Chapter 1 导言、算法分析与STL

1.1 OJ

- Codeforces
- Virtual Judge
- 牛客网 - 找工作神器|笔试题库|面试经验|实习招聘内推，求职就业一站解决_牛客网
- AtCoder
- 首页 - 洛谷 | 计算机科学教育新生态
- Welcome To PKU JudgeOnline
- Welcome to Hangzhou Dianzi University Online Judge
- 力扣 (LeetCode) 全球极客挚爱的技术成长平台

1.2 Resources

- OI Wiki - OI Wiki

1.3 算法分析

- 复杂度简介 - OI Wiki

1.3.1 渐进分析

1. 渐进记号在数学上实际是集合，**等于**实际上是**属于**或**包含**
2. Θ符号：

> 若存在正常量 c_1 和 c_2，使得对于足够大的 n，函数 $f(n)$能"夹人"$c_1 g(n)$与 $c_2 g(n)$之间，则$f(n)$属于集合 $\Theta(g(n))$。因为 $\Theta(g(n))$是一个集合，所以可以记"$f(n)\in\Theta(g(n))$"，以指出 $f(n)$是 $\Theta(g(n))$的成员。作为替代，我们通常记"$f(n)=\Theta(g(n))$"以表达相同的概念。因为我们按这种方式活用了等式，所以你可能感到困惑，但是在本节的后面我们将看到这样做有其好处。

3. O符号：表示函数的一个在常量因子内的上界
4. Ω符号：表示函数的一个在常量因子内的下界

1.3.2 求解递归式

1. 递归式的编写
2. 主方法解递归式：主定理(Master Theorem) - 知乎

1.4 STL

参考C++ 参考手册 - cppreference.com

1.5 Question

- 我们学习了三种渐进符号：渐进紧确界、渐进上界与渐进下界。那么使用这些渐进分析算法有什么意义呢？
- 为什么其中渐进上界是较为常用的渐进符号呢？
- 还有哪些算法分析方法？适用于哪些情况？

Chapter 2 模拟、枚举与贪心

2. 1 模拟

- P1138 第 k 小整数 - 洛谷 | 计算机科学教育新生态
- Problem - A - Codeforces
- Problem - B - Codeforces
- T539827 202411H Enemy - 洛谷 | 计算机科学教育新生态

2.2 枚举

- 试除法：【算法拾遗】试除法 - Trial Division - 知乎
- P2241 统计方形（数据加强版） - 洛谷 | 计算机科学教育新生态

2.3 贪心

- 部分背包：算法设计与分析_中国大学MOOC(慕课)
- Problem - A - Codeforces

Chapter 3 双指针、滑动窗口

3.1 双指针

1. 快慢指针 27. 移除元素 - 力扣（LeetCode）
2. 左右指针 977. 有序数组的平方 - 力扣（LeetCode）

3.2 滑动窗口

1. 209. 长度最小的子数组 - 力扣（LeetCode）

Chapter 4 高精度

见板子

Chapter 5 二分、前缀和与差分

略

Chapter 6 分治与搜索

6.1 分治

- 归并排序

6.2 搜索

- dfs本质上是n叉树前序遍历，bfs本质上是n叉树层序遍历
- 回溯时，相关量的改变在递归返回是需要撤销
- dfs可以用stack实现（或者说递归本来就是在运行栈运行），bfs可以用queue实现
- 特别的，lambda表达式可用于实现递归，相对来说较方便

Chapter 7 数论

模运算

模运算是大数运算中常见的操作，在算法竞赛中面对数据过大的时候，题目往往需要给出需要取模的要求，取模数一般为$1e9+7$、$1e9+9$、998244353。

这三个数比较大，但小于int32位最大值，并且三个数都是质数，至于为什么要质数，后面学习到其他算法会讲到。

定义取模运算为求a除以m的余数，记作：

$$a \bmod m = a\%m$$

一般的取余都是正整数的操作，对于负数求余数，不同的编译语言结果可能不同。

取模的性质：

加：$(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$

减：$(a-b) \bmod m = ((a \bmod m) - (b \bmod m)) \bmod m$

乘：$(a * b) \bmod m = ((a \bmod m) * (b \bmod m)) \bmod m$

对于除法取模是错误的，除法取模需要使用“逆”，在后续会讲到。

在运算过程中，对于大数相乘取模，会出现溢出的情况，即$a * b$或$(a \bmod m) * (b \bmod m)$可能会发生溢出。

此时我们可以不直接计算$a * b$，改为计算$(a * 2 * 2 \ldots\ldots 2 * 2) * (b/2/2 \ldots\ldots /2/2)$直到$b$减少为0为止。对于这种/2的操作我们可以采用二进制来处理。

