Lineer Zamanda Eratosthenes Kalburu

Size bir n sayısı veriliyor. Sizden $[2; n]$ aralığındaki tüm asal sayıları bulmanız isteniyor.

Bu problemin klasik çözümü Eratosthenes Kalburu'dur. Algoritma çok basittir ve $O(n \log \log n)$ maliyette çalışır. Bu algoritmanın güzel bir yanı da $[2; n]$ arasındaki sayıların tüm çarpanlarını bulmamıza yarıyor.

Klasik Eratosthenes kalburunun kullandığı hafıza n (32bit integer) dir.

Bu algoritma 10^7 kadar olan sayılar için kullanılabilir.

Biz daha iyileştirilmiş ve maliyeti $O(N)$ olan bir Eratosthenes Kalburu anlatacağız. Klasik Eratosthenes Kalburunun kodunu da bu bölümde bulabilirsiniz.

Algoritmanın Tanımı

Bizim hedefimiz her i öyle ki $[2; n]$ aralığındaki sayılar için $lp[i]$ (i'nin en küçük asal çarpanını) bulmaktır**.**

Ek olarak tüm bulduğumuz asal sayıları $pr[]$ dizisinde tutalım.

Başta tüm $lp[i]$ değerleri 0 olsun. Algoritma sonuçlandığında dizimiz doğru değerleri tutuyor olacak.

Tüm i değerlerini küçükten büyüğe gezelim.

Bir i değeri için durumları inceleyelim.

Eğer $lp[i] = 0$: Bu durumda i sayısı için kendinden küçük asal çarpan bulunamamıştır. O zaman i sayısı asal sayı olur; $lp[i] = i$ olur ve i sayısını $pr[]$ dizisine ekleriz.

$lp[i] \neq 0$: Bu durumda i sayısının en küçük asal çarpanını bulmuş oluruz. i asal sayı değildir.

Her iki durumda da $lp[i]$ den küçük eşit olan tüm p_j asal sayılarını geziyoruz. $x_j = i * p_j$ şartının sağlayan tüm x_j sayılarının $lp[x_j]$ değerleri p_j sayısına eşit olur. Çünkü $lp[p_j] = p_j \leq lp_i$ olduğunu kabul ettik. Bu durumda $lp[x_j] = min(lp[p_j], lp[i]) = lp[p_j] = p_j$ olur. İspatı aşağıda daha ayrıntılı olarak verilmiştir.

Algoritmanın Uygulanması

Belli bir N sayısı için için Algoritmaları çalıştıracağız.

Klasik Eratosthenes Kalburu

```
int n;
vector<char> prime (n+1, true);
prime[0] = prime[1] = false;
for (int i=2; i<=n; ++i)
	if (prime[i])
		if (i * 1ll * i <= n)
			for (int j=i*i; j<=n; j+=i)
				prime[j] = false;
```

İyileştirilmiş Eratosthenes Kalburu

```
const int N = 10000000;
int lp[N+1];
vector<int> pr;

for (int i=2; i<=N; ++i) {
	if (lp[i] == 0) {
		lp[i] = i;
		pr.push_back (i);
	}
	for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[j]<=N; ++j)
		lp[i * pr[j]] = pr[j];
}
```

Algoritmanın Doğruluğunun İspatı

Kafaya takılan bir kaç nokta var.

* Bir x sayısının $lp[x]$ değerini neden sadece bir kere atarız ve bu değer neden doğrudur?
* Algoritmanın maliyeti neden $O(N)$ dir?

Bir x sayısının değerini sadece $i = x / lp[x]$ iken atarız. Çünkü $lp[x]$ değerinden büyük bir p_j değeri atayabilmemiz için $i_2 = x / p_j$ olmalı. Bu durumda x sayısının p_j den küçük bir $lp[x]$ asal çarpanı vardır. Bu $lp[x]$ çarpanı p_j asal sayısının çarpanı olamayacağı için i_2 sayısının asal çarpanıdır. Öyleyse $lp[x] < p_j$ ve $lp[i_2] < p_j$ olduğundan i_2 sayısı için $lp[i_2]$ den küçük eşit asal sayıları gezerken p_j asal sayısıyla karşılaşmayız.

Gelelim maliyetine. Biz her elemanın en küçük asal çarpanın bir kere hesaplıyoruz. Ve hiçbir boş işlem yok. Öyleyse maliyet $O(N)$ dir.

Zaman ve Gerekli Hafıza

Klasik Eratosthenes Kalburunda hafıza karmaşıklığı $O(N)$ fakat zaman karmaşıklığı $O(n \log \log n)$ dir.

İyileştirilmiş kalburda ise zaman karmaşıklığı $O(N)$ olur fakat hafıza karmaşıklığı $O(N+P)$ olur.(P $[2; n]$ aralığındaki asal sayıların sayısı) P yaklaşık N / lnN olur.

Açıklama

- David Gries, Jayadev Misra. **A Linear Sieve Algorithm for Finding Prime Numbers** [1978]
- http://e-maxx.ru/algo/prime_sieve_linear
- http://e-maxx.ru/algo/eratosthenes_sieve