Задача A. Считаем конфеты

Достаточно перебрать все возможные места расположения конфет и проверить, есть ли там конфета. Такое решение работает за $O(nm)$.

Задача B. Дерево и бензобаки

В задаче требуется посчитать сумму c_v для вершин на пути в дереве с шагом в k_i. Давайте по отдельности рассматривать те запросы, в которых $k_i > \sqrt{n}$, и те, в которых $k_i \leqslant \sqrt{n}$. Назовем шаг k_i в этих запросах длинным и коротким, соответственно.

После нахождения наименьшего общего предка один запрос с произвольным путем в дереве можно свести к не более, чем двум запросам с вертикальными путями в дереве. Это можно сделать за время $O(n \log n)$.

Запросы даны в оффлайне, поэтому будем отвечать на них одновременно. Будем спускаться по дереву обходом в глубину и отвечать на запрос, когда находимся в вершине — нижнем конце запроса. При этом будем хранить стек вершин в текущем пути до корня. Тогда, чтобы ответить на запрос с длинным шагом, будем просто подниматься от нижнего конца до верхнего с шагом в k_i, находя очередную вершину при помощи стека. Так как $k_i > \sqrt{n}$, то всего шагов будет сделано меньше, чем $\sqrt{n}$.

Для того, чтобы отвечать на запросы с коротким шагом, будем хранить несколько стеков — аналогов префиксных сумм для всех возможных $1 \leqslant k \leqslant \sqrt{n}$. В стеке с номером k на позиции i будет храниться сумма c_v для всех вершин на текущем пути до корня с глубинами не большими i и имеющими такой же как i остаток по модулю k. Тогда ответ на запрос это разность двух элементов стека с номером k_i. Пересчет этих стеков при спуске в вершину это добавление элемента в каждый стек, а при подъеме — удаление. Таким образом, ответ на запрос работает за $O(1)$ времени, но операции со стеком суммарно работают $O(n\sqrt{n})$.

Итоговое время работы $O(n\sqrt{n})$.

Задача C. Теперь про время

Зафиксируем какой-то порядок следования отрезков в системе A и в системе B. Можно проверять, что этот порядок соответствует всем известным соотношениям следующим алгоритмом. Изначально установим оба указателя на начало дня в обеих системах. Будем двигать указатели постепенно. Назовем системы первой и второй так, что в первой системе от начала дня прошло t_1 часов, а во второй t_2, причем $t_1 \leqslant t_2$ (в случае равенства для однозначности скажем, что первая система это A). Подвинем указатель в первой системе. После этого проверим корректность некоторых соотношений, будем двигать указатели дальше.

На очередном шаге, пусть следующий отрезок в первой системе имеет номер i и идет τ_1 часов, а последний отрезок, который мы уже учли, во второй системе имеет номер j и идет τ_2 часов. Из алгоритма видно, что между t_1 и $\min(t_1 + \tau, t_2)$ часами от начала дня в первой системе будет все время часть i, а во второй — все время часть j. В момент передвижения указателя будем проверять корректность всех тех соотношений, которые должны произойти между временем t_1 и $\min(t_1+\tau_1, t_2)$ от начала дня, то есть все те соотношения, у которых в первой системе идет i-я часть, и время в которых указано от 0 до $\min(\tau_1, t_2 - t_1)$, и все те соотношения, у которых во второй системе идет j-я часть, и время в которых указано от $t_1 - (t_2 - \tau_2)$ до $\min(t_1 + \tau_1, t_2) - (t_2 - \tau_2)$. Во всех таких соотношениях должны быть в первой системе i-я часть и во второй системе j-я часть, и должна быть правильная разница во времени между ними — в $t_1 - (t_2 - \tau_2)$ часов.

Понятно, что эти условия необходимы и достаточны для корректности порядка следования.