但是如果b是奇数的话，$b/2$会取整，丢弃余数1，所以要判断b的奇偶性。

如果b是偶数就是$(a * 2) * (b/2)$。

如果b是奇数就是$(a * 2) * (b/2 + 1) = (a * 2) * (b/2) + (a * 2)$。

```
long long mul (long long a,long long b, long long m)
{
    a = a % m;
    b = b % m;
    long long res = 0;
    while(b > 0)
    {
        if(b & 1)//判断奇偶
            res = (res + a) % m;
        a = (a + a) % m;//a*2
        b >>=1;//b向右边移动一位，即去掉已经处理过的最后一位。
    }
    return res;
}
int main()
{
    long long a = 0x7877665544332211;
    long long b = 0x7988776655443322;
    long long m =  0x998776655443322;
    cout<<a*b%m<<'\n';//318333883276991760
    cout<<mul(a, b, m);//411509877096934416
}
```

快速幂

对于幂运算求a^n，如果我们直接一个一个的乘，时间复杂度为$O(n)$。如果使用快速幂，时间复杂度为$O(log_2 n)$。

快速幂标准做法采用位运算实现，下面介绍快速幂如何实现。

例如要计算a^{11}。我们可以分成$a^{11} = a^8 * a^2 * a^1$。其中8、2、1都是2的几次方，得出二进制为1101。

那么如何把11分解成$8 + 2 + 1$？把n按二进制处理即可。

遇到没有的幂次，我们进行判断一下跳过就可以了，比如a^{11}中没有a^4，检测到位为0，即可跳过。

[P1226 【模板】快速幂 - 洛谷 | 计算机科学教育新生态]

```
long long fastpow(long long a,long long b,long long mod)
{
    long long ans = 1;
    a = a % mod;
    while(b)
    {
        if(b & 1)//检测最后一位是0还是1。
            ans = (ans * a) % mod;//如果是1，就乘以当前位数。
        a = (a * a) % mod;//递推a的次方。
        b >>= 1;//n向右边移动一位，即去掉已经处理过的最后一位。
    }
    return ans;
}
```

GCD和LCM

最大公约数(GCD)和最小公倍数(LCM)是比赛中的高频考点。

整除

定义：a能整除b，记为$a|b$。其中，a、b为整数，且$a \neq b$，b是a的倍数。

性质:

1.若a、b、c为整数，且$a|b$、$b|c$，则$a|c$。

2.若a、b、m、n为整数，且$c|a$、$c|b$，则$c|(ma+nb)$；

GCD

GCD的定义：整数a和整数b的最大公约数能同时整除a和b的最大整数，记为$gcd(a,b)$。

例如：$gcd(15,81)=3$，$gcd(0,0)=0, gcd(0,3)=3, gcd(-6,-15)=3$。$-a$和$a$的因子相同，$gcd(a,b)=gcd(|a|,|b|)$所以我们在比赛中只考虑正整数的GCD。

GCD的性质

1.$gcd(a,b)=gcd(a,a+b)=gcd(a,k*a+b)$

2.$gcd(ka,kb)=k*gcd(a,b)$

3.定义多个整数的GCD：$gcd(a,b,c)=gcd[gcd(a,b),c]$

4.若$gcd(a,b)=d$，则$gcd(a/d,b/d)=1$,即a/d与b/d互素

5.$gcd(a+cb,b)=gcd(a,b)$

求GCD

求GCD常用方法有很多种，这里我只介绍两种较为重要的。

1）欧几里得

即辗转相除法，$gcd(a,b)=gcd(b, a \bmod b)$。

下面给出证明:

