Задача A. Треугольники

Автор задачи: Даниил Орешников, разработчик: Константин Бац

Для того, чтобы числа x, y и z образовывали стороны треугольника, должно выполняться $x + y > z$, $x + z > y$ и $y + z > x$. Соответственно, если нам уже даны x и y, ограничения на z следующие: $z < x + y$ и $z > \max(x, y) - \min(x, y)$.

Соответственно, для решения **первой группы** просто переберем все числа до $2 \cdot \max(a)$, после чего для каждого из них переберем все возможные пары (a_i, a_j) и проверим, что эти три пары образуют треугольник. Это решение за время $\mathcal{O}(n^2 \max(a))$.

Соптимизировать напрямую это решение можно двумя способами. Например, можно перебрать x и a_i и заметить, что осталось проверить, правда ли, что все a_j лежат между $\max(x, a_i) - \min(x, a_i)$ и $a_i + x$. Если это верно для всех a_i, то такой x подходит, иначе — нет. Альтернативно, можно перебрать пару (a_i, a_j) и заметить, что она накладывает ограничение на x вида $\max(a_i, a_j) - \min(a_i, a_j) < x < a_i + a_j$, то есть задает определенный отрезок целых чисел, в котором x должно находиться. Пересечь n^2 таких отрезков можно за $\mathcal{O}(n^2)$: $[l_1, r_1] \cap [l_2, r_2] = [\max(l_1, l_2), \min(r_1, r_2)]$. Оба таких решения проходили **вторую группу**.

Третья группа решалась разбором случаев: можно рассмотреть случаи каждого подмножества набора $[1, 2, 3, 4]$, вошедшего в последовательность a_i, а можно было просто отталкиваться от минимума и максимума. **Четвертая группа** тоже была частным случаем, но в ней всегда ответ получался равным $2a_1 - 1$: все x от 1 до $2a_i - 1$ подходят.

Проанализировав какие-то простые примеры, подходящие под ограничения **пятой группы**, можно было быстро прийти к ее решению. Действительно, для пятой группы, если есть $a_1 = 1$, то вместе с любым a_j мы получаем ограничение, что $a_j - 1 < x < a_j + 1$, то есть $a_j = x$. Иными словами, если не все оставшиеся числа равны, то ответ будет 0, иначе — 1.

Ну и совместив все описанные выше идеи, можно было прийти к **полному решению**. Достаточно заметить, что при пересечении ограничений вида $x \in [\max(a_i, a_j) - \min(a_i, a_j) + 1), a_i + a_j - 1]$, максимальная левая граница получается из $a_n - a_1$, а минимальная правая — из $a_1 + a_2$, если a_i упорядочить по возрастанию.

Таким образом, ответ равен $(a_1 + a_2) - (a_n - a_1) - 1$, где a_1 и a_2 — два минимальных элемента, а a_n — максимальный. Найти их можно либо за линейный проход по массиву, либо с помощью сортировки. Разумеется, также нужно учесть, что итоговая разность может получиться отрицательной — это просто означает, что нижняя граница оказалась больше правой, и ответ равен 0.

Задача B. Взлом сейфа

Автор задачи и разработчик: Артем Васильев

Поэкспериментировав с разными строками, можно заметить, что ответ бывает «`NO`» только в том случае, когда из s нельзя получить t произвольной перестановкой букв, то есть количество вхождений какой-то буквы в s и t отличается. В ином случае всегда можно получить t из s, и это можно показать конструктивно.

Для **первой группы**, например, достаточно полным перебором попробовать все возможные действия. Поскольку полный перебор может быть бесконечным, имеет смысл сделать мемоизацию, то есть запоминать посещенные состояния, не посещая их заново. Таким образом, алгоритм будет похож на `dfs` по возможным состояниям строки, которых в этой группе не больше 8!.

