MO’nun Algoritması

Offline sorgular için çalısan $O(\sqrt{N})$ ’lik bir algoritmadır. Şimdi algoritmanın çalışmasını anlayabilmek için bir örnekle birlikte inceleyelim. Çalışma prensibini ve uygulamasını adım adım anlatalım:

1) Problemin tanımlanması
2) Problem için $O(N^2)$ lik basti bir çözüm bulunması
3) Bulduğumuz algoritmanın düzenlenmesi
4) Yukarıdaki problem için çalışan bir algoritmanın buluması ve doğruluğunun ispatlanması
5) Bulunan son çözüm için çalışma zamanının ispatlanması $O(N\sqrt{N})$.
6) Nerede ne zaman kullanılabileceğinin açıklanması
7) Örnek problemler

Problemin tanımı

N elemanlı bir dizi verilmektedir. Bu dizinin bütün elemanları 1 ile N arasındadır. Sizden M adet sorguya cevap vermeniz istenmektedir.Her sorgu L,R şeklindedir ve sizden [L,R] aralığındaki en az üç defa tekrar eden sayıları bulmanız istenmektedir.

Örnek: Dizi şu şekilde olsun: `{1, 2, 3, 1, 1, 2, 1, 2, 3, 1}` (0 indisli olarak tanımlanmıştır).

Sorgu: L=0,R=4. Cevap = 1. Bu aralıkta sadece 1 sayısı 3 defa geçmektedir.
Sorgu: L=1,R=8. Cevap = 2. Bu aralıkta 2 ve 1 sayıları 3’er defa geçmektedir.

Problem için $O(N^2)$ lik basti bir çözüm bulunması

Her sorgu için, L ve R aralığındaki tüm sayıları tek tek inceleyelim ve geçme sıklıklarına bakarak 3 veya daha fazla sayıda geçenleri çözüm olarak belirleyelim. M=N şeklinde kabul edersek algoritmamızın çalışma zamanı $O(N^2)$ olacaktır.

```
for each query:
  answer = 0
  count[] = 0
  for i in {l..r}:
    count[array[i]]++
    if count[array[i]] == 3:
      answer++
```

Bulduğumuz algoritmanın düzenlenmesi

```
add(position):
  count[array[position]]++
  if count[array[position]] == 3:
    answer++

remove(position):
  count[array[position]]--
  if count[array[position]] == 2:
    answer--

currentL = 0
currentR = 0
answer = 0
count[] = 0
for each query:
  // currentL should go to L, currentR should go to R
  while currentL < L:
    remove(currentL)
    currentL++
  while currentL > L:
    add(currentL)
    currentL--
  while currentR < R:
    add(currentR)
    currentR++
  while currentR > R:
    remove(currentR)
    currentR--
  output answer
```

İlk olarak biz L,R arasındaki bütün değerleri tek tek gezerek yine sayma işlemini gerçekleştireceğiz ancak bu sefer diğer sorguları kendinden öncekilere bakarak ayarlayacağız.

Eğer önceki girdi L=3 ve R=10 sa ve biz currentL=3 ve currentR=10 a sahip olacağız demektir belirtilen sorgunun sonunda. Eğer bir sonraki sorgu L=5, R=7 ise bu sefer currentL değerini 3 ten 5 e taşıyacağız ve currentR değerini de 10 dan 7 te taşıyacağız.

Add fonksiyonu: bu fonksiyon ile eklemek istediğimiz değerleri saklamak istediğimiz kümemize ekleyeceğiz. Bu şekilde de cevabımızı güncelleyeceğiz.

Remove fonksiyonu: bu fonksiyon ile saklamak istediğimiz elemanlar kümesinden sorgular arasındaki geçişlerde çıkması gereken değerleri siliyoruz.

Yukarıdaki problem için çalışan bir algoritmanın bulunması ve doğruluğunun ispatlanması

Mo'nun algoritması verilen sorguları belirli bir sıraya sokarak çalışmaktadır. Bize verilen M sorgu için sorguları belirli bir sıraya sokacağız ve ardından sorguları hesaplamaya çalışacağız. Ve açıkca görüldüğü gibi offline bir algoritmadır. Tüm sorgular L,R şeklinde temsil edilmektedir ve biz onları başlangıç ve bitiş olarak değerlendireceğiz. Şimdi verilen N diziyi $\sqrt{N}$ gruba ayıracağız. Her bir grup $N/\sqrt{N} = \sqrt{N}$ uzunluğunda olacaktır. Her başlangıç bu gruplardan birine yerleşecektir. Aynı şekilde her bitiş de bu gruplardan birine yerleşecektir.

Bir sorgu eğer başlangıcı P. grubun içerisindeyse P. gruba ait olacaktır. Bu algoritmada ilk olarak birinci grubun sorgularını cevaplandıracağız ardından 2. grubun sorgularını ve bu şekilde tüm grupların sorgularını cevaplayacağız. Biz neredeyse sorguları sıraladık. Bulundukları gruplar açısından incelersek her ne kadar aynı grupta olan sorgular da olsa sorgular başlangıçlarına göre sıralı sayılmaktadır.

Şimdi buradan itibaren $\sqrt{N}$ grupla ilgilenmek yerine bir grup içerisindeki sorguları nasıl yapacağımız üzerine yoğunlaşacağız. Ardından geliştireceğimiz algoritmayı tüm gruplar üzerinde çalışarak bütün sorguları cevaplayacağız. Belirlediğimiz grup içinde bulunan bütün sorguların başlangıçları aynı grubun içinde olacaktır ve bitişleri herhangi bir grubun içinde olabilir. Şimdi gruptaki sorguları R değerlerine göre artan sırada olacak şekilde sıralayalım.

