优化

取消同步流

```
ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
```

读入优化

朴素读入优化

```
int read() {
  int x = 0, w = 1;
  char ch = 0;
  while (ch < '0' || ch > '9') {
    if (ch == '-') w = -1;
    ch = getchar();
  }
  while (ch >= '0' && ch <= '9') {
    x = x * 10 + (ch - '0');
    ch = getchar();
  }
  return x * w;
}
```

fread读入优化

```
inline char getcha(){
    static char buf[100000],*p1=buf,*p2=buf;
    return p1==p2&&(p2=(p1=buf)+fread(buf,1,100000,stdin),p1==p2)?EOF:*p1++;
}
int read(){
    int res=0;char ch=getcha();bool XX=false;
    for(;!isdigit(ch);ch=getcha())
      (ch=='-') && (XX=true);
    for(;isdigit(ch);ch=getcha())
      res=(res<<3)+(res<<1)+(ch^48);
    return XX?-res:res;
}
```

FastIO

```
#define FI(n) FastIO::read(n)

namespace FastIO {
    const int SIZE = 1 << 16;
```

```
    char buf[SIZE], obuf[SIZE], str[60];
    int bi = SIZE, bn = SIZE, opt;
    int read(char *s) {
        while (bn) {
            for (; bi < bn && buf[bi] <= ' '; bi++);
            if (bi < bn) break;
            bn = fread(buf, 1, SIZE, stdin);
            bi = 0;
        }
        int sn = 0;
        while (bn) {
            for (; bi < bn && buf[bi] > ' '; bi++) s[sn++] = buf[bi];
            if (bi < bn) break;
            bn = fread(buf, 1, SIZE, stdin);
            bi = 0;
        }
        s[sn] = 0;
        return sn;
    }
    bool read(int& x) {
        int n = read(str), bf;

        if (!n) return 0;
        int i = 0; if (str[i] == '-') bf = -1, i++; else bf = 1;
        for (x = 0; i < n; i++) x = x * 10 + str[i] - '0';
        if (bf < 0) x = -x;
        return 1;
    }
};
```

实数读入优化

```
inline double dbread(){
    double X=0,Y=1.0; int w=0; char ch=0;
    while(!isdigit(ch)) {w|=ch=='-';ch=getchar();}
    while(isdigit(ch)) X=X*10+(ch^48),ch=getchar();
    ch=getchar();//读入小数点
    while(isdigit(ch)) X+=(Y/=10)*(ch^48),ch=getchar();
    return w?-X:X;
}
```

输出优化

朴素输出优化

```
void write(int x) {
  if (x < 0) {
    x = -x;
    putchar('-');
  }
  if (x > 9) write(x / 10);
  putchar(x % 10 + '0');
}
```

write输出优化

```
char pbuf[100000],*pp=pbuf;
void push(const char c) {
    if(pp-pbuf==100000) fwrite(pbuf,1,100000,stdout),pp=pbuf;
    *pp++=c;
}
void write(int x){
    static int sta[35];
    int top=0;
    do{sta[top++]=x%10,x/=10;}while(x);
    while(top) push(sta[--top]+'0');
}
//请大家在程序结束前加上一句fwrite(pbuf,1,pp-pbuf,stdout);pp=pbuf;
//防止出现没输出完成的类似错误
```

O2O3优化

```
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
```

黑科技

```
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
#pragma GCC optimize("unroll-loops")
```

```
//Example: 可以用于分割被空格、制表符等符号分割的字符串
#include<iostream>
#include<sstream>        //istringstream 必须包含这个头文件
#include<string>
using namespace std;
int main(){
    string str="i am a boy";
    istringstream is(str);
    string s;
    while(is>>s)  {
        cout<<s<<endl;
    }
}
```

数论模板

欧几里得算法(辗转相除法)

> gcd(a,b)=gcd(b,a%b)

```
int gcd(int a, int b){
    return !b ? a : gcd (b, a % b);
}
```

a,b的最小公倍数

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

扩展欧几里得算法

> **裴蜀定理：若 a,b 是整数,且 gcd(a,b)=d，那么对于任意的整数 x,y, ax+by 都一定是 d 的倍数，特别地，一定存在整数 x,y，使 ax+by=d 成立。**

扩展欧几里德常用在求解模线性方程及方程组中。

```
#include <bits/stdc++.h>
using namespace std;
int exgcd(int a,int b,int &x,int &y){
    if(!b){
        x=1,y=0;
        return a;
    }
    int gcd=exgcd(b,a%b,x,y),temp=x;
    x=y,y=temp-a/b*y;
    return gcd;
}

int main(){
    int a,b,x,y;
    cin>>a>>b;
    exgcd(a,b,x,y);
    cout<<(x%b+b)%b<<"\n";
    return 0;
}
```

高精度gcd(Stein算法)

```
int stein(int a,int b){
    if(a<b) a^=b,b^=a,a^=b;        //交换，使a为较大数；
    if(!b) return a;                    //当相减为零，即两数相等时，gcd=a；
    if((!(a&1))&&(!(b&1))) return stein(a>>1,b>>1)<<1;    //s1,注意最后的左移，在递归返回过程中
将2因子乘上；
    else if((a&1)&&(!(b&1))) return stein(a,b>>1);            //s2;
    else if((!(a&1))&&(b&1)) return stein(a>>1,b);
    else return stein(a-b,b);                                    //s3;
}
```

龟速乘(a*b % mod)

```
ll mul(ll a,ll b,ll mod){
    ll ans=0; //保存结果
    ll base=a; //每一次进行加的数字
    while(b){
        if(b&1) ans=(ans+base)%mod;
        base=(base*2)%mod;
        b>>=1;
    }
    return ans;
}
```

快速幂(a^b % mod)

```
ll fpow(ll a,ll b,ll mod){
    if(mod==1) return 0;
    ll ans=1%mod;
    while(b){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ans;
}
```

矩阵快速幂

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod=1e9+7;
struct node{
    ll mat[105][105];
};
int n;
node mul(node x,node y){
    node tmp;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            tmp.mat[i][j]=0;
            for(int k=0;k<n;k++){
                tmp.mat[i][j]+=(x.mat[i][k]*y.mat[k][j])%mod;
                tmp.mat[i][j]%=mod;
            }
        }
    }
    return tmp;
}
node matpow(node x,node y,ll num){
    while(num){
        if(num&1){
            y=mul(x,y);
        }
        x=mul(x,x);
        num=num>>1;
    }
    return y;
}
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    node x,y;//x是系数矩阵,y是单位矩阵
    ll k;
    cin>>n>>k;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cin>>x.mat[i][j];
            if(i==j) y.mat[i][j]=1;
        }
    }
    node c=matpow(x,y,k);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cout<<c.mat[i][j]<<" ";
```

```
        }
        cout<<"\n";
    }
    return 0;
}
```

逆元

拓展欧几里得求逆元

```
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(!b){
        x=1,y=0;
        return a;
    }
    ll gcd=exgcd(b,a%b,x,y),temp=x;
    x=y,y=temp-a/b*y;
    return gcd;
}

ll inv(ll a,ll p){
    ll x,y;
    if(exgcd(a,p,x,y)!=1) return -1; //无解的情况
    return (x%p+p)%p;
}
```

快速幂逆元

条件:mod是质数

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll mod;
ll fpow(ll a,ll b){
    ll ans=1%mod;
    while(b){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ans;
}

int main(){
    int n; cin>>n;
    while(n--){
        int a;
        cin>>a>>mod;
        if(a%mod==0) puts("impossible");
        else cout<<fpow(a,mod-2)<<"\n";
    }
    return 0;
}
```

线性递推求逆元

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=3e6+10;
ll n,mod,inv[N];
int main(){
    cin>>n>>mod;
    inv[1]=1;
    for(int i=2;i<=n;i++) inv[i]=(mod-(mod/i))*inv[mod%i]%mod;
    for(int i=1;i<=n;i++) cout<<inv[i]<<"\n";
    return 0;
}
```

组合数的计算

> $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$

递归

时间复杂度：小于$O(n^2)$

```
long long res[67][67];
long long C(long long n,long long m){
    if(!m||m==n) return 1;
    if(res[n][m]) return res[n][m];
    return res[n][m]=C(n-1,m)+C(n-1,m-1);
}
```

递推 将整张表都计算出来（杨辉三角）

时间复杂度：$O(n^2)$

```
long long res[67][67];
const int N = 60;
void calc(){
    for(int i=0;i<N;i++){
        res[i][0]=res[i][i]=1;  //初始化边界
    }
    for(int i=2;i<N;i++){
        for(int j=0;j<=i/2;j++){
            res[i][j]=res[i-1][j]+res[i-1][j-1];
            res[i][i-j] = res[i][j]; //C(i,i-j) = C(i,j)
        }
    }
}
```

通过定义式的变形来计算

时间复杂度：$O(m)$

```
ll C(ll n,ll m){
    ll ans = 1;
    for(ll i=1;i<=m;i++){
        ans=ans*(n-m+i)/i;  //注意一定要先乘再除
    }
    return ans;
}
```

计算C(n,m)%p

根据定义式计算

要求：$n \leq 10^6, m \leq 10^6, p \leq 10^9$
时间复杂度：$O(klogn)$,其中k为不超过n的质数个数

```
//使用筛法得到素数表prime，注意表中最大素数不得小于n
int prime[maxn];

ll cal(ll n,ll p){
    ll ans=0;
    while(n){
        ans+=n/p;
        n/=p;
    }
    return ans;
}

ll C(ll n,ll m,ll p){
    ll ans=1;
    //遍历不超过n的所有质数
    for(ll i=0;prime[i]<=n;i++){
        //计算C(n,m)中prime[i]的指数c,cal(n,k)为n!中含质因子k的个数
        ll c=cal(n,prime[i])-cal(m,prime[i])-cal(n-m,prime[i]);
        //快速幂计算prime[i]^c%p
        ans=ans*fpow(prime[i],c,p)%p;
    }
    return ans;
}
```

通过定义式变形来计算

情况① $m < p$，且p是素数

要求：$n \leq 10^9, m \leq 10^5, m < p \leq 10^9$,p是素数
时间复杂度：$O(mlogm)$

```
ll C(ll n,ll m,ll p){
    ll ans=1;
    for(ll i=1;i<=m;i++){
        ans=ans*(n-m+i)%p;
        ans=ans*fpow(i,p-2)%p;
    }
    return ans;
}
```

情况② m任意，且p是素数

要求：$n \leq 10^9, m \leq 10^5, p \leq 10^9$,p是素数
时间复杂度：$O(mlogn)$

```
ll C(ll n,ll m,ll p){
    //ans存放计算结果，numP统计分子中的p比分母中的p多几个
    ll ans=1,numP=0;
    for(ll i=1;i<=m;i++){
        int temp=n-m+i;  //分子
        while(temp%p==0){  //去除分子中的所有p，同时累计numP
            numP++;
            temp/=p;
        }
        ans=ans*temp%p;  //乘以分子中除了p以外的部分
```

```
            temp=i;//分母
            while(temp%p==0){   //去除分母中的所有p，同时减少numP
                numP--;
                temp/=p;
            }
            ans=ans*fpow(temp,p-2)%p;   //除以分母中除了p以外的部分
        }
        if(numP>0) return 0;   //分子中p的个数多于分母，直接返回0
        else return ans;   //分子中p的个数等于分母，返回计算的结果
}
```

预处理

```
ll Finv[N],fac[N],inv[N];

void init(int n){//n<N
    inv[1]=1;
    for(int i=2;i<=n;i++) inv[i]=((mod-mod/i)*inv[mod%i])%mod;
    fac[0]=Finv[0]=1;
    for(int i=1;i<=n;i++) fac[i]=fac[i-1]*i%mod,Finv[i]=Finv[i-1]*inv[i]%mod;
    //Finv[n]=fpow(fac[n],mod-2);
    //for(int i=n-1;i>=1;i--) Finv[i]=Finv[i+1]*(i+1)%mod;
}

ll C(ll n,ll m){
    if(m<0||m>n) return 0;
    return fac[n]*Finv[n-m]%mod*Finv[m]%mod;
}
```

Lucas定理

要求：$n \le 10^{18}, m \le 10^{18}, p \le 10^5$,p是素数

如果 p 是素数，将 m 和 n 表示为 p 进制：

$$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_0$$

$$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_0$$

那么 Lucas（卢卡斯）定理告诉我们，$C_n^m \equiv C_{n_k}^{m_k} \times C_{n_{k-1}}^{m_{k-1}} \times \cdots \times C_{n_0}^{m_0} \pmod p$ 成立。

例如对 $C_8^3\%5$ 来说，m=3、n = 8，将 m 和 n 表示为五进制：

$$m = 3 = 0 \times 5^1 + 3$$

$$n = 8 = 1 \times 5^1 + 3$$

于是有 $C_8^3\%5 = C_1^0 \times C_3^3\%5 = 1$。

看起来很复杂，那么这个式子意味着什么呢？由于 n 和 m 的 p 进制表示的项数为 $O(\log n)$ 级别，因此 Lucas 定理意味着将 $C_n^m\%p$ 分解为 $O(\log n)$ 级别个小组合数的乘积的模。显然，分解出的小组合数 $C_{n_i}^{m_i}$ 均满足 $n_i < p$，因此 Lucas 定理非常适合处理 $p \leqslant 10^5$ 级别的大组合数取模问题，此时能够支持 long long 级别的 n 和 m，也就是 $m \leqslant n \leqslant 10^{18}$ 级别的数据范围。唯一的要求是 p 是素数。

```
ll p;
ll lucas(ll n,ll m) {
    if(!m) return 1;
    return C(n%p,m%p)*lucas(n/p,m/p)%p;
}
```

exLucas定理

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e6 + 5;

ll Pre[N];

ll extend_gcd(ll a, ll b, ll &x, ll &y) {
    if(!b) {
        x = 1; y = 0;
        return a;
    }
    ll d = extend_gcd(b, a % b, x, y);
    ll t = x;
    x = y; y = t - (a / b) * y;
    return d;
}

ll fpow(ll a, ll b, ll p) {
    ll ans = 1;
    a %= p;
    while(b) {
        if(b & 1) ans = (ans * a) % p;
        a = (a * a) % p;
        b >>= 1;
    }
    return ans;
}

ll getInv(ll a, ll p) {
    ll x, y;
    extend_gcd(a, p, x, y);
    x = (x % p + p) % p;
    return x;
}

ll Mul(ll n, ll pi, ll pk) {
    if(n <= 1) return 1;
    ll ans = 1;
    if(n >= pk) {
        ans = Pre[pk - 1];
        ans = fpow(ans, n / pk, pk);
    }
    if(n % pk) ans = ans * Pre[n % pk] % pk;
    return ans * Mul(n / pi, pi, pk) % pk;
}

ll getC(ll n, ll m, ll pi, ll pk) {
    Pre[0] = Pre[1] = 1;
    for (ll i = 2; i < pk; i ++){
        Pre[i] = Pre[i - 1];
        if(i % pi) Pre[i] = Pre[i] * i % pk;
    }
    ll a = Mul(n, pi, pk);
    ll b = getInv(Mul(m, pi, pk), pk);
    ll c = getInv(Mul(n - m, pi, pk), pk);
    ll ans = 1ll * a * b % pk * c % pk;
    ll k = 0;
    for (ll i = n / pi; i; i /= pi) k += i;
    for (ll i = m / pi; i; i /= pi) k -= i;
```

```
    for (ll i = (n - m) / pi; i; i /= pi) k -= i;
    return ans * fpow(pi, k, pk) % pk;
}

ll exlucas(ll n, ll m, ll P) {
    ll p = P;
    ll ans = 0;
    for (ll i = 2; i <= p; i ++) {
        if(p % i == 0) {
            ll pi = i, pk = 1;
            while(p % i == 0) {
                p /= i;
                pk *= i;
            }
            ans = (ans + 1ll * getC(n, m, pi, pk) * (P / pk) % P * getInv(P / pk, pk) %
P) % P;
        }
    }
    return ans;
}


int main(){
    ll n,m,p;cin>>n>>m>>p;
    cout<<exlucas(n,m,p)<<"\n";
    return 0;
}
```

扩展欧拉定理

> 费马小定理：a是不能被质数p整除的正整数，$a^{p-1} \equiv 1 \pmod p$
>
> 推论：$a^b \equiv a^{b \bmod (p-1)} \pmod p$
>
> 欧拉定理：若m,a为正整数，且m,a互质，$a^{\varphi(m)} \equiv 1 \pmod m$
>
> 推论：$a^b \equiv a^{b \bmod \varphi(m)} \pmod m$
>
> 扩展欧拉定理：a和m不互质，$a^b \equiv \begin{cases} a^b & ,b < \varphi(m) \\ a^{b \bmod \varphi(m) + \varphi(m)} & ,b \geqslant \varphi(m) \end{cases} \pmod m$

```
//求a^b mod m
//如果爆ll可用龟速乘替换快速幂中的乘法
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

ll euler(ll n){
    ll ans=n;
    for(ll p=2;p*p<=n;p++){
        if(n%p==0){
            ans=ans/p*(p-1);
            while(n%p==0) n/=p;
        }
    }
    if(n!=1) ans=ans/n*(n-1);
    return ans;
}

ll fpow(ll a,ll b,ll mod){
    if(mod==1) return 0;
```

```
    ll ans=1%mod;
    while(b){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ans;
}

ll read(ll m){
    register ll x=0,f=0;char ch=getchar();
    while(!isdigit(ch)) ch=getchar();
    while(isdigit(ch)){
        x=x*10+ch-'0';
        if(x>=m) f=1;
        x%=m;ch=getchar();
    }
    return x+(f==1?m:0);
}

int main(){
    ll a,b,m;cin>>a>>m;
    a%=m;
    ll phi=euler(m);
    b=read(phi);
    cout<<fpow(a,b,m)<<"\n";
    return 0;
}
```

素数筛

埃氏筛

时间复杂度：$O(nloglogn)$

```
const int N=1e6+10; //表长
int prime[N],cnt=0; //prime数组存放所以素数，cnt为素数个数
bool st[N]; //false为素数
void get_prime(int n){
    st[1]=1;
    for(int i=2;i<=n;i++){
        if(!st[i]){
            prime[cnt++]=i; //把素数i存到prime数组中
            for(int j=i+i;j<=n;j+=i) st[j]=true; //筛去所有i的倍数
        }
    }
}
```

欧拉筛(线性筛)

时间复杂度：$O(n)$

```
const int N=1e6+10; //表长
int prime[N],cnt=0; //prime数组存放所以素数，cnt为素数个数
bool st[N]; //false为素数
void get_prime(int n){
    st[1]=1;
    for(int i=2;i<=n;i++){
        if(!st[i]) prime[cnt++]=i; //把素数i存到prime数组中
        for(int j=0;j<cnt&&i*prime[j]<=n;j++){
            st[i*prime[j]]=true; //找到的素数的倍数不访问
            if(i%prime[j]==0) break; //关键代码
        }
    }
}
```

区间筛法

求区间[a,b)内有多少个素数

```
typedef long long ll;
bool is_prime[N],is_prime_small[N];
// is_prime[i-a]表示i是素数
//[a,b)
void segnment_sieve(ll a,ll b){
    for(ll i=0;i*i<b;i++) 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) is_prime_small[j]=false; //筛[2,√b)
            for(ll j=max(2LL,(a+i-1)/i)*i;j<b;j+=i) is_prime[j-a]=false; //筛[a,b)
        }
    }
}
```

欧拉函数

$$\varphi(n)=\sum_{i=1}^{n}[\gcd(n,i)=1]$$

1~n 中与 n 互质的数的个数

求单个欧拉函数

$\varphi(n)=n\prod_{i=1}^{k}(1-\frac{1}{p_i})$

```
int euler(int n){
    int ans=n;
    for(int p=2;p*p<=n;p++){
        if(n%p==0){
            ans=ans/p*(p-1);
            while(n%p==0) n/=p;
        }
    }
    if(n!=1) ans=ans/n*(n-1);
    return ans;
}
```

筛法求欧拉函数

```
ll phi[N];
int prime[N],cnt=0; //prime数组存放所以素数，cnt为素数个数
bool st[N]; //false为素数
void get_phi(int n){
    st[1]=1;phi[1]=1;
    for(int i=2;i<=n;i++){
        if(!st[i]) prime[cnt++]=i,phi[i]=i-1; //把素数i存到prime数组中
        for(int j=0;j<cnt&&i*prime[j]<=n;j++){
            st[i*prime[j]]=true; //找到的素数的倍数不访问
            if(i%prime[j]==0){
                phi[i*prime[j]]=phi[i]*prime[j];
                break;
            }
            phi[i*prime[j]]=phi[i]*phi[prime[j]];
        }
    }
}
```

莫比乌斯函数

设$n=\prod_{i=1}^{m} p_i^{c_i}$

$$\mu(n)=\begin{cases}1 & n=1\\ (-1)^m & n\text{含有次数大于 1 的质因子(即}c_1=c_2=\cdots=c_m=1)\\ 0 & \text{其他情况（n 不为 1 且不含次数大于 1 的质因子）}\end{cases}$$

筛法求莫比乌斯函数

```
ll mu[N];
int prime[N],cnt=0; //prime数组存放所以素数，cnt为素数个数
bool st[N]; //false为素数
void get_mu(int n){
    st[1]=1;mu[1]=1;
    for(int i=2;i<=n;i++){
        if(!st[i]) prime[cnt++]=i,mu[i]=-1; //把素数i存到prime数组中
        for(int j=0;j<cnt&&i*prime[j]<=n;j++){
            st[i*prime[j]]=true; //找到的素数的倍数不访问
            if(i%prime[j]==0){
                mu[i*prime[j]]=0;
                break;
            }
            mu[i*prime[j]]=-mu[i];
        }
    }
}
```

线性同余方程

关于x的线性同余方程$ax\equiv b \pmod m$的最小正整数解

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

ll exgcd(ll a, ll b, ll &x, ll &y){
    if (b == 0){
        x = 1;
        y = 0;
```

```
        return a;
    }
    ll g = exgcd(b, a % b, x, y);
    ll temp = x;     //存放x的值
    x = y;     //更新x = y(old)
    y = temp - a / b * y;     //更新y = x(old) - a / b * y(old)
    return g;
}

int main(){
    ll a,b,m,x,y;
    cin>>a>>b>>m;
    ll gcd=exgcd(a,m,x,y);
    if(b%gcd!=0) cout<<"impossible\n";
    else x=x*b/gcd%m,cout<<(x%m+m)%m<<"\n";
    return 0;
}
```

中国剩余定理

$$\begin{cases} x \equiv a_1(mod\ m_1) \\ x \equiv a_2(mod\ m_2) \\ x \equiv a_3(mod\ m_3) \\ \quad ... \\ x \equiv a_n(mod\ m_n) \end{cases}$$

$m_1, m_2, m_3, .., m_n$两两互质

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=105;
int n,m[N],a[N],d;
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(!b){
        x=1,y=0;
        return a;
    }
    ll gcd=exgcd(b,a%b,x,y),temp=x;
    x=y,y=temp-a/b*y;
    return gcd;
}

ll inv(ll a,ll p){
    ll x,y;
    if(exgcd(a,p,x,y)!=1) return -1; //无解的情况
    return (x%p+p)%p;
}

ll CRT(){
    ll lcm=1,x=0;
    for(int i=1;i<=n;i++) lcm=lcm*m[i];
    for(int i=1;i<=n;i++){
        ll r=lcm/m[i];
        x+=a[i]%lcm*r%lcm*inv(r,m[i])%lcm;
        x%=lcm;
    }
    return x;
}

int main(){
    cin>>n;
```

```
    for(int i=1;i<=n;i++){
        cin>>m[i]>>a[i];
    }
    cout<<CRT()<<"\n";
    return 0;
}
```

