

# MAC0345 - Desafios 2

## Editorial Lista 3

Nathan Luiz e Enrique Junchaya

May 11, 2023

### A. Distinct Values Queries .

*Tópicos: MO.*

Este problema já aprendemos a resolver com segtree persistente de maneira online. Com MO, só podemos resolver offline.

A ideia é que temos que ter uma estrutura que representa um conjunto de números e que é possível adicionar/remover um elemento no conjunto e também saber quantos elementos distintos o conjunto tem. Primeiramente, podemos fazer compressão de coordenadas. Depois, basta manter um vetor de frequência e uma variável `cur` que representa o número atual de elementos distintos. Sempre que adicionamos um elemento no conjunto e a frequência anterior dele era 0, então incrementamos `cur`. Se removemos um elemento e a frequência passou a ser 0, então decrementamos `cur`.

Complexidade final  $O((n + q)\sqrt{n})$ .

### B. XOR and Favorite Number .

*Tópicos: MO.*

Vamos fazer uma redução desse problema. Fazendo *xor-sum* de prefixo do vetor, ou seja,  $pref_i$  é o xor de todos os elementos do prefixo  $i$ , então para achar o *xor* do range de  $l$  até  $r$  basta pegar  $pref_{l-1} \oplus pref_r$ . Portanto, para cada query  $l_i, r_i$ , queremos achar o número de pares  $i, j$  tal que  $pref_i \oplus pref_j = k$ ,  $i, j \in [l_i - 1, r_i]$  e  $i \leq j$ . Portanto, nossas queries agora serão até  $l_i - 1$ .

Como o problema anterior, podemos usar um vetor de frequência e uma variável `cur` com a resposta atual. Então, para ir mantendo a resposta atual atualizada, sempre que adicionamos um elemento  $x$  no conjunto, somamos  $freq[k \oplus x]$  na resposta. Analogamente quando retiramos. Isso funciona por causa das operações de *xor*.

$$x \oplus y = k \iff y = k \oplus x$$

Só precisa ter cuidado com o caso  $k = 0$ . Complexidade final  $O((n + q)\sqrt{n})$ .

### C. Little Elephant and Array .

*Tópicos: MO.*

Essa questão vai usar o mesmo truque dos problemas anteriores. Só precisamos ter um vetor de frequência e salvar a resposta atual. Para não ter problema de acesso a posição inválida do array de frequência, podemos fazer cada elemento  $a_i = \min(a_i, n + 1)$ .

Complexidade final  $O((n + q)\sqrt{n})$ .

### D. Enumerate Triangles .

*Tópicos: SQRT.*

Esse problema é bem parecido com o de contar o número de triângulos. A ideia é separar os vértices de pesados e leves. Os vértices pesados são aqueles que tem grau alto e os leves os que não são pesados. Podemos definir o limite próximo a  $\sqrt{m}$ . Sendo  $d_u$  o grau de um vértice, então como  $\sum_{u \in G} d_u = 2 \cdot m$ , o número de vértices pesados não pode ser muito grande.

Portanto, vamos iterar por cada aresta e verificar o tipo dela.

- Se ela é pesado-pesado:

Iteramos por todos os vértices pesados e vemos se eles são adjacentes aos vértices dessa aresta. Se for, adicionamos na resposta. Esse caso 3 vezes o número de triângulos pesado-pesado-pesado.

- Se ela é pesado-leve:

Novamente iteramos pelos vértices pesados e adicionamos na resposta. Esse conta 2 vezes os triângulos pesado-pesado-leve.

- Se ela é leve-leve:

Como os vértices leves tem no máximo  $\sqrt{m}$  arestas, então podemos iterar pelos adjacentes dos 2 vértices e ver quais são em comum. Se o vértice em comum for pesado, contamos uma vez os triângulos leve-leve-pesado e caso contrário 3 vezes os triângulos leve-leve-leve.

A complexidade final é  $O(\sqrt{m} \cdot m)$  ou  $O(\sqrt{m} \cdot \log n \cdot m)$ , dependendo da implementação. É importante notar que o número total de triângulos é  $O(m)$ , já que estamos iterando por todos eles pelo menos uma vez.

## E. Holes .

*Tópicos: SQRT.*

Vamos dividir o array em buckets de tamanho *magic*. Para cada vértice, vamos salvar uma aresta que vai dele até o próximo elemento em um bucket diferente e com peso o número de pulos necessários para chegar nesse bucket. Dessa forma, as queries serão feitas simulando, já que iteramos por cada bucket ( $n/magic$ ) no máximo uma vez. Um detalhe de implementação é que quando chegar no último bucket antes de sair do array, podemos ir pulando de um em um para achar a resposta. Com isso fazemos no máximo *magic* operações.

Para o update, temos que atualizar as arestas somente dos elementos que estão no mesmo bucket que  $x$  e que vem antes de  $x$ . Portanto, iteramos pelos elementos de frente para trás fazendo o update de cada aresta em  $O(1)$  (tente pensar como, algo parecido com dp). Como o tamanho do bucket é *magic*, o update fica  $O(magic)$ .

Portanto, a complexidade final é  $O(q \cdot (magic + \frac{n}{magic}))$ .

## F. Tear It Apart .

*Tópicos: Ad-hoc, Guloso.*

Olhar o [editorial](#).