设整数a和$b(a>b)$，则有$a=k*b+q$，其中a、b、k、q分别为被除数，除数，商和余数。

当q为0，显然$gcd(a,b)=k$为k。

当q不为0，设a和b最大公约数为d。则有$a=d*k_1$，$b=d*k_2$。

由$a=k*b+q$可以得到，$q=a-k*b=d*k_1-k*k_2*d=d(k_1-k*k_2)$。

可以看出d能整除q，所以a与b最大公约数等于b与q最大公约数。

反之我们也需要证明，即a和b的最大公约数就是b和q的最大公约数。上面证明，我们得出d是q的一个因子。

已知q是b和d的因子，那么q就可以整除$k*b+d$。此时$a=k*d+b$就可以得到q整除a。

所以a、b中任意一个公因数也必定是b、q的任意一个公因数，即a、b的公因数集等于b、q的公因数集。那么两个集合里的最大值也相等。

[B4025 最大公约数 - 洛谷 | 计算机科学教育新生态]

```
int mygcd(int a,int b)//一直递归找gcd，回溯条件是b为0，此时可以得知gcd为a，因为gcd(a,0)=a。
{
    return b ? mygcd(b, a % b):a;
}
```

另外，对于 C++17，我们可以使用 `<numeric>` 头中的 `std::gcd` 与 `std::lcm` 来求最大公约数和最小公倍数。

2）更相减损术

即辗转相减法，由GCD的性质：$gcd(a,b) = gcd(b,a-b) = gcd(a,a-b)$。用大数减去小数，把所得的差与较小的数比较，继续减法，直到减数与差相等。

```
int mygcd(int a,int b)
{ r424
    while(a != b)
    {
        if(a > b)
            a = a- b;
        else
            b = b- a;
    }
    return a;
}
```

更相减损术避免了欧几里得取模计算，但是计算次数更多。由于避免了取模操作，所以常用用于高精度计算GCD问题。

LCM

a和b的最小公倍数表示为$lcm(a,b)$。

设$a = p_1^{c_1}p_2^{c_2}......p_m^{c_m}$，$b = p_1^{f_1}p_2^{f_2}......p_m^{f_m}$，则$gcd(a,b) = p_1^{min\{c_1,f_1\}}p_2^{min\{c_2,f_2\}}......p_m^{min\{c_3,f_3\}}$，$lcm(a,b) = p_1^{max\{c_1,f_1\}}p_2^{max\{c_2,f_2\}}......p_m^{max\{c_3,f_3\}}$。于是可以得出$gcd(a,b)lcm(a,b) = ab$。

注意要先除法后乘法，防止溢出。

```
int lcm(int a,int b)
{
    return a / gcd(a,b) * b;
}
```

裴蜀定理

定义：如果a与b都为整数，则有整数x和y使$ax+by = gcd(a,b)$。

推论：整数a与整数b互素当且仅当存在整数x和y，使$ax+by = 1$。

证明：使得$ax+by = d$，d一定是$gcd(a,b)$的整数倍。因为$a = gcd(a,b)*k_1$，$b = gcd(a,b)*k_2, ax+by = gcd(a,b)*k_1*x+gcd(a,b)*k_2*y = gcd(a,b)*(k_1+k_2+x+y) = d$。

所以最小的d是$gcd(a,b)$。

[P4549 【模板】裴蜀定理 - 洛谷 | 计算机科学教育新生态]

```
#include <iostream>
using namespace std;
int a[100003];
long long gcd(int a,int b)
{
    return b ? gcd(b,a%b):a;
}
int main ()
{
    int n;
    int i;
    cin>>n;
    for(i = 0;i < n;i++)
    {
        cin>>a[i];
    }
    long long sum = 0;
    sum = gcd(abs(a[0]),abs(a[1]));
    for(i = 1;i < n;i++)
    {
```

```
        sum = gcd(abs(a[i]),sum);
    }
    cout<<sum;
}
```

二元线性丢番图方程

方程$ax+by=c$成为二元性丢番图方程，x和y是变量，其他是已知整数，问是否有整数解。

该方程是二维平面上的一条直线，如果直线上有整数坐标点，就有解，没有则无解，如果存在一个解就有无数个解。

定理：设a，b为整数且$gcd(a,b)=d$。如果d不能整除c，那么方程$ax+by=c$没有整数，否则存在无穷多个解。如果（x_0，y_0）是方程的一个特解，那么所有解可以表示为