扩展中国剩余定理

$$\begin{cases} x \equiv a_1(mod\ m_1) \\ x \equiv a_2(mod\ m_2) \\ x \equiv a_3(mod\ m_3) \\ \ \ \cdots \\ x \equiv a_n(mod\ m_n) \end{cases}$$

其中m_1,m_2,m_3...m_k为**不一定两两互质**的整数

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5+5;
int n;
ll m[N],a[N];

ll exgcd(ll a, ll b, ll &x, ll &y){
    if (b == 0){
        x = 1;
        y = 0;
        return a;
    }
    ll g = exgcd(b, a % b, x, y);
    ll temp = x;    //存放x的值
    x = y;    //更新x = y(old)
    y = temp - a / b * y;    //更新y = x(old) - a / b * y(old)
    return g;
}

ll mul(ll a,ll b,ll mod){
    ll ans=0; //保存结果
    ll base=a; //每一次进行加的数字
    while(b){
        if(b&1) ans=(ans+base)%mod;
        base=(base*2)%mod;
        b>>=1;
    }
    return ans;
}

ll excrt(){
    ll A=a[1],M=m[1];
    ll x,y;
    for(int i=2;i<=n;i++){
        ll ai=M,bi=m[i],c=((a[i]-A)%bi+bi)%bi;
        ll gcd=exgcd(ai,bi,x,y),bg=bi/gcd;
        if(c%gcd) return -1;
        x=mul(x,c/gcd,bg);
        A+=x*M;//更新前k个方程组的答案
        M*=bg;//M为前k个m的lcm
        A=(A%M+M)%M;
    }
    return (A%M+M)%M;
}
```

```
int main(){
    cin>>n;
    for(int i=1;i<=n;i++) cin>>m[i]>>a[i];
    cout<<excrt()<<"\n";
    return 0;
}
```

BSGS

求最小的非负整数x,使得$A^x \equiv B(mod\ P)$ (P是素数)

```
#include <bits/stdc++.h>
using namespace std;
__int128 fpow(__int128 a,__int128 b,__int128 mod){
    if(mod==1) return 0;
    __int128 ans=1%mod;
    while(b){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ans;
}

__int128 BSGS(__int128 a,__int128 b,__int128 p){
    if(b==1) return 0;
    unordered_map<__int128,__int128> hash;
    b%=p;
    __int128 t=(__int128)ceil(sqrt((double)p));
    __int128 val=b;
    for(__int128 j=0;j<t;j++){
        hash[val]=j;
        val=val*a%p;
    }
    a=fpow(a,t,p);
    if(!a) return(b==0)?1:-1;
    val=1;
    for(__int128 i=0;i<=t;i++){
        __int128 j=hash.find(val)==hash.end()?-1:hash[val];
        if(j>=0&&i*t-j>=0) return i*t-j;
        val=val*a%p;
    }
    return -1;
}

inline __int128 read() {
    __int128 x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9') {
        if(ch=='-')
            f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9') {
        x=x*10+ch-'0';
        ch=getchar();
    }
    return x*f;
}

inline void write(__int128 x) {
```

```
    if(x<0) {
        putchar('-');
        x=-x;
    }
    if(x>9) write(x/10);
    putchar(x%10+'0');
}

__int128 A,B,P;

int main(){
    A=read(),B=read(),P=read();
    __int128 x=BSGS(A,B,P);
    write(x);
    return 0;
}
```

```
const int UN = 77773;
int hs[UN],head[UN], nxt[UN], id[UN],tp;
void insert(int x,int y) {
    int k=x%UN;
    hs[tp]=x,id[tp]=y,nxt[tp]=head[k],head[k]=tp++;
}

int find(int x) {
    int k=x%UN;
    for(int i=head[k];i!=-1;i=nxt[i]) if(hs[i]==x) return id[i];
    return -1;
}

ll BSGS(ll a,ll b,ll p) {
    memset(head,-1,sizeof head);
    tp=1;b%=p;
    if(b==1) return 0;
    ll m=sqrt(p*1.0),x=1,val=1;
    for(ll i=0;i<m;i++,val=val*a%p) insert(val*b%p,i);
    for(ll i=m,j;;i+=m) {
        if((j=find(x=x*val%p))!=-1) return i-j;
        if(i>p) break;
    }
    return -1;
}
```

```
struct Hash{
    static const int MOD=114514;  //77773
    static const int maxn=1e7+5;
    int tot,head[MOD+10],next[maxn],h[maxn],val[maxn];
    void clear(){tot=0;memset(head,-1,sizeof(head));}
    inline void insert(int H,int VAL){
        for(int i=head[H%MOD];i!=-1;i=next[i])
        if(h[i]==H){
            val[i]=VAL;return;
        }
        h[++tot]=H;val[tot]=VAL;next[tot]=head[H%MOD];head[H%MOD]=tot;
    }
    inline int find(int H){
        for(int i=head[H%MOD];i!=-1;i=next[i])if(h[i]==H)return val[i];
        return -1;
    }
}Hash;
```

```
ll BSGS(ll a,ll b,ll p){
    if(b==1) return 0;
    Hash.clear();
    b%=p;
    ll t=(ll)ceil(sqrt((double)p));
    ll val=b;
    for(ll j=0;j<t;j++){
        Hash.insert(val,j);
        val=val*a%p;
    }
    a=fpow(a,t,p);
    if(!a) return(!b)?1:-1;
    val=1;
    for(ll i=0;i<=t;i++){
        ll j=Hash.find(val);
        if(j>=0&&i*t-j>=0) return i*t-j;
        val=val*a%p;
    }
    return -1;
}
```

扩展BSGS

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

ll fpow(ll a,ll b,ll mod){
    if(mod==1) return 0;
    ll ans=1%mod;
    while(b){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ans;
}

ll exgcd(ll a,ll b,ll &x,ll &y){
    if(!b){
        x=1,y=0;
        return a;
    }
    ll gcd=exgcd(b,a%b,x,y),temp=x;
    x=y,y=temp-a/b*y;
    return gcd;
}

ll inv(ll a,ll p){
    ll x,y;
    if(exgcd(a,p,x,y)!=1) return -1; //无解的情况
    return (x%p+p)%p;
}

ll BSGS(ll a,ll b,ll p){
    unordered_map<ll,ll> hash;
    b%=p;
    ll t=(ll)ceil(sqrt((double)p));
    ll val=b;
    for(ll j=0;j<t;j++){
        hash[val]=j;
```

```
        val=val*a%p;
    }
    a=fpow(a,t,p);
    if(!a) return(!b)?1:-1;
    val=1;
    for(ll i=0;i<=t;i++){
        ll j=hash.find(val)==hash.end()?-1:hash[val];
        if(j>=0&&i*t-j>=0) return i*t-j;
        val=val*a%p;
    }
    return -1;
}

ll EXBSGS(ll a,ll b,ll p){
    ll tot=1,cnt=0,g=__gcd(a,p);
    a%=p;b%=p;                   //防止超时
    if(b==1||p==1) return 0;
    while(g>1){
        if(b%g) return -1;    //无法整除则无解
        cnt++;b/=g;p/=g;tot=tot*(a/g)%p;
        if(tot==b) return cnt;   //tot=b说明前面的a的次数为0，只需要返回k
        g=__gcd(a,p);
    }
    ll res=BSGS(a,b*inv(tot,p)%p,p);
    if(~res) return res+cnt;
    return res;
}

ll A,B,P;

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    while(cin>>A>>P>>B){
        if(!A&&!B&&!P) break;
        ll x=EXBSGS(A,B,P);
        if(~x) cout<<x<<"\n";
        else cout<<"No Solution\n";
    }
    return 0;
}
```

数论分块

$\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    long long n,ans=0;
    cin >> n;
    for(long long l=1,r;l<=n;l=r+1){
        r = n/(n/l);             //计算r，让分块右移
        ans += (r-l+1)*(n/l);    //求和
        cerr << l <<" "<< r <<": "<< n/r << endl;   //打印分块
    }
    cout << ans;                 //打印和
}
```

$\sum_{i=1}^{n}\lfloor\frac{w}{i}\rfloor$

```
ll ans=0;
for(ll l=1,r;l<=n;l=r+1){
    if(w/l) r=w/(w/l);
    else r=n;
    r=min(r,n);
    ans+=(r-l+1)*(w/l);
}
```

$\sum_{i=1}^{n}\lfloor\frac{a}{i}\rfloor\lfloor\frac{b}{i}\rfloor$

```
ll ans=0;
for(ll l=1,r,r1,r2;l<=n;l=r+1){
    if(a/l) r1=a/(a/l);
    else r1=n;
    if(b/l) r2=b/(b/l);
    else r2=n;
    r=min(r1,r2);
    r=min(r,n);
    ans+=(r-l+1)*(a/l)*(b/l);
}
```

$\sum_{i=1}^{n}\lceil\frac{w}{i}\rceil=\sum_{i=1}^{n}\lfloor\frac{w+i-1}{i}\rfloor=\sum_{i=1}^{n}\lfloor\frac{w-1}{i}\rfloor+1$

```
ll ans=0;
for(ll l=1,r;l<=n;l=r+1){
    if((w-1)/l) r=(w-1)/((w-1)/l);
    else r=n;
    r=min(r,n);
    ans+=(r-l+1)*((w-1)/l+1);
}

```

$\sum_{i=1}^{n}i\lfloor\frac{w}{i}\rfloor$

```
ll ans=0;
for(ll l=1,r;l<=n;l=r+1){
    if(w/l) r=w/(w/l);
    else r=n;
    r=min(r,n);
    ans+=(r-l+1)*(l+r)/2*(w/l);
}
```

高斯消元

高斯消元解线性方程组

$$\begin{cases}a_{1,1}x_1+a_{1,2}x_2+\cdots+a_{1,n}x_n=b_1\\a_{2,1}x_1+a_{2,2}x_2+\cdots+a_{2,n}x_n=b_2\\\cdots\\a_{n,1}x_1+a_{n,2}x_2+\cdots+a_{n,n}x_n=b_n\end{cases}$$

```
#include <bits/stdc++.h>
using namespace std;
const int N=105;
const double eps=1e-6;

int n;
```

```
double a[N][N];

int guess(){
    int r,c; //r代表行,c代表列
    for(c=0,r=0;c<n;c++){
        int t=r;
        //找当这一列绝对值最大的这一行
        for(int i=r;i<n;i++){
            if(fabs(a[i][c])>fabs(a[t][c])) t=i;
        }
        //如果最小的是0
        if(fabs(a[t][c])<eps) continue;
        //把这一行换到最上面
        for(int i=c;i<=n;i++) swap(a[t][i],a[r][i]);
        for(int i=n;i>=c;i--) a[r][i]/=a[r][c];
        for(int i=r+1;i<n;i++){
            if(fabs(a[i][c])>eps){
                for(int j=n;j>=c;j--){
                    a[i][j]-=a[r][j]*a[i][c];
                }
            }
        }
        r++;
    }
    if(r<n){
        for(int i=r;i<n;i++){
            if(fabs(a[i][n])>eps) return -1;
        }
        return 0;
    }
    for(int i=n-1;i>=0;i--){
        for(int j=i+1;j<n;j++){
            a[i][n]-=a[i][j]*a[j][n];
        }
    }
    return 1;
}

void solve(){
    cin>>n;
    for(int i=0;i<n;i++)
        for(int j=0;j<=n;j++)
            cin>>a[i][j];
    int t=guess(); //t=-1无解 t=0无数解 t=1唯一解
    if(!t) cout<<"Infinite group solutions\n";
    else if(t==-1) cout<<"No solution\n";
    else for(int i=0;i<n;i++) cout<<fixed<<setprecision(2)<<a[i][n]<<"\n";
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    solve();
    return 0;
}
```

高斯消元解异或线性方程组

$$\begin{cases} a_{1,1}x_1 \bigoplus a_{1,2}x_2 \bigoplus \cdots \bigoplus a_{1,n}x_n = b_1 \\ a_{2,1}x_1 \bigoplus a_{2,2}x_2 \bigoplus \cdots \bigoplus a_{2,n}x_n = b_2 \\ \cdots \\ a_{n,1}x_1 \bigoplus a_{n,2}x_2 \bigoplus \cdots \bigoplus a_{n,n}x_n = b_n \end{cases}$$

```
#include <bits/stdc++.h>
using namespace std;
const int N=105;

int n;
bitset<N> a[N];

int guass(){
    int r,c; //r代表行,c代表列
    for(c=0,r=0;c<n;c++){
        int t=-1;
        //找当这一列绝对值最大的这一行
        for(int i=r;i<n;i++){
            if(a[i][c]){
                t=i;
                break;
            }
        }
        //如果最小的是0
        if(t==-1) continue;
        //把这一行换到最上面
        swap(a[t],a[r]);

        for(int i=r+1;i<n;i++){
            if(a[i][c]){
                a[i]^=a[r];
            }
        }
        r++;
    }
    if(r<n){
        for(int i=r;i<n;i++){
            if(a[i][n]) return -1;
        }
        return 0;
    }

    for(int i=n-1;i>=0;i--){
        if(!a[i][i]) a[i][n]=1; //无数解 add
        for(int j=i+1;j<n;j++){
            a[i][n]=a[i][n]^(a[i][j]&a[j][n]);
        }
    }
    return 1;
}

void solve(){
    cin>>n;
    for(int i=0;i<n;i++){
        for(int j=0;j<=n;j++){
            int c;cin>>c;
            a[i][j]=c;
        }
    }
    int t=guass(); //t=-1无解 t=0无数解 t=1唯一解
    if(!t) cout<<"Multiple sets of solutions\n";
    else if(t==-1) cout<<"No solution\n";
    else for(int i=0;i<n;i++) cout<<a[i][n]<<"\n";
}


int main(){
```

```
    ios::sync_with_stdio(0);cin.tie(0);
    solve();
    return 0;
}
```

线性基

性质

1. 线性基没有异或为0的子集。
2. 线性基的异或集合中每个元素的异或方案唯一，其实这个跟性质1是等价的。
3. 线性基二进制最高位互不相同。
4. 线性基中元素互相异或，异或集合不变。

```
struct Linear_Basis{
    int zero;
    ll p[65],d[65];//d[i]是重建后的线性基
    int cnt;
    Linear_Basis() {memset(p,0,sizeof p);cnt = 0;zero=0;}
    ~Linear_Basis() {}
    void init(){
        memset(p,0,sizeof p);
        cnt = 0;
        zero = 0;
    }
    bool ins(ll x){ //向线性基中插入一个数
        for(int i = 62; i >= 0; i--){
            if(x&(1ll<<i)){
                if(!p[i]) {p[i] = x; break;}
                x ^= p[i];
            }
        }
        if(!x) zero=1;
        return x > 0ll;
    }
    ll MAX(ll x){ //求线性空间与ans异或的最大值
        for(int i = 62; i >= 0; i--)
            if((x^p[i]) > x) x ^= p[i];
        return x;
    }
    //如果是求一个数与线性基的异或最小值，则需要先rebuild，再从高位向低位依次进行异或
    ll MIN(){
        if(zero) return 0;
        rep(i,0,62)
            if(p[i]) return p[i];
    }
    //将线性基改造成每一位相互独立，即对于二进制的某一位i，只有pi的这一位是1，其它都是0
    void rebuild(){
        cnt = 0;
        for(int i = 62; i >= 0; i--)
            for(int j = i-1; j >= 0; j--)
                if(p[i]&(1ll<<j))
                    p[i]^=p[j];
        rep(i,0,62)
            if(p[i]) d[cnt++] = p[i];
    }
    //求线性基能够组成的数中的第K小
    ll Kth(ll k){
        k=k-zero;
        if(!k) return 0ll;
```

```
        ll ret = 0;
        if(k >= (1ll<<cnt)) return -1; //k大于子集总数, 找不到
        for(int i = 62; i >= 0; i--)
            if(k&(1ll<<i)) ret ^= d[i];
        return ret;
    }
    //查询排名
    int rank(ll x) {
        ll ans = 0;
        for(int i = cnt - 1; i >= 0; i --)
            if(x >= d[i]) ans += (1 << i), x ^= d[i];
        return ans + zero;
    }
    //合并两个线性基
    Linear_Basis& merge(const Linear_Basis &xx){
        for(int i = 62; i >= 0; i--)
            if(xx.p[i]) ins(xx.p[i]);
        return *this;
    }
}LB;

//两个线性基求交
Linear_Basis merge(Linear_Basis a, Linear_Basis b){
    Linear_Basis A = a, tmp = a, ans; //tmp不断构建A+(B\ans)
    ll cur,d;
    rep(i,0,33) //从低到高，使得不存在一个基底可以仅由(tmp\A)表示
        if(b.p[i]){ //b中有这个基底
            cur = 0, d = b.p[i];
            per(j,i,0)
                if((d>>j)&1){
                    if(tmp.p[j]){
                        d ^= tmp.p[j], cur ^= A.p[j];
                        if(d) continue;
                        ans.p[i] = cur; //cur的第i位不为0
                    }
                    else tmp.p[j] = d, A.p[j] = cur; //如果不能被表示, A的赋值是为了让高位中含有j这位的基底下放到A中j的位置
                    break;
                }
        }
    return ans;
}
```

特殊计数

Catalan数(卡特兰数)

公式

1. $H_n = \frac{C_{2n}^n}{n+1}, n = 0, 1, 2, \cdots.$

前一部分Catalan数是
1,1,2,5,14,42,132,429,1430,4862,16796,58786,208012,742900,2674440,9694845,35357670

2. $H_n = \frac{4n-2}{n+1} H_{n-1}$
3. $H_n = C_{2n}^n - C_{2n}^{n+1} = C_{2n}^n - C_{2n}^{n-1}$
4. $H_n = H_0 H_{n-1} + H_1 H_{n-2} + \cdots + H_{n-2} H_1 + H_{n-1} H_0 = \sum_{i=0}^n H_i H_{n-i}$， $H_0 = 1$
5. $H_n = \frac{(2n)!}{(n+1)!n!}$

Stirling数(斯特林数)

第一类

定义:把n个不同的元素分配到k个圆的排序里，圆不能为空的分法数量

公式:$S(n,k)=S(n-1,k-1)+(n-1)S(n-1,k), 1\leqslant k\leqslant n$

$S(0,0)=1, S(k,0)=0, 1\leqslant k\leqslant n$

第二类

定义:把n个不同的球分配到k个相同的盒子里，不能有空盒子的分法数量

公式:$S(n,k)=kS(n-1,k)+S(n-1,k-1), 1\leqslant k\leqslant n$

$S(0,0)=1, S(i,0)=0, 1\leqslant i\leqslant n$

数据结构

分块

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+5;
ll a[N];
int op[N],ed[N];//第i个块的左右端点
int id[N];//a[i]属于哪一块
ll sum[N];//第i个块的和
ll mark[N];//第i个块的标记

void add(int l,int r,int k){
    if(id[l]==id[r]){
        for(int i=l;i<=r;i++){
            a[i]+=k;
            sum[id[i]]+=k;
        }
        return;
    }
    for(int i=l;i<=ed[id[l]];i++){
        a[i]+=k;
        sum[id[i]]+=k;
    }
    for(int i=op[id[r]];i<=r;i++){
        a[i]+=k;
        sum[id[i]]+=k;
    }
    for(int i=id[l]+1;i<id[r];i++) mark[i]+=k;
}

//[l,r]的区间和
ll query(int l,int r){
    ll ans=0;
    if(id[l]==id[r]){
        for(int i=l;i<=r;i++){
            ans+=a[i]+mark[id[i]];
        }
        return ans;
    }
    for(int i=l;i<=ed[id[l]];i++){
        ans+=a[i]+mark[id[i]];
```

```cpp
    }
    for(int i=op[id[r]];i<=r;i++){
        ans+=a[i]+mark[id[i]];
    }
    for(int i=id[l]+1;i<id[r];i++) ans+=sum[i]+mark[i]*(ed[i]-op[i]+1);
    return ans;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    int n,m;cin>>n>>m;
    int len=sqrt(n),siz=(n+len-1)/len;
    for(int i=1;i<=siz;i++){
        op[i]=len*(i-1)+1;
        ed[i]=len*i;
    }
    ed[siz]=n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        id[i]=(i-1)/len+1;
        sum[id[i]]+=a[i];
    }
    while(m--){
        char op;cin>>op;
        if(op=='C'){
            int l,r,c;cin>>l>>r>>c;
            add(l,r,c);
        }
        else {
            int l,r;cin>>l>>r;
            cout<<query(l,r)<<"\n";
        }
    }
    return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e6+5;
ll a[N];
int op[N],ed[N];//第i个块的左右端点
int id[N];//a[i]属于哪一块
ll mark[N];//第i个块的标记
vector<ll> v[N];

void update(int x){
    for(int i=0;i<v[x].size();i++) v[x][i]=a[op[x]+i];
    sort(v[x].begin(),v[x].end());
}

void add(int l,int r,int k){
    if(id[l]==id[r]){
        for(int i=l;i<=r;i++) a[i]+=k;
        update(id[l]);
        return;
    }
    for(int i=l;i<=ed[id[l]];i++) a[i]+=k;
    update(id[l]);
    for(int i=op[id[r]];i<=r;i++) a[i]+=k;
    update(id[r]);
    for(int i=id[l]+1;i<id[r];i++) mark[i]+=k;
```

```
}

//[l,r]大于等于k的个数
ll query(int l,int r,ll k){
    ll ans=0;
    if(id[l]==id[r]){
        for(int i=l;i<=r;i++){
            if(a[i]+mark[id[l]]>=k) ans++;
        }
        return ans;
    }
    for(int i=l;i<=ed[id[l]];i++){
        if(a[i]+mark[id[i]]>=k) ans++;
    }
    for(int i=op[id[r]];i<=r;i++){
        if(a[i]+mark[id[i]]>=k) ans++;
    }
    for(int i=id[l]+1;i<id[r];i++){
        ans+=v[i].end()-lower_bound(v[i].begin(),v[i].end(),k-mark[i]);
    }
    return ans;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    int n,m;cin>>n>>m;
    int len=sqrt(n),siz=(n+len-1)/len;
    for(int i=1;i<=siz;i++){
        op[i]=len*(i-1)+1;
        ed[i]=len*i;
    }
    ed[siz]=n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        id[i]=(i-1)/len+1;
        v[id[i]].push_back(a[i]);
    }
    for(int i=1;i<=siz;i++) sort(v[i].begin(),v[i].end());
    while(m--){
        char op;cin>>op;
        if(op=='M'){
            int l,r,c;cin>>l>>r>>c;
            add(l,r,c);
        }
        else {
            int l,r,c;cin>>l>>r>>c;
            cout<<query(l,r,1ll*c)<<"\n";
        }
    }
    return 0;
}
```

