Задача A. Билли Бонс и монеты

Идея: Артём Абатуров
Разработка: Артём Абатуров

Наша задача найти минимальное число y, такое что оно строго больше, чем x, а также выполняется следующее условие: любая цифра из этого числа должна отличаться от цифры, которая находится в том же разряде числа x. Чтобы это условие выполнялось, у числа y точно должна отличаться первая цифра. Так как мы хотим минимизировать число y, давайте просто возьмем первую цифру числа x и увеличим её на 1. Это и будет первая цифра числа y. В случае, если первая цифра числа x равна 9, число y должно начинаться с 10.

Теперь, вне зависимости от остальных цифр y это число будет точно больше числа x. Поэтому давайте просто поставим 0 в тех разрядах, в которых у числа x ненулевые цифры, и 1 во всех остальных разрядах. Нетрудно понять, что сконструированное нами число будет минимально возможным.

Задача B. Шприцы

Идея: Антон Лахтин
Разработка: Антон Лахтин, Артём Степанов

Очевидно, что если длина хотя бы 1 из сторон делится на k, то прямоугольник можно разбить на прямоугольники $1 \times k$. Нетрудно доказать, что если прямоугольник $a \times b$ можно разбить на прямоугольники $1 \times k$, то хотя бы 1 из сторон делится на k. Для этого рассмотрим вертикальную раскраску доски в k цветов и будем следить за количеством свободных клеток каждого цвета по модулю k. Несложно заметить, что среди этих количеств всегда найдутся 2 разных значения, а значит все нули получить нельзя. Более формальное доказательство этого факта оставим в качестве упражнения читателю.

То есть нам нужно посчитать количество пар чисел $(a,\ b)$, таких что $1 \leqslant a, b \leqslant n$ и хотя бы одно из чисел a, b делится на k. Пусть A – множество пар, в которых a делится на k, B – множество пар, в которых b делится на k. Тогда ответ это $|A| + |B| - |A \cap B|$. $|A| = |B| = [\frac{n}{k}] \cdot n$. $|A \cap B| = [\frac{n}{k}]^2$. Отсюда получаем ответ: $2[\frac{n}{k}] \cdot n - [\frac{n}{k}]^2$.

Задача C. Битый ром

Идея: Артём Кутузов
Разработка: Дмитрий Зайцев

Задача сводится к тому, что мы для каждого числа k от 1 до $2^n - 1$ выполняем алгоритм:

```
a[k] += 1;
while (f(k) != k) {
    k = f(k);
    a[k] += 1;
}
```

Где $f(k)$ — количество битов в числе k, а a — массив целых чисел с индексами от 1 до $2^n - 1$.

Заметим, что для индексов $> n$ в массиве a будут лежать только единицы.

Тогда нам осталось посчитать ответ для индексов $\leqslant n$. Сделаем это таким образом: в порядке от n до 1 выполним наш алгоритм, только вместо единицы будем прибавлять $C_k^n + 1 - cntBad$, где $cntBad$ — количество чисел с номерами $\leqslant n$, в которых k единичных битов, и C_k^n — сочетания из n по k. Число C_k^n равно количеству всех n битных (учитываем ведущие нули) чисел, содержащих k единичных битов.

Так как $n \leqslant 60$, то C_k^n можно посчитать с помощью динамики: $C_k^n = C_k^{n-1} + C_{k-1}^{n-1}$, $C_n^n = 1$, $C_0^n = 1$.

Задача D. Хорошее настроение

Идея: Артём Абатуров
Разработка: Виктор Кривощеков

Найдем все вхождения минимального числа. Тогда изначально левый конец подотрезка будет равен первому вхождению минимума, а правый конец — последнему. Отсортируем массив чисел по модулю числа. Когда будем идти по массиву, будет возникать два случая:

1. $a_i \geqslant 0$. Тогда, если это число входит в текущий подотрезок изменения, то оно стало отрицательным и будет новым минимумом массива.
2. $a_i < 0$. Если текущее число попало в подотрезок, то значит, оно стало положительным, и нет смысла менять текущий подотрезок. Иначе, раз мы идем в порядке возрастания модулей, то чтобы сделать ответ больше, нужно изменить знак этого числа. Будем расширять подотрезок по одному элементу. Если встречаем положительное число, смена знака которого делает ответ меньше, то значит, неоптимально было расширять отрезок, и ответ это a_i.

