Math 数学

Modint

```
template<typename I,typename L,I mod> struct Modint {
    I v;
    I pow(L b) const {
        L res=1,a=v;
        while(b) { if(b&1) res=res*a%mod; b>>=1; a=a*a%mod; }
        return res;
    }
    I inv() const { return pow(mod-2); }

    using M=Modint;
    M &operator+=(const M &x) { v+=x.v; v-=v>=mod?mod:0; return *this; }
    M &operator-=(const M &x) { v-=x.v; v+=v<0?mod:0; return *this; }
    M &operator*=(const M &x) { v=L(1)*v*x.v%mod; return *this; }
    M &operator/=(const M &x) { v=L(1)*v*x.inv()%mod; return *this; }

    friend M operator+(M l,const M &r) { return l+=r; }
    friend M operator-(M l,const M &r) { return l-=r; }
    friend M operator*(M l,const M &r) { return l*=r; }
    friend M operator/(M l,const M &r) { return l/=r; }
    friend M operator-(M r) { r.v=mod-r.v; return r; }

    friend ostream &operator<<(ostream &os,const M &x) { return os<<x.v; }
    constexpr Modint(L x=0): v((x%=mod)<0?x+mod:x) {}
}; using Mint=Modint<int,long long,998244353>;
```

附加函数

```
M operator++(int) { auto res=*this; ++*this; return res; }
M operator--(int) { auto res=*this; --*this; return res; }
M &operator++() { v=v==mod-1?0:v+1; return *this; }
M &operator--() { v=v?v-1:mod-1; return *this; }
```

```
bool operator< (const M &x) const { return v< x.v; }
bool operator> (const M &x) const { return v> x.v; }
bool operator<=(const M &x) const { return v<=x.v; }
bool operator>=(const M &x) const { return v>=x.v; }
bool operator==(const M &x) const { return v==x.v; }
bool operator!=(const M &x) const { return v!=x.v; }

friend istream &operator>>(istream &is,M &x) { is>>x.v; x=M(x.v); return
is; }
```

组合数

时间复杂度 $\mathcal{O}(n)$。

```
struct Binom {
    vector<Mint> faet,infaet;

    Mint operator()(int n,int m) {
        if(n<0||m<0||n<m) return {};
        return faet[n]*infaet[n-m]*infaet[m];
    }

    explicit Binom(int sz) {
        faet.resize(sz+1);
        infaet.resize(sz+1);

        faet[0]=faet[1]=1;
        infaet[0]=infaet[1]=1;
        for(int i=2;i<=sz;i++){
            faet[i]=faet[i-1]*i;
            infaet[i]=infaet[i-1]/i;
        }
    }
} binom(N);
```

质数筛与质因数分解

欧拉筛

时间复杂度 $\mathcal{O}(n)$。

```
constexpr int M=1e6+10;
int prime[M],idx;
bool isnp[M];
```

```
void get_prime(int n=M-1) {
    isnp[1]=1;
    for(int i=2;i<=n;i++) {
        if(!isnp[i]) prime[++idx]=i;
        for(int j=1;prime[j]<=n/i;j++) {
            isnp[prime[j]*i]=true;
            if(i%prime[j]==0) break;
        }
    }
}
```

欧拉函数

线性筛求欧拉函数 $\varphi(i)$ 表示 小于等于 i 和 i 互质的个数。

时间复杂度 $\mathcal{O}(n)$。

```
constexpr int M=1e6+10;
int prime[M],phi[M],idx;
bool isnp[M];

void get_prime(int n=M-1) {
    isnp[1]=phi[1]=1;
    for(int i=2;i<=n;i++) {
        if(!isnp[i]) prime[++idx]=i,phi[i]=i-1;
        for(int j=1;prime[j]<=n/i;j++) {
            isnp[prime[j]*i]=true;
            if(i%prime[j]==0) {
                phi[i*prime[j]]=phi[i]*prime[j];
                break;
            }
            else phi[i*prime[j]]=phi[i]*(prime[j]-1);
        }
    }
}
```

