Всероссийская олимпиада школьников по информатике 2018–2019

Региональный этап

Разбор задач

Разбор задач подготовил Андрей Станкевич, помощь в подготовке разбора оказали Степан Филиппов, Александра Дроздова, Михаил Путилин.

Тесты и решения подготовили Михаил Анопренко, Николай Будин, Александра Дроздова, Арсений Кириллов, Михаил Путилин, Андрей Станкевич, Дмитрий Филиппов, Степан Филиппов.

Разбор задачи «Два измерения»

Автор задачи: Иван Сафонов

Для удобства оценки времени работы решений обозначим количество возможных моментов, в которые можно проводить измерения, равное $(r - l + 1)$, как n.

Для решения первой подзадачи достаточно перебрать все пары часов, в которые проводятся измерения, первое в час i от l до $r - a$, второе в час j от $i + a$ до r. Для каждой пары проверим, что разность $(j - i)$ делится на a, и в случае положительного результата проверки, увеличим ответ на 1. Время работы этого решения $O(n^2)$.

Избавимся от перебора времени второго измерения. Переберём время первого измерения i. Второе могло проводиться в моменты $(i + a), (i + 2a), \ldots$, которые не превышают r. Количество таких моментов равно $\lfloor (r - i)/a \rfloor$, где $\lfloor x \rfloor$ означает округление числа x вниз. С точки зрения реализации это означает, что $(r - i)$ следует поделить на a нацело. Получим решение за $O(n)$, оно подходит для первых двух подзадач.

```
long long ans = 0;
for (int i = l; i <= r; i++) {
    ans += (r - i) / a;
}
```

Следует обратить внимание на использование 64-битного типа, поскольку суммарное количество пар может быть порядка 10^{18}.

Заметим, что возможных значений i все еще слишком много, чтобы уложиться в ограничение по времени для третьей подзадачи. Ускорим решение. Рассмотрим два значения i_1 и $i_2 = i_1 + a$. Заметим, что множества всех подходящих пар измерений вида (i_1, j) и (i_2, j) различаются только наличием в первом случае пары (i_1, i_2).

Пар вида (i_1, j) подходит $k = \lfloor (r - i)/a \rfloor$, следовательно пар вида (i_2, j) подходит $(k - 1)$. Значит чтобы учесть в ответе все пары измерений с $i = i_1$, $i = i_1 + a$, $i = i_1 + 2a$, и так далее, надо прибавить к ответу $k + (k - 1) + (k - 2) + \ldots = k(k + 1)/2$.

Будем перебирать i от l до $l + a - 1$. Для каждого значения i воспользуемся описанным выше методом, чтобы обработать все значения вида $i + ta$. Получим решение за $O(a)$, которое, правда, все еще недостаточно производительное.

```
long long ans = 0;
for (int i = l; i <= r && i < l + a; i++) {
    long long k = (r - i) / a;
    ans += k * (k + 1) / 2;
}
```

Сделаем теперь финальное наблюдение. Для первого значения i выполнено $k = \lfloor (r - l)/a \rfloor$. Последнее значение равно $k = \lfloor (r - (l + a - 1))/a \rfloor$. Эти значения различаются не более чем на 1. Уменьшение значения на 1 происходит после того, как разность $(r - i)$ становится кратна a. Количество i, для которых используется исходное значение k равно, таким образом, $(r - l + 1) \bmod a$, а остальные используют значение k на единицу меньше. Итого, получаем решение за $O(1)$:

```
long long big = (r - l + 1) % a;
long long small = a - big;
long long k = (r - l + 1) / a;
long long ans = big * k * (k + 1) / 2 + small * k * (k - 1) / 2;
```

Разбор задачи «Полные квадраты»

Автор задачи: Олег Христенко

Как известно из условия задачи, $1+3+\ldots+(2n-1) = n^2$. Пусть квадрат целого числа достигается после прибавления a последовательных нечетных натуральных чисел, тогда $k+a^2 = b^2$. Пусть $a \geqslant 0$ и $b \geqslant 0$. Рассмотрим разность $b^2 - a^2 = (b+a)(b-a)$.