Если после обхода массива не вышли за границы массива, значит, что получилось все числа сделать положительными, и ответ будет минимум из модулей всего массива.

Задача E. Портальные сокровища

Идея: Антон Лахтин
Разработка: Артем Степанов

В данной задаче надо было понять, в каком случае мы можем вернуться обратно в вершину. Действительно, в одной компоненте рёберной двусвязности мы можем пройтись по каждой вершине и вернуться в неё. А также давайте поймём, что если мы зашли в компоненту, то мы можем её обойти и вернуться назад.

На самом деле, в листья нашего графа мы не сможем зайти, так как потом из них не сможем выйти. Это значит, что такие листовые вершины мы можем выкинуть из графа. Тогда отсюда складывается решение: выкидываем из графа листья, пока можем. Для этого можно было поддерживать степени вершин и добавлять их по мере необходимости. Тогда утверждается, что оставшийся граф и будет ответом на задачу. Так как у нас не осталось листовых вершин. Это значит, что в дереве рёберной двусвязности, листья нашего графа это компоненты размера хотя бы 2. А через них мы сможем пройтись по всем вершинам.

Итоговая сложность $O(n + m)$.

Задача F. Дробитель

Идея: Артём Кутузов
Разработка: Виктор Кривощеков

Пусть после какого-то количества действий получили дроби $\frac{a_1}{b_1}, \frac{a_2}{b_2}, \ldots, \frac{a_k}{b_k}$. Тогда, если есть $a_i > b_i$, то выгодно перевернуть дробь. От уменьшения слагаемого итоговая сумма уменьшится. Теперь поймем, что итоговый ответ состоит всего из одной монеты, так как

$\frac{a_1}{b_1} + \frac{a_2}{b_2} = \frac{a_1 b_2 + a_2 b_1}{b_1 b_2} > \frac{a_1 + a_2}{b_1 + b_2}$. Из-за того, что верно неравенство
$(a_1 b_2 + a_2 b_1)(b_1 + b_2) = a_1 b_1 b_2 + a_2 b_1^2 + a_1 b_2^2 + a_2 b_1 b_2 > a_1 b_1 b_2 + a_2 b_1 b_2 = (a_1 + a_2) b_1 b_2$.

Поэтому итоговый алгоритм следующий. Сделать во всех дробях числители меньше или равными, чем в знаменателях, и сложить все дроби.

Задача G. Любимое число Флинта

Идея: Артём Степанов
Разработка: Антон Лахтин

По сути задача такова. Есть n чисел. 2 из них можно сложить. Какой наибольший НОД можно таким образом получить?

Рассмотрим пару чисел, которая дает оптимальный ответ, пусть это числа a и b. Заметим, что среди первых 3 не нулевых чисел хотя бы одно не участвовало в этой паре (если 3 не нулевых чисел нет, то ответ – сумма всех чисел). Тогда ответ – это делитель этого числа. Обозначим ответ за d

Пусть d делится на простое число p и степень вхождения p в d равна k. Тогда все числа, кроме возможно a и b, делятся на p^k и $(a + b)$ делится на p^k. При этом для p^{k+1} это уже не верно. А так как p^k – делитель ответа, то оно также является делителем хотя бы одного из первых 3 чисел. То есть достаточно рассмотреть все простые числа p, являющиеся делителем хотя бы одного из первых 3 чисел.

То есть алгоритм такой:

Берем 3 не нулевых числа, раскладываем их на простые множители.

Затем рассматриваем все найденные простые числа. Пусть простое число p имеет степень вхождения не больше k в рассматриваемые 3 числа. Тогда перебираем все степени от 0 до k и находим наибольшую – $m \leqslant k$, для которой все числа, кроме возможно двух, делятся на p^m.