$x=x_0+\frac{b}{d}*n$,

$y=y_0-\frac{a}{d}*n$,

n为任意整数。

例如：
（1）方程$18x+3y=7$没有整数解，因为$gcd(18,3)=3$，3不能整除7；
（2）方程$25x+15y=70$存在无穷个解，因为$gcd(25,15)=5$且5整除70，一个特解是$x_0=4$，$y_0=-2$，通解是$x=4+3n$，$y=-2-5n$。

证明：令$a=da',b=db'$，有$ax+by=d(a'x+b'y)=c$。如果x、y、a'、b'是整数，那么c必须是d的整数倍数，才有整数解。

对于通解的形式，x值按b/d递增，y值按$-a/d$递增。我们可以设(x_0,y_0)是这条直线上的一个整数点，那么它移动到另一个点后的坐标为$(x_0+\Delta x,y_0+\Delta y)$。此时$\Delta x,\Delta y$必须是整数，$(x_0+\Delta x,y_0+\Delta y)$才是另一个整数点，我们知道这个直线必定通过原点，那么可以得到$a\Delta x+b\Delta y=0$。

那么Δx最小值是多少？因为a/d与b/d互素，只有$\Delta x=b/d$，$\Delta y=-a/d$时候，才为整数，并且满足$a\Delta x+b\Delta y=0$

扩展欧几里得

方程$ax+by=gcd(a,b)$，根据前一节的定理，它有整数解。扩展欧几里得算法求一个特解(x_0,y_0)。

下面证明为什么可以得到特解。

已知$ax+by=gcd(a,b)$,$bx'+(a\%b)y'=gcd(b,a\%b)$。

那么$ax+by=bx'+(a\%b)y'=bx'+(a-b(a/b)y')=ay'+b(x'-(a/b)y')$

所以可以得到$x=y',y=x'-(a/b)y'$。

递归终止条件$gcd(a,0)=a$,此时易知$x=1,y=0。

```
long long exgcd(long long a,long long b,long long &x,long long &y)
{
    if(b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    long long d = exgcd(b, a%b,y,x);
    y -= a/b*x;
    return d;
}
```

二元丢番图方程$ax+by=c$的解

用扩展欧几里德算法得到$ax+by=gcd(a,b)$的一个特解后，再利用它求方程$ax+by=c$的一个特解。步骤如下：
（1）判断方程$ax+by=c$是否有整数解，即$gcd(a,b)$能整除c。记$d=gcd(a,b)$。

（2）用扩展欧几里得算法求$ax+by=d$的一个特解x_0，y_0。

（3）在$ax_0+by_0=d$两边同时乘以c/d，得：$ax_0c/d+by_0c/d=c$。

（4）对照$ax+by=c$，得到它的一个解(x'_0,y'_0)为：$x'_0=x_0c/d$、$y'_0=y_0c/d$。

(5) 方程$ax+by=c$的通解：$x=x_0'+(b/d)n$，$y=y_0'-(a/d)n$。

已知通解为$x=x_0+(b/d)n$要使x最小，则需要x_0不断的减b/d。那么此过程相当于取模，可以简单化为$((x_0\%(b/d)+b/d)\%(b/d))$，多加一次再去模防止负数。

[P1516 青蛙的约会 - 洛谷 | 计算机科学教育新生态]

```cpp
#include <iostream>

using namespace std;
long long exgcd(long long a,long long b,long long &x,long long &y)
{
    if(b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    long long d = exgcd(b,a%b,y,x);
    y -= a/b*x;
    return d;
}
int main () {
    long long x, y, m, n, l;
    cin >> x >> y >> m >> n>>l;
    long long  c = x - y;
    long long a = n - m;
    if (a < 0) {
        a = -a;
        c = -c;
    }
    long long d = exgcd(a, l, x, y);
    if (c % d != 0)//判断方程有无整数解
        cout << "Impossible";
    else
        cout << ((x*c/d ) % (l / d) + (l / d)) % (l / d);
}
```

同余

设m是正整数，若a和b是整数，且$m|(a-b)$，则称a和b模m同余。即a除以m得到的余数和b除以m的余数相同，或者说$a-b$除以m，余数为0。

把a和b模m同余记为$a\equiv b(\mod m)$，m称为同余的模。

一元线性同余方程

设x是未知数，给定a、b、m，求整数x，满足$ax\equiv b(\mod m)$。

$ax\equiv b(\mod m)$表示$ax-b$是m的倍数，设为$-y$倍，则有$ax+my=b$，这就是二元线性丢番图方程。所以求解一元线性同余方程等价于求解二元线性丢番图方程。