莫队

```
//求区间内这一段中有多少不同的数字
#include<bits/stdc++.h>
typedef long long ll;
using namespace std;
const int N=5e4+5,M=2e5+5,W=1e6+5;

ll a[N],id[N];
```

```
ll res[W],ans[M];
ll cnt;

struct query {
    int l,r,id;
}q[M];

// bool cmp(query a, query b) {
//     if(id[a.l]==id[b.l]) {
//         if(id[a.l]&1) return a.r<b.r;
//         return a.r>b.r;
//     }
//     return id[a.l]<id[b.l];
// }

bool cmp(query a, query b) {
    return (id[a.l] ^ id[b.l]) ? id[a.l] < id[b.l] : ((id[a.l] & 1) ? a.r < b.r : a.r >
b.r);
}

// void del(int pos){
//     res[a[pos]]--;
//     if(!res[a[pos]]) cnt--;
// }
// void add(int pos){
//  if(!res[a[pos]]) cnt++;
//     res[a[pos]]++;
// }

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    int n;cin>>n;
    int len=sqrt(n),siz=(n+len-1)/len;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        id[i]=(i-1)/len+1;
    }
    int m;cin>>m;
    for(int i=1;i<=m;i++) {
        cin>>q[i].l>>q[i].r;
        q[i].id=i;
    }
    sort(q+1,q+1+m,cmp);
    int l=1,r=0;
    for(int i=1;i<=m;i++) {
        int ql=q[i].l,qr=q[i].r;
        while(l < ql) cnt -= !--res[a[l++]];
        while(l > ql) cnt += !res[a[--l]]++;
        while(r < qr) cnt += !res[a[++r]]++;
        while(r > qr) cnt -= !--res[a[r--]];
//      while(l>q[i].l) add(--l);
//      while(r<q[i].r) add(++r);
//      while(l<q[i].l) del(l++);
//      while(r>q[i].r) del(r--);
        ans[q[i].id]=cnt;
    }
    for(int i=1;i<=m;i++) cout<<ans[i]<<"\n";
    return 0;
}
```

并查集

```
void init(int n){
    for(int i=1;i<=n;i++) fa[i]=i;
}
//查询树的根
int find(int x){
    if(fa[x]==x) return x;
    return fa[x] = find(fa[x]);
}
//合并a和b所属的集合
void unite(int a,int b){
    a=find(a),b=find(b);
    fa[a]=fa[b];
}
//判断a和b是否属于同一个集合
bool same(int a,int b){
    return find(a)==find(b);
}
```

```
struct DSU {
    vector<int> f, siz;
    DSU(int n) : f(n), siz(n, 1) {iota(f.begin(), f.end(), 0);}
    int find(int x) {return x == f[x] ? x : f[x] = find(f[x]);}
    bool same(int x, int y) {return find(x) == find(y);}
    bool merge(int x, int y) {
        x = find(x), y = find(y);
        if (x == y) return false;
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }
    int size(int x) {return siz[find(x)];}
};
```

路径压缩（查询）+启发式合并（按秩合并）

```
int fa[N],depth[N];
void init(int n){
    for(int i=1;i<=n;i++)
        fa[i]=i,depth[i]=1;
}
//查询树的根
int find(int x){
    if(x!=fa[x]) fa[x]=find(fa[x]);
    return fa[x];
}
//合并a和b所属的集合
void unite(int a,int b){
    a=find(a),b=find(b);
    if(depth[a]==depth[b]){
        depth[a]=depth[a]+1;
        fa[b]=a;
    }
    else{
        if(depth[a]<depth[b]) fa[a]=b;
        else fa[b]=a;
    }
}
//判断a和b是否属于同一个集合
```

```
bool same(int a,int b){
    return find(a)==find(b);
}
```

带权并查集（维护连通块大小）

```
int find(int x){
    if(x!=fa[x]){
        int root=find(fa[x]);
        d[x]+=d[fa[x]];
        fa[x]=root;
    }
    return fa[x];
}

void unite(int a,int b){
    a=find(a),b=find(b);
    d[a]=siz[b];
    siz[b]+=siz[a];
    fa[a]=fa[b];
}
```

树状数组

单点修改，区间查询

```
#include <bits/stdc++.h>
using namespace std;
const int N=500010;
int q[N],t[N];
int n,m;

// 算出x二进制的从右往左出现第一个1以及这个1之后的那些0组成数的二进制对应的十进制的数
inline int lowbit(int x){
    return x&(-x);
}

void add(int x,int v){    //在x位置加上v
    while(x<=n){
        t[x]+=v;
        x+=lowbit(x);
    }
}

//求前缀和
int getsum(int x){
    int res=0;
    while(x>0){
        res+=t[x];
        x-=lowbit(x);
    }
    return res;
}

int main(){
    int k,a,b;
    cin>>n>>m;
    for(int i=1;i<=n;i++) {
        scanf("%d",&q[i]);
        add(i,q[i]);   //树状数组
    }
```

```
    for(int i=0;i<m;i++){
        scanf("%d%d%d",&k,&a,&b);
        if(k==2) printf("%d\n",getsum(b)-getsum(a-1));
        else if(k==1) add(a,b);
    }
    return 0;
}
```

```
struct BIT{
    vector<int> bit;

    BIT(int n) : bit(n + 1) {}

    void update(int pos, int val) {
        for (int i = pos; i < (int)bit.size(); i += i & (-i)) {
            bit[i] += val;
        }
    }

    int query(int l, int r) {
        return query(r) - query(l - 1);
    }

    int query(int pos) {
        int res = 0;
        for (int i = pos; i >= 1; i -= i & (-i)) {
            res += bit[i];
        }
        return res;
    }
};
```

区间修改，单点查询

```
#include <bits/stdc++.h>
using namespace std;
const int N=500010;
int q[N],d[N],tree[N],n,m;

inline int lowbit(int x){
    return x&(-x);
}

void add(int p,int x){
    for(int i=p;i<=n;i+=lowbit(i)) tree[i]+=x;
}

int getsum(int p){
    int ans=0;
    for(int i=p;i;i-=lowbit(i)) ans+=tree[i];
    return ans;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    for(int i=1;i<=n;i++) {
        cin>>q[i];
        d[i]=q[i]-q[i-1];
        add(i,d[i]);
```

```
    }
    for(int i=0;i<m;i++){
        int op,x,y,k;
        cin>>op;
        if(op==1) {
            cin>>x>>y>>k;
            add(x,k),add(y+1,-k);
        }
        else{
            cin>>x;
            cout<<getsum(x)<<"\n";
        }
    }
    return 0;
}
```

线段树

区间修改，区间查询

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+10;
ll a[N];
ll tree[N<<2];
ll lazy[N<<2];

inline ll ls(ll x){return x<<1;}
inline ll rs(ll x){return x<<1|1;}

inline void pushup(ll u){
    tree[u]=tree[ls(u)]+tree[rs(u)];
}

inline void build(ll u,ll ul,ll ur){
    lazy[u]=0;
    if(ul==ur){tree[u]=a[ul];return;}
    ll mid=(ul+ur)>>1;
    build(ls(u),ul,mid);
    build(rs(u),mid+1,ur);
    pushup(u);
}

inline void addlazy(ll u,ll ul,ll ur,ll v){
    lazy[u]+=v;
    tree[u]+=v*(ur-ul+1);
}

inline void pushdown(ll u,ll ul,ll ur){
    if(lazy[u]){
        ll mid=(ul+ur)>>1;
        addlazy(ls(u),ul,mid,lazy[u]);
        addlazy(rs(u),mid+1,ur,lazy[u]);
        lazy[u]=0;
    }
}

//区间修改
inline void update(ll l,ll r,ll u,ll ul,ll ur,ll v){
    if(l<=ul&&ur<=r){
```

```
        addlazy(u,ul,ur,v);
        return;
    }
    pushdown(u,ul,ur);
    ll mid=(ul+ur)>>1;
    if(l<=mid) update(l,r,ls(u),ul,mid,v);
    if(r>mid) update(l,r,rs(u),mid+1,ur,v);
    pushup(u);
}

//区间查询
inline ll query(ll l,ll r,ll u,ll ul,ll ur){
    if(l<=ul&&ur<=r) return tree[u];
    pushdown(u,ul,ur);
    ll res=0;
    ll mid=(ul+ur)>>1;
    if(l<=mid) res+=query(l,r,ls(u),ul,mid);
    if(r>mid) res+=query(l,r,rs(u),mid+1,ur);
    return res;
}

int main(){
    ll n,m;cin>>n>>m;
    for(ll i=1;i<=n;i++) scanf("%lld",&a[i]);
    build(1,1,n);
    while(m--){
        int t;scanf("%d",&t);
        ll l,r,v;
        if(t==1){
            scanf("%lld%lld%lld",&l,&r,&v);
            update(l,r,1,1,n,v);
        }
        else{
            scanf("%lld%lld",&l,&r);
            printf("%lld\n",query(l,r,1,1,n));
        }
    }
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=200005;

int n,m;
ll a[N];
struct node{
    int l,r;
    ll lazy,v;
}tree[N<<2];

void pushup(int u){
    tree[u].v=tree[u<<1].v+tree[u<<1|1].v;
}

void addlazy(int u,ll v){
    tree[u].lazy+=v;
    tree[u].v+=v*(tree[u].r-tree[u].l+1);
}

void pushdown(int u){
```

```
    if(tree[u].lazy){
        addlazy(u<<1,tree[u].lazy);
        addlazy(u<<1|1,tree[u].lazy);
        tree[u].lazy=0;
    }
}

void build(int u,int l,int r){
    tree[u].l=l,tree[u].r=r;
    if(l==r){
        tree[u].v=a[l];
        return;
    }
    int mid=(l+r)>>1;
    build(u<<1,l,mid);build(u<<1|1,mid+1,r);
    pushup(u);
}

void modify(int u,int l,int r,ll v){
    if(l<=tree[u].l&&tree[u].r<=r){
        addlazy(u,v);
        return;
    }
    pushdown(u);
    int mid=(tree[u].l+tree[u].r)>>1;
    if(r<=mid) modify(u<<1,l,r,v);
    else if(l>mid) modify(u<<1|1,l,r,v);
    else modify(u<<1,l,r,v),modify(u<<1|1,l,r,v);
    pushup(u);
}

ll query(int u,int l,int r){
    if(l<=tree[u].l&&tree[u].r<=r) return tree[u].v;
    pushdown(u);
    int mid=(tree[u].l+tree[u].r)>>1;
    if(r<=mid) return query(u<<1,l,r);
    else if(l>mid) return query(u<<1|1,l,r);
    else return query(u<<1,l,r)+query(u<<1|1,l,r);
}

void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>a[i];
    build(1,1,n);
    while(m--){
        int op;cin>>op;
        if(op==1){
            int l,r,k;cin>>l>>r>>k;
            modify(1,l,r,k);
        }
        else{
            int l,r;cin>>l>>r;
            cout<<query(1,l,r)<<"\n";
        }
    }
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
//  int t;cin>>t;while(t--)
    solve();
    return 0;
```

```
}
```

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=100010;
ll q[N];
int p;
struct node{
    int l,r; //tree[i].l和tree[i].r分别表示这个点代表的线段的左右下标
    ll sum; //tree[i].sum表示这个节点表示的线段和
    ll addlazy,mullazy; //懒标记
}tree[N<<2];  //开四倍空间

inline void pushup(int u){        //更新函数
    tree[u].sum=(tree[u<<1].sum+tree[u<<1|1].sum)%p; //父节点的和等于两个子节点之和
}

inline void pushdown(int u){
    tree[u<<1].sum=((tree[u].mullazy*tree[u<<1].sum)%p+(tree[u].addlazy*(tree[u<<1].r-
tree[u<<1].l+1))%p)%p;
    tree[u<<1|1].sum=((tree[u].mullazy*tree[u<<1|1].sum)%p+(tree[u].addlazy*
(tree[u<<1|1].r-tree[u<<1|1].l+1))%p)%p;

    tree[u<<1].mullazy=(tree[u<<1].mullazy*tree[u].mullazy)%p;
    tree[u<<1|1].mullazy=(tree[u<<1|1].mullazy*tree[u].mullazy)%p;

    tree[u<<1].addlazy=((tree[u<<1].addlazy*tree[u].mullazy)%p+tree[u].addlazy)%p;
    tree[u<<1|1].addlazy=((tree[u<<1|1].addlazy*tree[u].mullazy)%p+tree[u].addlazy)%p;

    tree[u].mullazy=1;
    tree[u].addlazy=0;
}

//一个节点为x 它的父节点为x/2（x>>1） 左儿子2x(x<<1) 右儿子2x+1(x<<1|1)
inline void build(int u,int l,int r){
    tree[u].l=l;
    tree[u].r=r;
    tree[u].mullazy=1;
    if(l==r){     //左端点等于右端点，即为叶子节点，直接赋值即可
        tree[u].sum=q[l]%p;
        return;
    }

    int mid=(l+r)>>1;    //mid则为中间点，左儿子的结点区间为[l,mid],右儿子的结点区间为[m+1,r]
    build(u<<1,l,mid);    //递归构造左儿子结点
    build(u<<1|1,mid+1,r);    //递归构造右儿子结点
    pushup(u);    //更新父节点
}

inline void add(int u,int l,int r,ll v) {  //u为结点下标，[l,r]为修改区间，v为要加上的值
    if(l<=tree[u].l&&r>=tree[u].r){
        tree[u].sum=(tree[u].sum+((tree[u].r-tree[u].l+1)*v)%p)%p;
        tree[u].addlazy=(tree[u].addlazy+v)%p;
        return;
    }
    pushdown(u);
    int mid=(tree[u].l+tree[u].r)>>1;
    if(l<=mid) add(u<<1,l,r,v);
    if(r>mid) add(u<<1|1,l,r,v);
    pushup(u);
}
```

```
inline void mul(int u,int l,int r,ll v) {  //u为结点下标，[l,r]为修改区间，v为要乘上的值
    if(l<=tree[u].l&&r>=tree[u].r){
        tree[u].sum=(tree[u].sum*v)%p;
        tree[u].mullazy=(tree[u].mullazy*v)%p;
        tree[u].addlazy=(tree[u].addlazy*v)%p;
        return;
    }
    pushdown(u);
    int mid=(tree[u].l+tree[u].r)>>1;
    if(l<=mid) mul(u<<1,l,r,v);
    if(r>mid) mul(u<<1|1,l,r,v);
    pushup(u);
}

//区间查询
inline ll query(int u,int l,int r){    //u为结点下标, [l,r]即为要查询的区间
    if(tree[u].l>=l&&tree[u].r<=r)    //如果当前结点的区间包含于(?)要查询的区间内，则返回结点信息且不需要往下递归
        return tree[u].sum;
    ll sum=0;

    pushdown(u);

    int mid=(tree[u].l+tree[u].r)>>1;    //mid则为中间点，左儿子的结点区间为[l,mid],右儿子的结点区间为[mid+1,r]

    if(l<=mid)   //先找和左边无交集
        sum=(sum+query(u<<1,l,r))%p; //左儿子

    if(r>mid)   //再找和右边无交集
        sum=(sum+query(u<<1|1,l,r))%p; //加上右儿子

    return sum;    //返回当前结点得到的信息
}


int main(){
    int n,m;
    int t,x,y;
    ll k;
    cin>>n>>m>>p;
    for(int i=1;i<=n;i++) scanf("%lld",&q[i]);

    build(1,1,n);

    for(int i=0;i<m;i++){
        scanf("%d",&t);
        if(t==1){
            scanf("%d%d%lld",&x,&y,&k);
            mul(1,x,y,k);
        }
        else if(t==2){
            scanf("%d%d%lld",&x,&y,&k);
            add(1,x,y,k);
        }
        else if(t==3){
            scanf("%d%d",&x,&y);
            printf("%lld\n",query(1,x,y));
        }
    }
    return 0;
```

```
}
```

维护区间最值操作与区间历史最值

- `1 l r k`：对于所有的 $i \in [l,r]$，将 A_i 加上 k（k 可以为负数）。
- `2 l r v`：对于所有的 $i \in [l,r]$，将 A_i 变成 $\min(A_i, v)$。
- `3 l r`：求 $\sum_{i=l}^{r} A_i$。
- `4 l r`：对于所有的 $i \in [l,r]$，求 A_i 的最大值。
- `5 l r`：对于所有的 $i \in [l,r]$，求 B_i 的最大值。

在每一次操作后，我们都进行一次更新，让 $B_i \leftarrow \max(B_i, A_i)$。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=5e5+10,INF=0x3f3f3f3f;

struct SegmentTree{
    struct Node{
        int l, r;
        int mx, mx_, se, cnt; ll sum;
        int add1, add1_, add2, add2_;
    } tr[N<<2];
    #define lc (o<<1)
    #define rc (o<<1|1)
    void pushup(int o){
        tr[o].sum=tr[lc].sum+tr[rc].sum;
        tr[o].mx_=max(tr[lc].mx_, tr[rc].mx_);
        if (tr[lc].mx==tr[rc].mx){
            tr[o].mx=tr[lc].mx;
            tr[o].se=max(tr[lc].se, tr[rc].se);
            tr[o].cnt=tr[lc].cnt+tr[rc].cnt;
        }
        else if (tr[lc].mx>tr[rc].mx){
            tr[o].mx=tr[lc].mx;
            tr[o].se=max(tr[lc].se, tr[rc].mx);
            tr[o].cnt=tr[lc].cnt;
        }
        else{
            tr[o].mx=tr[rc].mx;
            tr[o].se=max(tr[lc].mx, tr[rc].se);
            tr[o].cnt=tr[rc].cnt;
        }
    }
    void update(int o, int k1, int k1_, int k2, int k2_){
        tr[o].sum+=1ll*k1*tr[o].cnt+1ll*k2*(tr[o].r-tr[o].l+1-tr[o].cnt);
        tr[o].mx_=max(tr[o].mx_, tr[o].mx+k1_);
        tr[o].add1_=max(tr[o].add1_, tr[o].add1+k1_);
        tr[o].mx+=k1, tr[o].add1+=k1;
        tr[o].add2_=max(tr[o].add2_, tr[o].add2+k2_);
        if (tr[o].se!=-INF) tr[o].se+=k2;
        tr[o].add2+=k2;
    }
    void pushdown(int o){
        int tmp=max(tr[lc].mx, tr[rc].mx);
        if (tr[lc].mx==tmp)
            update(lc, tr[o].add1, tr[o].add1_, tr[o].add2, tr[o].add2_);
        else update(lc, tr[o].add2, tr[o].add2_, tr[o].add2, tr[o].add2_);
        if (tr[rc].mx==tmp)
            update(rc, tr[o].add1, tr[o].add1_, tr[o].add2, tr[o].add2_);
        else update(rc, tr[o].add2, tr[o].add2_, tr[o].add2, tr[o].add2_);
```

```
        tr[o].add1=tr[o].add1_=tr[o].add2=tr[o].add2_=0;
    }
    void build(int o, int l, int r, int* a){
        tr[o].l=l, tr[o].r=r;
        tr[o].add1=tr[o].add1_=tr[o].add2=tr[o].add2_=0;
        if (l==r){
            tr[o].sum=tr[o].mx_=tr[o].mx=a[l];
            tr[o].se=-INF, tr[o].cnt=1;
            return;
        }
        int mid=l+r>>1;
        build(lc, l, mid, a);
        build(rc, mid+1, r, a);
        pushup(o);
    }
    void modify1(int o, int l, int r, int k){
        if (tr[o].l>r||tr[o].r<l) return;
        if (l<=tr[o].l&&tr[o].r<=r)
            { update(o, k, k, k, k); return; }
        pushdown(o);
        modify1(lc, l, r, k), modify1(rc, l, r, k);
        pushup(o);
    }
    void modify2(int o, int l, int r, int k){
        if (tr[o].l>r||tr[o].r<l||k>=tr[o].mx) return;
        if (l<=tr[o].l&&tr[o].r<=r&&k>tr[o].se)
            { update(o, k-tr[o].mx, k-tr[o].mx, 0, 0); return; }
        pushdown(o);
        modify2(lc, l, r, k), modify2(rc, l, r, k);
        pushup(o);
    }
    ll query3(int o, int l, int r){
        if (tr[o].l>r||tr[o].r<l) return 0;
        if (l<=tr[o].l&&tr[o].r<=r) return tr[o].sum;
        pushdown(o);
        return query3(lc, l, r)+query3(rc, l, r);
    }
    int query4(int o, int l, int r){
        if (tr[o].l>r||tr[o].r<l) return -INF;
        if (l<=tr[o].l&&tr[o].r<=r) return tr[o].mx;
        pushdown(o);
        return max(query4(lc, l, r), query4(rc, l, r));
    }
    int query5(int o, int l, int r){
        if (tr[o].l>r||tr[o].r<l) return -INF;
        if (l<=tr[o].l&&tr[o].r<=r) return tr[o].mx_;
        pushdown(o);
        return max(query5(lc, l, r), query5(rc, l, r));
    }
    #undef lc
    #undef rc
} sgt;
int a[N];
int main(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>a[i];
    sgt.build(1, 1, n, a);
    while (m--){
        int op, l, r, k;
        cin>>op;
        switch (op){
```