В случае $k = 0$ ответ $b = 0$ подходит, пусть далее $k \neq 0$.

В подзадачах 1–3 выполнено $k > 0$, значит $b > a$. Следовательно $k = (b+a)(b-a) \geqslant b+a \geqslant a$. Таким образом достаточно рассматривать $a \leqslant k$.

В подзадаче 1 будем перебирать значения a по возрастанию, для каждого получившегося числа будем проверять, что оно является полным квадратом, первое найденное число будет ответом. Из приведенного выше рассуждения видно, что если среди первых k чисел полного квадрата не оказалось, то ответа не существует. Время $O(k^2)$ или $O(k^3)$.

В подзадаче 2 необходимо выполнять тот же процесс немного эффективнее. Будем перебирать значения a по возрастанию, но во-первых, не будем каждый раз пересчитывать сумму, а во-вторых будем поддерживать, как в методе двух указателей, максимальный квадрат целого числа, меньший или равный текущей суммы. Заметим, что совпадение этих двух значений означает, что ответ найден. Время работы $O(k)$.

Решения, аналогичные изложенным, работают и для подзадач 4 и 5, однако требуется оценить максимальное возможное значение a для отрицательных k. Поскольку $k < 0$, $k = (b+a)(b-a)$, то $b < a$. Значит $|k| = (b+a)(a-b) \geqslant (b+a) \geqslant a$. Таким образом осмысленно перебирать значения a только до $|k|$.

Отметим в заключение этой части, что для получения баллов за подзадачи 1, 2, 4 и 5 можно и не доказывать оценку на максимальное значение a и просто выполнить, например, 10^6 итераций вычисления суммы.

Перейдем теперь к решению полному решению. Требуется найти минимальное неотрицательное целое b, для которого существует a, такое что $k = (b+a)(b-a)$. Пусть для начала $k > 0$, как в подзадачах 1–3. Рассмотрим все делители k, это можно сделать за $O(\sqrt{k})$.

Пусть u — делитель k, обозначим как $v = k/u$, пусть $u \leqslant v$. Тогда предположим, что $b+a = v$, $b-a = u$. Тогда $b = (u+v)/2$, $a = (v-u)/2$. Чтобы значения a и b были целыми, необходимо, чтобы u и v имели одинаковую четность. Из этого, в частности, следует, что если $k \bmod 4 = 2$, то решения нет, хотя это наблюдение и не требуется для полного решения.

Итак, если целые неотрицательные u и v имеют одинаковую четность, $u \leqslant v$ и $uv = k$, то число $b = (u+v)/2$ — один из полных квадратов, которые встречаются в последовательности. Переберём все такие варианты и выберем минимальный.

Наконец, рассмотрим случай $k < 0$. Пусть снова u и v целые, $uv = k$, $u < v$, u и v имеют одинаковую четность. Заметим, что на этот раз u и v должны иметь разный знак, значит $u < 0$, $v > 0$. Тогда число $b = (u+v)/2$ — один из полных квадратов, встречающихся в последовательности, переберём такие варианты и выберем из них минимальный.

Время работы получившихся решений $O(\sqrt{k})$.

Отметим, что чтобы избежать трудностей с рассмотрением случаев можно проверять обе подходящих комбинации знаков u и v, как, например, в приведенном ниже коде.

```
long long ans;

void relax(long long u, long long v, long long k) {
    if (u * v != k) return;
    a = (v - u) / 2;
    b = (v + u) / 2;
    if (a >= 0 && b >= 0 && b < ans) {
        ans = b;
    }
}

...

for (long long u = 1; u * u <= abs(k); u++) {
    if (k % u == 0) {
        long long v = abs(k) / u;
        if (u % 2 == v % 2) {
            relax(u, v, k);
            relax(u, -v, k);
            relax(-u, v, k);
            relax(-u, -v, k);
            relax(v, u, k);
            relax(v, -u, k);
            relax(-v, u, k);
            relax(-v, -u, k);
        }
    }
}
```

Разбор задачи «Автоматизация склада»

Автор задачи: Александра Дроздова

В подзадаче 1 карты в отсеке для хранения уже лежат в том порядке, в котором роботу необходимо открывать помещения. Каждый раз помещая карту на последнее место, он не испортит порядок остальных карт, таким образом для решения этой подзадачи достаточно n раз вывести число n.

