ACM Template

Shanghai University

Team Untitled

July 20, 2020

Contents

0 Header

```cpp
#include <bits/stdc++.h> // #include <algorithm>
using namespace std;
#define clr(a, x) memset(a, x, sizeof(a))
#define pb(x) push_back(x)
#define X first
#define Y second
typedef pair<int, int> PII;
typedef vector<int> VI;
typedef long long ll;
const int INF = 0x3f3f3f3f;
const int minINF = 0xc0c0c0c0;
const int mod = 1e9 + 7;
const double eps = 1e-6; //若题目要求保留到小数点后6位，则为1e-7
const double PI = acos(-1.0);
const double E = exp(1);

//关闭同步/解除绑定
ios_base::sync_with_stdio(false);
cin.tie(0);

// 字符串转数字要考虑负号
```

1 Math

1.1 Prime

1.1.1 Eular Sieve

原理：对于任意合数，必定可以有最小质因子乘以最大因子的分解方式 $O(n)$ 因此，对于每个合数，只要用最大因子筛一遍，枚举时只要枚举最小质因子即可。因为一般有 $prime[i] * prime[i] <= n$ ，所以筛选范围为 $\sqrt{n}$ 1e7–665000 1e6–80000 1e5–1e4

```
int prime[cnt]; //prime[0]储存素数的个数，素数下标从1开始
bool heshu[maxn];
void getPrime(int n)
{
    for (int i = 2; i <= n; ++i)
    {
        if (!heshu[i])
            prime[++prime[0]] = i;
        for (int j = 1; j <= prime[0] && i * prime[j] <= n; ++j)
        {
            heshu[i * prime[j]] = true; //找到的素数的倍数不访问
            if (i % prime[j] == 0)
                break;
        }
    }
}
```

1.1.2 有关素数的基础算法

素数判定

```
bool isPrime(int n)
{
    if (n == 1)
        return false;
    if (n == 2 || n == 3)
        return true;
    if (n % 6 != 1 && n % 6 != 5)
        return false;
    for (int i = 5; i * i <= n; i += 6)
        if (n % i == 0 || n % (i + 2) == 0)
            return false;
    return true;
}
```

约数枚举

```
vector<int> res;
void divisor(int n)
{
    for (int i = 1; i * i <= n; ++i)
        if (n % i == 0)
        {
            res.push_back(i);
            if (i != n / i)
                res.push_back(n / i);
        }
}
```

1.1.3 Miller Rabin

```cpp
typedef long long ll;
ll Mul(ll a, ll b, ll mod)
{
    if (mod <= 1000000000)
        return a * b % mod;
    else if (mod <= 1000000000000LL)
        return (((a * (b >> 20) % mod) << 20) + (a * (b & ((1 << 20) - 1)))) % mod;
    else
    {
        ll d = (ll)floor(a * (long double)b / mod + 0.5);
        ll res = (a * b - d * mod) % mod;
        if (res < 0)
            res += mod;
        return res;
    }
}

bool Miller_Rabin(ll n, int s) //s >= 8
{
    if (n == 2)
        return true;
    if (n < 2 || !(n & 1))
        return false;
    int t = 0;
    ll u = n - 1;
    while ((u & 1) == 0)
        ++t, u >>= 1;
    for (int i = 0; i < s; ++i)
    {
        ll a = rand() % (n - 1) + 1;
        ll x = mod_pow(a, u, n);
        for (int j = 0; j < t; ++j)
        {
            ll y = Mul(x, x, n);
            if (y == 1 && x != 1 && x != n - 1)
                return false;
            x = y;
        }
        if (x != 1)
            return false;
    }
    return true;
}
```

1.1.4 区间素数筛

```cpp
bool is_prime[1000]; //is_prime[i-a]=true <=> i是素数
bool is_prime_small[100];
ll prime[maxn];                 //只求素数个数时可省略
int segment_sieve(ll a, ll b) //对区间[a,b)内的整数执行筛法
{
    for (ll i = 0; i * i < b; ++i) //直接 i*i 会溢出
        is_prime_small[i] = true;
    for (ll i = 0; i < b - a; ++i)
        is_prime[i] = true;
    for (ll i = 2; i * i < b; ++i)
```

```
        if (is_prime_small[i])
        {
            for (ll j = 2 * i; j * j < b; j += i) //筛[2,sqrt(b))
                is_prime_small[i] = false;
            for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i) //筛[a,b)
                is_prime[j - a] = false;
        }
    int cnt = 0;
    for (ll i = 0; i < b - a; ++i)
        if (is_prime[i])
            prime[cnt++] = i + a; // ++cnt;
    return cnt;
}
```

```
ans = segment_sieve(a, b + 1) - (a == 1)); //[a,b]
```

1.2 Euler

1.2.1 欧拉函数

欧拉函数的值等于不超过 m 并且和 m 互素的数的个数

```
ll euler(ll n) //求欧拉函数值
{
    ll res = n;
    for (ll i = 2; i * i <= n; ++i)
        if (n % i == 0)
        {
            res = res / i * (i - 1);
            while (n % i == 0)
                n /= i;
        }
    if (n != 1)
        res = res / n * (n - 1);
    return res;
}
int phi[maxn];
void euler_phi(int n) //筛出欧拉函数值的表
{
    for (int i = 0; i <= n; ++i)
        phi[i] = i;
    for (int i = 2; i <= n; ++i)
        if (phi[i] == i)
            for (int j = i; j <= n; j += i)
                phi[j] = phi[j] / i * (i - 1);
}
```

1.2.2 欧拉降幂

Euler Theorem 费马小定理 (p 为素数) $a^b = a^{bmod(p-1)}modp$ 扩展欧拉定理（欧拉降幂）(a 和 p 不互质) $a^bmodp = a^bmodP(b < phi(p))$ $a^bmodp = a^{bmodphi(p)+phi(p)}modp(b >= phi(p))$ 指数循环节：从 a^0 到 $a^{phi(p)-1}$ 不是重复的，从 $a^{phi(p)}$ 开始出现循环节，长度为 $phi(p)$ $a^bmodp(1 <= a <= 1e9\ 1 <= b <= 10^2e7\ 1 <= p <= 1e6)$

```
ll solve(ll a, string &str, ll p)
{
    int phi = euler_phi(p);
    bool flag = false;
    ll b = 0;
    for (int i = 0; i < str.length(); ++i)
    {
```