1) Если все числа делятся на p^m, то умножаем общий ответ на p^m, так как эта степень простого подходит для любой пары чисел.

2) Если ровно 2 числа (a_p и b_p) не делятся на p^m, а их сумма делится на p^m, то ответ для найденной пары умножим на p^m. (Параллельно для каждой возможной пары чисел храним ответ для нее, например используя unordered_map)

В конце берем наибольший из ответов для найденных пар и умножаем его на общий ответ.

Посчитаем вычислительную сложность этого алгоритма. Разложение на множители работает за $O(\sqrt{A})$, где A – наибольшее из чисел. Суммарная степень вхождения простых $O(log(A))$. Проверка конкретной степени простого работает за $O(n)$. Получаем алгоритм, который работает за $O(\sqrt{A} + nlog(A))$

Задача H. Путешествие к кладу

Идея: Артём Степанов, Артём Абатуров
Разработка: Артём Степанов

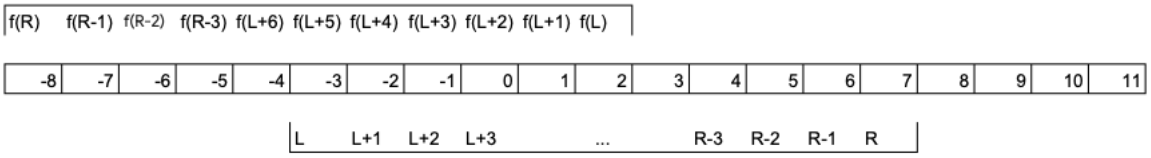
Ключевая идея заключается в том, чтобы аккуратно предпосчитать ответ для каждого x и далее уже выводить посчитанные значения.

Давайте заведем функцию $f(x) = -(x+1)$. Тогда давайте поймем, что если мы получили, что ответ для $x \to a$, то ответ для $f(x) \to f(a)$, обратите внимание, что $f(f(x)) = x$. Это действительно так, так как по факту все наши действия с числами симметричны относительно -0.5. Тогда мы можем посчитать ответ только для неотрицательных x и мы будем знать ответ для всех чисел.

Чтобы предпосчитать ответ, давайте заведем 2 отрезка $[L; R]$ и $[L_{cur}, R_{cur}]$. Они будут означать, что

числа из отрезка $[L; R]$ переходят в отрезок $[L_{cur}; R_{cur}]$. Изначально $L_{cur} = L = 0, R_{cur} = R = X_{max}$. Тогда после очередного a_i отрезок $[L_{cur}; R_{cur}] \to [L_{cur} \pm a_i; R_{cur} \pm a_i]$. Будем поддерживать наши отрезки так, чтобы наш отрезок $[L_{cur}; R_{cur}]$ всегда был по одну сторону от 0.

Теперь надо понять, что делать, если наш отрезок стал по разные стороны от 0, то есть $L_{cur} < 0$ и $R_{cur} \geqslant 0$. Тогда давайте поймем, что вместе с нашим отрезком мы знаем и ответ для отрезка $[f(R), f(L)]$ (в силу симметрии). Давайте считать, что большая часть отрезка выходит за ноль направо (т.е. $|L_{cur}| \leqslant R_{cur}|$), случай $|R_{cur}| < |L_{cur}|$ аналогичный.
Поймем, что числа $L \dots L'$ перейдут в какие-то числа $f(X) \dots f(Y)$. Где $L' = L + |L_{cur}| - 1, X = L + 2|L_{cur}| - 1, Y = L + |L_{cur}|$. (то есть у них будет ответ такой же как и у них в итоговом подсчете). Тогда мы можем за линию обозначить для чисел $L \dots L'$ на кого надо будет ссылаться, чтобы узнать для них ответ.



Так как суммарно наш отрезок уменьшится не более чем на X_{max}, значит наш алгоритм будет работать за $O(n + X_{max} + Q)$.

Задача I. А ну-ка, руки вверх!

