Разбор задачи «Отделы»

Решение задачи подразумевала следующую жадную стратегию. Каждый отдел, кроме последнего, сделаем размера два (соответственно, последний отдел будет размера $\sum_{i=1}^{n} a_i$). Затем будем добавлять коэффициенты в порядке их неубывания последовательно в отделы.

Докажем, что эта жадность работает.

Заметим, что если у нас есть два отдела размерами k_1 и k_2 и при этом $k_1 < k_2$, то нам выгодно будет расположить работников так, чтобы все коэффициенты во втором отделе были больше или равны всех коэффициентов в первом отделе. В самом деле, если бы это было не так, то нашлись работник из первого отдела с коэффициентом x и работник из второго отдела с коэффициентом y такие, что $x < y$. В общую сумму эти работники дают сумму $\frac{x}{k_1} + \frac{y}{k_2}$, однако работников можно поменять местами и тогда они будут давать сумму $\frac{y}{k_1} + \frac{x}{k_2}$. Легко проверить, что вторая сумма окажется больше первой.

Таким образом, мы можем посортировать отделы по количеству работников в них и факт про оптимальный ответ будет заключаться в том, что если два отдела имеют размеры k_1 и k_2 и $k_1 < k_2$, то минимум в первом отделе будет меньше или равен максимуму из второго.

Теперь докажем, что если у нас есть два отдела размерами k_1 и k_2 и $k_1 < k_2$, то мы можем убрать минимум из первого отдела и положить его во второй, и это не ухудшит ответ. За x обозначим тот самый минимум, который мы хотим перенести. Остальные коэффициенты первого отдела обозначим как $a_1, a_2, \ldots, a_{k_1-1}$, а коэффициенты второго как $b_1, b_2, \ldots, b_{k_2}$. Отметим, что $x \leqslant \min_{i=1}^{k_1-1} a_i$ и $\max_{i=1}^{k_2} b_i \leqslant x$. Имея ввиду этот факт, легко видеть, что среднее значение в обоих отделах могло только увеличится.

Итак, мы доказали, что мы можем брать сотрудника с минимальным коэффициентом из отдела с меньшим количеством работников и переносить его в отдел с большим количеством. Делать это можно до тех пор, пока каждый отдел, за исключением последнего, не станет иметь у себя всего двух работников.

Разбор задачи «Возвращение к мечте»

Заметим, что на вторую и третью подгруппы можно было сдать программу, написанную непосредственно в условии.

Для решения первой подгруппы можно было попробовать предподсчитать значения для всех степеней двойки. Поскольку их немного, все значения можно было заполнить.

Для решения задачи на полный балл необходимо было написать решение, работающее быстрее, чем алгоритм из условия. Для этого предлагается перебрать префикс битовой маски числа $\lfloor\sqrt{n}\rfloor$ (это мотивируется тем, что алгоритм из условия пробегает все значения до этого числа) и проверить для каждого префикса, сколько чисел нам подходит. Здесь стоит отметить, что если у нас в числе есть k нулевых битов и нам нужно найти количество чисел, которые имеют нули в других битах, но могут иметь любое значение в каждом из этих k битов, то это количество есть 2^k.

Кроме того, требуется обработать отдельный случай, когда число n является точным квадратом. Для лучшего понимания прилагается код функции на языке C++, которая возвращает ответ на задачу для заданного аргумента x:

```
#define int long long

int fst(int x) {
    int cnt = 0;
    int _sqrt = sqrt(x);
    for (int i = 31; i >= 0; --i) {
        if ((_sqrt & (1ll << i)) == 0) {
            continue;
        } else {
            // put 0 here
            int c = 0;
```

```
            for (int j = i - 1; j >= 0; --j) {
                if ((x & (1ll << j)) == 0) ++c;
            }
            cnt += 1ll << c;
            // can't put 1 here
            if (x & (1ll << i)) break;
        }
    }
    if ((_sqrt & x) == 0) cnt++;
    cnt--; // zero
    cnt *= 2;
    if (_sqrt * _sqrt == x && (_sqrt & x) == 0) --cnt;

    return cnt;
}
```

Разбор задачи «Симус, Петрозаводск и числа на доске»

Во второй подгруппе можно было перебрать все пары за $\mathcal{O}(n^2)$.

Далее обозначим за C максимальное значение среди всех элементов по всем мультитестам.