```
            case 1:
                cin>>l>>r>>k;
                sgt.modify1(1, l, r, k);
                break;
            case 2:
                cin>>l>>r>>k;
                sgt.modify2(1, l, r, k);
                break;
            case 3:
                cin>>l>>r;
                cout<<sgt.query3(1, l, r)<<"\n";
                break;
            case 4:
                cin>>l>>r;
                cout<<sgt.query4(1, l, r)<<"\n";
                break;
            case 5:
                cin>>l>>r;
                cout<<sgt.query5(1, l, r)<<"\n";
                break;
        }
    }
    return 0;
}
```

李超线段树

https://www.luogu.com.cn/problem/P4097

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<double,int> pdi;
const int N=1e5+10,mod1=39989,mod2=1e9,INF=0x3f3f3f3f;

int cnt,lastans;
double k[N],b[N];
int tag[N<<2];

inline int ls(int x){return x<<1;}
inline int rs(int x){return x<<1|1;}
inline double calc(int i,int x){return b[i]+k[i]*x;}

void add(int x0,int y0,int x1,int y1) {
    cnt++;
    if(x0==x1){ //斜率不存在
        k[cnt]=0;
        b[cnt]=max(y1,y0);
    }
    else{
        k[cnt]=1.0*(y1-y0)/(x1-x0);
        b[cnt]=y0-k[cnt]*x0;
    }
}

void update(int l,int r,int p,int pl,int pr,int x){
    int mid=(pl+pr)>>1;
    if(l<=pl&&pr<=r){
        if(pl==pr){
            if(calc(tag[p],pl)<calc(x,pl)) tag[p]=x;
            return;
        }
```

```
        if(!tag[p]){tag[p]=x;return;}
        else{
            double y1=calc(tag[p],mid),y2=calc(x,mid);
            if(k[tag[p]]<k[x]) {
                if(y1<=y2) {update(l,r,ls(p),pl,mid,tag[p]);tag[p]=x;}
                else {update(l,r,rs(p),mid+1,pr,x);}
            }
            else if(k[tag[p]]>k[x]) {
                if(y1<=y2) {update(l,r,rs(p),mid+1,pr,tag[p]);tag[p]=x;}
                else {update(l,r,ls(p),pl,mid,x);}
            }
            else if(b[tag[p]]>b[x]){tag[p]=x;}
        }
        return;
    }
    if(l<=mid) update(l,r,ls(p),pl,mid,x);
    if(r>mid) update(l,r,rs(p),mid+1,pr,x);
}

pdi query(int p,int l,int r,int x){
    if (r<x||x<l) return {0,0};
    if(l==r) {
        return {calc(tag[p],l),tag[p]};
    }
    double res=calc(tag[p],x);
    int ansid=tag[p];
    int mid=(l+r)>>1;
    if(x<=mid){
        auto temp=query(ls(p),l,mid,x);
        if(res<temp.first){
            res=temp.first;
            ansid=temp.second;
        }
        else if(res==temp.first) {
            ansid=min(ansid,temp.second);
        }
    }
    else {
        auto temp=query(rs(p),mid+1,r,x);
        if(res<temp.first){
            res=temp.first;
            ansid=temp.second;
        }
        else if(res==temp.first){
            ansid=min(ansid,temp.second);
        }
    }
    return {res,ansid};
}

int main() {
    int n;cin>>n;
    while(n--){
        int op;cin>>op;
        if(op==1){
            int x0,y0,x1,y1;
            cin>>x0>>y0>>x1>>y1;
            x0=(x0+lastans-1)%mod1+1;
            x1=(x1+lastans-1)%mod1+1;
            y0=(y0+lastans-1)%mod2+1;
            y1=(y1+lastans-1)%mod2+1;
            if(x0>x1) swap(x0,x1),swap(y0,y1);
```

```
            add(x0,y0,x1,y1);
            update(x0,x1,1,1,mod1,cnt);
        }
        else {
            int x;cin>>x;
            x=(x+lastans-1)%mod1+1;
            lastans=query(1,1,mod1,x).second;
            cout<<lastans<<"\n";
        }
    }
    return 0;
}
```

扫描线

矩形面积并

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=100010;
int n,m;
int y[N<<1];

struct Segment{
    int x;
    int y1,y2;
    int state;//是左边还是右边
    bool operator< (const Segment &t)const{
        return x<t.x;
    }
}seg[N<<1];

struct node{
    int l,r;
    int cover;//当前区间覆盖次数
    ll len;//至少被覆盖1次的区间长度
}tree[N<<4];

inline void pushup(int u){
    if(tree[u].cover) tree[u].len=tree[u].r-tree[u].l;
    else tree[u].len=tree[u<<1].len+tree[u<<1|1].len;
}

void build(int u,int l,int r){
    tree[u].l=y[l],tree[u].r=y[r];
    if(r-l<=1) return;
    int mid=(l+r)>>1;
    build(u<<1,l,mid),build(u<<1|1,mid,r);
}

void modify(int u,int l,int r,int k){
    int x=tree[u].l,y=tree[u].r;
    if(x>=l&&y<=r){
        tree[u].cover+=k;
        pushup(u);
    }
    else{
        if(l<tree[u<<1].r) modify(u<<1,l,r,k);
        if(r>tree[u<<1|1].l) modify(u<<1|1,l,r,k);
        pushup(u);
    }
```

```
}

int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        int x1,y1,x2,y2;
        cin>>x1>>y1>>x2>>y2;
        y[i]=y1,y[n+i]=y2;
        seg[m++]={x1,y1,y2,1};
        seg[m++]={x2,y1,y2,-1};
    }
    sort(y+1,y+m+1);
    sort(seg,seg+m);

    build(1,1,m);

    ll ans=0;
    modify(1,seg[0].y1,seg[0].y2,seg[0].state);
    for(int i=1;i<m;i++){
        ans+=tree[1].len*(seg[i].x-seg[i-1].x);
        modify(1,seg[i].y1,seg[i].y2,seg[i].state);
    }
    cout<<ans<<"\n";
    return 0;
}
```

矩形周长并

```
#include<bits/stdc++.h>
using namespace std;
int ls(int x){ return x<<1;  }
int rs(int x){ return x<<1|1;}
const int MAXN = 200005;
struct ScanLine {
    int l, r, h, inout;  //inout=1 下边, inout=-1 上边
    ScanLine() {}
    ScanLine(int a, int b, int c, int d) :l(a), r(b), h(c), inout(d) {}
}line[MAXN];
bool cmp(ScanLine &a, ScanLine &b) {
    if(a.h==b.h) return a.inout>b.inout;
    return a.h<b.h;
}   //y坐标排序

bool lbd[MAXN << 2], rbd[MAXN << 2];//标记这个结点的左右两个端点是否被覆盖（0表示没有，1表示有）
int num[MAXN << 2];    //这个区间有多少条独立的边
int Tag[MAXN << 2];    //标记这个结点是否有效
int length[MAXN << 2]; //这个区间的有效宽度
void pushup(int p, int pl, int pr) {
    if (Tag[p]) {                //结点的Tag为正，这个线段对计算宽度有效
        lbd[p] = rbd[p] = 1;
        length[p] = pr - pl + 1;
        num[p] = 1;              //每条边有两个端点
    }
    else if (pl == pr) length[p]=num[p]=lbd[p]=rbd[p]=0;//叶子结点
    else {
        lbd[p] = lbd[ls(p)];      // 和左儿子共左端点
        rbd[p] = rbd[rs(p)];      //和右儿子共右端点
        length[p] = length[ls(p)] + length[rs(p)];
        num[p] = num[ls(p)] + num[rs(p)];
        if (lbd[rs(p)] && rbd[ls(p)]) num[p] -= 1;   //合并边
    }
}
```

```
void update(int L, int R, int io, int p, int pl, int pr) {
    if(L<=pl && pr<=R){    //完全覆盖
        Tag[p] += io;
        pushup(p, pl, pr);
        return;
    }
    int mid  = (pl + pr) >> 1;
    if (L<= mid) update(L, R, io, ls(p), pl, mid);
    if (mid < R) update(L, R, io, rs(p), mid+1, pr);
    pushup(p, pl, pr);
}
int main() {
    int n;
    while (~scanf("%d", &n)) {
        int cnt  = 0;
        int lbd = 10000, rbd = -10000;
        for (int i = 0; i < n; i++) {
            int x1, y1, x2, y2;
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);   //输入矩形
            lbd = min(lbd, x1);                      //横线最小x坐标
            rbd = max(rbd, x2);                      //横线最大x坐标
            line[++cnt] = ScanLine(x1, x2, y1, 1);   //给入边赋值
            line[++cnt] = ScanLine(x1, x2, y2, -1);  //给出边赋值
        }
        sort(line+1, line + cnt+1, cmp);           //排序。数据小，不用离散化
        int ans = 0, last = 0;                     //last：上一次总区间被覆盖长度
        for (int i = 1; i <= cnt ; i++){           //扫描所有入边和出边
            if (line[i].l < line[i].r)
                update(line[i].l, line[i].r-1, line[i].inout, 1, lbd, rbd-1);
            ans += num[1]*2 * (line[i + 1].h - line[i].h);  //竖线
            ans += abs(length[1] - last);                //横线
            last = length[1];
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

可持久化线段树(主席树)

```
//区间第k小
#include <bits/stdc++.h>
using namespace std;
const int N=200010;

int cnt=0; //用cnt标记可以使用的新结点
int a[N],root[N]; //root[i]记录第i棵线段树的根节点编号
vector<int> v; //排序后的数组

int getid(int x){
    return lower_bound(v.begin(),v.end(),x)-v.begin()+1;
}

struct node{
    int l,r,sum; //l左儿子, r右儿子，sum[i]是结点i的权值
}tree[N<<5]; //需要开nlogn空间

//建空树
int build(int l,int r){
    int rt = ++cnt; //cnt为当前节点编号
    tree[rt].sum=0;
    if(l==r) return rt;
```

```
    int mid=(l+r)>>1;
    tree[rt].l=build(l,mid);
    tree[rt].r=build(mid+1,r);
    return rt;  //返回当前节点的编号
}

//建一棵只有logn个结点的新线段树
int update(int pre,int l,int r,int k){
    int rt = ++cnt;
    tree[rt]=tree[pre];
    tree[rt].sum++; //插了1个数，在前一棵树的相同结点加1
    if(l==r) return rt;
    int mid=(l+r)>>1;
    if(k<=mid) tree[rt].l=update(tree[pre].l,l,mid,k);
    else tree[rt].r=update(tree[pre].r,mid+1,r,k);
    return rt; //返回当前分配使用的新结点的编号
}

//查询区间[u,v]第k小
int query(int u,int v,int l,int r,int k){
    if(l==r) return l; //到达叶子结点,找到第k小,l是节点编号,答案是b[l]
    int mid=(l+r)>>1;
    int x=tree[tree[v].l].sum-tree[tree[u].l].sum; //线段树相减
    if(x>=k) //左儿子数字大于等于k时，说明第k小的数字在左子树
        return query(tree[u].l,tree[v].l,l,mid,k);
    else  //否则在右子树找第k-x小的数字
        return query(tree[u].r,tree[v].r,mid+1,r,k-x);
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    int n,m;cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        v.push_back(a[i]);
    }
    sort(v.begin(),v.end());
    v.erase(unique(v.begin(),v.end()),v.end());
    int siz=v.size();
    root[0]=build(1,siz);

    for(int i=1;i<=n;i++){
        int x=getid(a[i]);
        root[i]=update(root[i-1],1,siz,x);
    }

    while(m--) {
        int x,y,k;cin>>x>>y>>k;
        //第y棵线段树减第x-1棵线段树,就是区间[x,y]的线段树
        int t=query(root[x-1],root[y],1,siz,k);
        cout<<v[t-1]<<"\n";
    }
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std ;
typedef long long ll;
const int N = 1e5+10;
int cas,cnt;
ll a[N];
```

```
int root[N];
int n,m;

struct node {
    int L, R;
    ll lazy,sum;
} tree[N<<5];    //需要开 nlogn空间

void pushup(int u){
    tree[u].sum=tree[tree[u].L].sum+tree[tree[u].R].sum;
}

int build(int pl, int pr) {
    int rt = ++ cnt;              //cnt为当前节点编号
    tree[rt].L=tree[rt].R=tree[rt].lazy=tree[rt].sum=0;
    if(pl==pr){
        tree[rt].sum=a[pl];
        return rt;
    }
    int mid=(pl+pr)>>1;
    tree[rt].L = build(pl, mid);
    tree[rt].R = build(mid+1, pr);
    pushup(rt);
    return rt;  //返回当前节点的编号
}

int update(int pre, int pl, int pr, int l, int r, ll v) {  //建一棵只有logn个结点的新线段树
    int rt = ++cnt;
    tree[rt] = tree[pre];
    tree[rt].sum+=v*(r-l+1);
    if(l==pl&&r==pr){
        tree[rt].lazy += v;
        return rt;
    }
    int mid=(pl+pr)>>1;
    if(r<=mid) tree[rt].L = update(tree[pre].L,pl,mid,l,r,v);
    else if(l>mid) tree[rt].R = update(tree[pre].R,mid+1,pr,l,r,v);
    else{
        tree[rt].L = update(tree[pre].L,pl,mid,l,mid,v);
        tree[rt].R = update(tree[pre].R,mid+1,pr,mid+1,r,v);
    }
//    pushup(rt);
    return rt;              //返回当前分配使用的新结点的编号
}

//区间查询
ll query(int rt,int pl,int pr,int l,int r){
    if(pl>=l&&pr<=r) return tree[rt].sum;
    int mid=(pl+pr)>>1;
    ll res=tree[rt].lazy*(r-l+1);
    if(r<=mid) res+=query(tree[rt].L,pl,mid,l,r);
    else if(l>mid) res+=query(tree[rt].R,mid+1,pr,l,r);
    else{
        res+=query(tree[rt].L,pl,mid,l,mid);
        res+=query(tree[rt].R,mid+1,pr,mid+1,r);
    }
    return res;    //返回当前结点得到的信息
}

void solve(){
    if(cas++) cout<<"\n";
    cnt=0;
```

```
    for(int i=1;i<=n;i++) cin>>a[i];
    root[0]=build(1,n);
    int time=0;
    while(m--){
        char op;cin>>op;
        if(op=='C'){
            int l,r;ll d;cin>>l>>r>>d;
            time++;
            root[time]=update(root[time-1],1,n,l,r,d);
        }
        else if(op=='Q'){
            int l,r;cin>>l>>r;
            cout<<query(root[time],1,n,l,r)<<"\n";
        }
        else if(op=='H'){
            int l,r,t;cin>>l>>r>>t;
            cout<<query(root[t],1,n,l,r)<<"\n";
        }
        else{
            int t;cin>>t;
            time=t;
        }
    }
}

int main() {
    while(cin>>n>>m)
    solve();
    return 0;
}
```

珂朵莉树(ODT)

推平一段区间

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+10;

struct node{
    int l,r;
    mutable ll v;
    node(int L,int R=-1,ll V=0):l(L), r(R), v(V) {}
    bool operator<(const node& o) const{
        return l < o.l;
    }
};
set<node> s;
int n,m;
ll seed,vmax;
ll a[N];

ll fpow(ll a,ll b,ll mod){
    if(mod==1) return 0;
    ll ans=1%mod;
    a%=mod;
    while(b){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
        b>>=1;
    }
```

```cpp
    return ans;
}

set<node>::iterator split(int pos){
    auto it=s.lower_bound(node(pos));
    if(it!=s.end() && it->l==pos) return it;
    --it;
    int L=it->l,R=it->r;
    ll V=it->v;
    s.erase(it);
    s.insert(node(L,pos-1,V));
    return s.insert(node(pos,R,V)).first;
}

//推平
void assign(int l, int r, ll val){
    auto itr=split(r+1),itl=split(l);
    s.erase(itl,itr);
    s.insert(node(l,r,val));
}

//区间加
void add(int l,int r,ll val){
    auto itr=split(r+1),itl=split(l);
    for(;itl!=itr;++itl) itl->v+=val;
}

//区间第k小
ll _rank(int l,int r,int k){
    vector<pair<ll,int> >vp;
    auto itr=split(r+1),itl=split(l);
    vp.clear();
    for(;itl!=itr;++itl) vp.push_back({itl->v,itl->r-itl->l+1});
    sort(vp.begin(),vp.end());
    for(auto i:vp){
        k-=i.second;
        if(k<=0) return i.first;
    }
    return -1;
}

//区间幂次和
ll sum(int l,int r,ll ex,ll mod){
    auto itr=split(r+1),itl=split(l);
    ll res=0;
    for(;itl!=itr;++itl)
        res=(res+(ll)(itl->r - itl->l + 1)*fpow(itl->v,ex,mod))%mod;
    return res;
}

ll rnd(){
    ll ret=seed;
    seed=(seed*7+13)%1000000007;
    return ret;
}

int main(){
    cin>>n>>m>>seed>>vmax;
    for(int i=1;i<=n;i++){
        a[i]=(rnd()%vmax)+1;
        s.insert(node(i,i,a[i]));
    }
```

```
    for(int i=1;i<=m;i++){
        int op=(rnd()%4)+1;
        int l=(rnd()%n)+1;
        int r=(rnd()%n)+1;
        if(l>r) swap(l,r);
        ll x,y;
        if(op==3) x=(rnd()%(r-l+1))+1;
        else x=(rnd()%vmax)+1;
        if(op==4) y=(rnd()%vmax)+1;

        if(op==1) add(l,r,x);
        else if(op==2) assign(l,r,x);
        else if(op==3) cout<<_rank(l,r,x)<<"\n";
        else cout<<sum(l,r,x,y)<<"\n";
    }
    return 0;
}
```

ST表

$O(nlogn)$预处理，$O(1)$查询最值

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int n,m,f[N][20],lg[N];

void init(){
    //log2(i)
    for(int i=2;i<=n;i++) lg[i]=lg[i>>1]+1;
}

int main(){
    cin>>n>>m;
    init();
    for(int i=1;i<=n;i++) cin>>f[i][0];
    for(int j=1;j<=lg[n];j++){
        for(int i=1;i<=n-(1<<j)+1;i++){
            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
        }
    }
    while(m--){
        int l,r;cin>>l>>r;
        int s=lg[r-l+1];
        cout<<max(f[l][s],f[r-(1<<s)+1][s])<<"\n";
    }
    return 0;
}
```

FHQ Treap

普通平衡树

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;

int cnt,x,y,z,root;
struct FHQTreap{
    int ch[N][2];//0左孩子，1右孩子
    int val[N];//每个点的权值
```

```
int rnd[N];//每个点的随机权值
int siz[N];//以每个点为根的树的大小

inline void update(int x){
    siz[x]=1+siz[ch[x][0]]+siz[ch[x][1]];
}

inline int new_node(int x){  //新建一个节点
    siz[++cnt]=1;
    val[cnt]=x;
    rnd[cnt]=rand();
    return cnt;
}

int merge(int A,int B){  //合并
    if(!A||!B) return A+B;
    if(rnd[A]<rnd[B]){
        ch[A][1]=merge(ch[A][1],B);
        update(A);
        return A;
    }
    else{
        ch[B][0]=merge(A,ch[B][0]);
        update(B);
        return B;
    }
}

void split(int now,int k,int &x,int &y){  //权值分裂
    if(!now) x=y=0;
    else{
        if(val[now]<=k) x=now,split(ch[now][1],k,ch[now][1],y);
        else y=now,split(ch[now][0],k,x,ch[now][0]);
        update(now);
    }
}

void insert(int a){  //插入a
    split(root,a,x,y);
    root=merge(merge(x,new_node(a)),y);
}

void del(int a){  //删除a(若有多个相同的数，只删除一个)
    split(root,a,x,z);
    split(x,a-1,x,y);
    y=merge(ch[y][0],ch[y][1]);
    root=merge(merge(x,y),z);
}

int myrank(int a){ //查询a的排名
    split(root,a-1,x,y);
    int res=siz[x]+1;
    root=merge(x,y);
    return res;
}

int kth(int now,int k){  //k小值
    while(1){
        if(k<=siz[ch[now][0]]) now=ch[now][0];
        else if(k==siz[ch[now][0]]+1) return now;
        else k-=siz[ch[now][0]]+1,now=ch[now][1];
    }
```

```
    }

    int pre(int a){  //a的前驱
        split(root,a-1,x,y);
        int res=val[kth(x,siz[x])];
        root=merge(x,y);
        return res;
    }

    int nxt(int a){  //a的后继
        split(root,a,x,y);
        int res=val[kth(y,1)];
        root=merge(x,y);
        return res;
    }

}Treap;

int main(){
    srand(time(0));
    int t;cin>>t;
    while(t--){
        int op,v;cin>>op>>v;
        if(op==1) Treap.insert(v);
        else if(op==2) Treap.del(v);
        else if(op==3) cout<<Treap.myrank(v)<<"\n";
        else if(op==4) cout<<Treap.val[Treap.kth(root,v)]<<"\n";
        else if(op==5) cout<<Treap.pre(v)<<"\n";
        else cout<<Treap.nxt(v)<<"\n";
    }
    return 0;
}
```

区间操作

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;

int n,m;
int cnt,x,y,z,root;
struct FHQTreap{
    int ch[N][2];//0左孩子，1右孩子
    int val[N];//每个点的权值
    int rnd[N];//每个点的随机权值
    int siz[N];//以每个点为根的树的大小
    bool lazy[N];

    inline void update(int x){
        siz[x]=1+siz[ch[x][0]]+siz[ch[x][1]];
    }

    inline int new_node(int x){  //新建一个节点
        siz[++cnt]=1;
        val[cnt]=x;
        rnd[cnt]=rand();
        return cnt;
    }

    inline void pushdown(int x){  //下传懒标记
        if(x&&lazy[x]){
            swap(ch[x][0],ch[x][1]);
```

```
            if(ch[x][0]) lazy[ch[x][0]]^=1;
            if(ch[x][1]) lazy[ch[x][1]]^=1;
            lazy[x]=0;
        }
    }

    int merge(int A,int B){   //合并
        if(!A||!B) return A+B;
        pushdown(A);pushdown(B);
        if(rnd[A]<rnd[B]){
            ch[A][1]=merge(ch[A][1],B);
            update(A);
            return A;
        }
        else{
            ch[B][0]=merge(A,ch[B][0]);
            update(B);
            return B;
        }
    }

    void split(int now,int k,int &x,int &y){   //权值分裂
        if(!now) x=y=0;
        else{
            pushdown(now);
            if(siz[ch[now][0]]<k) x=now,split(ch[now][1],k-siz[ch[now][0]]-1,ch[now]
[1],y);
            else y=now,split(ch[now][0],k,x,ch[now][0]);
            update(now);
        }
    }

    void insert(int a){   //插入a
        split(root,a,x,y);
        root=merge(merge(x,new_node(a)),y);
    }

    void reverse(int l,int r){   //翻转区间
        split(root,r,x,z);
        split(x,l-1,x,y);
        lazy[y]^=1;
        root=merge(merge(x,y),z);
    }

    int kth(int now,int k){   //k小值
        while(1){
            pushdown(now);
            if(k<=siz[ch[now][0]]) now=ch[now][0];
            else if(k==siz[ch[now][0]]+1) return now;
            else k-=siz[ch[now][0]]+1,now=ch[now][1];
        }
    }

}Treap;

int main(){
    srand(time(0));
    cin>>n>>m;
    for(int i=1;i<=n;i++) Treap.insert(i);
    while(m--){
        int l,r;cin>>l>>r;
        Treap.reverse(l,r);
```

```
    }
    for(int i=1;i<=n;i++) cout<<Treap.val[Treap.kth(root,i)]<<" \n"[i==n];
    return 0;
}
```