Во подзадаче 2 карта от первого помещения, которое необходимо открыть, лежит последней, поэтому перед тем как её вынуть, необходимо вынуть все остальные карты. Они в свою очередь лежат в обратном порядке, следовательно если мы в процессе перемещения карт заодно развернем их порядок, далее каждый раз очередная карта будет первой в отсеке и лишних действий выполнено не будет. Для того, чтобы выполнить разворот последовательности, извлекая i-ю по счёту карту, робот должен вернуть её на $n - i + 1$-е место. После того, как эта операция будет проделана $n - 1$ раз, карты в отсеке лежат в том порядке, в котором необходимо открывать помещения, и следует действовать как в подзадаче 1.

Для решения оставшихся подзадач необходимо сделать ключевое наблюдение. Заметим, что после того как робот взял из отсека карту, он всегда может положить её обратно в отсек таким образом, чтобы его нужно было вынимать из отсека только чтобы открыть дверь в соответствующее помещение.

Чтобы достичь этого, будем действовать следующим образом. Пусть робот извлек карту от помещения номер i из отсека для их хранения и необходимо вернуть её обратно. Рассмотрим все номера помещений, которые будут открыты до очередного момента, когда необходимо будет открыть помещение с номером i, пусть это помещения $j_1, j_2, \ldots, j_k$. Рассмотрим места, где в отсеке для хранения лежат карты от этих помещений. Поместим нашу карту сразу после последней из них. Несложно доказать по индукции, что при таком алгоритме возвращения карт никакая карта не будет извлечена

из отсека для хранения, кроме как для открывания двери, более одного раза.

Наивная реализация описанного алгоритма работает за $O(n^2)$ и решает подзадачу 3. Для решения подзадач 4, 5 и 6 необходимо выполнять указанные действия эффективнее.

Первым действием построим последовательность, в которой из отсека будут извлекаться карты. Для этого рассмотрим по очереди все помещения, которые надо открыть. Если карта для этого помещения уже извлекалась, то наш алгоритм разместит её таким образом, чтобы надо было извлечь только её. Иначе будем извлекать карты из начала отсека, пока не дойдём до карты от требуемого помещения.

```
vector<int> k;
vector<bool> used(n);
for (auto x : a) {
    if (!used[x]) {
        while (true) {
            int y = b.front();
            bool end = y == x;
            k.push_back(y);
            used[y] = true;
            b.pop();
            if (end) {
                break;
            }
        }
    } else {
        k.push_back(x);
    }
}
```

Мы получили массив $[k_1, k_2, \ldots, k_t]$, в котором хранится информация, в каком порядке карты будут извлекаться из отсека. Теперь рассмотрим очередную карту k_i, которая была извлечена, пусть следующее её вхождение в массив k на позиции j: $k_j = k_i$, $j > i$. Тогда позиция, на которую её следут поместить в точности равна количеству различных значений между позициями i и j, включительно.

Получили множество запросов вида «найти количество различных значений на отрезке» в постановке оффлайн. Это стандартная задача, которую можно решить с использованием корневой декомпозиции, дерева отрезков или декартова дерева, в зависимости от эффективности реализации тесты для подзадачи 6 могут пройти или не пройти по времени.

Наконец, дополнительное ограничение подзадачи 4, что все значения a_i различны позволяет немного упростить реализацию в части построения множества отрезков, на которых необходимо искать количество различных элементов.

Разбор задачи «Машинное обучение»

Автор задачи: Степан Филиппов

Немного переформулируем условие эффективности плана. Будем требовать, чтобы равенство $(a_i \text{ and } a_{i+1}) = a_i$ было верно для любого $1 \leqslant i \leqslant n-1$. Нетрудно видеть, что данное условие равносильно условию, сформулированному в задаче.

