разложить число на простые множители, сложить получившиеся степени различных простых, получив $n_{\min} = p_1^{\alpha_1} + p_2^{\alpha_2} + \ldots + p_s^{\alpha_s}$, сравнить с n, и, если n не превосходит $n_{\min}$, то вывести, что ответ существует. Пример таков: выбрать из чисел $1, 2, \ldots, n$ отрезки длиной $p_1^{\alpha_1}, p_2^{\alpha_2}, \ldots, p_s^{\alpha_s}$, на каждом запустить подстановку по циклу, а остальные элементы перевести к себя.

К сожалению, чтобы профакторизовать число m с ограничением 10^{18}, воспользоваться тривиальным алгоритмом с перебором делителей до корня не получится, так как он работает за время $\mathcal{O}(\sqrt{m})$, что не укладывается в ограничение по времени, а более продвинутые алгоритмы разложения на простые множители технически и идейно сложнее. Но они тут и не требуются: если число m имеет простой делитель, превосходящий n, то мы можем быть сразу уверены, что нужной подстановки не существует. Так что достаточно перебрать все делители m от 2 до n, каждый раз максимальное число раз деля m на найденные делители, и, если после всего этого m всё ещё не равно единице, то ответ — не существует. Если же оно оказалось равно единице, то мы нашли факторизацию m и можем её применить, как описано выше.

«Факторизация» m (с остановкой после перешагивания через n) описанным выше способом работает за $\mathcal{O}(n + \log m)$, дальнейшие действия, в том числе построение и вывод ответа — за $\mathcal{O}(n)$, итоговая асимптотика $\mathcal{O}(n + \log m)$.

Разбор задачи «Покер»

Чтобы ответить на какой-то запрос, достаточно хранить множество рангов карт на этом отрезке в какой-то структуре данных.

Давайте предложим такую структуру данных, предназначенную для хранения непересекающихся отрезков. Структура состоит из массива `go`, который для каждой правой границы хранит парную

ей левую, а для каждой левой границы — парную правую, а также дополнительный булевский массив, в котором хранится, покрыта точка отрезком или нет.

Такая структура позволяет производить операцию добавления точки: для этого нужно сначала нужно посмотреть в булевский массив, чтобы проверить, не включена ли точка уже, а затем завести новый тривиальный отрезок (`go[pos] := pos`) и, возможно, склеить его с соседними отрезками слева и справа (посмотрев в `go[pos - 1]` и `go[pos + 1]`).

Например, с помощью такой структуры данных мы уже научились отвечать на запросы за их длину (в предположении того, что структура уже создана). Для этого нужно просто последовательно добавить в структуру все точки на соответствующем отрезке, попутно поддерживая текущую максимальную длину отрезка: поскольку она только возрастает, это можно делать с помощью одной целочисленной переменной.

Дальнейшее развитие решения основано на алгоритме Мо. Несмотря на то, что в такую структуру очень легко добавлять новые точки (за $\mathcal{O}(1)$), удалить произвольную старую точку непросто. С другой стороны, эта структура данных является *откатываемой*: если вести лог всех присваиваний (и предыдущих значений), то достаточно легко откатиться назад до нужной версии.

Воспользуемся этим фактом.

Заведём глобальную структуру данных и будем периодически её откатывать.

Отдельно решим все короткие запросы (длины не более $\sqrt{n}$).

Все остальные запросы сгруппируем по левой границе в $\sqrt{n}$ групп: запрос с началом в l положим в группу с номером $\lfloor l/\sqrt{n} \rfloor$. Внутри каждой группы отсортируем все запросы по правой границе.

Решим для каждой группы запросов задачу независимо. Так как длина запросов хотя бы $\sqrt{n}$, то каждый запрос пересекает границу со следующим блоком массива.

Чтобы решить задачу для одной группы, будем обрабатывать запросы в порядке увеличения правой границы.

Сначала для запроса добавим весь ещё не добавленный кусок от пересечения со следующим блоком до конца правой границы. Так как правая граница монотонна, то суммарно таких добавлений внутри группы не больше n.

Затем запомним текущее состояние, и добавим все элементы от левой границы до пересечения. Получим ответ и сохраним его. Затем откатим все добавленные *левые* элементы и перейдём к следующему запросу.