可持久化

```
#include<bits/stdc++.h>
using namespace std;
const int N=5e5+5;

int cnt,x,y,z,root,rt[N];
namespace Treap{
    struct node{
        int ch[2];//0左孩子，1右孩子
        int val;//每个点的权值
        int rnd;//每个点的随机权值
        int siz;//以每个点为根的树的大小
    }t[N<<6];

    inline void update(int x){
        t[x].siz=1+t[t[x].ch[0]].siz+t[t[x].ch[1]].siz;
    }

    inline int new_node(int x){  //新建一个节点
        cnt++;
        t[cnt].siz=1;
        t[cnt].val=x;
        t[cnt].rnd=rand();
        return cnt;
    }

    int merge(int A,int B){  //合并
        if(!A||!B) return A+B;
        if(t[A].rnd<t[B].rnd){
            int id=new_node(0);
            t[id]=t[A];
            t[id].ch[1]=merge(t[id].ch[1],B);
            update(id);
            return id;
        }
        else{
            int id=new_node(0);
            t[id]=t[B];
            t[id].ch[0]=merge(A,t[id].ch[0]);
            update(id);
            return id;
        }
    }

    void split(int now,int k,int &x,int &y){  //权值分裂
        if(!now) x=y=0;
        else{
            if(t[now].val<=k){
                x=new_node(0);
                t[x]=t[now];
                split(t[now].ch[1],k,t[x].ch[1],y);
                update(x);
            }
            else{
                y=new_node(0);
                t[y]=t[now];
```

```
                split(t[now].ch[0],k,x,t[y].ch[0]);
                update(y);
            }
        }
    }

    void insert(int a,int &root){  //插入a
        split(root,a,x,y);
        root=merge(merge(x,new_node(a)),y);
    }

    void del(int a,int &root){  //删除a(若有多个相同的数，只删除一个)
        split(root,a,x,z);
        split(x,a-1,x,y);
        y=merge(t[y].ch[0],t[y].ch[1]);
        root=merge(merge(x,y),z);
    }

    int myrank(int a,int &root){ //查询a的排名
        split(root,a-1,x,y);
        int res=t[x].siz+1;
        root=merge(x,y);
        return res;
    }

    int kth(int now,int k){  //k小值
        while(1){
            if(k<=t[t[now].ch[0]].siz) now=t[now].ch[0];
            else if(k==t[t[now].ch[0]].siz+1) return now;
            else k-=t[t[now].ch[0]].siz+1,now=t[now].ch[1];
        }
    }

    int pre(int a,int &root){  //a的前驱
        split(root,a-1,x,y);
        int res;
        if(!x) res=-2147483647;
        else res=t[kth(x,t[x].siz)].val;
        root=merge(x,y);
        return res;
    }

    int nxt(int a,int &root){  //a的后继
        split(root,a,x,y);
        int res;
        if(!y) res=2147483647;
        else res=t[kth(y,1)].val;
        root=merge(x,y);
        return res;
    }

}

int main(){
    srand(time(0));
    int n;cin>>n;
    for(int i=1;i<=n;i++){
        int ver,op,w;cin>>ver>>op>>w;
        rt[i]=rt[ver];
        if(op==1) Treap::insert(w,rt[i]);
        else if(op==2) Treap::del(w,rt[i]);
        else if(op==3) cout<<Treap::myrank(w,rt[i])<<"\n";
```

```
        else if(op==4) cout<<Treap::t[Treap::kth(rt[i],w)].val<<"\n";
        else if(op==5) cout<<Treap::pre(w,rt[i])<<"\n";
        else cout<<Treap::nxt(w,rt[i])<<"\n";
    }
    return 0;
}
```

```

```

图论模板

最短路

Floyd

图中可能存在重边和自环，边权可能为负数。

```
void init(){
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(i==j) d[i][j]=0;
            else d[i][j]=INF;
        }
    }
}

void floyd(){
    for(int k=1;k<=n;k++){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                // d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
                if(d[i][k]<INF&&d[k][j]<INF&&d[i][j]>d[i][k]+d[k][j])
                    d[i][j]=d[i][k]+d[k][j];
            }
        }
    }
}

d[x][y]=min(d[x][y],z);
```

Bellman-Ford

图中可能存在重边和自环， **边权可能为负数。**

```
//最多经过k条边的最短距离
void init(){
    memset(dis,INF,sizeof dis);
    dis[1]=0;
}

void Bellman_Ford(){
    for(int kk=0;kk<k;kk++){
        memcpy(back,dis,sizeof dis);
        for(int i=0;i<m;i++){
            if(back[u[i]]!=INF) dis[v[i]] = min(dis[v[i]],back[u[i]]+w[i]);
        }
    }
}
```

SPFA

朴素

图中可能存在重边和自环， **边权可能为负数。**

```
int w[N],idx,dis[N],ne[N],h[N],e[N];
bool st[N];
int n,m;
queue<int> q;
void init(){
    memset(dis,INF,sizeof dis);
    dis[1]=0;
    memset(h,-1,sizeof h);
}

void add(int a,int b,int c){
    e[idx]=b;w[idx]=c;ne[idx]=h[a];h[a]=idx++;
}

void spfa(){
    q.push(1);st[1]=1;
    while(q.size()){
        int t=q.front();q.pop();
        st[t]=0;
        for(int i=h[t];~i;i=ne[i]){
            int j=e[i];
            if(dis[j]>dis[t]+w[i]){
                dis[j]=dis[t]+w[i];
                if(!st[j]){
                    q.push(j);
                    st[j]=1;
                }
            }
        }
    }
}

add(u,v,w);
```

```
int dis[N];
struct edge{
    int to; //连接的节点
    int cost; //边长
};
vector<edge> g[N];
bool st[N];
int n,m;
queue<int> q;

void init(){
    memset(dis,INF,sizeof dis);
    dis[1]=0;
}

void spfa(){
    init();
    q.push(1);st[1]=1;
    while(q.size()){
        int t=q.front();q.pop();
        st[t]=0;
        for(auto x:g[t]){
```

```
            int v=x.to,w=x.cost;
            if(dis[v]>dis[t]+w){
                dis[v]=dis[t]+w;
                if(!st[v]){
                    q.push(v);
                    st[v]=1;
                }
            }
        }
    }
}
```

SLF优化

它是一种利用双端队列算法处理的问题。如果说当前点所花费的值少于我们当前队头点的值的话，那么我们就将这个节点插入到队头去，否则我们还是插入到队尾。

```
//s为起点
struct edge{
    int to;
    int cost;
};
vector<edge> g[N];
deque<int> q;
bool vis[N];
int dis[N];

void init(int s){
    memset(dis,INF,sizeof dis);
    dis[s]=0;
}

void spfa(int s){
    init(s);
    q.push_back(s);
    vis[s]=1;
    while(q.size()){
        int t=q.front();q.pop_front();
        vis[t]=0;
        for(auto x:g[t]){
            int v=x.to,w=x.cost;
            if(dis[v]>dis[t]+w){
                dis[v]=dis[t]+w;
                if(!vis[v]){
                    if(!q.empty()&&dis[v]<dis[q.front()]) q.push_front(v);
                    else q.push_back(v);
                    vis[v]=1;
                }
            }
        }
    }
}
```

SLF+LLL优化

LLL优化：记录现在队列中元素所代表值的平均值，和要压入元素的值相比较，如果大于平均值，直接压入对列尾部

ps:可能会变慢

```
//s为起点
struct edge{
```

```
    int to;
    int cost;
};
vector<edge> g[N];
deque<int> q;
bool vis[N];
int dis[N];

void init(int s){
    memset(dis,INF,sizeof dis);
    dis[s]=0;
}

void spfa(int s){
    init(s);
    q.push_back(s);
    vis[s]=1;
    int sum=dis[s],len=1;
    while(q.size()){
        int t=q.front();
        //LLL优化
        while(dis[t]*len>sum){
            q.pop_front();
            q.push_back(t);
            t=q.front();
        }
        q.pop_front();
        vis[t]=0;
        len--;
        sum-=dis[t];
        for(auto x:g[t]){
            int v=x.to,w=x.cost;
            if(dis[v]>dis[t]+w){
                dis[v]=dis[t]+w;
                if(!vis[v]){
                    if(q.size()&&dis[v]<dis[q.front()]) q.push_front(v);
                    else q.push_back(v);
                    vis[v]=1;
                    //LLL优化
                    sum+=dis[v];
                    len++;
                }
            }
        }
    }
}
```

判负环（TLE可以尝试把队列换成栈）

```
//队列写法
struct edge{
    int to,cost;
};
vector<edge> g[N];
int dis[N],cnt[N];
bool st[N];
void init(){
    memset(dis,INF,sizeof dis);
    dis[1]=0;
    memset(st,0,sizeof st);
    memset(cnt,0,sizeof cnt);
    for(int i=1;i<=n;i++) g[i].clear();
```

```
}

bool spfa(){
    queue<int> q;
    q.push(1);st[1]=1; //判断是否存在从1开始的负环

    //判断是否存在负环,需要将所有的点都加入队列中,其中dis初始化全为0即可
    /*
    for(int i=1;i<=n;i++){
        q.push(i);
        st[i]=1;
    }
    */
    while(q.size()){
        int t=q.front();q.pop();
        st[t]=0;
        for(auto x:g[t]){
            int v=x.to,w=x.cost;
            if(dis[v]>dis[t]+w){
                dis[v]=dis[t]+w;
                cnt[v]=cnt[t]+1;
                if(cnt[v]>=n) return true;
                if(!st[v]){
                    q.push(v);
                    st[v]=1;
                }
            }
        }
    }
    return false;
}
```

```
//栈写法
struct edge{
    int to,cost;
};
vector<edge> g[N];
int dis[N],cnt[N];
bool st[N];
void init(){
    memset(dis,INF,sizeof dis);
    dis[1]=0;
    memset(st,0,sizeof st);
    memset(cnt,0,sizeof cnt);
    for(int i=1;i<=n;i++) g[i].clear();
}

bool spfa(){
    stack<int> q;
    q.push(1);st[1]=1; //判断是否存在从1开始的负环

    //判断是否存在负环,需要将所有的点都加入队列中,其中dis初始化全为0即可
    /*
    for(int i=1;i<=n;i++){
        q.push(i);
        st[i]=1;
    }
    */
    while(q.size()){
        int t=q.top();q.pop();
        st[t]=0;
        for(auto x:g[t]){
```

```
            int v=x.to,w=x.cost;
            if(dis[v]>dis[t]+w){
                dis[v]=dis[t]+w;
                cnt[v]=cnt[t]+1;
                if(cnt[v]>=n) return true;
                if(!st[v]){
                    q.push(v);
                    st[v]=1;
                }
            }
        }
    }
    return false;
}
```

Dijkstra

单源最短路径

朴素

图中可能存在重边和自环，所有边权均为正值。

```
void init(){
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(i==j) g[i][j]=0;
            else g[i][j]=INF;
        }
    }

    memset(dis,INF,sizeof dis);
    dis[1]=0;
    memset(visited,false,sizeof visited);
}

void Dijkstra(){
    for(int i=0;i<n;i++){
        int u=-1;
        for(int j=1;j<=n;j++){
            if(!visited[j] && (u==-1||dis[u]>dis[j])){
                u=j;
            }
        }
        visited[u]=true;
        for(int v=1;v<=n;v++){
            if(g[u][v]!=INF&&dis[u]+g[u][v]<dis[v])
                dis[v]=dis[u]+g[u][v];
        }
    }
    if(dis[n]==INF) dis[n]=-1;
}

g[x][y]=min(g[x][y],z);
```

堆优化

图中可能存在重边和自环，所有边权均为非负值。

```
struct edge{
    int to; //连接的节点
    int cost; //边长
```

```
};
vector<edge> g[N];
int dis[N];
bool vis[N];
int n,m;

void init(){
    memset(dis,INF,sizeof dis);
    dis[1]=0;
}

void Dijkstra(){
    init();
    priority_queue<PII,vector<PII>,greater<PII> > que; //按first从小到大
    que.push({0,1});
    while(!que.empty()){
        auto t=que.top();que.pop();
        int v=t.second;
        // if(dis[v]<t.first) continue;
        if(vis[v]) continue;
        vis[v]=1;
        for(int i=0;i<g[v].size();i++){
            auto e=g[v][i];
            if(dis[e.to] > dis[v]+e.cost){
                dis[e.to] = dis[v]+e.cost;
                que.push({dis[e.to],e.to});
            }
        }
    }
}

g[x].push_back({y,z});
```

A*

```
//第k短路
#include <bits/stdc++.h>
using namespace std;
const int N = 1005;
const int INF = 0x3f3f3f3f;
typedef pair<int, int> PII;
typedef pair<int, PII> PIII;

struct edge{
    int to; //连接的节点
    int cost; //边长
};
vector<edge> g1[N],g2[N];//g1正向 g2反向
int dis[N],cnt[N];
bool vis[N];
int n,m;
int op,ed,k;

void init(){
    memset(dis,INF,sizeof dis);
    dis[ed]=0;
}

//从终点反向跑dij
void Dijkstra(){
    init();
    priority_queue<PII,vector<PII>,greater<PII> > que; //按first从小到大
```

```
    que.push({0,ed});
    while(!que.empty()){
        auto t=que.top();que.pop();
        int v=t.second;
        if(vis[v]) continue;
        vis[v]=1;
        for(int i=0;i<g2[v].size();i++){
            auto e=g2[v][i];
            if(dis[e.to] > dis[v]+e.cost){
                dis[e.to] = dis[v]+e.cost;
                que.push({dis[e.to],e.to});
            }
        }
    }
}

int astar(){
    priority_queue<PIII,vector<PIII>,greater<PIII> > que;
    // 谁的dis[u]+f[u]更小 谁先出队列
    que.push({dis[op],{0,op}});
    while(que.size()){
        auto t=que.top();que.pop();
        int v = t.second.second,distance=t.second.first;
        cnt[v]++;
        //如果终点已经被访问过k次了 则此时的ver就是终点t 返回答案
        if(cnt[ed]==k) return distance;
        for(int i=0;i<g1[v].size();i++){
            auto e=g1[v][i];
            if(cnt[e.to]<k){
                que.push({distance+e.cost+dis[e.to],{distance+e.cost,e.to}});
            }
        }
    }
    return -1;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    while(m--){
        int u,v,w;cin>>u>>v>>w;
        g1[u].push_back({v,w});
        g2[v].push_back({u,w});
    }
    cin>>op>>ed>>k;
    if(op==ed) k++;
    Dijkstra();
    cout<<astar()<<"\n";
    return 0;
}
```

差分约束

下面以求最大值为例（最小值替换成$x[a] \geq x[b] + c[k]$）

1. 求不等式组的可行解
   源点需要满足的条件: 从源点出发，一定可以走到所有的边。
   步骤:
   1. 先把每个$x[a] \leq x[b] + c[k]$不等式转化为一条从$x[b]$走到$x[a]$长度为$c[k]$的边
   2. 然后在这个图上找一个超级源点,使得该源点一定可以遍历到所有边
   3. 从源点求一遍单源最短路

结果1：如果存在负环，则原不等式一定无解

结果2：则dis[a]就是原不等式组的一个可行解

2. 如何求最大值或者最小值

结论：如果求最小值，则应该求最长路；如果求最大值，则应该求最短路

问题：如何转化$x[i] \le c$其中c是一个常数这类的不等式

方法：建立一个超级源点，0号点x[0]，然后建立0→i长度是c的边即可

题意	转化	连边
x[a] - x[b] ≤ c	x[a] ≤ x[b] + c	add(b, a, c);
x[a] - x[b] ≥ c	x[b] ≤ x[a] - c	add(a, b, -c);
x[a] - x[b] < c	x[a] ≤ x[b] + c -1	add(b, a, c-1);
x[a] - x[b] > c	x[b] ≤ x[a] - c -1	add(a, b, -c-1);
x[a] = x[b]	x[a] ≤ x[b] , x[b] ≤ x[a]	add(b, a, 0),add(a, b, 0);

```
#include <bits/stdc++.h>
using namespace std;
const int N=5010,INF=0x3f3f3f3f;

int n,m;
struct edge{
    int to,cost;
};
vector<edge> g[N];
int dis[N],cnt[N];
bool st[N];
void init(){
    memset(dis,INF,sizeof dis);
    dis[0]=0;
    memset(st,0,sizeof st);
    memset(cnt,0,sizeof cnt);
    for(int i=0;i<=n;i++) g[i].clear();
}

bool spfa(){
    queue<int> q;
    q.push(0);st[0]=1;
    while(q.size()){
        int t=q.front();q.pop();
        st[t]=0;
        for(auto x:g[t]){
            int v=x.to,w=x.cost;
            if(dis[v]>dis[t]+w){
                dis[v]=dis[t]+w;
                cnt[v]=cnt[t]+1;
                if(cnt[v]>=n+1) return true;
                //这里是n+1因为多了一个超级源点
                if(!st[v]){
                    q.push(v);
                    st[v]=1;
                }
            }
        }
    }
    return false;
}
```

```
void add(int a,int b,int c){
    g[a].push_back({b,c});
}

void solve(){
    cin>>n>>m;
    init();
    while(m--){
        int a,b,c;cin>>a>>b>>c;
        add(b,a,c);
    }
    for(int i=1;i<=n;i++) add(0,i,0);//超级源点
    if(spfa()) cout<<"NO\n";
    else{
        for(int i=1;i<=n;i++) cout<<dis[i]<<" \n"[i==n];
    }
}


int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    solve();
    return 0;
}
```

最近公共祖先(LCA)

倍增

```
#include <bits/stdc++.h>
using namespace std;
const int N=500010;

int n,m,root;
vector<int> g[N];
int depth[N],fa[N][20],lg[N];

void init(){
    // log2(i)+1
    // for(int i=1;i<=n;i++){
        // lg[i]=lg[i-1]+(1<<lg[i-1]==i);
    // }

    // log2(i)
    for(int i=2;i<=n;i++){
        lg[i]=lg[i>>1]+1;
    }
}

void dfs(int u,int father){
    depth[u]=depth[father]+1;
    fa[u][0]=father;
    // 2^i祖先为2^i-1级祖先的2^i-1级祖先
    for(int i=1;i<=lg[depth[u]];i++){
        fa[u][i]=fa[fa[u][i-1]][i-1];
    }
    for(int v:g[u]){
        if(v==father) continue;
        dfs(v,u);
    }
```

```
}

int lca(int a,int b){
    if(depth[a]<depth[b]) swap(a,b);
    // while(depth[a]>depth[b]){
        // a=fa[a][lg[depth[a]-depth[b]]-1];
    // }
    int k=depth[a]-depth[b];
    for(int i=lg[k];~i;i--){  //向上走k步
        if((k>>i)&1) a=fa[a][i];
    }
    if(a==b) return a;
    for(int i=lg[depth[a]];~i;i--){
        if(fa[a][i]^fa[b][i]){  //fa[a][i]!=fa[b][i]
            a=fa[a][i];
            b=fa[b][i];
        }
    }
    return fa[a][0];
}

int main(){
    cin>>n>>m>>root;
    init();
    for(int i=0;i<n-1;i++){
        int x,y;cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
    dfs(root,0);
    for(int i=0;i<m;i++){
        int a,b;cin>>a>>b;
        cout<<lca(a,b)<<"\n";
    }
    return 0;
}
```

性质

```
int dis(int a,int b){
    return depth[a]+depth[b]-2*depth[lca(a,b)];
}
```

```
//x向上走k步
int up_pos(int x,int k){
    for(int i=lg[k];~i;i--){
        if((k>>i)&1) x=fa[x][i];
    }
    return x;
}
```

树上两点之间最短距离

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e4+5;

struct node{
    int ne,w;
};
```

```cpp
int n,m;
vector<node> g[N];
int depth[N],fa[N][20],lg[N],cost[N][20];

void init(){
    // log2(i)
    for(int i=2;i<=n;i++){
        lg[i]=lg[i>>1]+1;
    }
}

void dfs(int u,int father){
    depth[u]=depth[father]+1;
    fa[u][0]=father;
    // 2^i祖先为2^i-1级祖先的2^i-1级祖先
    for(int i=1;(1<<i)<=depth[u];i++){
        fa[u][i]=fa[fa[u][i-1]][i-1];
        cost[u][i]=cost[fa[u][i-1]][i-1]+cost[u][i-1];
    }
    for(auto v:g[u]){
        if(v.ne==father) continue;
        cost[v.ne][0]=v.w;
        dfs(v.ne,u);
    }
}

int lca(int a,int b){
    int ans=0;
    if(depth[a]<depth[b]) swap(a,b);
    int k=depth[a]-depth[b];
    for(int i=lg[k];~i;i--){  //向上走k步
        if((k>>i)&1){
            ans+=cost[a][i];
            a=fa[a][i];
        }
    }
    if(a==b) return ans;
    for(int i=lg[depth[a]];i>=0;i--){
        if(fa[a][i]^fa[b][i]){
            ans+=cost[a][i]+cost[b][i];
            a=fa[a][i];
            b=fa[b][i];
        }
    }
    ans+=cost[a][0]+cost[b][0];
    return ans;
}

void solve(){
    cin>>n>>m;
    init();
    for(int i=0;i<n-1;i++){
        int x,y,k;cin>>x>>y>>k;
        g[x].push_back({y,k});
        g[y].push_back({x,k});
    }
    dfs(1,0);
    for(int i=0;i<m;i++){
        int a,b;cin>>a>>b;
        cout<<lca(a,b)<<"\n";
    }
}
```

```
int main(){
    solve();
    return 0;
}
```

树的直径

两次 DFS

```
#include<bits/stdc++.h>
using namespace std;
const int N=100005;

int c,d[N];
vector<int> g[N];

void dfs(int u,int fa){
    for(int v:g[u]){
        if(v==fa) continue;
        d[v]=d[u]+1;
        if(d[v]>d[c]) c=v;
        dfs(v,u);
    }
}

int main(){
    int n;cin>>n;
    for(int i=1;i<n;i++){
        int u,v;cin>>u>>v;
        g[u].push_back(v);g[v].push_back(u);
    }
    dfs(1,0);
    d[c]=0;dfs(c,0);
    cout<<d[c]<<"\n";
    return 0;
}
```

树形 DP

```
#include<bits/stdc++.h>
using namespace std;
const int N=100005;

int d1[N],d2[N],d;//最远距离d1 次远距离d2
vector<int> g[N];

void dfs(int u,int fa){
    d1[u]=d2[u]=0;
    for(int v:g[u]){
        if(v==fa) continue;
        dfs(v,u);
        int t=d1[v]+1;
        if(t>d1[u]) d2[u]=d1[u],d1[u]=t;
        else if(t>d2[u]) d2[u]=t;
    }
    d=max(d,d1[u]+d2[u]);
}

int main(){
    int n;cin>>n;
```

```
    for(int i=1;i<n;i++){
        int u,v;cin>>u>>v;
        g[u].push_back(v);g[v].push_back(u);
    }
    dfs(1,0);
    cout<<d<<"\n";
    return 0;
}
```

树的重心

重心定义：重心是指树中的一个结点，如果将这个点删除后，剩余各个连通块中点数的最大值最小，那么这个节点被称为树的重心。

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+10;

int n;
vector<int> g[N];
int siz[N],weight[N],centroid[2];
//siz[u]以u为根的树的节点数,包括u
//weight[u]记录删除u点后最大的连通子图节点数
//centroid[0]用于记录树的重心

void dfs(int u,int fa){
    siz[u]=1;weight[u]=0;
    for(int v:g[u]){
        if(v==fa) continue;
        dfs(v,u);
        siz[u]+=siz[v];
        weight[u]=max(weight[u],siz[v]);
    }
    weight[u]=max(weight[u],n-siz[u]);
    if(weight[u]<=n/2){
        centroid[centroid[0]!=0]=u;
    }
}

int main(){
    cin>>n;
    for(int i=1;i<n;i++){
        int a,b;cin>>a>>b;
        g[a].push_back(b);g[b].push_back(a);
    }
    dfs(1,0);
    cout<<weight[centroid[0]]<<"\n";
    return 0;
}
```