Теперь легко понять, какой вид имеют все эффективные планы. Массив, описывающий эффективный план, разбивается на последовательные блоки, состоящие из одинаковых чисел, причем битовые представления элементов последующих блоков являются надмасками битовых представлений элементов предыдущих блоков. Также заметим, что количество блоков не может превышать $\log_2 k$ — двоичного логарифма k, так как битовое представление элемента очередного блока содержит строго больше единиц, чем битовые представления элементов предыдущих блоков.

Этого достаточно, чтобы научиться решать подзадачи 1 и 4 с помощью идеи динамического программирования. Для 1-й подзадачи можно посчитать значения $dp[i][j]$ - количество искомых

массивов длины i, таких что последний элемент равен j. Для 4-й подзадачи придется добавить еще один параметр в динамическое программирование и посчитать значения $dp[i][j][k]$ - количество искомых массивов длины i, таких что последний элемент снова равен j, а k — индекс последнего элемента не равного j. Данной информации достаточно, чтобы при пересчете новых значений через предыдущие гарантировать выполнение дополнительных требований к массиву, а именно требований неравенства некоторых пар элементов.

Для решения остальных подзадач нужно перестать хранить значение последнего элемента в явном виде. Ключевым наблюдением является то, что про последнее число достаточно знать количество единиц в его битовом представлении. Предположим, что мы научились считать значения $dp[i][j]$ - количество массивов длины i, таких что они удовлетворяют всем условиям задачи, а также количество единиц в битовом представлении последнего числа равно j. Важно что мы не уточняем какое именно число стоит на последней позиции, мы считаем, что оно фиксировано, но нам не важна его величина. Тогда, чтобы решить искомую задачу, достаточно перебрать x в диапазоне от 0 до k - последнее число в искомом массиве, и прибавить к ответу значение $dp[n][cnt(x)]$, где функция $cnt(x)$ возвращает количество единиц в битовом представлении x.

Научимся считать значения $dp[i][j]$ из предыдущего абзаца.

```
dp[i][j] = 0;
if (check(1, i)) {
    dp[i][j] += 1;
}
for (int p = 1; l < i; p++) {
    if (check(p + 1, i) {
        for (int h = 0; h < j; h++) {
            dp[i][j] += dp[p][h] * C(j, h);
        }
    }
}
```

Функция $check(l, r)$ проверяет, можно ли сделать все элементы отрезка $[l, r]$ равными, не нарушив требований неравенства некоторых пар элементов. Для этого достаточно проверить, что не существует такого требования i, что $l \leqslant l_i < r_i \leqslant r$.

Изначально проверяется, можно ли сделать все элементы с 1-го по i-й равными. Если да, то прибавим 1 к $dp[i][j]$. Таким образом мы учли массивы, состоящие из одного блока одинаковых элементов. Теперь считаем, что массив состоит из нескольких блоков и переберем p - конец предыдущего блока. С помощью функции $check(p + 1, i)$ проверим, может ли конец предыдущего блока быть в позиции p, и если да, переберем h — количество единиц в битовом представлении числа предыдущего блока и добавим к ответу величину $dp[p][h] \cdot C(j, h)$, где функция $C(n, k)$ возвращает C_n^k — количество способов выбрать k элементов из n. И правда, при фиксированном числе с j единицами в битовом представлении существует ровно C_j^h чисел, которые содержат h единиц в битовом представлении и при этом являются подмаской данного числа.

Мы научились решать задачу при больших k, но все еще делаем это медленно, чтобы решить остальные подзадачи. Наша цель будет избавиться от хранения количества единиц в битовом представлении последнего числа. Чтобы сделать это и при этом иметь возможность решать задачу, добавим в динамическое программирование еще один параметр, который будет отвечать за количество блоков, на которые разбивается искомый массив. Теперь у нас считаются значения $dp[i][j][c]$ - количество массивов длины i, которые удовлетворяют всем требованиям и при этом состоят из c блоков одинаковых элементов и элемент последнего блока имеет j единиц в битовом представлении. Формулы пересчета почти не поменяются по сравнению с предыдущим решением, если раньше мы пересчитывали значение $dp[i][j]$ через $dp[p][h]$, то теперь будем пересчитывать $dp[i][j][c]$ через $dp[p][h][c-1]$.