Идея: Кутузов Артём
Разработка: Кутузов Артём

Для каждого i от 0 до n найдём ans_i — на какой наименьший угол нужно повернуть форт, чтобы в Джона Сильвера было направлено хотя бы i ружей. Проинициализируем ans числами 360, что будет означать, что добиться желаемого невозможно.

Посчитаем s — сколько ружей изначально направлено в Джона Сильвера. Чтобы проверить, направлено ли ружьё в Джона Сильвера, достаточно проверить, что вектор направлен между касательными к окружности. Положим $ans_s = 0$.

Пусть мы повернули форт на какой-то оптимальный угол $\alpha > 0$. Тогда какой-то из векторов должен совпасть с правой касательной к окружности, ведь иначе мы могли бы повернуть на меньший угол. Поэтому давайте для каждого i от 1 до n предподсчитаем res_i — сколько ружей будет направлено в Джона Сильвера, если ружьё номер i совпадёт с правой касательной к окружности и α_i — угол, на который для этого нужно повернуть. Все эти значения можно найти за методом двух указателей, отсортировав ружья по углу. Затем для каждого i от 1 до n положим $ans_{res_i} = min(ans_{res_i}, \alpha_i)$.

Пройдём по массиву res от конца к началу и положим $ans_i = min(ans_i, ans_{i+1})$. В итоге мы получим готовый массив ans.

Для каждого p_i найдём минимальное число ружей g, такое что если все они направлены в Джона Сильвера, то он будет поражён с вероятностью хотя бы p_i. Если направлено g ружей из n, то вероятность не попасть за один выстрел равна $\frac{n-g}{n}$, а не попасть за k выстрелов $\left(\frac{n-g}{n}\right)^k$. Таким образом вероятность попасть будет $1 - \left(\frac{n-g}{n}\right)^k \geqslant p_i$. Откуда находим наименьшее подходящее $g = \lceil n - n\sqrt[k]{1 - p_i} \rceil$. Остаётся только вывести ans_g (или -1, если $ang_g = 360$).

Итоговая асимптотика $O(n \log n + O(q))$.

Задача J. Нужно больше золота

Идея: Артём Абатуров
Разработка: Артём Абатуров

Для начала давайте посчитаем степень каждой вершины. Обозначим за deg_v степень вершины v.

Теперь на самом деле мы можем ориентировать каждое ребро. Рассмотрим ребро, соединяющее вершины u и v. Тогда у нас есть три случая:

1. $deg_u < deg_v$. В этом случае мы можем перейти только из вершины u в вершину v, поэтому ориентируем ребро таким образом.
2. $deg_u > deg_v$. По аналогичной причине ориентируем ребро из v в u.
3. $deg_u = deg_v$. В этом случае заметим, что мы никак не можем переходить по этому ребру, а значит давайте просто удалим его.

Нетрудно понять, что граф, который получается после такой ориентации рёбер, является ациклическим. Еще одно важное свойство этого графа, которое нам понадобится — длина максимального пути не превосходит $2\sqrt{m}$. Почему это так? Суммарная степень вершин на пути длины k в нашем графе будет равна хотя бы

$$1 + 2 + 3 + \cdots + k = \frac{k \cdot (k+1)}{2}$$

Значит количество уникальных рёбер, у которых хотя бы 1 сосед находится в этом пути равно хотя бы

$$\frac{k \cdot (k+1)}{4}$$

Естественным образом

$$\frac{k \cdot (k+1)}{4} \leqslant m$$

$$k^2 \leqslant k \cdot (k+1) \leqslant 4m$$

$$k \leqslant 2\sqrt{m}$$

Далее заведем следующую динамику: $dp[v][d]$ будет равно максимальному количеству пиастров, которое мы можем заработать, начиная в вершине v при условии, что первое путешествие будет иметь длину равную d. Также для удобства будем поддерживать $opt[v]$ — максимальное количество пиастров, которое можно заработать, начиная в вершине v без всяких дополнительных условий. $opt[v]$ просто равно максимальному значению $dp[v][d]$ при фиксированном d. Эту динамику можно пересчитывать следующим образом:

$$dp[v][0] = a[v]$$

$$dp[v][1] = \max_u opt[u] + a[v] - b[1]$$

$$dp[v][d] = \max_u (dp[u][d-1] - a[u]) + b[d-1] + a[v] - b[d]$$

Где $d > 1$, а u — сосед вершины v.

