Задача A. Tribute

Отсортируем все суммы подмножеств.

Пусть исходный массив был $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_n$.

Посмотрим на наименьший элемент. Так как все числа положительные, то этот элемент — сумма подмножства размера 1, причем минимального из чисел в массиве — a_1. Отлично, удалим его. Посмотрим на новый минимум. Это обязательно второй по минимальности элемент — a_2. Теперь мы знаем два минимальных элемента. Удалим из множества значение $a_1 + a_2$. Тогда минимальный элемент из оставшихся — a_3. Удалим все значения $a_3 + a_1$, $a_3 + a_2$, $a_3 + a_1 + a_2$. Теперь минимальный элемент из оставшихся — a_4. Удалим все суммы $a_4 + a_1$, $a_4 + a_2$, $a_4 + a_3$, $a_4 + a_1 + a_2$, $a_4 + a_1 + a_3$, $a_4 + a_2 + a_3$, $a_4 + a_1 + a_2 + a_3$. Теперь минимальный элемент во множестве — a_5. Будем повторять процесс, пока не удалим все элементы.

Код: `https://ideone.com/JS3sbL`

Задача B. A Good Problem

Наверное, вы не ждали, но решение на 100 баллов использует разделяй и властвуй `:)`.

Разделяй и властвуй в этой задаче будет немного не такое, как в предыдущих задачах. Более простая задача на такой разделяй и властвуй: `https://codeforces.com/problemset/problem/549/F`

В чем суть. Возьмем максимальный элемент в массиве, пусть он находится на позиции m. Тогда элемент с индексом m является максимумом для всех отрезков массива $[l..r]$, таких, что $1 \leqslant l \leqslant m \leqslant r \leqslant n$. На каждом таком отрезке значение максимума равно A_m. Тогда сумма $MAX(l, r) \cdot F(l, r)$ по всем этим отрезкам равна $A_m \cdot k$, где k — количество пар индексов (l, r), таких, что:

- $1 \leqslant l \leqslant m$
- $m \leqslant r \leqslant n$
- $(A_l \texttt{ and } A_r) = A_l$ (или наоборот).

Давайте для удобства обозначим за $F(l_1, r_1, l_2, r_2)$ количество таких пар (i, j), что $l_1 \leqslant i \leqslant r_1, l_2 \leqslant j \leqslant r_2$ и $(A_i \texttt{ and } A_j) = A_j$ или $(A_i \texttt{ and } A_j) = A_i$. В частности, значение k выше равно $k = F(1, m, m, n)$.

Теперь рекурсивно разделимся на два массива — на подотрезки $[1..m-1]$ и $[m+1..n]$. В них нужно будет сделать то же самое. Для примера, возьмем левую часть. Пусть m_2 — максимум на отрезке $[1..m-1]$. Тогда m_2 увеличивает ответ на $A_{m_2} \cdot F(1, m_2, m_2, m)$. Аналогично для правой части и рекурсивно дальше.

Итого у нас получится ровно n запросов вида $F(l_1, r_1, l_2, r_2)$. Их ровно n потому что в ходе рекурсии каждый элемент побывает максимумом ровно один раз.

Введем еще обозначение: $G(x, l, r)$ будет означать количество таких позиций i, что $l \leqslant i \leqslant r$ и $(A_i \texttt{ and } x) = x$ или $(A_i \texttt{ and } x) = A_i$. Обратите внимание, что в отличие от F, x означает не индекс элемента массива, а непосредственно его значение.

На запросы, очевидно, можно отвечать в оффлайне, ведь мы сами их создали `:)`.