Давайте теперь просто выкинем параметр j из динамического программирования и перестанем при пересчете $dp[i][c]$ домножать на $C(j, h)$. Код примет такой вид:

```
for (int i = 1; i <= n; i++) {
    if (check(1, i)) {
        dp[i][1] += 1;
    }
    for (int j = 2; j <= B; j++) {
        for (int p = 1; p < i; p++) {
            if (check(p + 1, i)) {
                dp[i][j] += dp[p][j - 1];
            }
        }
    }
}
```

Нетрудно видеть, что получившийся код считает количество разбиений массива на заданное число блоков, так чтобы выполнялись все требования на неравенства заданных пар элементов. Предыдущие варианты решения помогут нам понять, на что нужно домножить получившиеся значения, чтобы решить изначальную задачу. Раньше, когда мы к блоку элементов с h единицами в битовом представлении дописывали блок элементов с j единицами в битовом представлении, то мы домножали количество способов на C_j^h. Значит, если мы теперь зафиксируем количество блоков b и массив $(c_1, c_2, \ldots, c_b)$, означающий количество единиц в битовом представлении элемента соответствующего блока, а также число с c_b единицами в битовом представлении, которое будет последним элементов искомого массива, то к ответу нужно добавить величину $dp[n][b] \cdot C_{c_2}^{c_1} \cdot C_{c_3}^{c_2} \cdot \cdots \cdot C_{c_b}^{c_{b-1}}$.

Посчитаем массив $cnt[j][b]$: сумму $C_{c_2}^{c_1} \cdot C_{c_3}^{c_2} \cdot \cdots \cdot C_{c_b}^{c_{b-1}}$ по всем возможным массивам $(c_1, c_2, \ldots, c_b)$. Такие массивы должны удовлетворять двум условиям: элементы в них должны идти по возрастанию и последний элемент должен быть равен j. Это также можно сделать с помощью динамического программирования. Теперь количество эффективных массивов, состоящих из b блоков, у которых в последнем блоке стоит зафиксированное число с j единицами в битовом представлении равно $dp[n][b] \cdot cnt[j][b]$. Если еще посчитать значения $cntBits[j]$ - количество чисел от 0 до k с j единицами в битовом представлении, то ответ на задачу может быть посчитан следующим лаконичным кодом:

```
int ans = 0;
for (int j = 0; j <= B; j++) {
    for (int b = 1; b <= min(n, j + 1); b++) {
        ans += dp[n][b] * cnt[j][b] * cntBits[j];
    }
}
```

Для того, чтобы получить полное решение, достаточно научиться быстро считать массив $cntBits$, быстро вычислять функцию $check(l, r)$ и избавиться от вложенного цикла по p в вычислении $dp[i][j]$, заметив, что нас интересует сумма значений $dp[p][j-1]$ для p, принадлежащих некоторому промежутку.

В итоге можно получить решение, работающее за $O(n \cdot log(k) + \log^2 k)$ времени и за $O(n \cdot \log k)$ памяти, которое можно оптимизировать до $O(n)$ памяти.

Важно заметить, что в блоках кода, приведенных в разборе, опущено взятие по модулю, в реальности нужно при любом вычислении брать результат по модулю и использовать 64-битный тип данных, чтобы избежать переполнения.

Разбор задачи «Неисправный марсоход»

Автор задачи: Андрей Станкевич

Для решения подзадачи 1 можно использовать полный перебор вариантов очередного сигнала. Время работы полного перебора составляет $O(2^{b-a})$ и не превышает 2^{15}.

Для решения подзадачи 2 можно использовать динамическое программирование. Обозначим как $dp[i]$ минимальное число сигналов, необходимое, чтобы мощность батареи марсохода составляла i.