定理：设a、b和m是整数，$m>0$，$gcd(a,m)=d$。若d不能整除b，则$ax\equiv b(\mod m)$无解；若d能整除b，则$ax\equiv b(\mod m)$有d个模m不同余的解。

此定理可以参考前面的线性丢番图方程来理解，如果有一个特解是x_0，那么通解是$x=x_0+(m/d)n$，当$n=0$，1，2，$\ldots\ldots$，$d-1$时，有d个模m不同余的解。

推论：a和m互素时，因为$d=gcd(a,m)=1$，所以线性同余方程$ax\equiv b(\mod m)$有唯一的模m不同余的解。

逆

> 给定整数a，且满足 $gcd(a,m)=1$，称$ax\equiv 1(\mod m)$的一个解为 a模m的逆。记为a^{-1}。
> 例如：$8x\equiv 1(\mod 31)$，有一个解是$x=4$，4是8模31的逆。所有的解，例如35、66等，都是8模31的逆。

所以从逆的要求来看，$gcd(a,m)=1$可以看出，模m最好是一个大于a的素数，才能保证$gcd(a,m)=1$。

求逆

1）扩展欧几里得求逆

$ax \equiv 1(\bmod m)$，即丢番图方程$ax+my=1$，先用扩展欧几里得求$ax+my=1$的一个特解x_0，通解$x=x_0+mn$。然后通过取模计算最小整数解$((x_0 \bmod m)+m) \bmod m$。

```
#include <iostream>
using namespace std;
long long exgcd(long long a,long long b,long long &x,long long &y)
{
    if(b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    long long d = exgcd(b, a%b,y,x);
    y -= a/b*x;
    return d;
}
int main ()
{
    long long a,b;
    long long y,x;
    cin>>a>>b;
    long long d = exgcd(a,b,x,y);

    cout<<(x % b + b) %b;
}
```

2）费马小定理求逆

设n是素数，a是正整数且与n互素，有$a^{n-1} \equiv 1(\bmod n)$。

$a*a^{n-2} \equiv 1(\quad \bmod n)$，那么$a^{n-2} \bmod n$，就是$a$模$n$的逆。

计算时候采用快速幂取模。

```
long long mod_inverse(long long a,long long mod)
{
    return fast_pow(a,mod - 2,mod);
}
```

逆与除法取模

求$(a/b) \bmod m$，即使a除以b，然后对m取模。这里a、b都是很大的数，会溢出导致取模出错。用逆可以避免除法计算。

设b的逆元是b^{-1}，有$(a/b) \bmod m=((a/b) \bmod m)((bb^{-1}) \bmod m)=(a/b*bb^{-1}) \bmod m=(ab^{-1}) \bmod m$

除法的模运算转换为乘法模运算，即

$(a/b) \bmod m=(ab^{-1}) \bmod m=(a \bmod m)(b^{-1} \bmod m) \bmod m$

2024ICPC湖北省赛[Problem - J - Codeforces]

```
#include <bits/stdc++.h>

using namespace std;
long long m = 998244353;
long long fastpow(long long a,long long b,long long mod)
{
    long long ans = 1;
    a = a % mod;
    while(b)
    {
        if(b & 1)
```

```
            ans = ans * a % mod;
        a = a * a % mod;
        b = b >> 1;
    }
    return ans;
}
long long a[1000003];
int main ()
{
    int n;
    long long sum = 0;
    int i;
    cin>>n;
    for(i = 0;i < n;i++)
    {
        cin>>a[i];
        sum +=a[i];
        sum = sum%m;
    }
    cout<<sum*fastpow(n,m - 2,m)%m;
    return 0;
}
```

素数

定义：只能被1和自己整除的正整数。

素数判定

1）试除法

时间复杂度$O(\sqrt{n})$，$n <= 10^{12}$。

```
bool is_prime(long long n)
{
    if(n <= 1)
        return false;
    for(long long i = 2;i<= sqrt(n);i++)//i*i<=n
    {
        if(n % i ==0)
            return false;
    }
    return true;
}
```