最小生成树(MST)

Kruskal算法常用于稀疏图(边少)，而Prim算法常用于稠密图(点少)

Kruskal 算法

$O(mlogm)$

```
#include <bits/stdc++.h>
using namespace std;
const int N=2e5+10,INF=0x3f3f3f3f;
```

```
int n,m,fa[N],depth[N];
struct edge{
    int u,v,w;
}e[N];

bool cmp(edge x,edge y){
    return x.w<y.w;
}

void init(int n){
    for(int i=1;i<=n;i++)
        fa[i]=i,depth[i]=1;
}
//查询树的根
int find(int x){
    if(x!=fa[x]) fa[x]=find(fa[x]);
    return fa[x];
}
//合并a和b所属的集合
void unite(int a,int b){
    a=find(a),b=find(b);
    if(depth[a]==depth[b]){
        depth[a]=depth[a]+1;
        fa[b]=a;
    }
    else{
        if(depth[a]<depth[b]) fa[a]=b;
        else fa[b]=a;
    }
}
//判断a和b是否属于同一个集合
bool same(int a,int b){
    return find(a)==find(b);
}

int Kruskal(){
    int res=0,count=0;
    for(int i=0;i<m;i++){
        if(!same(e[i].u,e[i].v)){
            unite(e[i].u,e[i].v);
            res+=e[i].w;
            count++;
        }
        if(count==n-1) return res;
    }
    return -1;
}

int main(){
    cin>>n>>m;
    for(int i=0;i<m;i++){
        scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);
    }
    sort(e,e+m,cmp);
    init(n);
    int k=Kruskal();
    if(k!=-1) cout<<k;
    else puts("impossible");
    return 0;
}
```

Prim 算法

朴素

```
int e[N][N],dis[N];
bool st[N];

void init(){
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(i==j) e[i][j]=0;
            else e[i][j]=INF;
        }
    }
    memset(dis,INF,sizeof dis);
}

int prim(){
    int sum=0;
    for(int i=0;i<n;i++){
        int t=-1;
        for(int j=1;j<=n;j++){
            if(!st[j]&&(t==-1||dis[j]<dis[t])) t=j;
        }
        if(i&&dis[t]==INF) return INF;
        st[t]=1;
        if(i) sum+=dis[t];

        for(int k=1;k<=n;k++){
            if(!st[k]&&dis[k]>e[t][k]){
                dis[k]=e[t][k];
            }
        }
    }
    return sum;
}

e[v][u]=e[u][v]=min(e[u][v],w);
```

```
int dis[N],tot;
bool vis[N];
struct node{
    int to,cost;
};
vector<node> g[N];

void init(){
    memset(dis,INF,sizeof dis);
    dis[1]=0;
}

int prim(){
    init();
    int u=1,sum=0;
    for(auto x:g[u]){
        int v=x.to,w=x.cost;
        dis[v]=min(dis[v],w);
    }
    while(++tot<n){
        int minn=INF;
        vis[u]=1;
```

```
        u=-1;
        for(int i=1;i<=n;i++){
            if(!vis[i]&&minn>dis[i]){
                minn=dis[i];
                u=i;
            }
        }
        if(u==-1) return INF;
        sum+=minn;
        for(auto x:g[u]){
            int v=x.to,w=x.cost;
            if(!vis[v]&&dis[v]>w){
                dis[v]=w;
            }
        }
    }
    return sum;
}

g[u].push_back({v,w});
g[v].push_back({u,w});
```

堆优化

```
typedef pair<int, int> PII;
int e[N][N],dis[N];
bool vis[N];
struct node{
    int to,cost;
};
vector<node> g[N];
priority_queue<PII, vector<PII>, greater<PII>> q;

void init(){
    memset(dis,INF,sizeof dis);
    dis[1]=0;
}

int prim(){
    init();
    int sum=0,cnt=0;
    q.push({0,1});
    while(q.size()&&cnt<n){
        auto t=q.top();q.pop();
        int u=t.second,d=t.first;
        if(vis[u]) continue;
        vis[u]=1;sum+=d;cnt++;
        for(auto x:g[u]){
            int v=x.to,w=x.cost;
            if(dis[v]>w){
                dis[v]=w;
                q.push({dis[v],v});
            }
        }
    }
    if(cnt!=n) return INF;
    return sum;
}

g[u].push_back({v,w});
g[v].push_back({u,w});
```

Borůvka (Sollin) 算法

```
#include <bits/stdc++.h>
using namespace std;

const int MaxN = 5000 + 5, MaxM = 200000 + 5;

int N, M;
int U[MaxM], V[MaxM], W[MaxM];
bool used[MaxM];
int par[MaxN], Best[MaxN];

void init() {
    scanf("%d %d", &N, &M);

    for (int i = 1; i <= M; ++i)
        scanf("%d %d %d", &U[i], &V[i], &W[i]);
}

void init_dsu() {
    for (int i = 1; i <= N; ++i)
        par[i] = i;
}

int get_par(int x) {
    if (x == par[x]) return x;
    else return par[x] = get_par(par[x]);
}

inline bool Better(int x, int y) {
    if (y == 0) return true;
    if (W[x] != W[y]) return W[x] < W[y];
    return x < y;
}

void Boruvka() {
    init_dsu();
    int merged = 0, sum = 0;
    bool update = true;
    while (update) {
        update = false;
        memset(Best, 0, sizeof Best);
        for (int i = 1; i <= M; ++i) {
            if (used[i] == true) continue;
            int p = get_par(U[i]), q = get_par(V[i]);
            if (p == q) continue;
            if (Better(i, Best[p]) == true) Best[p] = i;
            if (Better(i, Best[q]) == true) Best[q] = i;
        }
        for (int i = 1; i <= N; ++i)
            if (Best[i] != 0 && used[Best[i]] == false) {
                update = true;
                merged++; sum += W[Best[i]];
                used[Best[i]] = true;
                par[get_par(U[Best[i]])] = get_par(V[Best[i]]);
            }
    }
    if (merged == N - 1) printf("%d\n", sum);
    else puts("orz");
}

int main() {
```

```
    init();
    Boruvka();
    return 0;
}
```

非严格次小生成树

```
int n,m,fa[N];
ll d[N][N];//用来维护u到v的距离，
vector<int> p[N]; //扩展路径

struct edge{
    int u,v;
    ll w;
    bool vis;//判断边是否加入最小生成树的集合
}e[M];

bool cmp(edge x,edge y){
    return x.w<y.w;
}

int find(int x){
    if(x!=fa[x]) fa[x]=find(fa[x]);
    return fa[x];
}

void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        p[i].clear();
        p[i].push_back(i);
        fa[i]=i;
    }
    for(int i=1;i<=m;i++){
        cin>>e[i].u>>e[i].v>>e[i].w;
        e[i].vis=0;
    }
    sort(e+1,e+1+m,cmp);
    ll ans=0;
    int tot=0;
    for(int i=1;i<=m;i++){
        int u=find(e[i].u),v=find(e[i].v);
        if(u==v) continue;
        ans+=e[i].w;
        e[i].vis=1;
        fa[u]=v;
        tot++;
        for(int x:p[u]){
            for(int y:p[v]){
                d[x][y]=d[y][x]=e[i].w;
            }
        }
        for(int x:p[u]) p[v].push_back(x);
        if(tot==n-1) break;
    }
    ll res=INF; //非严格次小生成树
    for(int i=1;i<=m;i++){
        if(!e[i].vis) res=min(res,ans+e[i].w-d[e[i].u][e[i].v]);
    }
    if(res==ans) cout<<"Not Unique!\n";
    else cout<<ans<<"\n";
}
```

严格次小生成树

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+10,M=3e5+10,INF=0x3f3f3f3f;
int n,m,fa[N],depth[N];
ll mst,ans=1e18;
int p[N][25];
ll max1[N][25],max2[N][25];
bool used[M];
struct edge{
    int u,v,w;
}e[M];

struct node{
    int to,cost;
};
vector<node> g[N];

bool cmp(edge x,edge y){
    return x.w<y.w;
}

void init(int n){
    for(int i=1;i<=n;i++) fa[i]=i;
}
//查询树的根
int find(int x){
    if(x!=fa[x]) fa[x]=find(fa[x]);
    return fa[x];
}
//合并a和b所属的集合
void unite(int a,int b){
    a=find(a),b=find(b);
    fa[a]=fa[b];
}
//判断a和b是否属于同一个集合
bool same(int a,int b){
    return find(a)==find(b);
}

void Kruskal(){
    int count=0;
    for(int i=0;i<m;i++){
        if(!same(e[i].u,e[i].v)){
            unite(e[i].u,e[i].v);
            mst+=e[i].w;
            g[e[i].u].push_back({e[i].v,e[i].w});
            g[e[i].v].push_back({e[i].u,e[i].w});
            used[i]=1;
            count++;
        }
        if(count==n-1) return;
    }
}

void dfs(int u,int fa,ll w){
    p[u][0]=fa;
    depth[u]=depth[fa]+1;
    max1[u][0]=w;//最大边权
```

```
    max2[u][0]=-INF;//次大边权
    for(int i=1;(1<<i)<=depth[u];i++){
        p[u][i]=p[p[u][i-1]][i-1];
//      max1[u][i]=max(max1[u][i-1],max1[p[u][i-1]][i-1]);
//      max2[u][i]=max(max2[u][i-1],max2[p[u][i-1]][i-1]);
//      if(max1[u][i-1]>max1[p[u][i-1]][i-1]){
//          max2[u][i]=max(max2[u][i],max1[p[u][i-1]][i-1]);
//      }
//      else if(max1[u][i-1]<max1[p[u][i-1]][i-1]){
//          max2[u][i]=max(max2[u][i],max1[u][i-1]);
//      }
        int kk[4]={max1[u][i-1],max1[p[u][i-1]][i-1],
                    max2[u][i-1],max2[p[u][i-1]][i-1]};
        sort(kk,kk+4);
        max1[u][i]=kk[3];
        int ptr=2;
        while(ptr>=0&&kk[ptr]==kk[3]) ptr--;
        max2[u][i]=(ptr==-1?-INF:kk[ptr]);
    }
    for(auto x:g[u]){
        int v=x.to,w=x.cost;
        if(v==fa) continue;
        dfs(v,u,w);
    }
}

int lca(int a,int b){
    if(depth[a]<depth[b]) swap(a,b);
    for(int i=21;i>=0;i--){
        if(depth[p[a][i]]>=depth[b]) a=p[a][i];
    }
    if(a==b) return a;
    for(int i=21;i>=0;i--){
        if(p[a][i]!=p[b][i]) a=p[a][i],b=p[b][i];
    }
    return p[a][0];
}

ll query(int a,int b,ll ma){
    ll res=-INF;
    for(int i=21;i>=0;i--){
        if(depth[p[a][i]]>=depth[b]){
            if(max1[a][i]!=ma) res=max(res,max1[a][i]);
            else res=max(res,max2[a][i]);
            a=p[a][i];
        }
    }
    return res;
}

int main(){
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v,w;cin>>u>>v>>w;
        e[i]={u,v,w};
    }
    sort(e,e+m,cmp);
    init(n);
    Kruskal();

    dfs(1,0,-INF);
```

```
    for(int i=0;i<m;i++){
        if(!used[i]){
            int u=e[i].u,v=e[i].v,w=e[i].w;
            int _lca=lca(u,v);
            ll tmpa=query(e[i].u,_lca,w);
            ll tmpb=query(e[i].v,_lca,w);
            if(max(tmpa,tmpb)>-INF) ans=min(ans,mst-max(tmpa,tmpb)+w);
        }
    }
    cout<<(ans==1e18?-1:ans)<<'\n';
    return 0;
}
```

最小生成树计数（具有相同权值的边不会超过10条）

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=110,M=1010,mod=31011;
struct edge{
    int u,v,w;
    bool operator < (const edge &t){
        return w<t.w;
    }
}e[M];

struct node{
    int l,r,num; //起点,终点,最小生成树中所包含的数量
}a[M];

int fa[N],n,m;

void init(int n){
    for(int i=1;i<=n;i++) fa[i]=i;
}

int find(int x){ //在有dfs回溯的时侯,不能采用压缩路径
    return fa[x]==x ? x : find(fa[x]);
}

ll dfs(int i,int now,int k){  //在第几层(按权值分层) 边集中的位置 当前层已选择的数量
    ll sum=0;
    if (now>a[i].r) return k==a[i].num;
    int u=find(e[now].u),v=find(e[now].v);
    if(u!=v){
        fa[u]=v;
        sum+=dfs(i,now+1,k+1);
        fa[u]=u,fa[v]=v;
    }
    return sum+dfs(i,now+1,k);
}
int main() {
    cin>>n>>m;
    init(n);
    for(int i=1;i<=m;i++) cin>>e[i].u>>e[i].v>>e[i].w;
    sort(e+1,e+1+m);
    int tot=0,cnt=0;
    for(int i=1;i<=m;i++){
        if(e[i].w!=e[i-1].w) a[++cnt].l=i,a[cnt-1].r=i-1;
        int u=find(e[i].u),v=find(e[i].v);
        if(u==v) continue;
        tot++;
```

```
            fa[u]=v;
            a[cnt].num++;
        }
        if(tot<n-1){  //无解
            return cout<<"0\n",0;
        }
        a[cnt].r=m;

        init(n);
        ll ans=1;
        for(int i=1;i<=cnt;i++){
            ans=ans*dfs(i,a[i].l,0)%mod;
            for(int j=a[i].l;j<=a[i].r;j++){
                int u=find(e[j].u),v=find(e[j].v);
                if(u!=v) fa[u]=v;
            }
        }
        cout<<ans<<"\n";
        return 0;
}
```

普通生成树计数(矩阵树定理)

```
ll d[N],g[N][N],a[N][N];//d入度 g邻接矩阵 a基尔霍夫矩阵

//求基尔霍夫矩阵的任意n-1阶余子式
ll gauss(int n) {
    ll sum = 1;
    for (int i = 2; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) {
            while (a[j][i]) {
                ll t = a[i][i] / a[j][i];
                for (int k = i; k <= n; k++) a[i][k] = (a[i][k] - a[j][k] * t);
                for (int k = i; k <= n; k++) swap(a[i][k], a[j][k]);
                sum = -sum;
            }
        }
        if (a[i][i] == 0) return 0;
        sum *= a[i][i];
    }
    return abs(sum);
}

void solve(){
    int n,m;cin>>n>>m;
    for(int i=1;i<=n;i++){
        d[i]=0;
        for(int j=1;j<=n;j++){
            g[i][j]=0;
        }
    }
    while(m--){
        int u,v;cin>>u>>v;
        g[u][v]=g[v][u]=1;
        d[u]++,d[v]++;
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
             a[i][j]= i==j ? d[i] : -g[i][j];
        }
    }
    cout<<gauss(n)<<"\n";
```

```
}
```

最小生成树计数(矩阵树定理)

```
#include <bits/stdc++.h>
using namespace std;
#define mod 31011
#define N 101
#define M 1001
struct Edge {
    int x, y, w;
} e[M], se[N];
bool cmp(Edge a, Edge b) { return a.w < b.w; }
int f[N];
int find(int x) { return f[x] == x ? x : find(f[x]); }
int a[N][N];
int gauss(int n) {
    int res = 1;
    for (int i = 1; i < n; i++) {
        for (int j = i + 1; j < n; j++)
            while (a[j][i]) {
                int t = a[i][i] / a[j][i];
                for (int k = i; k < n; k++)
                    a[i][k] = (a[i][k] - t * a[j][k] + mod) % mod;
                swap(a[j], a[i]);
                res = -res;
            }
        res = res * a[i][i] % mod;
    }
    return (res + mod) % mod;
}
int col[M], cw[M], n, m;
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) f[i] = i;
    for (int i = 1; i <= m; i++) scanf("%d%d%d", &e[i].x, &e[i].y, &e[i].w);
    sort(e + 1, e + m + 1, cmp);
    int num = 1, w_num = 0;
    for (int i = 1; num < n && i <= m; i++) {
        int fx = find(e[i].x), fy = find(e[i].y);
        if (fx == fy) continue;
        f[fx] = fy, se[num++] = e[i];
        if (e[i].w != cw[w_num]) cw[++w_num] = e[i].w;
    }
    if (num < n) {
        printf("0\n");
        return 0;
    }
    int ans = 1;
    for (int i = 1; i <= w_num; i++) {
        for (int j = 1; j <= n; j++) f[j] = j;
        for (int j = 1; j < n; j++)
            if (se[j].w != cw[i]) {
                int fx = find(se[j].x), fy = find(se[j].y);
                if (fx != fy) f[fx] = fy;
            }
        int col_num = 0;
        for (int j = 1; j <= n; j++)
            if (find(j) == j) col[j] = ++col_num;
        for (int j = 1; j <= n; j++) col[j] = col[find(j)];
        memset(a, 0, sizeof(a));
        for (int j = 1; j <= m; j++)
```

```
            if (e[j].w == cw[i]) {
                int x = col[e[j].x], y = col[e[j].y];
                a[x][x]++, a[y][y]++, a[x][y]--, a[y][x]--;
            }
        ans = ans * gauss(col_num) % mod;
    }
    printf("%d\n", (ans + mod) % mod);
    return 0;
}
```

有向图的强连通分量

Tarjan算法

```
int n,m;
vector<int> g[N];
int dfn[N],low[N]; //dfn[u]:遍历到u的时间 low[u]:u通过有向边可回溯的最早次序(环中最小的dfn)
bool vis[N];//当前这点是不是在栈当中
int cnt,timetag;
int col[N],siz[N];//col:表示在哪一块 siz:大小
int in[N],out[N];
stack<int> s;

void tarjan(int u){
    dfn[u]=low[u]=++timetag;
    vis[u]=1;
    s.push(u);
    for(auto v:g[u]){
        if(!dfn[v]){
            tarjan(v);
            low[u]=min(low[u],low[v]); //回溯
        }
        else if(vis[v]){ //是不是在这个环中
            low[u]=min(low[u],low[v]);
        }
    }
    if(dfn[u]==low[u]){
        cnt++;
        while(1){
            int t=s.top();s.pop();
            vis[t]=0;
            col[t]=cnt;
            siz[cnt]++;
            if(t==u) break;
        }
    }
}

void solve(){
    cin>>n>>m;
    while(m--){
        int a,b;cin>>a>>b;
        g[a].push_back(b);
    }
    for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i);
    for(int u=1;u<=n;u++){
        for(auto v:g[u]){
            if(col[u]!=col[v]) in[col[v]]++,out[col[u]]++;
        }
    }
```

```
}
```

缩点

```
int n,m;
int w[N],dp[N],dis[N];
vector<int> g[N],g1[N],g2[N];
int dfn[N],low[N]; //dfn[u]:遍历到u的时间 low[u]:u通过有向边可回溯的最早次序(环中最小的dfn)
bool vis[N];//当前这点是不是在栈当中
int cnt,timetag;
int col[N];//col:表示在哪一块
int in[N];
stack<int> s;
vector<int> topo;

void tarjan(int u){
    dfn[u]=low[u]=++timetag;
    vis[u]=1;
    s.push(u);
    for(auto v:g[u]){
        if(!dfn[v]){
            tarjan(v);
            low[u]=min(low[u],low[v]); //回溯
        }
        else if(vis[v]){ //是不是在这个环中
            low[u]=min(low[u],low[v]);
        }
    }
    if(dfn[u]==low[u]){
        cnt++;
        while(1){
            int t=s.top();s.pop();
            vis[t]=0;
            col[t]=cnt;
            dis[cnt]+=w[t];
            if(t==u) break;
        }
    }
}

void toposort(){
    queue<int> q;
    for(int i=1;i<=n;i++) if(!in[i]) q.push(i);
    while(!q.empty()){
        int u=q.front();q.pop();
        topo.push_back(u);
        for(auto v:g1[u]){
            in[v]--;
            if(!in[v]) q.push(v);
        }
    }
}

void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>w[i];
    while(m--){
        int u,v;cin>>u>>v;
        g[u].push_back(v);
    }
    for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i);
    for(int u=1;u<=n;u++){
```

```
        for(auto v:g[u]){
            if(col[u]!=col[v]){
                g1[col[u]].push_back(col[v]);
                g2[col[v]].push_back(col[u]);
                in[col[v]]++;
            }
        }
    }
    toposort();
    for(auto u:topo){
        dp[u]=dis[u];
        for(auto v:g2[u]){
            dp[u]=max(dp[u],dp[v]+dis[u]);
        }
    }
    int ans=0;
    for(int i=1;i<=cnt;i++) ans=max(ans,dp[i]);
    cout<<ans<<"\n";
}
```

无向图的双连通分量

```
int n,m;
int h[N],e[M],ne[M],idx;
int dfn[N],low[N]; //dfn[u]:遍历到u的时间 low[u]:u通过有向边可回溯的最早次序(环中最小的dfn)
bool bridge[M];
int cnt,timetag;
int col[N];//col:表示在哪一块
int d[N]; //度数
stack<int> s;

void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}

void tarjan(int u,int from){
    dfn[u]=low[u]=++timetag;
    s.push(u);
    for(int i=h[u];~i;i=ne[i]){
        int v=e[i];
        if(!dfn[v]){
            tarjan(v,i);
            low[u]=min(low[u],low[v]); //回溯
            if(dfn[u]<low[v]) bridge[i]=bridge[i^1]=1; //双向边^1
        }
        else if(i!=(from^1)){ //上一次已经遍历过反向边
            low[u]=min(low[u],dfn[v]);
        }
    }
    if(dfn[u]==low[u]){
        cnt++;
        while(1){
            int t=s.top();s.pop();
            col[t]=cnt;
            if(t==u) break;
        }
    }
}

void solve(){
    memset(h,-1,sizeof h);
    cin>>n>>m;
```

```
    while(m--){
        int a,b;cin>>a>>b;
        add(a,b),add(b,a);
    }
    tarjan(1,-1);
    for(int i=0;i<idx;i++){
        if(bridge[i]) d[col[e[i]]]++;
    }
    int ans=0;
    for(int i=1;i<=cnt;i++){
        if(d[i]==1) ans++;
    }
    cout<<(ans+1)/2<<"\n";
}
```