Тогда $dp[i] = \min(dp[i-1], dp[i-2]) + 1$, если i не кратно c и $dp[i] = +\infty$, если i кратно c. Время работы этого решения $O(b-a)$.

Для подзадачи 3, где $c = 2$ оптимальна следующая последовательность: необходимо $(b-a)/2$ раз отправить на марсоход сигнал Y, поскольку b и a оба нечетны, все промежуточные значения будут нечетными.

Перейдем теперь к полному решению.

Рассмотрим два соседних значения, кратных c и числа между ними: $ck, ck+1, ck+2, \ldots, c(k+1)$. Заметим, что, поскольку значение мощности равное ck запрещено, первое из этих значений, которое примет мощность батареи, равно $ck+1$. Будем далее отправлять на марсоход сигналы Y, пока мощность не примет одно из двух значений: $ck+k-2$ или $ck+k-1$. В первом случае далее следует подать сигнал X, чтобы мощность приняла значение $ck+k-1$. Теперь, подав сигнал Y, мы переводим мощность в следующий отрезок чисел между кратными c.

Итого для преодоления такого отрезка требуется $\lfloor (c+1)/2 \rfloor$ сигналов (здесь $\lfloor x \rfloor$ означает x, округленное вниз).

Осталось разобраться с начальным отрезком от a до первого числа, кратного c и с заключительным отрезком от последнего числа кратного c до b. Это можно сделать аналогично. Наконец, следует не забыть о случае, когда между aи b совсем нет чисел, кратных c, в этом случае ответ равен $\lfloor (b-a)/2 \rfloor$.

Разбор задачи «Интервальные тренировки»

Автор задачи: Андрей Станкевич

Для решения подзадачи 1 можно применить рекурсивный перебор вариантов.

Тот же метод решает и подзадачу 2 при использовании аккуратного перебора с отсечениями. Если для n порядка 30 перебор не успевает найти ответ за отведенное ограничение по времени, можно применить метод предподсчёта: заранее на своём компьютере найти ответы для всех тестов с $1 \leqslant k \leqslant n \leqslant 30$ и отправить на проверку программу, которая выводит найденный заранее ответ.

Для решения подзадач 3 и 4 необходимо применить динамическое программирование. Рассмотрим сначала решение за $O(n^3)$, которое решает подзадачу 3.

Обозначим как $up[n][k]$ количество планов тренировок, которые удовлетворяют описанным ограничениям, суммарная нагрузка которых равна n, нагрузка в первый день k, а нагрузка во второй день строго больше нагрузки в первый день. Аналогично введём величину $down[n][k]$, для количества таких планов, где нагрузка во второй день строго меньше нагрузки в первый день.

Заметим, что $up[n][n] = down[n][n] = 1$. Для $k < n$ значения можно вычислить по формулам:

$$up[n][k] = \sum_{i=k+1}^{n-k} down[n-k][i],$$

$$down[n][k] = \sum_{i=1}^{k-1} up[n-k][i].$$

Величина, которую требуется найти по условию задачи, равна $up[n][k] + down[n][k]$, не забудем также отдельный случай $n = k$, когда ответ равен 1.

Чтобы получить решение для подзадачи 4, для которой решение за $O(n^3)$ слишком медленное, можно использовать два подхода.

Подход 1: другая формула пересчёта. Заметим, что формула для $up[n][k]$ отличается от $up[n][k+1]$ одним слагаемым: $down[n-k][k+1]$, поэтому

$$up[n][k] = up[n][k+1] + down[n-k][k+1].$$

Аналогично,

$$down[n][k] = down[n][k-1] + up[n][k-1].$$

Теперь значения вычисляются за $O(1)$ и время подсчёта всех значений есть $O(n^2)$. Следует обратить внимание, что значения $up[n][k]$ следует вычислять по убыванию k, а значения $down[n][k]$ по возрастанию k.