Итоговая асимптотика решения — $O(n\sqrt{m})$

Задача K. Они заряжают пушку... ЗАЧЕМ?!

Идея: Кутузов Артём
Разработка: Кутузов Артём

Посчитаем $c_i = (a_1 - b_1) + (a_2 - b_2) + \ldots + (a_i - b_i)$ — то, насколько Джим обгонит пиратов, если будет ехать максимально быстро.

Предподсчитаем суффиксные минимумы $s_i = min(c_i, c_{i+1}, \ldots, c_n)$.

По массиву c построим массив p таким образом: $p_1 = 1$ и p_{i+1} — наименьший индекс, больший p_i, такой, что $c_{p_{i+1}} > c_{p_i}$. Это можно сделать жадно одним проходом.

Джим может действовать таким образом: пусть в i-й день он попробует проплыть a_i. Если он при этом не обогнал пиратов, то плыть медленнее не имеет смысла. Если при этом он обогнал пиратов, то возможны два случая: если он сможет дальше им не уступать, то ответ i. Иначе ему оптимально проплыть столько, чтобы сравняться с пиратами.

Заметим, что в силу такой стратегии, ответ на задачу всегда будет одним из чисел массива p. Пусть теперь нам дано число D. Найдём бинпоиском первое p_i, такое что $c_{p_i} > D$, то есть первый день, когда у Джима есть шанс обогнать пиратов. Если $s_{p_i} > D$, то это значит, что потом Джим может плыть максимально быстро и никогда не уступить пиратам, поэтому ответ в таком случае p_i.

Иначе в момент времени p_i Джим сравняется с пиратами. В таком случае в каждый из моментов $p_{i+1}, p_{i+2}, \ldots$ Джим либо останется наравне с пиратами, либо обгонит их. Пусть он обгонит их в момент p_j $(j > i)$. Это значит, что в момент p_{j-1} он был наравне с пиратами. Значит в момент p_j преимущество Джима будет равно $c_{p_j} - c_{p_{j-1}}$. Для того, чтобы проверить, хватит ли этого преимущества, нужно проверить, что $s_{p_j} > c_{p_{j-1}}$.

Для каждого p_i, можно предподсчитать ответ на задачу p_j в случае, если Джим сравнялся с пиратами. Для этого надо найти минимальное $j > i$, такое что $s_{p_j} > c_{p_{j-1}}$. Такой предподсчёт можно сделать одним проходом с конца к началу.

Итоговая асимптотика $O(n + q\,log\,n)$.

Задача L. Мне не нравятся эти матросы!

Идея: Артём Абатуров, Артём Степанов
Разработка: Артём Абатуров

Давайте обозначим за cnt_x количество подотрезков в нашем массиве, которые не содержат число x. Ключевой факт в данной задаче — после одного запроса у нас меняется не более 2 значений cnt_x. Тогда, если мы сможем как-то быстро посчитать значения cnt_x, а затем как-то быстро их обновлять, то мы сможем решить нашу задачу.

Заметим, что количество подотрезков, не содержащих число x, можно вычислить, зная позиции, на которых число x встречается в массиве. Давайте для каждого числа x, присутствующего в массиве a, будем хранить отсортированный список позиций, на которых оно находится.

Подотрезки, не содержащие число x, — это подотрезки, полностью лежащие между позициями появления x. Если обозначить позиции появления x как $p_1, p_2, \ldots, p_k$, то промежутки между ними будут:

- От 1 до $p_1 - 1$
- От $p_1 + 1$ до $p_2 - 1$

- ...
- От $p_k + 1$ до n

Количество подотрезков в каждом таком промежутке длины L_i равно $\frac{L_i \cdot (L_i+1)}{2}$.