В следующих группах уже более-менее необходимо переходить к конструктивному решению. Частая идея в таких задачах — строить ответ посимвольно, например, постепенно увеличивать суффикс, совпадающий с соответствующим суффиксом t. Если делать `bfs` вместо `dfs`, то получится за достаточно небольшое количество итераций находить последовательность действий, которая этот суффикс увеличивает, и если применить его $n - 1$ раз, s станет равна t, что при разной эффективности реализации проходит **вторую или третью группу**.

Далее приведем явное конструктивное решение для **пятой группы**. Заметим, что пара операций с параметрами $n - k$ и n разворачивает префикс длины k, оставляя остальную строку без изменений. Действительно, если $s = \alpha\beta$, где $|\beta| = n - k$, после первой операции строка равна $\beta^R\alpha$, а после

второй — $\alpha^R\beta$.

А используя такое действие («развернуть префикс»), можно за несколько операций увеличить суффикс, совпадающий с суффиксом t. Пусть сейчас у s и t совпадает префикс длины k, а очередной символ стоит на позиции i. Тогда сначала развернем префикс длины i, а затем развернем префикс длины $n-k$. Сначала нужный символ переместится в позицию 1, а затем — в $n-k$, делая суффиксы s и t длины $k+1$ равными.

Таким образом, за $4(n-1)$ операций можно из строки s получить строку t, потому что когда $n-1$ символ совпадет, оставшийся автоматически будет тоже совпадающим.

Различные оптимизации этого решения могли пройти **шестую группу**, но мы сразу приведем **полное решение**, позволяющее получить t за $\frac{5}{2}n$ операций. Для этого нужно научиться за пять операций увеличивать суффикс на два. Один из способов это сделать показан ниже (подчеркнута строка, которая выбирается в качестве β):

1. `...x<u>...abc</u>` → `cba......x`
2. `<u>cba......x</u>` → `x......abc`
3. `x......<u>abc</u>` → `cbax......`
4. `cbax...<u>y..</u>` → `..ycbax...`
5. `..ycbax<u>...</u>` → `.....ycbax`

Таким образом, из последовательности `abc` на конце строки, мы получили последовательность `ycbax`, на два символа длиннее. Стоит отметить, что здесь мы не «наращиваем» совпадающий суффикс влево, а наращиваем совпадающую с частью t строку одновременно влево и вправо.

Тогда можно либо начать с «центрального» символа t и «наращивать» общую строку в обе стороны, либо выбирать '`y`' так, чтобы влево рос суффикс строки t, а '`x`' — так, чтобы вправо рос префикс. В таком случае в конце получится циклический сдвиг t, который можно за еще за три операции преобразовать в t.

Задача C. Шестиугольный рисунок

Автор задачи: Даниил Орешников, разработчики: Даниил Орешников и Константин Бац

В **первой группе тестов** в данной задаче достаточно научиться определять, являются ли три данные ячейки соседними или нет. Это легко сделать, посмотрев на координаты клетки: у клетки с координатами (x, y) соседи имеют коодинаты $(x, y \pm 1)$, $(x+1, y+[-1, 0])$ и $(x-1, y+[0, 1])$. Для каждой клетки проверим, что обе из оставшихся являются ее соседями, тогда любую из них надо удалить, иначе можно не удалять ничего.

Разумеется, есть более простой способ: для любой точки на решетке три соседние клетки имеют ровно две с одинаковым x, ровно две с одинаковым y и ровно две с одинаковым $x+y$. Действительно, $x = \text{const}$, $y = \text{const}$ и $x + y = \text{const}$ задают три типа множеств ячеек, расположенных последовательно «в прямую».

Вторая группа расчитана на полный перебор. Можно перебрать все подмножества ячеек, и для каждого, пользуясь решеним первой группы, проверить, что в нем нет треугольных скоплений. Среди всех таких множеств останется выбрать максимальное, и получится минимальное число удаленных клеток.

