$\pi(n)$的计算

2015-03-11 09:23:26 By **matthew99**

$\pi(n)\neq\pi n$，更不是$3.141592655\cdots(n)$(←_←)，而是小于等于n的质数个数，n可以是任意实数。

普通算法

直接用线性筛就可以做到$O(n)$。

逗比算法

用$O(n\log n)$的筛法！

不对我会$O(n\log\log n)$(只用质数筛就是$O(n\log\log n)$)

帅气的算法一

根据容斥原理可得，所有小于等于x的数中，不能被n个不同的质数$p_1,p_2,\cdots,p_n$(p_i是第i个质数)中的任何一个整除的个数是

$$\lfloor x\rfloor-\sum_i\lfloor\frac{x}{p_i}\rfloor+\sum_{i<j}\lfloor\frac{x}{p_ip_j}\rfloor-\sum_{i<j<k}\lfloor\frac{x}{p_ip_jp_k}\rfloor+\cdots$$

若$n=\lfloor\sqrt{x}\rfloor$这个数显然就是$\pi(n)-\pi(\sqrt{n})+1$。

设$\Phi(x',n')$表示小于等于x'的数中(x'可以是任意实数)，不能被n个不同的质数$p_1,p_2,\cdots,p_n$中的前n'个整除的个数，则根据上面的式子容易推得

$$\Phi(x',n')=\Phi(x',n'-1)-\Phi(\frac{x'}{p_{n'}},n'-1)$$

递归边界为

$$\Phi(x',0)=\lfloor x'\rfloor$$

这个考虑一下上式中每一个p_k对式子的贡献即可。

注意到在递归过程中，第一个参数永远是$\lfloor\frac{x}{a}\rfloor$，$a$是某些因数的乘积，而即便$a$取所有值，上述值也不超过$O(\sqrt{x})$个，第二个参数显然也是$O(\sqrt{x})$个，这么一来如果加上记忆化或者直接递推，时间复杂度上界是$O(x)$，或许存在更低的上界，比如比线性低一个$O(\log x)$或者$O(\log\log x)$？(大雾)。

帅气的算法二

令$P_k(x,n)$表示小于等于x的数中恰好含有k个大于p_n的(可以相同的)质因子且不含有小于等于p_n的质因子的个数，又不妨令$P_0(x,n)=1$，则：

$$\Phi(x,n)=\sum_{k=0}^{+\infty}P_k(x,n)$$

令$y=\sqrt[3]{x}$，$n=\pi(y)$，则$P_k(x,n)=0(k\geq 3)$。

而显然有

$$P_1(x,n)=\pi(x)-n$$

$$P_2(x,n)=\sum_{y<p\leq\sqrt{x},p是质数}\pi(\frac{m}{p})-\pi(p)+1$$

(设$x=ab$(a,b是质数)是满足条件的数，那么当且仅当$p=\min\{a,b\}$时统计了x一次)

所以 $\Phi(x,n)=1+\pi(x)-n+\left(\sum_{y<p\le\sqrt{x}}\pi\left(\frac{m}{p}\right)-\pi(p)+1\right)$

移项得

$$\pi(x)=\Phi(x,n)-1+n-\left(\sum_{y<p\le\sqrt{x}}\pi\left(\frac{m}{p}\right)-\pi(p)+1\right)$$

预处理 $\pi(1)$ 到 $\pi(x^{\frac{2}{3}})$ 的值，那么除了 $\Phi(x,n)$ 之外，其他的项都可以在 $O(n^{\frac{2}{3}})$ 内求出。

对于 $\Phi(x,n)$，由于 $n=O(x^{\frac{1}{3}})$，而第一个参数上界是 $O(x^{\frac{1}{2}})$，因此总的复杂度不超过 $O(x^{\frac{5}{6}})$，且有可能更优？（大雾）

总之还是做到比线性低啦！

实现

实现成功后我们发现，正如题目中所述，时间复杂度只是个上界，实际上这个算法跑的很快。以下是我用来计算 $\pi(10^9)$ 的代码，实际上 $calc$ 中的不同参数只有不到70万个，比理论值 $10^{7.5}$ 不知道少到哪里去了，因此如果有更好的复杂度上界，请告诉我。

变量定义和该文章中的定义有所不同，请大家注意分别。

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <cctype>
#include <ctime>
#include <cassert>
#include <ext/pb_ds/assoc_container.hpp>
#include <iostream>

#define REP(i, a, b) for (int i = (a), _end_ = (b); i != _end_; ++i)
#define debug(...) fprintf(stderr, __VA_ARGS__)

using namespace std;

typedef long long LL;

const int maxn = 1e6, max0 = 1e5;

int sum[maxn + 5];
int prime[max0 + 5];
int pn = 0;

LL n = 1000000000LL;

int Max = -1;

__gnu_pbds::gp_hash_table<int, int> id;

LL a[40000][1000];
```

```
int an = 0;

LL calc(const int &n, const int &x)
{
    if (x < 0) return n;
    if (!id[n]) id[n] = ++an;
    int tmp = id[n];
    if (a[tmp][x] != -1) return a[tmp][x];
    return a[tmp][x] = calc(n, x - 1) - calc(n / prime[x], x - 1);
}

int main()
{
    memset(a, -1, sizeof a);
    for (int i = 2; i <= maxn; ++i)
    {
        if (!sum[i])
        {
            if ((LL)i * i * i <= n) Max = pn;
            prime[pn++] = i;
        }
        sum[i] = sum[i - 1] + !sum[i];
        for (int j = 0; j < pn && prime[j] * i <= maxn; ++j)
        {
            sum[i * prime[j]] = 1;
            if (!(i % prime[j])) break;
        }
    }
    LL ans = calc(n, Max) + Max;
    for (int p = Max + 1; (LL)prime[p] * prime[p] <= n; ++p) ans -= sum[n / prime
[p]] - sum[prime[p]] + 1;
    cout << ans << endl;
    return 0;
}
```

该程序的运行结果是50847534，与维基百科上的一致。在本机上(开启O2优化)运行时间为$53ms$，主要耗时依旧在Φ函数的计算中。

内存的话凑合着看吧，我又没说要直接用这个当模板交题。

参(chao)考(xi)文献

Prime-counting function

在该条目中还提到了更好的算法，有兴趣的人可以去深入研究。

UPD:本文章中的算法已被完++ ...更多内容参见π(n)的计算++。