Тогда нетрудно представить F следующим образом:

$$F(l_1, r_1, l_2, r_2) = \sum_{i=l_1}^{r_1} G(A_i, l_2, r_2).$$

Или наоборот:

$$F(l_1, r_1, l_2, r_2) = \sum_{j=l_2}^{r_2} G(A_j, l_1, r_1)$$

Таким образом, значение $F(l_1, r_1, l_2, r_2)$ можно представить в помощью $r_1 - l_1 + 1$ значений функции G. Или с помощью $r_2 - l_2 + 1$ значений. Здравый смысл подсказывает, что лучше представлять меньшим количеством `:)`. Оказывается, что если представить все n запросов F как запросами G, причем меньшей частью, то на самом деле суммарно получится не более $n \log n$ запросов G! Это так

примерно по тому же, почему суммарно получается $n \log n$ в обычном разделяй и властвуй. Давайте оценим, сколько каждый элемент A_i ($1 \leqslant i \leqslant n$) мог быть в роли x из G. Если A_i участвовал в G, то это значит, что он был в меньшей по длине части отрезка $[l_1..r_2]$ в вычислении $F(l_1, r_1, l_2, r_2)$. Очевидно, что меньшая часть (к примеру $[l_1..r_1]$) меньше всего отрезка $[l_1..r_2]$ хотя бы в два раза, на то она и меньшая. Более того, все отрезки-запросы в F либо не пересекаются вообще, либо вкладываются друг в друга. Поэтому если A_i был участником каких-то, пусть k, запросов вида $G(A_i, l, r)$, то каждая пара из этих k отрезков отличается по длине хотя бы в два раза. Следовательно, $k \leqslant \log_2 n$.

С разделяй и властвуй все. Можно почитать разбор задачи выше с КФа, там наверное понятнее будет, почему отрезков получится $O(n \log n)$.

Теперь переходим ко второй части задачи - ответы на запросы $G(x, l, r)$. Пусть их q ($q = O(n \log n)$). Мы больше не вспоминаем исходную задачу. У нас есть новая задача: есть массив длины n и q запросов:

- x l r — найти $G(x, l, r)$ — количество таких позиций i, что $l \leqslant i \leqslant r$ и (A_i `and` x) $= x$ или (A_i `and` x) $= A_i$.

Здесь воспользуемся идеей префиксных сумм. $G(x, l, r) = G(x, r) - G(x, l-1)$, где $G(x, i) = G(x, 1, i)$.

Отлично, теперь все запросы на префиксе массива. Так как мы можем отвечать в оффлайне, для каждого i запомним все x из запросов $G(x, i)$.

Идем по порядку (сканлайном) по массиву. Нужно уметь обрабатывать события:

- добавить элемент массива A_i
- ответить на все $G(x, i)$
- переместиться к $i + 1$

Чтобы обрабатывать эти события, воспользуемся идеями корневой декомпозиции! Давайте разобьем массив (он у нас длины n) на блоки размером b как в корневой. Будем выполнять все операции «блок за блоком». Когда мы отвечаем на запрос $G(x, i)$, все индексы j ($1 \leqslant j \leqslant i$, которые могли увеличить G на 1, делятся на два типа: те, которые лежат в том же блоке, что и i, или находятся в каком-то предыдущем блоке. Индексов, которые лежат в том же блоке, не больше, чем размер блока, то есть b. Их обработаем за $O(b)$ влоб — просто пробежимся по элементом из этого же блока и проверим за $O(1)$ иф на то, что одно число является подмаской другой. Числа из других блока будем обрабатывать целиком всем блоком. Когда мы переходим из одного блока в следующий, посчитаем как в задаче «Конкурс» областной олимпиады 2012 для каждого x количество его подмасок и надмасок среди чисел всех уже обработанных блоков за $O(3^{14}$ или $O(2^{14} \cdot 14)$ с помощью ДП, которое рассказывали на сборах. Такую операцию нужно будет производить каждый раз, когда меняется блок, то есть всего $\frac{n}{b}$ раз. Тогда каждый из q запросов отвечается за $O(b)$ для элементов из текущего блока + $O(1)$ для элементов из предыдущих блоков, и дополнительно придется $O(\frac{n}{b})$ раз считать все под(над)маски. Идейно это все.