Особый вид множества ячеек в **третьей группе** при более внимательном взгляде позволяет понять, что это множество «плотное», то есть без «тонких переходов», которые в графе связей были бы мостами. В таком множестве может прийти идея, близкая к одной половине полного решения: ячейки такой сетки можно правильно раскрасить в три цвета, где цвет ячейки (x, y) можно определить по $(x-y) \bmod 3$: эта величина различна для любых трех попарно соседних клеток.

Поскольку это правильная раскраска, то при удалении всех ячеек одного из цветов, треугольных скоплений не останется. А на множествах из третьей группы такое решение показывает себя достаточно хорошо, хоть и не всегда абсолютно эффективно.

В **четвертой группе** дано ограничение, что у каждой ячейки степень (количество соседей) не больше 3. Если степень какой-то ячейки меньше 2, ее можно убрать из рассмотрения: она не является частью никакого треугольного скопления. Ячейки со степенями 2 и 3, не являющиеся частями никаких треугольных скоплений, аналогично, можно убрать. А среди оставшихся ячеек можно показать, что размеры компонент двусвязности не превосходят небольшой константы, и в каждой такой компоненте можно найти ответ независимо одним из предыдущих решений.

Полное решение же получается как комбинация решения с цветами и жадного алгоритма удаления. Для решения с цветами выделим компоненты двувязности в графе связности ячеек (выделим и удалим мосты и после выделим компоненты связности), после чего в каждой из компонент удалим множество минимально представленного цвета. Жадный алгоритм убирает из рассмотрения ячейки, которые не являются частями треугольных скоплений, после чего сортирует ячейки по убыванию степени и удаляет в таком порядке, если удаление очередной ячейки уменьшает количество треугольных скоплений.

Задача D. Путешествие миньонов

Автор задачи и разработчик: Дмитрий Грунтов

Эту задачу можно воспринимать как упражнение на оптимизации динамического программирования, в частности, `convex hull trick`.

Во всех группах работает некоторая версия динамического программирования $\texttt{dp}[i]$ — максимальное удовольствие при посещении городов от 1 до i при условии, что город i посещается на день. Тогда можно перебрать город j, который был посещен на день перед i, и обновить $\texttt{dp}[i]$ через $\texttt{dp}[j] + (a_{j+1} - c)^2 + \ldots + (a_{i-1} - c)^2 + (a_i - a_j)^2$.

В **первой группе** $n \leqslant 500$, а в таком случае можно вычислять эту сумму напрямую, что дает решение за $\mathcal{O}(n^3)$. Если же заметить, что здесь большая часть слагаемых образует сумму $(a_k - c)^2$ на отрезке $j + 1 \leqslant k \leqslant i - 1$, то можно посчитать префиксные суммы на таких величинах, и делать пересчет за $\mathcal{O}(1)$, что даст общее время решения $\mathcal{O}(n^2)$.

Полное решение основывается на том, что

$$\texttt{dp}[j]+\left(\sum_{k=j+1}^{i-1}(a_k-c)^2\right)+(a_i-a_j)^2=\left[\texttt{dp}[j]+\left(\sum_{k=j+1}^{n}(a_k-c)^2\right)+a_j^2\right]+\left(a_i^2-\left(\sum_{k=i}^{n}(a_k-c)^2\right)\right)-2a_ia_j.$$

Здесь выделена часть, которая зависит только от j, $\texttt{value}(j)$, и часть, которая зависит тольк от i, $\texttt{delta}(i)$. Тогда

$$\texttt{dp}[i] = \min_{j<i}(\texttt{value}(j) - 2a_ia_j) + \texttt{delta}(i),$$

а это равносильно поиску минимального значения k_jx+b_j в точке $x = a_i$ по всем $j < i$ при $k_j = -2a_j$ и $b_j = \texttt{value}(j)$.

Можно поддерживать множество соответствующих прямых и искать такой минимум за $\mathcal{O}(\log n)$ с помощью дерева Ли Чао в **четвертой группе** или при помощи Convex Hull Trick в **третьей группе**.