割点

```
int n,m,ans;
vector<int> g[N];
int dfn[N],low[N],timetag;
//dfn[u]:遍历到u的时间 low[u]:u通过有向边可回溯的最早次序(环中最小的dfn)
bool flag[N];

void tarjan(int u,int fa){
    dfn[u]=low[u]=++timetag;
    int child=0;
    for(auto v:g[u]){
        if(!dfn[v]){
            child++;
            tarjan(v,u);
            low[u]=min(low[u],low[v]); //回溯
            if(u!=fa&&dfn[u]<=low[v]&&!flag[u]){
                flag[u]=1;
                ans++;
            }
        }
        else if(fa!=v){ //是不是在这个环中
            low[u]=min(low[u],dfn[v]);
        }
    }
    if(fa==u&&child>=2&&!flag[u]){
        flag[u]=1;
        ans++;
    }
}

void solve(){
    cin>>n>>m;
    while(m--){
        int a,b;cin>>a>>b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    for(int i=1;i<=n;i++){
        if(!dfn[i]){
            tarjan(i,i);
        }
    }
    cout<<ans<<"\n";
    for(int i=1;i<=n;i++){
        if(flag[i]) cout<<i<<" ";
    }
```

```
}
```

二分图

定理

> **二分图不存在长度为奇数的环**
>
> **最大匹配数**：最大匹配的匹配边的数目
>
> **最小点覆盖数**：选取最少的点，使任意一条边至少有一个端点被选择
>
> **最大独立数**：选取最多的点，使任意所选两点均不相连
>
> **最小路径覆盖数**：对于一个 DAG（有向无环图），选取最少条路径，使得每个顶点属于且仅属于一条路径。路径长可以为 0（即单个点）。
>
> **最小路径重复点覆盖**：在最小路径覆盖问题的基础上，去掉互不相交。
>
> 定理1：最大匹配数 = 最小点覆盖数
>
> 定理2：最大独立数 = 顶点数 - 最大匹配数
>
> 定理3：最小路径覆盖数 = 顶点数 - 最大匹配数
>
> 定理4：记原图G，求传递闭包后的图G'，则G的最小路径重复点覆盖=G'的最小路径覆盖

染色法判定二分图

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+10;
vector<int> g[N];
int color[N]; //顶点颜色(1 or -1)

bool dfs(int v,int c){
    color[v]=c;
    for(int i=0;i<g[v].size();i++){
        if(color[g[v][i]]==c) return false; //相邻点同色
        if(color[g[v][i]]==0 && !dfs(g[v][i],-c)) return false; //未染色,染成-c
    }
    return true;
}

int main(){
    int n,m;
    cin>>n>>m;
    while(m--){
        int u,v;
        scanf("%d%d",&u,&v);
        g[u].push_back(v);
        g[v].push_back(u);
    }

    for(int i=0;i<n;i++){
        if(color[i]==0 && !dfs(i,1)){
            printf("No\n");
            return 0;
        }
    }
    printf("Yes\n");
}
```

二分图的最大匹配数(匈牙利算法)

```
vector<int> g[N]; //图的邻接表
int match[N]; //所匹配的顶点
bool st[N]; //(临时)预定标记
int n1,n2,m;

bool dfs(int u){
    for(int i=0;i<g[u].size();i++){
        int v=g[u][i];
        if(!st[v]){
            st[v]=1;
            if(!match[v]||dfs(match[v])){
                match[v]=u;
                return true;
            }
        }
    }
    return false;
}

int bipartite_matching(){
    int res=0;
    for(int u=1;u<=n1;u++){
        memset(st,0,sizeof st);
        if(dfs(u)) res++;
    }
    return res;
}

void solve(){
    cin>>n1>>n2>>m;
    while(m--){
        int u,v;cin>>u>>v;
        g[u].push_back(v);
    }
    cout<<bipartite_matching()<<'\n';
}
```

二分图最大权完美匹配(KM算法)

dfs写法

```
//时间复杂度：O(n^3) 最坏O(n^4)
int nx,ny;
int g[N][N];
int match[N]; //左边所匹配的右边顶点
int lx[N],ly[N]; //左边的期望值   右边期望值
int slack[N]; //松弛   右边能被左边匹配最少还需要多少期望值
bool visx[N],visy[N];

bool dfs(int x){
    visx[x]=1;
    for(int y=0;y<ny;y++){
        if(visy[y]) continue;
        int gap=lx[x]+ly[y]-g[x][y];
        if(!gap){ //符合要求
            visy[y]=1;
            if(match[y]==-1||dfs(match[y])){
                match[y]=x;
                return 1;
```

```
            }
        }
        else slack[y]=min(slack[y],gap);
    }
    return 0;
}

int KM(){
    memset(match,-1,sizeof match);
    memset(ly,0,sizeof ly);
    for(int i=0;i<nx;i++){
        lx[i]=-INF;
        for(int j=0;j<ny;j++){
            lx[i]=max(lx[i],g[i][j]);
        }
    }
    for(int x=0;x<nx;x++){
        for(int j=0;j<ny;j++) slack[j]=INF;
        while(1){
            memset(visx,0,sizeof visx);
            memset(visy,0,sizeof visy);
            if(dfs(x)) break; //找到匹配的

            //找不到就降低期望值
            int d=INF;//最小可降低的期望值
            for(int j=0;j<ny;j++){
                if(!visy[j]) d=min(d,slack[j]);
            }

            //所有访问过的左边降低的期望值
            for(int j=0;j<nx;j++){
                if(visx[j]) lx[j]-=d;
            }

            for(int j=0;j<ny;j++){
                //所有访问过的右边增加的期望值
                if(visy[j]) ly[j]+=d;
                //没有访问过的右边  因为左边期望值降低  离右边匹配更近了
                else slack[j]-=d;
            }
        }
    }

    int ans=0;
    for(int i=0;i<ny;i++){
        if(~match[i]) ans+=g[match[i]][i];
    }
    return ans;
}

void solve(){
    int n;cin>>n;
    nx=ny=n;
    for(int i=0;i<nx;i++){
        for(int j=0;j<ny;j++){
            cin>>g[i][j];
        }
    }
    cout<<KM()<<"\n";
}
```

bfs写法

```
//时间复杂度：O(n^3)
int nx,ny;
ll g[N][N];
int match[N]; //左边所匹配的右边顶点
ll lx[N],ly[N]; //左边的期望值  右边期望值
ll slack[N]; //松弛  右边能被左边匹配最少还需要多少期望值
bool visy[N];
int pre[N]; //右边同侧的前驱点

void bfs(int u){
    memset(pre,0,sizeof pre);
    for(int j=1;j<=ny;j++) slack[j]=INF;
    int x,y=0,yy=0;
    match[y]=u;
    while(1){
        x=match[y];
        ll delta=INF;
        visy[y]=1;
        for(int i=1;i<=ny;i++){
            if(visy[i])continue;
            if(slack[i]>lx[x]+ly[i]-g[x][i]){ // 更新右边每个点的最小松弛值
                slack[i]=lx[x]+ly[i]-g[x][i];
                pre[i]=y;
            }
            if(slack[i]<delta){ //更新全局最小松弛值. 如果slack是0说明边在里面
                delta=slack[i];
                yy=i;
            }
        }
        for(int i=0;i<=ny;i++){
            if(visy[i]) lx[match[i]]-=delta,ly[i]+=delta; // v已经在增广路中了
            else slack[i]-=delta;
        }
        y=yy;
        if(match[y]==-1) break; // 找到了增广路
    }
    while(y){
        match[y]=match[pre[y]];
        y=pre[y];
    }
}

ll KM(){
    memset(match,-1,sizeof match);
    memset(lx,0,sizeof lx);
    memset(ly,0,sizeof ly);

    for(int x=1;x<=nx;x++){
        memset(visy,0,sizeof visy);
        bfs(x);
    }

    ll ans=0;
    for(int i=1;i<=ny;i++){
        if(~match[i]) ans+=g[match[i]][i];
    }
    return ans;
}

void solve(){
```

```
    int n,m;cin>>n>>m;
    nx=ny=n;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            g[i][j]=-INF;
        }
    }
    while(m--){
        int x,y;ll v;cin>>x>>y>>v;
        g[x][y]=max(g[x][y],v);
    }
    cout<<KM()<<"\n";
    for(int i=1;i<=n;i++) cout<<match[i]<<" \n"[i==n];
}
```

传递闭包

```
void Warshall(){
    for(int j=1;j<=n;j++){ //第1列到最后列
        for(int i=1;i<=n;i++){ //第j列从第1行到最后行
            if(b[i][j]){
                for(int k=1;k<=n;k++) {
                    b[i][k]|=b[j][k];
                }
            }
        }
    }
}
```

欧拉回路

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+5,M=4e5+5;

int n,m;
int h[N],e[M],ne[M],idx;
int deg[N],din[N],dout[N];
bool st[M];
vector<int> ans;

void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}

void dfs1(int u){
    while(~h[u]){
        int i=h[u];
        if(st[i]){
            h[u]=ne[i];
            continue;
        }
        h[u]=ne[i];
        st[i]=st[i^1]=1;
        dfs1(e[i]);
        int t=i/2+1;
        if(i&1) ans.push_back(-t);
        else ans.push_back(t);
    }
}
```

```
void dfs2(int u){
    while(~h[u]){
        int i=h[u];
        if(st[i]){
            h[u]=ne[i];
            continue;
        }
        h[u]=ne[i];
        st[i]=1;
        dfs2(e[i]);
        ans.push_back(i+1);
    }
}

void solve1(){
    memset(h,-1,sizeof h);
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v;cin>>u>>v;
        add(u,v);add(v,u);
        deg[u]++,deg[v]++;
    }
    for(int i=1;i<=n;i++){
        if(deg[i]&1){
            cout<<"NO\n";
            return;
        }
    }
    for(int i=1;i<=n;i++){
        if(deg[i]){dfs1(i);break;}
    }
    if(ans.size()==m){
        cout<<"YES\n";
        for(int i=m-1;~i;i--) cout<<ans[i]<<" ";
    }
    else cout<<"NO\n";
}

void solve2(){
    memset(h,-1,sizeof h);
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v;cin>>u>>v;
        add(u,v);
        dout[u]++,din[v]++;
    }
    for(int i=1;i<=n;i++){
        if(din[i]!=dout[i]){
            cout<<"NO\n";
            return;
        }
    }

    for(int i=1;i<=n;i++){
        if(din[i]+dout[i]){dfs2(i);break;}
    }
    if(ans.size()==m){
        cout<<"YES\n";
        for(int i=m-1;~i;i--) cout<<ans[i]<<" ";
    }
    else cout<<"NO\n";
}
```

```
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    int t;cin>>t;
    if(t==1) solve1();
    else solve2();
    return 0;
}
```

拓扑排序

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+10;

vector<int> g[N];
int depth[N];
vector<int> topo;
int n,m;

bool toposort(){
    queue<int> q;
    for(int i=1;i<=n;i++) if(!depth[i]) q.push(i);
    while(!q.empty()){
        int u=q.front();q.pop();
        topo.push_back(u);
        for(int v:g[u]){
            depth[v]--;
            if(!depth[v]) q.push(v);
        }
    }
    return topo.size()==n;
}

int main(){
    cin>>n>>m;
    while(m--){
        int a,b;cin>>a>>b;
        g[a].push_back(b);
        depth[b]++;
    }
    if(!toposort()) cout<<"-1";
    else for(int x:topo) cout<<x<<" ";
    return 0;
}
```

网络流

> 最大流等于最小割

最大流

EK算法

```
#include <bits/stdc++.h>
using namespace std;
int n,m,s,t,u,v;
long long w,ans,dis[520010];
int tot=1,vis[520010],pre[520010],head[520010],flag[2510][2510];

struct node{
```

```
    int to,net;
    long long val;
}e[520010];

void add(int u,int v,long long w){
    e[++tot].to=v;
    e[tot].val=w;
    e[tot].net=head[u];
    head[u]=tot;
    e[++tot].to=u;
    e[tot].val=0;
    e[tot].net=head[v];
    head[v]=tot;
}

int bfs(){  //bfs寻找增广路
    for(int i=1;i<=n;i++) vis[i]=0;
    queue<int> q;
    q.push(s);
    vis[s]=1;
    dis[s]=0x3f3f3f3f;
    while(!q.empty()){
        int x=q.front();
        q.pop();
        for(int i=head[x];i;i=e[i].net) {
            if(e[i].val==0) continue;  //我们只关心剩余流量>0的边
            int v=e[i].to;
            if(vis[v]==1) continue;  //这一条增广路没有访问过
            dis[v]=min(dis[x],e[i].val);
            pre[v]=i;  //记录前驱，方便修改边权
            q.push(v);
            vis[v]=1;
            if(v==t) return 1;  //找到了一条增广路
        }
    }
    return 0;
}

void update(){  //更新所经过边的正向边权以及反向边权
    int x=t;
    while(x!=s) {
        int v=pre[x];
        e[v].val-=dis[t];
        e[v^1].val+=dis[t]; //v^1即v的反向边
        x=e[v^1].to;
    }
    ans+=dis[t];   //累加每一条增广路经的最小流量值
}

int main(){
    cin>>n>>m>>s>>t;
    for(int i=1;i<=m;i++) {
        cin>>u>>v>>w;
        if(flag[u][v]==0){  //处理重边的操作（加上这个模板题就可以用Ek算法过了）
            add(u,v,w);
            flag[u][v]=tot;
        }
        else{
            e[flag[u][v]-1].val+=w;
        }
    }
    while(bfs()){  //直到网络中不存在增广路
```

```
        update();
    }
    cout<<ans<<"\n";
    return 0;
}
```

Dinic算法

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=10005,M=100005;
const ll INF=1e18;

struct NF{
    struct edge{
        int to;
        ll w;
        int nxt;
    }edge[M<<1];
    int head[N],tot=1;
    int depth[N],cur[N];
    int n,m,s,t;
    void init(int _n,int _m,int _s,int _t){ //初始化
        n=_n,m=_m,s=_s,t=_t;
    }
    void add(int u,int v,ll w){ //加边
        edge[++tot].w=w;
        edge[tot].to=v;
        edge[tot].nxt=head[u];
        head[u]=tot;
    }
    bool bfs(){  //分层
        for(int i=1;i<=n;i++) depth[i]=0;
        depth[s]=1;
        queue<int> q;
        q.push(s);
        while(!q.empty()){
            int u=q.front();q.pop();
            if(u==t) return 1;
            for (int i=head[u];i;i=edge[i].nxt) {
                int v=edge[i].to;
                if(edge[i].w>0&&!depth[v]){
                    depth[v]=depth[u]+1;
                    q.push(v);
                }
            }
        }
        return 0;
    }
    ll dfs(int u,ll dis){ //dis:该条路径当前最大允许流量
        if(u==t) return dis; //到达汇点后，成功流出流量即是该条路径最大允许流量
        ll sum=0;
        for(int i=cur[u];i&&dis;i=edge[i].nxt){ //多方向增广
            cur[u]=i; //当前弧优化
            int v=edge[i].to;
            if(depth[v]==depth[u]+1&&edge[i].w>0){ //可以往下增广
                ll d=dfs(v,min(dis,edge[i].w));
                if(!d) depth[v]=0;  //增广失败，删除该点
                edge[i].w-=d;     //增广成功
                edge[i^1].w+=d;
                sum+=d;    //成功流出的流量增加
```

```
                dis-=d;  //最大允许流量减少
            }
        }
        return sum;  //成功流出的流量总和
    }
    ll dinic(){
        ll ans=0;
        while(bfs()){
            for(int i=1;i<=n;i++) cur[i]=head[i];
            ans+=dfs(s,INF);
        }
        return ans;
    }
}fff;

int main() {
    int n,m,s,t;cin>>n>>m>>s>>t;
    fff.init(n,m,s,t);
    for(int i=1;i<=m;i++){
        int u,v;ll w;
        cin>>u>>v>>w;
        fff.add(u,v,w);
        fff.add(v,u,0);
    }
    cout<<fff.dinic()<<"\n";
    return 0;
}
```

atcoder最大流

```
namespace internal {
template <class T> struct simple_queue {
    std::vector<T> payload;
    int pos = 0;
    void reserve(int n) { payload.reserve(n); }
    int size() const { return int(payload.size()) - pos; }
    bool empty() const { return pos == int(payload.size()); }
    void push(const T& t) { payload.push_back(t); }
    T& front() { return payload[pos]; }
    void clear() {
        payload.clear();
        pos = 0;
    }
    void pop() { pos++; }
};
}  // namespace internal

namespace atcoder {

template <class Cap> struct mf_graph {
  public:
    mf_graph() : _n(0) {}
    explicit mf_graph(int n) : _n(n), g(n) {}

    int add_edge(int from, int to, Cap cap) {
        assert(0 <= from && from < _n);
        assert(0 <= to && to < _n);
        assert(0 <= cap);
        int m = int(pos.size());
        pos.push_back({from, int(g[from].size())});
        int from_id = int(g[from].size());
        int to_id = int(g[to].size());
```

```
        if (from == to) to_id++;
        g[from].push_back(_edge{to, to_id, cap});
        g[to].push_back(_edge{from, from_id, 0});
        return m;
    }

    struct edge {
        int from, to;
        Cap cap, flow;
    };

    edge get_edge(int i) {
        int m = int(pos.size());
        assert(0 <= i && i < m);
        auto _e = g[pos[i].first][pos[i].second];
        auto _re = g[_e.to][_e.rev];
        return edge{pos[i].first, _e.to, _e.cap + _re.cap, _re.cap};
    }
    std::vector<edge> edges() {
        int m = int(pos.size());
        std::vector<edge> result;
        for (int i = 0; i < m; i++) {
            result.push_back(get_edge(i));
        }
        return result;
    }
    void change_edge(int i, Cap new_cap, Cap new_flow) {
        int m = int(pos.size());
        assert(0 <= i && i < m);
        assert(0 <= new_flow && new_flow <= new_cap);
        auto& _e = g[pos[i].first][pos[i].second];
        auto& _re = g[_e.to][_e.rev];
        _e.cap = new_cap - new_flow;
        _re.cap = new_flow;
    }

    Cap flow(int s, int t) {
        return flow(s, t, std::numeric_limits<Cap>::max());
    }
    Cap flow(int s, int t, Cap flow_limit) {
        assert(0 <= s && s < _n);
        assert(0 <= t && t < _n);
        assert(s != t);

        std::vector<int> level(_n), iter(_n);
        internal::simple_queue<int> que;

        auto bfs = [&]() {
            std::fill(level.begin(), level.end(), -1);
            level[s] = 0;
            que.clear();
            que.push(s);
            while (!que.empty()) {
                int v = que.front();
                que.pop();
                for (auto e : g[v]) {
                    if (e.cap == 0 || level[e.to] >= 0) continue;
                    level[e.to] = level[v] + 1;
                    if (e.to == t) return;
                    que.push(e.to);
                }
            }
```

```cpp
        };
        auto dfs = [&](auto self, int v, Cap up) {
            if (v == s) return up;
            Cap res = 0;
            int level_v = level[v];
            for (int& i = iter[v]; i < int(g[v].size()); i++) {
                _edge& e = g[v][i];
                if (level_v <= level[e.to] || g[e.to][e.rev].cap == 0) continue;
                Cap d =
                    self(self, e.to, std::min(up - res, g[e.to][e.rev].cap));
                if (d <= 0) continue;
                g[v][i].cap += d;
                g[e.to][e.rev].cap -= d;
                res += d;
                if (res == up) return res;
            }
            level[v] = _n;
            return res;
        };

        Cap flow = 0;
        while (flow < flow_limit) {
            bfs();
            if (level[t] == -1) break;
            std::fill(iter.begin(), iter.end(), 0);
            Cap f = dfs(dfs, t, flow_limit - flow);
            if (!f) break;
            flow += f;
        }
        return flow;
    }

    std::vector<bool> min_cut(int s) {
        std::vector<bool> visited(_n);
        internal::simple_queue<int> que;
        que.push(s);
        while (!que.empty()) {
            int p = que.front();
            que.pop();
            visited[p] = true;
            for (auto e : g[p]) {
                if (e.cap && !visited[e.to]) {
                    visited[e.to] = true;
                    que.push(e.to);
                }
            }
        }
        return visited;
    }

  private:
    int _n;
    struct _edge {
        int to, rev;
        Cap cap;
    };
    std::vector<std::pair<int, int>> pos;
    std::vector<std::vector<_edge>> g;
};

}  // namespace atcoder
```

```
using namespace atcoder;

void solve(){
    int n,m,s,t;cin>>n>>m>>s>>t;
    mf_graph<ll> g(n+5);
    for(int i=1;i<=m;i++){
        int u,v,cap;cin>>u>>v>>cap;
        g.add_edge(u,v,cap);
    }
    ll cost=g.flow(s,t);
    cout<<cost<<"\n";
}
```

最小费用最大流

dinic算法

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=5005,M=50005;
const ll INF = 1e18;

struct MCMF {
    int n, s, t;
    ll maxFlow, minCost;
    int head[N], tot = 1;
    int last[N], pre[N], inq[N];
    ll flow[N], dis[N];
    struct Edge {
        int to;
        ll flow;
        ll cost;
        int nxt;
    } edge[M<<1];
    void add_edge(int u, int v, ll f, ll c) {
        edge[++tot].to = v;
        edge[tot].flow = f;
        edge[tot].cost = c;
        edge[tot].nxt = head[u];
        head[u] = tot;
    }
    void init(int _n, int _s, int _t) { //点数，边数，源点，汇点
        n = _n, s = _s, t = _t;
        maxFlow = minCost = 0LL;
    }
    bool spfa(int s, int t) {
        for (int i = 1; i <= n; i++)  //以cost为边权寻找最短增广路
            flow[i] = dis[i] = INF;
        queue<int> q;
        q.push(s);
        dis[s] = 0, inq[s] = 1, pre[t] = -1;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            inq[u] = 0;
            for (int i = head[u]; i; i = edge[i].nxt) {
                int v = edge[i].to;
                if (edge[i].flow > 0 && dis[v] > dis[u] + edge[i].cost) {  //成功找到增广路
                    dis[v] = dis[u] + edge[i].cost;
                    flow[v] = min(flow[u], edge[i].flow);
                    pre[v] = u;   //记录该点的上一个点，方便回溯。
```

```
                    last[v] = i;  //记录该点的上一条边，方便回溯
                    if (!inq[v]) {
                        inq[v] = 1;
                        q.push(v);
                    }
                }
            }
        }
        return pre[t] != -1;
    }
    void dinic() {
        while (spfa(s, t)) {
            int now = t;
            maxFlow += flow[t];           //最大流增加
            minCost += flow[t] * dis[t];  //最小费用 = 流量 * 单位时间最小的流量费用
            while (now != s) {            //回溯，构建残量网络
                edge[last[now]].flow -= flow[t];
                edge[last[now] ^ 1].flow += flow[t];
                now = pre[now];
            }
        }
    }
} fff;

int main() {
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    fff.init(n, s, t);
    for (int i = 1; i <= m; i++) {
        int u, v, f, c;
        cin >> u >> v >> f >> c;
        fff.add_edge(u, v, f, c);
        fff.add_edge(v, u, 0, -c);
    }
    fff.dinic();
    cout << fff.maxFlow << " " << fff.minCost << endl;
    return 0;
}
```

atcoder费用流