2)Miller-Rabin素性测试

```
#include <bits/stdc++.h>

using namespace std;
long long fast_pow(long long x,long long y,int m)
{
    long long res = 1;
    x = x % m;
    while(y)
    {
        if(y&1)
            res = (res * x) % m;
        x = (x * x) % m;
        y >>= 1;
    }
    return res;
```

```
}
bool witness(long long a,long long b)//素性测试，返回true表示n是合数
{
    long long u = n - 1;//u是n-1的二进制去掉末尾0
    int t = 0;//n-1的二进制，是奇数u的二进制后面加t个0
    while(u&1 == 0)//整数n-1末尾0的个数就是t
    {
        u = u>>1;
        t++;
    }
    long long x1,x2;
    x1 = fast_pow(a,u,n);
    for(int i = 1;i <= t;i++) //平方取模
    {
         x2 = fast_pow(x1,2,n);
         if(x2 == 1&&x1 != 1&&x1 != n-1)
         {
            return true;
         }
         x1 = x2;
    }
    if(x1 != -1)
        return true;//用费马测试判断是否为合数
    return false;
}
int miller_rabin(long long n,int s)
{
    if(n < 2)
        return 0;
    if(n == 2)
        return 1;
    if(n%2==0)
        return 0;
    for(int i = 0;i < s &&i < n;i++)//s次测试
    {
        long long a = rand()%(n-1) + 1; //a为随机数
        if(witness(a,n))
            return 0;
    }
    return 1;
}
int main()
{
    int m;
    cin>>m;
    int cnt = 0;
    for(int i = 0;i < m;i++)
    {
        long long n;
        cin>>n;
        int s = 50//50次测试
        cnt += miller_rabin(n,s);
    }
    cout<<cnt;
}
```

素数筛

给定n，求2 n内所有的素数。逐个判断时间复杂度非常高，我们可以筛法一起筛出所有整数，把非素数筛掉。

埃式筛

时间复杂度$O(nlog_2log_2^n)$

对于初始队列$\{2,3,4,5,6,7,8,9,10,11,12,13,......,n\}$。

(1)输出最小素数2，然后筛掉2的倍数，得$\{3,5,7,9,11,13......\}$。

(2)输出最小素数3，然后筛掉3的倍数，得$\{5,7,11,13,......\}$。

(3)输出最小素数5，然后筛掉5的倍数，得$\{7,11,13,......\}$。

```cpp
const int N=1e7;
int prime[N+1];//储存素数
bool visit[N+1];//true表示被筛掉，不是素数
int E_sieve(int n)
{
    for(int i = 0;i <= n;i++)
    {
        visit[i] = false;//初始化为false
    }
    for(int i = 2;i * i <= n;i++)//遍历用来做筛除的数2、3、5
    {
        if(!visit[i])
        {
            for(int j = i*i;j <= n;j+=i)
            {
                visit[j] = true;//标记为非素数
            }
        }
    }
    int k =  0;//记录素数个数
    //上面筛完了，下面开始记录素数
    for(int i = 2;i <= n;i++)
    {
        if(!visit[i])
            prime[k++] = i;//把筛出来的素数存入数组中
    }
    return k;
}
```

埃式筛中做了一些无用功，某个数会被筛到好几次，比如12就被2和3筛了两次，所以我们就可以使用更快的欧式筛来改进。

欧式筛

时间复杂度为$O(n)$。

一个合数肯定有一个最小质因数；让每个合数只被它的最小质因数筛选一次，达到不重复的目的。

[P3383 【模板】线性筛素数 - 洛谷 | 计算机科学教育新生态]

```cpp
int prime[N+1]; //保存素数
bool vis[N+1];//记录是否被筛
int euler_sieve(int n)
{
    int cnt = 0;
    memset(vis, 0, sizeof(vis));
    memset(prime,0,sizeof(prime));
    for(int i = 2;i <= n;i++)//检查每个数，筛去合数
    {
        if(!vis[i])
            prime[cnt++] = i;//如果没有被筛过，是素数，记录下来。
        for(int j = 0;j < cnt;j++)//用已经得到的素数去筛后面的数。
        {
            if(i * prime[j] > n) //只筛小于或等于n的数，大于直接退出
                break;
            vis[i * prime[j]] = 1;//用最小质因数筛去x
            if(i % prime[j] == 0)//如果i能整除它，表明i肯定不是x的最小质因数，
                break;
        }
    }
    return cnt;
}
```