质因数分解

预处理复杂度 $\mathcal{O}(n)$，分解复杂度 $\mathcal{O}(\log n)$。

```
constexpr int M=1e6+10;
int prime[M],minp[M],idx;
bool isnp[M];

void get_prime(int n=M-1) {
    isnp[1]=minp[1]=1;
    for(int i=2;i<=n;i++) {
        if(!isnp[i]) prime[++idx]=i,minp[i]=i;
        for(int j=1;prime[j]<=n/i;j++) {
```

```
            isnp[prime[j]*i]=true;
            minp[prime[j]*i]=prime[j];
            if(i%prime[j]==0) break;
        }
    }
}

vector<int> get_factor(int val) {
    vector<int> res;
    while(val>1) {
        int t=minp[val];
        res.push_back(t);
        while(minp[val]==t) val/=t;
    }
    return res;
}
```

复杂度 $\mathcal{O}(\sqrt{n})$。

```
template<typename T> vector<T> get_prime_factor(T x) {
    vector<T> res;
    for(T i=2;i*i<=x;i++) {
        if(x%i==0) {
            while(x%i==0) x/=i;
            res.push_back(i);
        }
    }
    if(x>1) res.push_back(x);
    return res;
}

template<typename T> vector<T> get_factor(T x) {
    vector<T> res;
    for(T i=1;i*i<=x;i++) {
        if(x%i==0) {
            res.push_back(i);
            if(x/i!=i) res.push_back(x/i);
        }
    }
    return res;
}
```

Exgcd

计算 $ax+by=\gcd(a,b)$ 的一组解，返回 $\gcd(a,b)$。

如果是计算 $ax+by=c$ 的一组解,那么将 x,y 再乘上 $\frac{c}{\gcd(a,b)}$ 即可,当且仅当 $\gcd(a,b)|c$ 时有解。

此外，x,y 的通解形式分别为 $x+\frac{kb}{\gcd(a,b)},y-\frac{ka}{\gcd(a,b)}$。

```
template<typename T> T exgcd(T a,T b,T &x,T &y) {
    if(!b) { x=1,y=0; return a; }
    T res=exgcd(b, a%b, x, y),t=x;
    x=y,y=t-(a/b)*y;
    return res;
}
```

数论分块

```
int next_floor(int k,int i) {
    return k/(k/i);
}

int next_ceil(int k,int i) {
    if(k-1<i) return i;
    return (k-1)/((k-1)/i);
}
```

矩阵

```
template<typename T,int R,int C=R> struct Matrix {
    array<array<T,C>,R> v;

    template<int Rr,int Cr> constexpr Matrix<T,R,Cr>
    operator*(const Matrix<T,Rr,Cr> &r) const {
        static_assert(C==Rr,"");
        array<array<T,Cr>,R> ans{};
        for(int i=0;i<R;i++) {
            for(int k=0;k<C;k++) {
                // if(v[i][k]==0) continue;
                for(int j=0;j<Cr;j++) {
                    ans[i][j]+=v[i][k]*r[k][j];
                }
            }
        }
        return ans;
    }

    constexpr Matrix operator+(const Matrix &r) const {
        array<array<T,C>,R> ans{};
        for(int i=0;i<R;i++)
            for(int j=0;j<C;j++)
                ans[i][j]=v[i][j]+r[i][j];
```

```
        return ans;
    }

    constexpr Matrix operator-(const Matrix &r) const {
        array<array<T,C>,R> ans{};
        for(int i=0;i<R;i++)
            for(int j=0;j<C;j++)
                ans[i][j]=v[i][j]-r[i][j];
        return ans;
    }

    constexpr Matrix &operator*=(const Matrix<T,C,C> &r) { return
*this=*this*r; }
    constexpr Matrix &operator+=(const Matrix &r) { return *this=*this+r; }
    constexpr Matrix &operator-=(const Matrix &r) { return *this=*this-r; }

    constexpr Matrix pow(long long k) const {
        Matrix res(1),x=*this;
        while(k) { if(k&1) res*=x; k>>=1; x*=x; }
        return res;
    }

    constexpr auto &operator[](int idx) { return v[idx]; }
    constexpr auto &operator[](int idx) const { return v[idx]; }

    constexpr void clear() { v={}; }
    constexpr void unit(T x=1) {
        static_assert(R==C,"");
        clear(); for(int i=0;i<R;i++) v[i][i]=x;
    }

    constexpr Matrix() { clear(); }
    constexpr Matrix(T x) { unit(x); }
    constexpr Matrix(const array<array<T,C>,R> &x): v(x) {}
};
```