В этом алгоритме проверки корректности порядка видно, что для всех порядков следования в двух системах, в которых мы уже прошли префиксы указателями, очередная проверка проводится абсолютно одинаково при фиксированных множествах уже пройденных указателями событий и фиксированных последних пройденных частях. Поэтому, можно считать следующую динамику по подмножествам. $dp[mask_A][mask_B][last_A][last_B]$ — количество способов упорядочить множество частей $mask_A$ в системе A и множество $mask_B$ в системе B, что последние элементы в этих порядках это $last_A$ и $last_B$, соответственно, и алгоритм проверки корректности на них дошел до этого состояния и не нашел ошибок. Эту динамику можно пересчитывать вперед, поскольку сумма длин частей в

подмножествах это и будут описанные выше t_1 и t_2, и по ним ясно, в какой из систем нужно двигать указатель, нужно просто перебрать следующую часть и проверить корректность для небольшого кусочка времени (сопоставив новую часть с $last_A$ или $last_B$).

В конце, нужно посмотреть на сумму всех возможных динамик, у которых пришли в конечное состояние проверки — прошли оба множества целиком. Если суммарно есть ровно один способ, то его можно восстановить, иначе, ответ Ambiguous или Inconsistent в зависимости от того, нулевое ли получилось количество способов. Итоговое время работы $O(2^{n+m}nm(n+m))$

Задача D. Самоописывающие числа

Поскольку задано все множество встречающихся цифр, итоговый ответ имеет вид $a_0b_0a_1b_1...a_{k-1}b_{k-1}$, где k — количество различных цифр в множестве, а $b_1...b_{k-1}$ — какая-то перестановка заданных цифр. При этом, естественно, все $a_0...a_{k-1}$ тоже должны быть из множества встречающихся цифр.

При фиксированных цифрах a_i легко проверить корректность числа — достаточно посчитать количество раз, которое встречается каждая цифра. При этом, переставлять пары a_ib_i можно в произвольном порядке, поэтому, для минимальности достаточно отсортировать их по возрастанию (как пары цифр), чтобы получить минимальное возможное число с такими частотами цифр.

Поэтому, для решения задачи, нужно перебрать частоты цифр такие, что сами частоты лежат в множестве, и сумма которых равна $2k$ (так как всего цифр в числе $2k$). Для корректных частот минимальное число получается сортировкой пар. Прорелаксируем ответ этим числом.

С условием того, что сумма равна $2k$ уже остается очень мало наборов частот — не больше чем C_{2k-1}^{k-1} (число способов представить $2k$ в виде суммы k натуральных слагаемых). А это 92378 при максимальном $k = 10$, поэтому можно просто перебрать все такие наборы рекурсивно.

Задача E. Соединяем точки с землёй

Отсортируем точки первого типа по координате x по возрастанию. Добавим к точкам первого типа две дополнительных точки — с координатами $(0, 0)$ и $(M, 0)$, а к точкам второго типа две точки с координатами $(0, M)$ и (M, M), где M — число, большее всех координат. Теперь обозначим точки первого типа за $A_0...A_{n+1}$ (в порядке возрастания), а второго типа за $B_0...B_{n+1}$ (новые точки это $A_0, A_{n+1}, B_0, B_{n+1}$).

Будем считать следующую динамику: $dp[left][right][left_b][right_b]$ — количество способов разбить на пары точки первого типа с номерами i, что $left < i < right$, и точки второго типа, y-координаты которых не выше, чем y-координаты точек второго типа с номерами $left_b$ и $right_b$, и которые находятся строго правее прямой, соединяющей A_{left} с B_{left_b}, и строго левее прямой, соединяющей A_{right} с B_{left_b}. Таким образом, ответ на задачу будет $dp[0][n+1][0][n+1]$.

Для того, чтобы пересчитать динамику, во-первых, посчитаем количество точек второго типа, удовлетворяющих условию. Если их количество не совпадает с количеством точек первого типа $(right - left - 1)$, то разбить на пары нельзя. Начальное значение динамики: когда и тех и других точек по ноль, есть ровно один способ разбить на пары. Если же есть какие-то точки, то пусть номер самой высокой из них второго типа это j. Переберем ту точку i, которая будет с ней в паре. Тогда, количество способов доразбить оставшиеся точки на пары это ровно $dp[left][i][left_b][j] \cdot dp[i][right][j][right_b]$, поскольку среди точек второго типа остались только точки с y-координатами от 0 и до y-координаты B_j, и, так как отрезки пар не должны пересекаться, а точки $A_{left+1}...A_{i-1}$ находятся строго левее от отрезка A_iB_j, то они должны быть в паре с теми точками, которые строго левее этого отрезка, с точками $A_{i+1}...A_{right-1}$ аналогично.