Так как мы сгруппировали все запросы по левой группе, то на каждый запрос растить левую границу не придётся больше чем $\sqrt{n}$ раз, а откаты занимают не больше времени, чем исходные операции.

Итоговая асимптотика: $\mathcal{O}(n\sqrt{n})$, в предположении, что $n \approx q$.

Разбор задачи «Фотоувеличение»

Переформулируем задачу формально. Это классическая задача выполнимости (SAT): на фотографиях люди в чёрных костюмах — переменные, люди в красных костюмах — их отрицания, а сами фотографии — это клозы (дизъюнкции литералов, то есть некоторых переменных и отрицаний некоторых других переменных), которые должны быть выполнены. Нам нужно найти выполняющий набор: истинные переменные будут соответствовать людям, которые на вечеринку наденут чёрные костюмы, а ложные — людям, которые наденут красные. От общей задачи нашу отличает одно дополнительное условие: количество клозов $m < 2^k$, где k — минимальное количество переменных в клозе.

Заметим, что решение существует. Действительно, математическое ожидание количества выполненных клозов при равновероятном выборе любого набора значений не меньше чем $m \cdot \left(1 - \frac{1}{2^k}\right) > m - 1$. Так как эта случайная величина принимает только целые значения, существует набор значений, выполняющий все клозы.

Заметим, что можно удалить из каждого клоза все литералы, кроме каких-нибудь k, и формула всё ещё будет выполнима.

Будем действовать жадно: перебирать переменные в любом порядке и присваивать каждой «более удачное значение». Чтобы выбрать, какое значение более удачное, будем поддерживать *опасность* клозов $w_1, \ldots, w_m$. Изначально, когда ни одно значение не подставлено, все $w_i = 1$. Далее будем поддерживать следующий инвариант: $w_i = 0$, если клоз выполнен, и $w_i = 2^{k-\ell}$, если в клозе осталось ℓ литералов. Будем также поддерживать неравенство $\sum\limits_{i=1}^{m} w_i \leqslant m$ для суммарной опасности в любой момент времени.

Заметим, что если нам удастся так подставлять значения, то итоговый набор будет выполнять все клозы. Действительно, если бы какой-то клоз обнулялся нашей подстановкой, то, поскольку в клозе осталось 0 литералов, его опасность $w_i = 2^k$. Но $w_i \leqslant \sum\limits_{i=1}^{m} w_i \leqslant m < 2^k$. Таким образом, итоговая подстановка выполняет все клозы.

Как выбирать значение очередной переменной x_i? Пусть $P \subseteq \{1, 2, \ldots, m\}$ — множество клозов, содержащих переменную x_i с положительным знаком, а N — с отрицательным. Тогда, если $\sum\limits_{j \in P} w_j \geqslant \sum\limits_{j \in N} w_j$, подставим $x_i = 1$, а иначе $x_i = 0$. Рассмотрим первый случай. Когда мы подставим $x_i = 1$, опасности w_j для $j \in N$ удвоятся, а для $j \in P$ обнулятся. Тогда суммарная опасность не уменьшится и, значит, инвариант сохранится.

Разбор задачи «Xor шаров»

Прибавление или вычитание единицы инвертирует в двоичном представлении числа какой-то суффикс. Нам нужно, имея изначально 0, получить в итоге число $t = x \oplus a_1 \oplus a_2 \oplus \ldots \oplus a_n$. Старший бит t мы обязательно должны изменить, а значит, и суффикс, соответствующий ему, и не изменять суффиксы для более старших битов. Значит, однозначно задаётся множество суффиксов, которые нужно изменить.

Давайте построим двудольный граф. В левой доле будут суффиксы, в правой — элементы массива, причём ребро есть тогда, когда $a_i \oplus (a_i \pm 1)$ равно соответствующему суффиксу. Найдём максимальное паросочетание. Оно и будет ответом.

Можно также заметить, что из каждого элемента массива идёт два ребра: одно инвертирует суффикс длины 1, а другое — суффикс какой-то большей длины. Поэтому паросочетание можно искать жадно: для каждого суффикса длины больше единицы выберем любой элемент массива связанный с ним, а дальше, если нужен суффикс длины 1, он связан с любым оставшимся элементом массива.