Подход 2: префиксные суммы. Введем дополнительные величины:

$$sumUp[n][k] = \sum_{i=1}^{k} up[n][i],$$

$$sumDown[n][k] = \sum_{i=1}^{k} down[n][i].$$

Тогда

$$up[n][k] = sumDown[n-k][n-k] - sumDown[n-k][k],$$

$$down[n][k] = sumUp[n][k-1].$$

Отметим, что во всех случаях следует аккуратно следить за тем, чтобы индексы значений, к которым происходит обращение в формулах, не выходили за пределы массивов и за порядком вычислений.

Разбор задачи «Экспедиция»

Автор задачи: Елена Андреева

Для решения подзадачи 1 можно перебрать все подмножества кандидатов и для каждого подмножества проверить, удовлетворяет ли оно требованиям к составу экспедиции. Время работы $O(n2^n)$.

В подзадаче 2 в экспедицию можно взять либо всех кандидатов с нечётными номерами, либо всех кандидатов с чётными номерами. Первое выгоднее, в этом случае в экспедицию отправятся $\lfloor (n+1)/2 \rfloor$ кандидатов.

Для решения подзадач 3, 4 и 5 переформулируем задачу в терминах теории графов. Построим ориентированный граф, у которого вершинами будут кандидаты в экспедицию, проведем из вершины i ребро в вершину a_i, если $a_i \neq -1$. Условие, что в экспедицию нельзя одновременно брать кандидатов i и a_i соответствует тому, что множество отправленных в экспедицию кандидатов должно образовывать *независимое множество* в графе.

В подзадаче 3 получается подвешенное дерево, где корень — вершина 1, а ребра ориентированы к корню. Для поиска независимого множества в дереве можно использовать жадный алгоритм или динамическое программирование. Жадный алгоритм: удалим ориентацию ребер и будем действовать следующим образом. Выберем все листья дерева в множество, удалим их и все индцидентные им вершины, повторим. Доказательство: если лист не выбран в независимое множество, если его сосед также не выбран, можно выбрать лист, увеличив размер независимого множества, если его сосед выбран, то выберем лист вместо соседа, размер не изменится.

Рассмотрим теперь решение с использованием динамического программирования на подвешенном дереве: хоть и избыточное для задачи на дереве, оно пригодится нам для решения подзадач 4 и 5. Обозначим как $with[u]$ размер максимального независимого множества в поддереве вершины u, содержащего вершину u, и как $out[u]$ размер максимального независимого множества в поддереве вершины u, не содержащего вершину u. Тогда

$$with[u] = \sum_{v-\text{сын } u} out[v],$$

$$out[u] = \sum_{v-\text{сын } u} \max(with[v], out[v]).$$

Для листа $with[u] = 1$, $out[u] = 0$.

Теперь ответ для дерева равен $\max(with[r], out[r])$ для корня дерева r.

Перейдем теперь к решению задачи для подзадач 4 и 5, где граф не обязан быть деревом. Заметим, что из каждой вершины выходит не более одного ребра. Каждая компонента связности

такого графа представляет собой либо дерево, либо цикл, к каждой вершине которого может быть подвешено дерево. Решим задачу независимо для каждой компоненты связности и сложим ответы для них. Для компонент, которые представляют собой дерево, решение уже описано выше.

Рассмотрим теперь компоненту, которая содержит цикл. Удалим ребра цикла и решим задачу для каждого из получившихся деревьев. Осталось решить следующую задачу на цикле: в каждой вершине есть два числа: $with[u]$ и $out[u]$ требуется для каждой вершины цикла одно из этих чисел, причём для двух соседних вершин нельзя одновременно выбрать $with$. Для решения этой задачи можно использовать динамическое программирование, аналогичное решению задачи для дерева: разрежем одно из ребер цикла, переберем, какое из двух значений взято для первой вершины, посчитаем значения вдоль получившегося пути, и найдем ответ.

В зависимости от реализации всех описанных методов, за $O(n^2)$ или $O(n)$, решение получает баллы за подзадачи 4 и 5.

Разбор задачи «Разбиение на пары»

Автор задачи: Иван Сафонов

Рассмотрим искомое разбиение на пары. Для каждого j есть $n/2$ индексов i, таких что $a_{i,j} \leqslant b_j$ и $n/2$ индексов i, таких что $a_{i,j} \geqslant b_j$.