Таким образом, у динамики всего $O(n^4)$ состояний и пересчет работает за $O(n)$, поэтому, итоговое время $O(n^5)$.

Задача F. Треугольники на плоскости

Докажем, что пересечение любого набора треугольников такого вида это либо треугольник такого вида, либо пустое множество. Действительно, треугольник с углом в (x_0, y_0) и стороной r_0 является пересечением трех полуплоскостей: $x \geqslant x_0$, $y \geqslant y_0$ и $x + y \leqslant x_0 + y_0 + r$. Тогда пересечение n таких треугольников это тоже пересечение трех полуплоскостей вида $x \geqslant A$, $y \geqslant B$, $x + y \leqslant C$ (где

$A = \max x_i$, $B = \max y_i$, $C = \min(x_i + y_i + r_i)$ для $1 \leqslant i \leqslant n$). А пересечение таких полуплоскостей это треугольник с углом в (A, B) и стороной $C - A - B$, если $C - A - B \geqslant 0$, или пустое множество иначе.

Тогда, можно легко посчитать для каждого подмножества треугольников площадь их пересечения за время $O(2^n \cdot n)$. Назовем эту величну $area(S)$, где S — подмножество. Далее посчитаем для каждого подмножества треугольников площадь той части плоскости, которая лежит в пересечении этого множества, и не лежит ни в каких других треугольниках. Назовем эту величину $unique(S)$. Она вычисляется так: $unique(S) = area(S) - \sum_{S \subsetneq T} unique(T)$. Тогда, перебирая множества по убыванию количества треугольников, ее можно посчитать за $O(3^n)$.

Итоговый ответ на задачу это сумма $unique(S)$ по всем множествам нечетного размера.

Задача G. Восстановить значения в дереве

Для начала, посчитаем динамическим программированием по поддеревьям такие величины: min_v и max_v — минимальное и максимальное возможное число, записанное в вершине v, если выполнены все условия в поддереве вершины v (в листьях положительные целые числа, в остальных вершинах числа — сумма чисел в потомках, в тех u, в которых $A_u \neq 0$, записано A_u), и то, возможно ли вообще так корректно расставить числа.

Значения в листьях такие: если $A_v \neq 0$, то $min_v = A_v, max_v = A_v$, иначе $min_v = 1, max_v = \infty$. Пересчет будет устроен таким образом: обозначим $L = \sum_{u \in children(v)} min_u$, $R = \sum_{u \in children(v)} max_u$

- $A_v = 0$. Возможно расставить, если возможно корректно расставить числа в поддеревьях всех детей, $min_v = L$, $max_v = R$.
- $A_v \neq 0$. Возможно расставить, если возможно корректно расставить числа в поддеревьях детей и $L \leqslant A_v \leqslant R$. Устанавливаем $min_v = A_v, max_v = A_v$.

По индукции по глубине дерева легко доказывается, что в поддереве вершины v можно корректно расставить числа тогда и только тогда, когда описанной выше динамикой по поддеревьям было посчитано, что это возможно, и все возможные значения числа в вершине v в корректных расстановках это все целые числа между min_v и max_v.

Понятно, что если эта динамика установила, что во всем дереве расставить числа невозможно, то ответ impossible. Также, если $min_1 \neq max_1$ (вершина 1 — корень дерева), то ответ восстанавливается неоднозначно, так как по доказанному выше можно корректно расставить числа как с min_1 в корне, так и с $min_1 + 1$ в корне.