Первая подгруппа подразумевала решение за $\mathcal{O}(C \log C)$ следующим образом. Отметим сначала, что Проверку того, есть ли в последовательности число y можно было совершать с помощью массива подсчета за $\mathcal{O}(1)$. Тогда можно было пробежаться по всем значениям x от 1 до C и перебрать все числа (меньшие или равные C), которые делятся на x: x, $2x$, $3x$, ... Это работает за $\frac{C}{1} + \frac{C}{2} + \ldots + \frac{C}{C} = C \cdot (\frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{C}) = C \log C$ (см. гармонический ряд).

В третьей и четвертой подгруппе необходимо было найти все делители каждого числа в последовательности и проверить, есть ли число в последовательности, равное этому делителю. Среди всех пар найти лучшую.

В третьей подгруппе можно было перебрать все делители i-го числа обычным алгоритмом поиска делителей за $\mathcal{O}(\sqrt{(a_i)})$.

В четвертой же требовалось применить следующую оптимизацию. Для начала разложим каждое число на простые с помощью решета Эратосфена, а затем, на основе разложения, рекурсивно найдем все делители числа. Если за $d(x)$ обозначить количество делителей числа x, то такая оптимизация позволит перебрать делители за $\mathcal{O}(d(x))$ и заранее предподсчитанным решетом Эратосфена за $\mathcal{O}(C \log \log C)$ или $\mathcal{O}(C)$. Отметим, что $d(x)$ можно очень грубо оценить как кубический корень из числа x, поэтому такое решение будет асимптотически лучше, чем решение для третьей подгруппы.

Разбор задачи «Две перестановки»

Для решения первой подзадачи можно было перебрать все пары чисел и запустить в каждой перестановке поиск в глубину и проверить, что мы натыкаемся на пару чисел как в первой перестановке, так и во второй.

Для решения второй подзадаче можно было заметить, что нам нет смысла запускать поиск в глубину каждый раз при фиксировании очередной пары чисел. Можно было заранее найти все циклы в первой перестановке и во второй, сохранив при этом все пары чисел, которые встречаются в цикле. Это можно было, например, реализовать следующим образом: завести два красно-черных дерева (`std::set<T>` в C++), храня в первом все возможные пары чисел из первой перестановки, а во втором из второй. Тогда при фиксировании очередной пары чисел нужно было бы проверить, встречается ли это пара в обоих деревьях.

Для решения задачи на полный балл предлагается пронумеровать циклы второй перестановки и для каждого числа запомнить, в каком цикле оно находится. Тогда можно перебрать все циклы первой перестановки и для чисел текущего цикла выписать их номера циклов во второй перестановке. Пусть номер цикла второй перестановки x встречается в текущем цикле первого встречается cnt_x раз. Тогда к ответу следует прибавить $\frac{cnt_x \cdot cnt_{x-1}}{2}$.

Разбор задачи «Новый год, подарки, розы»

В первой подгруппе можно было пройтись по всем отрезкам и честно проверить, является ли он *прекрасным*.

Вторая подгруппа решалась примерно таким же подходом, как и первая, но каждый отрезок не считался заново, а при переходе от отрезка $[l, r]$ к отрезку $[l, r+1]$ сохранять информацию о предыдущем отрезке: сколько каких чисел хранится на этом отрезке.

Решение на полный балл было существенно сложнее. Давайте поймем, какие отрезки вообще будут хорошими. Пусть мы зафиксировали число x, которое встречается в индексах $i_1, i_2, \ldots, i_j$. Тогда точно хорошими отрезками $[l, r]$ будут такие:

- $-1 < l \leqslant i_1;\ i_k \leqslant r < i_{k+1}$;
- $i_1 < l \leqslant i_2;\ i_{k+1} \leqslant r < i_{k+2}$;
- $\ldots$
- $i_{j-k} < l \leqslant i[j-k+1];\ i_k \leqslant r < n$.

Теперь давайте представим отрезок $[l, r]$ как точку на плоскости с координатами (l, r). Построим прямоугольники в соответствии с ограничениями выше. То есть одно неравенство означает один прямоугольник. Например, для ограничения $-1 < l \leqslant i_1;\ i_k \leqslant r < i_{k+1}$ Построим прямоугольник с координатами $((0, i_1), (i_k, i_{k+1} - 1))$.

Утверждается, что точка, которая является покрытой хотя бы одним прямоугольником, соответствует хорошему отрезку. Таким образом, нам нужно посчитать количество точек, покрытых хотя бы одним прямоугольником, или, что эквивалентно, посчитать площадь объединения всех прямоугольников, что является стандартной задачей на дерево отрезков и сканирующую прямую.