Давайте поймем, какое должно быть b. Предположим, что мы считаем подмаски за $O(2^{14} \cdot 14)$ (в задаче заходит это делать и за $O(3^{14})$). Тогда суммарно обработка всех запросов занимает $O\left(\frac{n}{b} \cdot 2^{14} \cdot 14 + qb\right)$. Чтобы минимизировать это значение, нужно приравнять слагаемые:

$$\frac{n}{b} \cdot 2^{14} \cdot 14 = qb$$
$$\frac{n}{q} \cdot 2^{14} \cdot 14 = b^2$$

Вспоминаем, что мы изначально рещали другую задачу и $q = n \log n$:

$$\frac{n}{n \log n} \cdot 2^{14} \cdot 14 = b^2$$
$$\frac{n}{n \log n} \cdot 2^{14} \cdot 14 = b^2$$

$$b = \sqrt{\frac{2^{14} \cdot 14}{\log n}}$$

Итого решение за

$$O\left(\frac{n}{b} \cdot 2^{14} \cdot 14 + qb\right)$$

$$O\left(\frac{n}{\sqrt{\frac{2^{14} \cdot 14}{\log n}}} \cdot 2^{14} \cdot 14 + q\sqrt{\frac{2^{14} \cdot 14}{\log n}}\right) =$$

$$O\left(\frac{n\sqrt{\log n}}{\sqrt{2^{14} \cdot 14}} \cdot 2^{14} \cdot 14 + n \log n\sqrt{\frac{2^{14} \cdot 14}{\log n}}\right) =$$

$$O\left(n\sqrt{2^{14} \cdot 14 \cdot \log n} + n\sqrt{2^{14} \cdot 14 \cdot \log n}\right) =$$

$$O\left(n\sqrt{2^{14} \cdot 14 \cdot \log n}\right)$$

На практике b проще выбрать константой, не зависящей от n. Код! `https://ideone.com/rxhFPp` (сорри, он без пробелов).

Пока писал разбор, вспомнил, что есть авторский на английском. Не уверен, что там такое же решение `https://discuss.codechef.com/t/goodprob-editorial/12980`

Задача C. Родственные языки

- Решение первой подзадачи.

  $dist(stringA, stringB)$ - количество символов в которых строки A и B различаются.

  $S_{l...r}$ - подстрока строки S, которая начинается в l, и заканчивается в r.

  Тогда просто в явном виде переберём все возможные пары подстрок и проверим то, что $dist(substringA, substringB) <= k$

  Сложность данного решения: $O(n^5)$.

- Решение второй подзадачи.

  Давайте применим метод Динамического программирования, а именно: $dp_{n,m,k}$ - такое наибольшее L, что $\text{dist}(A_{n...n+L-1}, B_{m...m+L-1}) = k$

  Как же это считать? Да очень просто!

  База: $dp_{n,m,dist(A_{n...n},B_{m...m})} = 1$

  Есть всего 2 случая:

  1. Если $A[n] = B[m]$ то $dp_{n,m,k} = dp_{n+1,m+1,k} + 1$
  2. Если $A[n] \neq B[m]$ то $dp_{n,m,k} = dp_{n+1,m+1,k-1} + 1$

  Всего получается $n \cdot m \cdot k$ состояний, каждый переход считается за $O(1)$. Из этого получается что сложность данного решения $O(n \cdot m \cdot k)$

«53 балла хорошо, а 100 ещё лучше»

©Я

- Решение последней подзадачи.

  Чтобы получить заветные 100 баллов, нужно модернизировать наше Динамическое программирование. Пускай теперь $dp_{n,m}$ - такое наибольшее L, что $dist(A_{n...n+L-1}, B_{m...m+L-1}) <= k$. Также заведём массив $count_{n,m}$ - количество позиций в которых подстроки $A_{n...n+dp_{n,m}-1}$ $B_{m...m+dp_{n,m}-1}$ различаются. Зная эту информацию можно пересчитывать состояния.

  1. Если $A[n] = B[m]$ то $dp_{n,m} = dp_{n+1,m+1}+1$, $count_{n,m} = count_{n+1,m+1}$
  2. Если $A[n] \neq B[m]$ то $dp_{n,m} = dp_{n+1,m+1}+1$ и $count_{n,m} = count_{n+1,m+1}+1$. Но могла возникнуть такая ситуация что $count_{n,m} > k$, а если быть точнее то $count_{n,m} = k+1$. В этой ситуации давайте просто уменьшать $dp_{n,m}$ пока $dist(A_{n...n+dp_{n,m}-1}, B_{m...m+dp_{n,m}-1}) > k$

  Легко видеть, что данное решение работает за $O(nm)$, по той же причине, почему работают два указателя на массиве за $O(n)$.