Опишем рекурсивную процедуру, которая будет проверять, что можно корректным образом расставить числа в поддереве вершины v с числом x в вершине v единственным способом, и находить этот способ. Как и выше, определим L и R. Считаем, что при запуске этой процедуры уже гарантируется, что $min_v \leqslant x \leqslant max_v$. Во всех случаях сначала пишем x в ответ в вершине v.

- Если v — лист, то расставить можно единственным способом всегда.
- Если $x = L$, то в каждой вершине u, являющейся ребенком вершины v, должно быть записано число min_u, соответственно. Сделаем рекурсивные вызовы для u, min_u, есть единственный способ расставить числа в поддереве v тогда и только тогда, когда есть единственный способ расставить числа в поддереве каждого ребенка u в соответствующих рекурсивных вызовах.
- Если $x = R$, то, аналогично, в каждой вершине u, являющейся ребенком вершины v, должно быть записано число max_u, соответственно. Сделаем рекурсивные вызовы для u, max_u, есть единственный способ расставить числа в поддереве v тогда и только тогда, когда есть единственный способ расставить числа в поддереве каждого ребенка u в соответствующих рекурсивных вызовах.
- Если у вершины v есть только один потомок u^*, у которого $min_{u^*} \neq max_{u^*}$, то в любом другом ребенке u однозначно определяется число (это $min_u = max_u$), которое должно быть записано, поэтому и в u^* число определено однозначно, так как сумма всех этих чисел x. Из корректности x попадает в возможный диапазон и число в u^*. Сделаем рекурсивные вызовы

для u^* и посчитанного значения y, а также для всех остальных u с min_u. Есть единственный способ расставить числа в поддереве v тогда и только тогда, когда есть единственный способ расставить числа в поддереве каждого ребенка u в соответствующих рекурсивных вызовах.

- Иначе, покажем, что расставить можно не единственным способом. Не выполнены все прошлые условия, поэтому $L < x < R$. Представим x в виде разбиения на целые слагаемые y_u такие, что $min_u \leqslant y_u \leqslant max_u$ (где u — ребенок v). Тогда, среди них есть u_1 такое, что $min_{u_1} \neq y_{u_1}$ и такое u_2, что $max_{u_2} \neq y_{u_2}$ поскольку $x \neq L, x \neq R$. При этом, можно выбрать различные u_1, u_2, так как если они совпали, то, так как есть больше одного ребенка, у которых $min_u \neq max_u$, то другая вершина u, удовлетворяющая этому условию, подходит в качестве u_1 или u_2, соответственно. Но тогда, существует способ корректно расставить числа y_u в u, как и способ расставить те же числа в детях кроме u_1, u_2, а в u_1 поставить $y_{u_1} - 1$, а в u_2 записать $y_{u_2} + 1$. Понятно, что эти способы различные.

Таким образом, доказана работа рекурсивной процедуры. Итоговое время $O(n)$.

Задача H. Упаковываем числа

Если бы все числа были различными, то их можно было бы как угодно распределять на наборы размера k. Поэтому, за какое-то количество операций точно можно добиться максимально возможного числа наборов $\lfloor \frac{\sum_{i=1}^{n} a_i}{k} \rfloor$. Обозначим это количество за b. Разберем два случая для нахождения необходимого числа операций.

Пусть $b = 1$. Рассмотрим число i. Пусть после всех операций оно встречается c_i раз. Тогда, оно встречается в наборе не более чем $k-1$ раз (так как не весь набор из одного числа), а также не более чем $\max(c_i - 1, 1)$ раз (так как если $c_i > 1$, то не могут все числа i лежать в одном наборе). Тогда, суммарно в единственном наборе могут находиться не более чем $\sum_{i=1}^{N} \min(k-1, \max(c_i-1, 1))$ чисел, где $c_1 ... c_N$ — итоговые количества чисел (считаем, что все числа от 1 до N встречаются). Посчитаем эту сумму.

- Если сумма не меньше k, то никаких операций делать не нужно — можно просто положить в набор $\min(k-1, \max(a_i - 1, 1))$ чисел, равных i. Оставив из них произвольные k, получим корректный набор.