Таким образом, общее количество подотрезков, не содержащих число x, можно вычислить как сумму по всем промежуткам:

$$cnt_x = \sum_i \frac{L_i \cdot (L_i + 1)}{2}$$

Теперь рассмотрим, как эффективно обновлять cnt_x при изменении элемента массива. При изменении роли s-го матроса с значения old_val на new_val нам нужно сделать следующее:

1. Удаление позиции s из списка позиций числа old_val:
   - Найдем в списке позиций числа old_val ближайшие позиции слева (prev) и справа (next) от позиции s.
   - До удаления s были два промежутка:
     - Левый: от prev $+ 1$ до $s - 1$ (длина L_1)
     - Правый: от $s + 1$ до next $- 1$ (длина L_2)
   - После удаления s эти промежутки объединяются в один новый промежуток: от prev $+ 1$ до next $- 1$ (длина $L = L_1 + L_2 + 1$)
   - Обновим $cnt_{\text{old\_val}}$:

$$cnt_{\text{old\_val}} = cnt_{\text{old\_val}} - \frac{L_1(L_1+1)}{2} - \frac{L_2(L_2+1)}{2} + \frac{L(L+1)}{2}$$

2. Добавление позиции s в список позиций числа new_val:
   - Найдем в списке позиций числа new_val ближайшие позиции слева (prev) и справа (next) от позиции s.
   - До добавления s был один промежуток — от prev $+ 1$ до next $- 1$ (длина L)
   - После добавления s этот промежуток разделяется на два:
     - Левый: от prev $+ 1$ до $s - 1$ (длина L_1)
     - Правый: от $s + 1$ до next $- 1$ (длина L_2)
   - Обновим $cnt_{\text{new\_val}}$:

$$cnt_{\text{new\_val}} = cnt_{\text{new\_val}} - \frac{L(L+1)}{2} + \frac{L_1(L_1+1)}{2} + \frac{L_2(L_2+1)}{2}$$

Для эффективного выполнения этих операций будем хранить списки позиций в структуре данных, позволяющей быстро выполнять вставки, удаления и поиск соседей — например, в C++ можно использовать set.

Кроме того, будем поддерживать мультимножество (в C++ можно использовать multiset) значений cnt_x для всех x. При каждом изменении cnt_x будем обновлять мультимножество: удалять старое значение и добавлять новое. Это позволит находить минимальное cnt_x после каждого запроса за $O(1)$, взяв первое (наименьшее) значение из мультимножества.

Таким образом, мы эффективно решаем задачу, поддерживая значения cnt_x и отслеживая минимальное из них после каждого изменения в массиве. Итоговая асимптотика решения — $O((n+q)\log n)$.

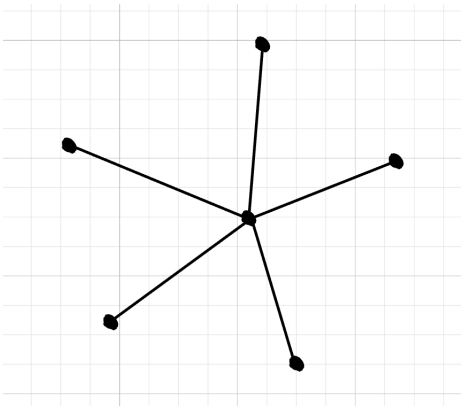
Задача M. Детство капитана Флинта

Идея: Артём Кутузов
Разработка: Михаил Бугрышев

Рассмотрим граф, в котором вершинами будут острова, а рёбрами — всевозможные прямые пары островов, между которыми можно путешествовать на корабле напрямую. Так как в нём по условию нет циклов, то этот граф разбивается на компоненты связности, каждая из которых представляет собой дерево. Тогда ответ — это сумма расстояний между каждой парой вершин в компоненте по всем компонентам (так как из одной компоненты связности мы никак не доберёмся до другой, значит пираты могли путешествовать только внутри одной компоненты).