代码vis[i*prime[j]] = 1中，表示用最小的质因数筛去了它的倍数，其中prime[j]是最小质因数。

后面一行中发现i能整除这个最小值质因数，那么可以说明i的最小是prime[j]，那么后面的prime[j+1]肯定不是最小质因数，于是可以退出循环了。比如当$i=4$时候，我们首先看$prime[j]=2$，筛去8，然后发现i，那么$prime[j+1]=3$就不会执行，因为$3*4=12$，而12的最小质因数是2，如果真的执行了，那么当$i=6$的时候，$prime[i]=2$，就会又执行一次，重复计算了。

质因数分解

前面我们讲LCM的时候提到了算术基本定理，即任何大于1的正整数n可以唯一分解为有限个素数的乘积：$n=p_1^{c_1}p_2^{c_2}\ldots\ldots p_m^{c_m}$，其中$c_i$都为正整数，$p_i$都为素数且从小到大。我们可以通过这个定理来分解质因数。

欧拉筛求最小质因数

我们知道欧拉筛是每次用最小质因数来操作，所以我们直接记录每次的最小质因数就可以了。

可以用来求$1-n$的每个数的最小质因数。

```
int prime[N+1]; //保存素数
int vis[N+1];// 记录最小质因数
int euler_sieve(int n)
{
    int cnt = 0;
    memset(vis, 0, sizeof(vis));
    memset(prime,0,sizeof(prime));
    for(int i = 2;i <= n;i++)//检查每个数，筛去合数
    {
        if(!vis[i])
        {
            prime[cnt++] = i;//如果没有被筛过，是素数，记录下来。
            vis[i] = i;//记录最小质因数
        }
        for(int j = 0;j < cnt;j++)//用已经得到的素数去筛后面的数。
        {
            if(i * prime[i] > n) //只筛小于或等于n的数，大于直接退出
                break;
            vis[i * prime[j]] = prime[j];//记录最小质因数
            if(i % prime[j] == 0)//如果i能整除它，表明i肯定不是x的最小质因数，
                break;
        }
    }
    return cnt;
}
```

试除法分解质因数

上面的欧拉筛是求最小质因数，分解质因数也可以采用试除法来求解。时间复杂度$O(\sqrt{n})$

（1）首先检查$2-\sqrt{n}$的所有素数，如果他能整除n，就是最小质因数。然后连续的用p_1去除n。目的是为了去掉n中的p_1，从而得到n_1

（2）然后对于n_1，可以从$p_1-\sqrt{n}$来检查所有素数，和步骤1一样。因为我们找到了最小质因数p_1，所以p_1前面的数就没有必要检查了。

最后做完之后，如果剩下一个大于1的数，那么它也是一个素数，是n的最大质因数。

```
int p[20];//记录因数，p[1]为最小因数
int c[40];//记录第i个因数的个数，
int  factor(int n)
{
    int m = 0;
    for(int i = 2;i <= sqrt(n);i++)
    {
        if(n % i == 0)
        {
            p[++m] = i;//记录最小因数
            c[m] = 0;//初始化有0个此因数
            while(n % i== 0)//把n中重复的去掉
            {
                n = n / i;
```

```
                c[m]++;//去一次个数加1
            }
        }
    }
    if(n > 1)
    {
        p[++m] = n;
        c[m] = 1;
    }
    return m;
}
```

Chapter 8 动态规划

8.1 dp基础

- dp问题具有最优子结构与重叠子问题
- 自上而下叫记忆化搜索，自下而上叫dp

8.2 线性dp

- [300. 最长递增子序列 - 力扣（LeetCode）]
- [1143. 最长公共子序列 - 力扣（LeetCode）]

8.3背包dp

8.3.1 01背包

- 模板题，网上随便找一道
- [416. 分割等和子集 - 力扣（LeetCode）]

8.3.2 完全背包

- 模板题，反转01背包容量遍历顺序即可重复

8.4 区间dp

- [P1880 NOI1995] 石子合并 - 洛谷 | 计算机科学教育新生态