- Если есть $a_i > k$, то это единственное встречающееся число (так как иначе сумма это уже $k-1$ от a_i и еще что-то положительное от других чисел, то есть больше или равна k). Тогда, достаточно поменять одно число на уникальное и положить в набор новое число и $k-1$ копию числа i. Числа, равные i, останутся вне набора, поэтому разбиение корректно.

- Иначе, изначальная сумма выглядит как $\sum_{i=1}^{n} \max(a_i - 1, 1) = \sum_{i=1}^{n} a_i - T$, где T — количество таких i, что $a_i > 1$. Тогда, чтобы добиться того, что такая сумма больше или равна k, нужно уменьшать количество $a_i > 1$. Если $a_i > 1$, то для того, чтобы количество чисел i стало не больше чем 1, нужно хотя бы $a_i - 1$ превращений (причем чисел, изначально равных i). При этом, ровно такого количества превращений хватит, если превращать все числа кроме одной копии a_i в новые уникальные числа. Поэтому, чтобы найти минимальное необходимое число операций, нужно посчитать то, насколько нужно уменьшить сумму D, после чего посчитать сумму $a_i - 1$ для D минимальных $a_i > 1$. Превращением $a_i - 1$ чисел i в новые уникальные числа мы как раз добьемся увеличения искомой суммы на D.

Пусть $b > 1$. Рассмотрим число, встречающееся чаще всего. Пусть это m. Тогда, должно быть хотя бы b чисел, не равных m, поскольку в каждом наборе должно быть число, не равное m. Значит, операций должно быть хотя бы $b - (\sum_{i=1}^{n} a_i - a_m)$ (вычитаемое это и есть количество чисел, не равных m). За такое число операций можно добиться того, чтобы было хотя бы b чисел, не равных m, просто превращая числа, равные m в новые уникальные. (так как всего хотя бы $2b$ чисел, а не равных m изначально $< b$, то к концу этого процесса все еще m — самое частое число, оно встречается хотя бы b раз).

Докажем, что если $\sum_{i=1}^{n} a_i - a_m \geqslant b$, то можно выделить b групп по k элементов, чтобы были выполнены все условия. Действительно, давайте упорядочим числа так, чтобы одинаковые числа

шли подряд, а первым шло число m. После этого, в таком порядке будем заполнять группы, добавляя числа по одному сначала в первую, потом во вторую ... в b-ю группу, потом заново в первую, вторую, и так далее. Если $a_i > 1$, то либо же не все числа i лежат в группах, либо оно встречается хотя бы в двух группах (так как второе добавленное число i точно не в той же группе, что первое).

Если m встречается реже, чем b раз, и есть какая-то группа целиком из одного числа, то это число встречается хотя бы k раз в этой группе, и хотя бы $k-1$ раз в других группах. Тогда суммарное количество этого числа это $bk - b + 1 \geqslant b + 1 > b$. Значит, m не самое частое число. Противоречие!

Если вдруг какая-то группа состоит целиком из числа m, то таким свойством обладают первые l групп, а в оставшихся число m встречается по $k - 1$ раз. Так как есть хотя бы b элементов, не равных m, а их всего $b - l$ в наборах, то есть еще хотя бы l. Значит, l не больше, чем остаток общего количества чисел при делении на b. Тогда, если просто выкинуть l копий m (то есть точно также бы расставляли числа по группам, но считая, что m встречается на l раз реже), то останется хотя бы bk элементов, а m все еще встречается хотя бы b раз.

Если m встречается хотя бы b раз и нет группы целиком из m, то в каждой группе встречаются разные числа, поскольку в каждой группе есть m.

Таким образом, решение доказано.

Задача I. Взаимопростое дерево на окружности

Сначала просто посчитаем все ребра, которые можно провести. После этого можно забыть про сами записанные числа и считать, что на окружности расположено n вершин графа, пронумерованных по кругу от 1 до n.