Таким образом, если решение существует, то в качестве b_j можно выбрать $\mathrm{median}(a_{1,j}, \ldots, a_{n,j})$, где median — медиана набора чисел, в наборе чётное количество чисел, поэтому медиана — это среднее арифметическое $n/2$-й и $n/2+1$-й порядковых статистик значений соответствующего параметра, хотя можно выбрать и просто $n/2$-ю порядковую статистику.

Для решения подзадачи 1 можно перебрать все разбиения на пары и для каждого проверить корректность.

В подзадаче 2 параметр всего один, поэтому артефакты можно разбить на пары жадно: первый и последний, второй и предпоследний, и так далее. Решение всегда существует.

В подзадаче 3 все значения каждого параметра различны. Для каждого артефакта построим битовый вектор длины k, j-я координата равна 0 или 1, в зависимости от того, больше или меньше значение j-го параметра, чем соответствующее медианное значение. Заметим, что артефакты надо разбить на пары, которым соответствуют битовые вектора, являющиеся отрицанием друг друга. Это можно попытаться сделать жадным алгоритмом, если решение не найдено, то его не существует.

Трудность решения оставшихся подзадач заключается в том, что все параметры, кроме первого, могут принимать равные значения. Каждый артефакт, значение параметра которого равно соответствующей медиане, может быть как меньшим, так и большим по этому параметру в паре.

В подзадаче 4 параметров два. Разделим артефакты по первому параметру на два множества, условно назовём их красным и синим: первые $n/2$ по первой координате и вторые $n/2$ по первой координате. В каждую пару должен войти один красный и один синий артефакт. Сопоставим каждому артефакту значение 0, 1 или -1, в зависимости от того, равно, больше или меньше значение его второго параметра медианному. Получаем 6 множеств: красные R_0, R_1, R_{-1} и, соответственно, синие B_0, B_1, B_{-1}. Артефакты из R_1 должны войти в пары с B_{-1}, из R_{-1} — в пары с B_1. Артефакты из R_0 могут войти в пары с любыми синими, а из B_0 с любыми красными. Их можно распределить жадно.

Альтернативное решение: отсортируем артефакты по второму параметру (при равенстве произвольным образом), и первые $n/2$ назовём малыми, а последние $n/2$ большими. Тогда красные большие входят в пары с синими малыми, а красные малые с синими большими. Легко показать, что их количество соответственно совпадает.

В подзадачах 5 и 6 параметров больше и жадный алгоритм перестаёт работать. Рассмотрим решение на основе алгоритмов поиска паросочетаний в двудольных графах и потоков.

Решение подзадачи 5. Для каждой пары артефактов проверим, можно ли создать из них пару.Проведём ребро между в случае, если можно. Как и прежде разделим артефакты на красные и синие, в зависимости от первого параметра. Заметим, что ребра соединяют артефакты разного цвета, следовательно граф двудольный. Найдём в получившемся графе полное паросочетание алгоритмом Куна. В графе $O(n^2)$ рёбер, поэтому решение работает за $O(n^3)$.

Решение подзадачи 6. Сопоставим каждому артефакту строку длины k из 0, 1 и -1: j-е число равно -1, если $a_{i,j} < b_j$, 1, если $a_{i,j} > b_j$ и 0, если $a_{i,j} = b_j$. Заметим, что первое число в строке не может быть равно 0. Таким образом, точки разбиваются на $2 \cdot 3^{k-1}$ групп. Заведём вершину для каждой группы. Вершины, которые соответствуют точкам, у которых $a_{i,1} < b_1$ — это первая доля, остальные — вторая. Также заведём исток и сток. В вершины первой доли проведём рёбра из истока с весом, равным числу точек в соответствующей группе. Из вершин второй доли в сток — аналогично. Между вершинами из разных долей проведём рёбра веса $+\infty$, если соответствующие им точки можно объединить в пары. Найдём в таком графе максимальный поток. Чтобы восстановить ответ, нужно найти декомпозицию этого потока.