Son durumda görüntü nasıl olacaktır?
Tüm sorgular kendi grup numaralarına göre artan sırada sıralanmışlardır. Aynı grupta olanlar ise R değerlerine göre kendi içinde artan sırada sıralıdır. Örneğin 3 büyüklüğündeki 3 gruptan oluşan bir dizimizin olduğunu düşünelim ve aşağıdaki örneği inceleyelim:

```
{0, 3} {1, 7} {2, 8} {7, 8} {4, 8} {4, 4} {1, 2}
```

Şimdi onları grup sıralarına göre sıralayalım:

```
{0, 3} {1, 7} {2, 8} {1, 2} {4, 8} {4, 4} {7, 8}
```

Şimdi de aynı grupta olanları R değerlerine göre sıralayalım:

```
{1, 2} {0, 3} {1, 7} {2, 8} {4, 4} {4, 8} {7, 8}
```

Şimdi üçüncü bölümde geliştirdiğimiz ve kodunu verdiğimiz algoritmayı burada kullanalım. Yukarıda verdiğimiz algoritma doğrudur ve herhangi başka bir düzeltme gerektirmemektedir.

Bulunan son çözüm için çalışma zamanının ispatlanması - $O(N\sqrt{N})$.

Ve geldik şimdi bütün bu anlatının en güzel bölümüne: çalışma zamanı analizi. Az önce yukarıda belittiğimiz ve $O(N^2)$ de çalışan algoritmamız belirlediğimiz şekilde soguların sıralanmasıyla $O(N\sqrt{N})$ de çalışmaktadır. Şimdi bunun nasıl böyle olduğunu inceleyelim.

Yukarıda yazdığımız koda tekrar bir bakalım. Total çalışma zamanını bulmak için 4 while döngüsünün ne kadar tekrar edeceğini hesaplamamız gerekmektedir. İlk iki while döngüsü başlangıçlar için gezen işaretçinin hamle sayısı kadar gezecektir. Ve aynı şekilde son iki while döngüsü de bitişler için gezen işaretçinin hamle sayısı kadar gezecektir. Bu iki hamle sayısının toplamı bize genel çalışma zamanını verecektir.

İlk olarak bitişler için gezen işaretçinin yapacağı hamle sayısını inceleyelim. Her bir gruptaki sorgular kendi içinde artan sırada olduğu için currentR değeri hep artarak ilerleyecektir. Bu da demektir ki en fazla $O(N)$ işlem yapacaktır. Ve ikinci grubun başı için ise currentR değeri en kötü durumda en başa dönebilir ve bu da yine $O(N)$ işlem sürecektir. Bunlardan çıkaracağımız sonuçları incelersek $\sqrt{N}$ grup için toplamda $O(N\sqrt{N})$ işlem yapacağız demektir.

Şimdi de başlangıçlar için gezecek olan işaretçinin kaç işlem yapacağına bakalım. Her bir grup için grup içindeki sorguların hepsinin başlangıç değeri yine grup içerisinde olacak şekilde ayarlamıştık. İki sorgu arasındaki geçiş için başlangıç işaretçisi yine grup içerisinde kalacak şekilde hareket etmektedir. Bu da demektir ki en fazla $\sqrt{N}$ işlem yapacaktır. Her bir grup için grup içerisindeki toplam işlem sayısı $O(\ Q\sqrt{N})$ olacaktır. Ve biz biliyoruz ki Q değerlerinin toplamı M olacaktır. Yani total çalışma zamanımız $O(M\sqrt{N})$ olacaktır.

Nerede ne zaman kullanılabileceğinin açıklanması

Daha önce de bahsettiğimiz gibi bu algoritma offline bir algoritmadır. Bu da demek oluyor ki bizden sorguları verildiği anda uygulama ya da cevaplama imkanı vermiyor. Bunlarla da kalmayıp bizden istenen sorgular için yukarıda tanımladığımız şekilde bir ekleme ve çıkarma işlemi tanımlamamız gerekmektedir. Ayrıca birçok durum için eklem işlemi kolay olurken çıkarma işlemi zor olabilmektedir. Buna örnek olarak, bir aralığın maksimum değerini almak verilebilir. Ekleme sırasında hala maksimum elemanı saklayabiliriz ancak elemanı silmek sandığımız kadar kolay değildir. Bu durumda ekleme ve çıkarma işlemlerini *set* veri yapısını kullanarak yapabiliriz ve bu da bizi bu işlemleri $O(logN)$ çalışma zamanında yapmak zorunda bırakır. Bu nedenle algoritmanın çalışma zamanı $O(N * \sqrt{N} * logN)$ olacaktır.

Bu algoritmayı kullanabileceğimiz bir çok yer mevcuttur. Ayrıca bazı problemler için farklı veri yapılarını da kullanma ihtimalimiz mevcuttur. Buna örnek olarak segment tree algoritmasını verebiliriz. Ancak bazı problemler de vardır ki Mo'nun algoriması kullanılmak zorundadır. Şimdi onlardan birkaçını konuşalım.

Örnek problemler

DQUERY – SPOJ
Powerful array – CF Div1 D
GERALD07 – Codechef
GERALD3 – Codechef
Tree and Queries – CF Div1 D
Sherlock and Inversions – Codechef
Jeff and Removing Periods – CF Div1 D