Будем считать следующую динамику по подотрезкам. $dp[left][right][can_{left}][can_{right}]$, где $1 \leqslant left < right \leqslant n$ соответствует подотрезку вершин, а can_{left} и can_{right} булевые флаги. Эта динамика будет считать количество корректных способов провести ребра так, чтобы получилось дерево на вершинах с $left$ по $right$ в предположении, что ребро, соединяющее $left$ и $right$ уже проведено (возможно, такого ребра на самом деле нельзя проводить, но мы все равно будем считать значения этой динамики), а из $left$ можно проводить еще ребра, если истинный флаг can_{left}, из $right$ можно проводить еще ребра, если истинный флаг can_{right}.

Базовые значения динамики: на $left + 1 = right$ и любых флагах есть ровно один способ, поскольку граф уже дерево и никаких еще ребер проводить нельзя.

Теперь посмотрим, как пересчитывается динамика в случае $right > left + 1$. Если есть какое-то ребро из $left$ (тогда обязательно истинен флаг can_{left}), то переберем ребро в максимальную соединенную с $left$ вершину i на подотрезке (выполнено $left < i < right$). Тогда таких деревьев ровно $dp[left][i][can_{left}][1] \cdot dp[i][right][1][can_{right}]$, поскольку нет ребер из $left$ в вершины с большим номером, то в части графа, оставшейся справа, можно мысленно заменить два ребра i — $left$ и $left$ — $right$ на одно ребро i — $right$, и дополнять его до дерева, способов столько же.

Если же нет ребер из $left$, то должно быть ребро из $right$ (так как дерево связано), переберем ребро в минимальную соединенную с $right$ вершину i. Аналогично, деревьев ровно $dp[left][i][0][1] \cdot dp[i][right][1][can_{right}]$. Тут важно, что в левом слагаемом вместо флага can_{left} в пересчете 0, так как мы предполагаем, что из вершины $left$ ребер больше нет.

Таким образом, можно посчитать эту динамику за $O(n^3)$, так как у нее $O(n^2)$ состояний и пересчет работает за $O(n)$. Для того, чтобы найти ответ на задачу, раздвоим первую вершину на две — еще и $(n+1)$-ю и мысленно соединим их ребром. Тогда число способов дополнить эту конструкцию до дерева это и есть ответ на задачу, и он лежит в $dp[1][n+1][1][0]$.

Задача J. Ценные точки

Задача решается двоичным поиском по ответу. Рассмотрим такую функцию от C: возможно ли выбрать m точек так, чтобы расстояние между любыми двумя было хотя бы d, а все их ценности хотя бы C. На значениях C вплоть до ответа на задачу это возможно, а на больших это сделать невозможно. Осталось реализовать эту функцию.

Проверить для фиксированного C можно жадно. Будем идти по точкам в порядке возрастания x координаты, и если у неё $c_i \geqslant C$, и расстояние до последней взятой точки $\geqslant d$, то будем брать её в множество. Если в итоге набрали хотя бы m точек, то C подходит.

Двоичный поиск по C можно делать только из набора c_i, поэтому итоговое решение работает за $O(n \log n)$.

Задача K. Всё битовое не ноль

Во-первых, заметим, что OR набора чисел всегда больше или равен их побитовому AND, поэтому достаточно добиться того, чтобы побитовый AND и XOR были ненулевыми. Пусть w таково, что $2^w > max(A[i])$. Будем перебирать ненулевой бит в итоговом AND от нулевого до w-го. Из дальнейшего решения будет понятно, что ответ всегда не больше, чем $n + 2$, поэтому перебирать большие биты бесполезно, так как для того, чтобы добиться такого единичного бита в AND, нужно сделать очень много операций.

Покажем, как найти минимальное количество операций, которое нужно сделать, чтобы установить k-й бит в AND равным 1, и чтобы XOR был ненулевым. Переберем все $A[i]$, у которых нулевой k-й бит. Для них найдем ближайший слева и ближайшее справо числа с единичным k-м битом (если $A[i] < 2^k$, то слева нет такого числа). Добавим к текущему ответу меньшее из этих двух расстояний, установим $A[i]$ в одно из этих чисел. Если же обе опции были на одном расстоянии, запомним это. Обозначим суммарное посчитанное расстояние за S.