Рассмотрим одну такую компоненту, пусть в ней расположено дерево размера k. Тогда очевидно, что в этой компоненте будет ровно $k-1$ ребро (так как в дереве на k вершинах столько рёбер). Значит во всех компонентах рёбер ровно $n-s$, где s — количество компонент (ведь каждая вершина входит ровно в одну компоненту — значит вычитаемое равно n, а -1 добавляется в сумму для каждой компоненты, то есть мы вычтем число 1 ровно s раз). По условию, количество рёбер равно $m \Longrightarrow m = n - s \Longrightarrow s = n - m$, то есть мы нашли количество компонент в графе.

Теперь рассмотрим одну компоненту размера k и посчитаем, какое минимальное суммарное расстояние между парами вершин в ней будет. В этой компоненте расположено дерево размера k (см. выше). У нас есть ровно $k-1$ пар вершин, между которыми расстояние равно 1 (так как если расстояние между какой-то парой вершин равно $1 \Longrightarrow$ они соединены ребром. Так как в дереве рёбер ровно $k-1$, то и пар островов, между которыми расстояние равно 1, составляет $k-1$). Все остальные пары вершин находятся хотя бы на расстоянии 2 друг от друга. А значит, минимальное суммарное расстояние точно не меньше, чем $(k-1)\cdot 1 + \left(\frac{k\cdot(k-1)}{2} - (k-1)\right)\cdot 2 = (k-1)^2$ (так как всего пар вершин в дереве — $\frac{k\cdot(k-1)}{2}$). Теперь построим дерево, на котором данная оценка достигается. Для этого давайте построим «ёжик»: зафиксируем одну вершину и соединим её со всеми остальными. Тогда каждые 2 вершины либо соединены ребром (тогда между ними расстояние 1), либо обе соединены с зафиксированной вершиной (тогда между ними расстояние 2).



Пример «ёжика»

Осталось понять, на какие именно размеры разбивать наши s компонент. Пусть нашлись 2 такие компоненты размеров a, b, где $b - a \geqslant 2$. Значит ответ для этих двух компонент был равен $(a-1)^2+(b-1)^2 = a^2-2\cdot a+1+b^2-2\cdot b+1$ (см. выше). Тогда мы можем «забрать» одну вершину у компоненты размера b и «отдать» ей компоненте размера a. Тогда после этой операции размеры компонент станут равными $a+1$ и $b-1$ (заметим, что размер второй компоненты до сих пор не меньше,

чем размер первой) и ответ равен $((a+1)-1)^2+((b-1)^2-1)^2 = a^2+b^2-4\cdot b+4$. Заметим, что ответ точно уменьшился, так как $(a^2-2\cdot a+1+b^2-2\cdot b+1)-(a^2+b^2-4\cdot b+4) = -2\cdot a+2\cdot b-2 = 2\cdot(b-a-1) \geqslant 2$, так как $b-a \geqslant 2$ — см. выше (только что мы из изначального ответа вычли тот ответ после операции). Значит, так как данная разница положительна, то наш ответ строго уменьшился, значит изначальный ответ не был оптимален. Значит либо во всех компонентах одинаковое количество вершин (в этом случае n делится на s), либо существует такое $k \in \mathbb{Z} : k \geqslant 0$ и все компоненты имеют размер k или $k+1$ (тогда n не делится на s и $k = \lfloor \frac{n}{s} \rfloor$). Вышеуказанные формулы верны, так как мы можем добавлять вершины по одной в компоненты — если сейчас есть 2 компоненты с различными размерами, то добавим эту вершину в компоненту с меньшим размером (иначе создадутся 2 компоненты, размеры которых отличаются хотя бы на 2 — противоречие, см. выше). Если же размеры всех компонент равны, то добавим данную вершину в любую компоненту — очевидно, что после этой операции не найдётся 2 компонент, размеры которых отличаются хотя бы на 2.

А значит мы получили решение, так как оценили суммарное расстояние между каждой парой вершин и привели пример графа, на котором этот ответ достигается.