Видно, что хотя бы S операций точно нужно было сделать, чтобы добиться единичного k-го бита во всех $A[i]$. Далее, есть несколько случаев

- Если XOR всех новых чисел ненулевой, то S это и есть ответ.
- Если же XOR всех новых нулевой, но было $A[i]$ у которого левый и правый ближайший на одном расстоянии, то S это тоже ответ (так как можно было вместо замены $A[i]$ на одно ближайшее число заменить на другое за то же самое число действий, итоговый XOR точно изменится и станет ненулевым).
- Если же у всех $A[i]$ был единственный ближайший вариант, то такие итоговые числа — единственные возможные значения $A[i]$ такие, что во всех них ненулевой k-й бит, которых можно достичь за S операций. Тогда, для ответа нужно точно сделать большее число операций. При $k > 0$ ответ это $S + 1$, так как число с единичным k-м битом всегда можно уменьшить или увеличить на 1, чтобы оно все еще имело единичный k-й бит. При $k = 0$ ответ это $S + 2$, поскольку каждое действие меняет суммарную четность чисел, поэтому четность количества операций фиксирована, при этом за еще две операции можно нечетное число превратить в другое нечетное число.

Задача L. Пирамидизация

Для начала поймем, что такое стоимость пирамидизации массива. Заметим, что в массиве $b_1...b_n$, являющемся пирамидой, выполнено следующее свойство: $b_i \geqslant \max(b_1...b_i)$ или $b_i \geqslant \max(b_i, b_{i+1}...b_n)$, так как элемент с номером i либо левее максимума, и тогда выполнено первое, либо правее, и тогда выполнено второе. Значит, $b_i \geqslant \min(\max(b_1...b_i), \max(b_i...b_n))$. Тогда, если массив b получен из массива a только прибавлениями единичек, то $b_i \geqslant \min(\max(a_1...a_i), \max(a_i...a_n))$. Значит, хотя бы $\min(\max(a_1...a_i), \max(a_i...a_n)) - a_i$ прибавлений нужно сделать на позиции i для того, чтобы сделать массив пирамидой. Докажем, что массив $b_i = \min(\max(a_1...a_i), \max(a_i...a_n))$ является пирамидой. Обозначим позицию максимума в массиве a за m. Тогда, для $i \leqslant m$ выполнено $b_i = \min(\max(a_1...a_i), \max(a_i...a_n)) = \min(\max(a_1...a_i), a_m) = \max(a_1...a_i)$, а для $i \geqslant m$, аналогично, выполнено $b_i = \max(a_i...a_n)$. Из этого легко видеть, что массив неубывает от позиции 1 до позиции m (так как берется максимум от расширяющегося множества), и невозрастает от позиции m до позиции n, аналогично.

Таким образом, чтобы сделать массив пирамидой, нужно увеличить элементы его префикса (до максимального элемента) до соответствующих префиксных максимумов, и его суффикса (до максимального элемента) до соответствующих суффиксных максимумов. Теперь давайте посмотрим, как меняется необходимое количество операций при добавлении одного элемента.

Обозначим позиции суффиксных максимумов в массиве $a_1...a_n$ за $i_1 < i_2... < i_k$ (таким образом, обязательно $i_k = n$, для всех j таких, что $i_{l-1} < j < i_l$, выполнено $a_j \leqslant a_{i_l}$, для всех l выполнено

$a_{i_{l-1}} > a_{i_l}$, а a_{i_1} — максимум в массиве). Дописываем элемент x к массиву. Обозначим старую стоимость пирамидизации за S. Есть два случая:

- $x > a_{i_1}$. Тогда новый массив, являющийся пирамидой, до которого нужно увеличить $a_1 ... a_n, x$ будет массивом префиксных максимумов. Так как до a_{i_1} уже был увеличен массив до массива префиксных максимумов, а на позициях с i_1 до n префиксные максимумы равны a_{i_1}, то к стоимости просто прибавится стоимость увеличения элементов правее максимума до a_{i_1}. В S уже учтена стоимость их увеличения до суффиксных максимумов, поэтому, остается добавить $\sum_{l=2}^{k}(i_l - i_{l-1}) \cdot (a_{i_1} - a_{i_l}) = (n - i_1)a_{i_1} - \sum_{l=2}^{k}(i_l - i_{l-1})a_{i_l}$
- $x \leqslant a_{i_1}$. Тогда пусть j максимальный, что $a_{i_j} \geqslant x$. Тогда, нужно увеличить элементы после a_{i_j} до x. Они уже были суффиксными максимумами, поэтому к S остается добавить $\sum_{l=j+1}^{k}(i_l - i_{l-1}) \cdot (x - a_{i_l}) = (n - i_j)x - \sum_{l=j+1}^{k}(i_l - i_{l-1})a_{i_l}$

Таким образом, изменение выражается через суффиксные максимумы, причем интересны только те суффиксные максимумы, которые не больше, чем x. Поэтому, будем идти слева направо по массиву и поддерживать сумму стоимостей пирамидизации у всех суффиксов текущего массива. Для этого будем поддерживать стек суффиксных максимумов. Тогда, так как индексы суффиксных максимумов на суффиксах это суффиксы стека, и для того, чтобы посчитать изменение, интересны только суффиксные максимумы, не большие x, то можно считать все изменения стоимостей пирамидизации прямо во время перестройки стека (когда мы как раз снимаем с него все элементы, меньшие или равные x). Остается только аккуратно посчитать эти суммы.

Все решение работает $O(n)$.

Задача M. Удаляем различные

Будем решать другую задачу — искать для каждого b максимальное количество чисел max_b, что можно набрать b групп из max_b различных чисел в массиве. Тогда ответ на задачу для k это наибольшее b такое, что $max_b \geqslant k$.

Для фиксированного b рассмотрим количества раз, которое встречались различные числа в массиве. Пусть число x встречалось $count[x]$ раз. Тогда, в b группах из различных чисел оно может встречаться не более, чем b раз (поскольку входит в каждую группу не более, чем один раз). Посчитаем сумму S по всем различным x-ам $\min(count[x], b)$. Это оценка сверху на максимальный суммарный размер групп, значит, max_b не больше, чем $\lfloor \frac{S}{b} \rfloor$. Эта оценка достигается, поскольку можно расставлять числа по группам так: выберем по $\min(count[x], b)$ копий числа x и упорядочим их так, чтобы одинаковые числа стояли подряд. Будем раскладывать числа в таком порядке по группам: одно число в первую группу, одно число во вторую группу ... одно число в b-ю группу, одно число в первую группу, одно во вторую ... и так далее. Тогда, так как все числа, равные x идут подотрезком длины не больше b, то они окажутся в различных группах. А из того, как мы раскладывали элементы, следует, что в нескольких первых группах окажется $\lfloor \frac{S}{b} \rfloor + 1$ элемент, а в остальных по $\lfloor \frac{S}{b} \rfloor$. Оставим только $\lfloor \frac{S}{b} \rfloor$ элементов — это и будут искомые группы.

Поэтому, для поиска max_b достаточно уметь считать сумму по различным x величин $\min(count[x], b)$, а это можно делать при помощи префиксных сумм и бинарных поисков, если отсортировать массив $count$, или же при помощи двух указателей. Потом из max_b легко построить ответ на задачу опять же с помощью двух указателей или бинарных поисков. Итоговое время $O(n \log n)$.

Задача N. Отрезки и окружность

Рассмотрим точку P вне круга радиуса r. Проведем из нее касательные к окружности. Эти касательные разбивают окружность на две дуги. Назовем одну из них видимой — ту, которая пересекается с отрезком, соединяющим центр окружности с точкой P (это и есть множество точек окружности, которые видно из точки P).

Утверждается, что отрезок, соединяющий точки P и Q, не пересекается с окружностью тогда и только тогда, когда соответствующие им видимые дуги пересекаются. Значит, задача состоит в том, чтобы посчитать количество пар пересекающихся дуг. Это можно сделать с помощью метода сканирующей прямой, двигая ее по окружности.