```
namespace atcoder {

template <class Cap, class Cost> struct mcf_graph {
  public:
    mcf_graph() {}
    mcf_graph(int n) : _n(n), g(n) {}

    int add_edge(int from, int to, Cap cap, Cost cost) {
        assert(0 <= from && from < _n);
        assert(0 <= to && to < _n);
        int m = int(pos.size());
        pos.push_back({from, int(g[from].size())});
        int from_id = int(g[from].size());
        int to_id = int(g[to].size());
        if (from == to) to_id++;
        g[from].push_back(_edge{to, to_id, cap, cost});
        g[to].push_back(_edge{from, from_id, 0, -cost});
        return m;
    }

    struct edge {
```

```
        int from, to;
        Cap cap, flow;
        Cost cost;
    };

    edge get_edge(int i) {
        int m = int(pos.size());
        assert(0 <= i && i < m);
        auto _e = g[pos[i].first][pos[i].second];
        auto _re = g[_e.to][_e.rev];
        return edge{
            pos[i].first, _e.to, _e.cap + _re.cap, _re.cap, _e.cost,
        };
    }
    std::vector<edge> edges() {
        int m = int(pos.size());
        std::vector<edge> result(m);
        for (int i = 0; i < m; i++) {
            result[i] = get_edge(i);
        }
        return result;
    }

    std::pair<Cap, Cost> flow(int s, int t) {
        return flow(s, t, std::numeric_limits<Cap>::max());
    }
    std::pair<Cap, Cost> flow(int s, int t, Cap flow_limit) {
        return slope(s, t, flow_limit).back();
    }
    std::vector<std::pair<Cap, Cost>> slope(int s, int t) {
        return slope(s, t, std::numeric_limits<Cap>::max());
    }
    std::vector<std::pair<Cap, Cost>> slope(int s, int t, Cap flow_limit) {
        assert(0 <= s && s < _n);
        assert(0 <= t && t < _n);
        assert(s != t);
        // variants (C = maxcost):
        // -(n-1)C <= dual[s] <= dual[i] <= dual[t] = 0
        // reduced cost (= e.cost + dual[e.from] - dual[e.to]) >= 0 for all edge
        std::vector<Cost> dual(_n, 0), dist(_n);
        std::vector<int> pv(_n), pe(_n);
        std::vector<bool> vis(_n);
        auto dual_ref = [&]() {
            std::fill(dist.begin(), dist.end(),
                      std::numeric_limits<Cost>::max());
            std::fill(pv.begin(), pv.end(), -1);
            std::fill(pe.begin(), pe.end(), -1);
            std::fill(vis.begin(), vis.end(), false);
            struct Q {
                Cost key;
                int to;
                bool operator<(Q r) const { return key > r.key; }
            };
            std::priority_queue<Q> que;
            dist[s] = 0;
            que.push(Q{0, s});
            while (!que.empty()) {
                int v = que.top().to;
                que.pop();
                if (vis[v]) continue;
                vis[v] = true;
                if (v == t) break;
```

```
                // dist[v] = shortest(s, v) + dual[s] - dual[v]
                // dist[v] >= 0 (all reduced cost are positive)
                // dist[v] <= (n-1)C
                for (int i = 0; i < int(g[v].size()); i++) {
                    auto e = g[v][i];
                    if (vis[e.to] || !e.cap) continue;
                    // |-dual[e.to] + dual[v]| <= (n-1)C
                    // cost <= C - -(n-1)C + 0 = nC
                    Cost cost = e.cost - dual[e.to] + dual[v];
                    if (dist[e.to] - dist[v] > cost) {
                        dist[e.to] = dist[v] + cost;
                        pv[e.to] = v;
                        pe[e.to] = i;
                        que.push(Q{dist[e.to], e.to});
                    }
                }
            }
            if (!vis[t]) {
                return false;
            }

            for (int v = 0; v < _n; v++) {
                if (!vis[v]) continue;
                // dual[v] = dual[v] - dist[t] + dist[v]
                //         = dual[v] - (shortest(s, t) + dual[s] - dual[t]) +
(shortest(s, v) + dual[s] - dual[v])
                //         = - shortest(s, t) + dual[t] + shortest(s, v)
                //         = shortest(s, v) - shortest(s, t) >= 0 - (n-1)C
                dual[v] -= dist[t] - dist[v];
            }
            return true;
        };
        Cap flow = 0;
        Cost cost = 0, prev_cost_per_flow = -1;
        std::vector<std::pair<Cap, Cost>> result;
        result.push_back({flow, cost});
        while (flow < flow_limit) {
            if (!dual_ref()) break;
            Cap c = flow_limit - flow;
            for (int v = t; v != s; v = pv[v]) {
                c = std::min(c, g[pv[v]][pe[v]].cap);
            }
            for (int v = t; v != s; v = pv[v]) {
                auto& e = g[pv[v]][pe[v]];
                e.cap -= c;
                g[v][e.rev].cap += c;
            }
            Cost d = -dual[s];
            flow += c;
            cost += c * d;
            if (prev_cost_per_flow == d) {
                result.pop_back();
            }
            result.push_back({flow, cost});
            prev_cost_per_flow = d;
        }
        return result;
    }

  private:
    int _n;
```

```
    struct _edge {
        int to, rev;
        Cap cap;
        Cost cost;
    };

    std::vector<std::pair<int, int>> pos;
    std::vector<std::vector<_edge>> g;
};

}  // namespace atcoder

using namespace atcoder;


void solve(){
    int n,m,s,t;cin>>n>>m>>s>>t;
    mcf_graph<ll,ll> g(n+5);
    for(int i=1;i<=m;i++){
        int u,v;ll cap,w;cin>>u>>v>>cap>>w;
        g.add_edge(u,v,cap,w);
    }
    auto [cap,cost]=g.flow(s,t);
    cout<<cap<<" "<<cost<<"\n";
}
```

无源汇上下界可行流

```
ll low[M],up[M];

struct NF{
    struct edge{
        int to;
        ll w;
        int nxt;
    }edge[M<<1];
    int head[N],tot=1;
    int depth[N],cur[N];
    int n,m,s,t;
    void init(int _n,int _m,int _s,int _t){ //初始化
        n=_n,m=_m,s=_s,t=_t;
    }
    void add(int u,int v,ll w){ //加边
        edge[++tot].w=w;
        edge[tot].to=v;
        edge[tot].nxt=head[u];
        head[u]=tot;
    }
    bool bfs(){  //分层
        for(int i=1;i<=n;i++) depth[i]=0;
        depth[s]=1;
        queue<int> q;
        q.push(s);
        while(!q.empty()){
            int u=q.front();q.pop();
            if(u==t) return 1;
            for (int i=head[u];i;i=edge[i].nxt) {
                int v=edge[i].to;
                if(edge[i].w>0&&!depth[v]){
                    depth[v]=depth[u]+1;
                    q.push(v);
                }
```

```
            }
        }
        return 0;
    }
    ll dfs(int u,ll dis){ //dis:该条路径当前最大允许流量
        if(u==t) return dis; //到达汇点后，成功流出流量即是该条路径最大允许流量
        ll sum=0;
        for(int i=cur[u];i&&dis;i=edge[i].nxt){ //多方向增广
            cur[u]=i; //当前弧优化
            int v=edge[i].to;
            if(depth[v]==depth[u]+1&&edge[i].w>0){ //可以往下增广
                ll d=dfs(v,min(dis,edge[i].w));
                if(!d) depth[v]=0;  //增广失败，删除该点
                edge[i].w-=d;    //增广成功
                edge[i^1].w+=d;
                sum+=d;   //成功流出的流量增加
                dis-=d;  //最大允许流量减少
            }
        }
        return sum;  //成功流出的流量总和
    }
    ll dinic(){
        ll ans=0;
        while(bfs()){
            for(int i=1;i<=n;i++) cur[i]=head[i];
            ans+=dfs(s,INF);
        }
        return ans;
    }
    void solve(){
        for(int i=2;i<=m*2;i+=2){
            cout<<edge[i^1].w+low[i>>1]<<"\n";
        }
    }
}fff;

ll in[N],out[N];
ll sum;

void solve(){
    int n,m,s,t;cin>>n>>m;
    s=n+1,t=n+2;
    fff.init(t,m,s,t);
    for(int i=1;i<=m;i++){
        int u,v;cin>>u>>v>>low[i]>>up[i];
        fff.add(u,v,up[i]-low[i]);
        fff.add(v,u,0);
        out[u]+=low[i];in[v]+=low[i]; //因为刚开始每条边流量设为low
    }
    for(int i=1;i<=n;i++){
        //初始流in[i]>=out[i] 附加流(反边)流入<流出
        if(in[i]>=out[i]){
            fff.add(s,i,in[i]-out[i]);
            fff.add(i,s,0);
            sum+=in[i]-out[i];
        }
        else{
            fff.add(i,t,out[i]-in[i]);
            fff.add(t,i,0);
        }
    }
    bool f=(fff.dinic()==sum);
```

```
    cout<<(f?"YES":"NO")<<"\n";
    if(f) fff.solve();
}
```

有源汇上下界最大流

```
ll low[M],up[M];

struct NF{
    struct edge{
        int to;
        ll w;
        int nxt;
    }edge[M<<1];
    int head[N],tot=1;
    int depth[N],cur[N];
    int n,m,s,t;
    void init(int _n,int _m,int _s,int _t){ //初始化
        n=_n,m=_m,s=_s,t=_t;
    }
    void add(int u,int v,ll w){ //加边
        edge[++tot].w=w;
        edge[tot].to=v;
        edge[tot].nxt=head[u];
        head[u]=tot;
    }
    bool bfs(){  //分层
        for(int i=1;i<=n;i++) depth[i]=0;
        depth[s]=1;
        queue<int> q;
        q.push(s);
        while(!q.empty()){
            int u=q.front();q.pop();
            if(u==t) return 1;
            for (int i=head[u];i;i=edge[i].nxt) {
                int v=edge[i].to;
                if(edge[i].w>0&&!depth[v]){
                    depth[v]=depth[u]+1;
                    q.push(v);
                }
            }
        }
        return 0;
    }
    ll dfs(int u,ll dis){ //dis:该条路径当前最大允许流量
        if(u==t) return dis; //到达汇点后，成功流出流量即是该条路径最大允许流量
        ll sum=0;
        for(int i=cur[u];i&&dis;i=edge[i].nxt){ //多方向增广
            cur[u]=i; //当前弧优化
            int v=edge[i].to;
            if(depth[v]==depth[u]+1&&edge[i].w>0){ //可以往下增广
                ll d=dfs(v,min(dis,edge[i].w));
                if(!d) depth[v]=0;  //增广失败，删除该点
                edge[i].w-=d;    //增广成功
                edge[i^1].w+=d;
                sum+=d;   //成功流出的流量增加
                dis-=d;  //最大允许流量减少
            }
        }
        return sum;  //成功流出的流量总和
    }
    ll dinic(){
```

```
        ll ans=0;
        while(bfs()){
            for(int i=1;i<=n;i++) cur[i]=head[i];
            ans+=dfs(s,INF);
        }
        return ans;
    }
}fff;

ll in[N],out[N];
ll sum;

void solve(){
    int n,m,s,t;cin>>n>>m>>s>>t;
    int S=n+1,T=n+2; //虚拟的源汇点
    fff.init(T,m,S,T);

    for(int i=1;i<=m;i++){
        int u,v;cin>>u>>v>>low[i]>>up[i];
        fff.add(u,v,up[i]-low[i]);
        fff.add(v,u,0);
        out[u]+=low[i];in[v]+=low[i]; //因为刚开始每条边流量设为low
    }
    for(int i=1;i<=n;i++){
        //初始流in[i]>=out[i] 附加流(反边)流入<流出
        if(in[i]>=out[i]){
            fff.add(S,i,in[i]-out[i]);
            fff.add(i,S,0);
            sum+=in[i]-out[i];
        }
        else{
            fff.add(i,T,out[i]-in[i]);
            fff.add(T,i,0);
        }
    }
    fff.add(t,s,INF);
    fff.add(s,t,0);
    if(fff.dinic()!=sum){
        cout<<"No Solution\n";
        return;
    }

    ll ans=fff.edge[fff.tot].w;
    fff.edge[fff.tot].w=fff.edge[fff.tot^1].w=0;
    fff.tot-=2; //删去(t->s)和(s->t)的连边
    fff.init(T,m,s,t);
    ans+=fff.dinic();
    cout<<ans<<"\n";
}
```

字符串

Trie树(字典树)

https://www.acwing.com/problem/content/837/

```
#include <bits/stdc++.h>
using namespace std;
```

```
struct Trie {
    int nex[1000010][26], cnt;
    int exist[1000010];  // 该结点结尾的字符串是否存在

    void insert(string s, int l) {  // 插入字符串
        int p = 0;
        for (int i = 0; i < l; i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) nex[p][c] = ++cnt;  // 如果没有，就添加结点
            p = nex[p][c];
        }
        exist[p] ++;
    }
    int find(string s, int l) {  // 查找字符串
        int p = 0;
        for (int i = 0; i < l; i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) return 0;
            p = nex[p][c];
        }
        return exist[p];
    }
} t;

int main() {
    ios::sync_with_stdio(0);cin.tie(0);
    int n;cin>>n;
    char c;
    string s;
    while(n--) {
        cin>>c>>s;
        if(c=='I') t.insert(s,s.length());
        else cout<<t.find(s,s.length())<<"\n";
    }
    return 0;
}
```

01Trie

```
struct Trie {
    int nex[N*32][2],cnt=0;

    void insert(int x) {
        int p = 0;
        for (int i = 30; i >= 0; i--) {
            int c = x>>i&1;
            if (!nex[p][c]) nex[p][c] = ++cnt;
            p = nex[p][c];
        }
    }
    int find(int x) {
        int p = 0,res = 0;
        for (int i = 30; i >= 0; i--) {
            int c = x>>i&1;
            if(nex[p][c^1]) p=nex[p][c^1],res|=(1<<i);
            else p=nex[p][c];
        }
        return res;
    }
} t;
```

字符串哈希

```
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;
const ull P = 131;
const ull N = 1e5+10;
ull powP[N],H[N];// H[i]前i个字符的hash值
int n,m;
string str;

void init(){
    powP[0]=1;
    for(int i=1;i<=n;i++){
        powP[i]=powP[i-1]*P;
    }
}

void calH(ull H[],string str){
    H[0]=0;
    for(int i=1;i<=n;i++){
        H[i]=H[i-1]*P+str[i];
    }
}

ull get(int l,int r){
    return H[r]-H[l-1]*powP[r-l+1];
}

int main(){
    cin>>n>>m;
    cin>>str;str=" "+str;
    init();
    calH(H,str);
    while(m--){
        int l1,l2,r1,r2;
        cin>>l1>>r1>>l2>>r2;
        if(get(l1,r1)==get(l2,r2)) puts("Yes");
        else puts("No");
    }
    return 0;
}
```

KMP算法

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e6+10;

int ne[N];

void getNext(string s,int len){
    int j=-1;
    ne[0]=-1;
    for(int i=1;i<len;i++){
        while(j!=-1&&s[i]!=s[j+1]){
            j=ne[j];
        }
        if(s[i]==s[j+1]) j++;
        ne[i]=j;
    }
```

```
}

int KMP(string text,string pattern){
    int n=text.length(),m=pattern.length();
    getNext(pattern,m);
    int ans=0,j=-1;
    for(int i=0;i<n;i++){
        while(j!=-1&&text[i]!=pattern[j+1]){
            j=ne[j];
        }
        if(text[i]==pattern[j+1]) j++;
        if(j==m-1) ans++,j=ne[j],printf("%d ", i-m+1);
    }
    return ans;
}

int main(){
    int n,m;
    string s1,s2;
    cin>>n>>s1>>m>>s2;
    KMP(s2,s1);
    return 0;
}
```

Manacher(马拉车)算法

最长回文子串

```
#include <bits/stdc++.h>
using namespace std;

int Manacher(string s){
    string str="@#";
    for(int i=0;i<s.length();i++) str+=s[i],str+='#';
    vector<int> p(str.length(),0);
    /*
    mx表示某个回文串延伸在最右端半径的下标
    id表示这个回文子串最中间位置下标
    reslen表示对应在s中的最大子回文串的半径
    rescenter表示最大子回文串的中间位置
    */
    int mx=0,id=0,reslen=0,rescenter=0;
    for(int i=1;i<str.length();i++){
        p[i] = mx>i ? min(p[2*id-i],mx-i):1;
        while(str[i+p[i]]==str[i-p[i]]) p[i]++;
        if(mx<i+p[i]) mx=i+p[i],id=i;
        if(reslen<p[i]) reslen=p[i],rescenter=i;
    }
    return reslen-1;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    string s;getline(cin,s);
    cout<<Manacher(s)<<"\n";
    return 0;
}
```

AC 自动机

```
#include <bits/stdc++.h>
using namespace std;
const int N = 5e5 + 5;

namespace AC {
    int nex[N][26], cnt;
    int exist[N], fail[N];
    void init() {
        memset(fail, 0, sizeof(fail));
        memset(nex, 0, sizeof(nex));
        memset(exist, 0, sizeof(exist));
        cnt = 0;
    }
    void insert(string s,int len) {
        int p = 0;
        for (int i = 0; i < len; i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) nex[p][c] = ++cnt;  // 如果没有，就添加结点
            p = nex[p][c];
        }
        exist[p] ++;
    }
    queue<int> q;
    void build() {
        for (int i = 0; i < 26; i++)
            if (nex[0][i]) fail[nex[0][i]]=0,q.push(nex[0][i]);
        while (!q.empty()) {
            int u = q.front();q.pop();
            for (int i = 0; i < 26; i++) {
                if (nex[u][i])
                    fail[nex[u][i]] = nex[fail[u]][i], q.push(nex[u][i]);
                else
                    nex[u][i] = nex[fail[u]][i];
            }
        }
    }
    int query(string t,int len) {
        int p = 0, res = 0;
        for (int i = 0; i < len; i++) {
            int c = t[i] - 'a';
            p = nex[p][c];  // 转移
            for (int j = p; j && exist[j] != -1; j = fail[j]) {
                res += exist[j], exist[j] = -1;
            }
        }
        return res;
    }
}  // namespace AC

string s,t;
int main() {
    int cas;cin>>cas;
    while(cas--){
        AC::init();
        int n;cin>>n;
        for (int i = 1; i <= n; i++) cin>>s, AC::insert(s,s.length());
        cin>>t;
        AC::build();
        cout<<AC::query(t,t.length())<<"\n";
    }
```

```
    return 0;
}
```

后缀数组(SA)

$O(nlog^2n)$

```
int n,k;
string s;
int rk[N<<1],oldrk[N<<1],sa[N];

bool compare_sa(int i,int j){
    if(rk[i]!=rk[j]) return rk[i]<rk[j];
    int ri=i+k<=n?rk[i+k]:-1;
    int rj=j+k<=n?rk[j+k]:-1;
    return ri<rj;
}

void construct_sa(){
    for(int i=0;i<=n;i++){
        sa[i]=i;
        rk[i]=i<n?s[i]:-1;
    }

    //k->2k
    for(k=1;k<=n;k*=2){
        sort(sa,sa+n+1,compare_sa);
        oldrk[sa[0]]=0;
        for(int i=1;i<=n;i++){
            oldrk[sa[i]]=oldrk[sa[i-1]]+(compare_sa(sa[i-1],sa[i])?1:0);
        }
        for(int i=0;i<=n;i++){
            rk[i]=oldrk[i];
        }
    }
}

int lcp[N];//lcp[i]:s[sa[i]...]与s[sa[i+1]...]的最长公共前缀的长度
void construct_lcp(){
    for(int i=0;i<=n;i++) rk[sa[i]]=i;
    int h=0;
    lcp[0]=0;
    for(int i=0;i<n;i++){
        int j=sa[rk[i]-1];
        if(h>0) h--;
        for(;j+h<n&&i+h<n;h++){
            if(s[j+h]!=s[i+h]) break;
        }
        lcp[rk[i]-1]=h;
    }
}

void solve(){
    cin>>s;
    n=s.length();
    construct_sa();
    construct_lcp();
    rep(i,0,n){
        if(rk[sa[i]]) cout<<sa[i]+1<<" \n"[i==n];
    }
    rep(i,0,n-1){
        cout<<lcp[i]<<" \n"[i==n-1];
```

```
    }
}
```

$O(nlogn)$

```
int rk[N<<1],oldrk[N<<1],sa[N];
int id[N], px[N], cnt[N];
// px[i] = rk[id[i]]（用于排序的数组所以叫 px）

bool cmp(int x,int y,int w){
    return oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w];
}

void get_SA(string s,int n){
    int p,m = max(n,300);
    for (int i=1;i<=n;i++) cnt[rk[i]=s[i]]++;   // 计数排序
    for (int i=1;i<=m;i++) cnt[i] += cnt[i-1];
    for (int i=n;i>=1;i--) sa[cnt[rk[i]]--] = i;

    //倍增
    for(int w=1;;w<<=1,m=p){// m=p 就是优化计数排序值域
        p=0;
        for(int i=n;i>n-w;i--) id[++p] = i; //先按第二关键字排序
        for(int i=1;i<=n;i++) if(sa[i]>w) id[++p]=sa[i]-w;
        assert(p==n);
        for(int i=1;i<=m;i++) cnt[i]=0; //再按照第一关键字基数排序
        for(int i=1;i<=n;i++) cnt[px[i]=rk[id[i]]]++; //px[i]为关键字
        for(int i=1;i<=m;i++) cnt[i] += cnt[i-1];
        for(int i=n;i>=1;i--) sa[cnt[px[i]]--]=id[i];
        for(int i=1;i<=n;i++) oldrk[i]=rk[i];
        p=0;
        for(int i=1;i<=n;i++) // 获得 rk 数组
            rk[sa[i]]=cmp(sa[i],sa[i-1],w)?p:++p;
        if(p==n){// 每个rk不相同，后缀已经有序
            for(int i=1;i<=n;i++) sa[rk[i]]=i;
            break;
        }
    }
}

int lcp[N];//lcp[i]:s[sa[i]...]与s[sa[i+1]...]的最长公共前缀的长度
void get_lcp(string s,int n){
    for(int i=1;i<=n;i++) rk[sa[i]]=i;
    for(int i=1,k=0;i<=n;i++){
        if(rk[i]==1) continue;
        if(k) k--;
        int j=sa[rk[i]-1];
        while(i+k<=n and j+k<=n and s[i+k]==s[j+k]) k++;
        lcp[rk[i]]=k;
    }
}

void solve(){
    string s;cin>>s;
    int n=s.length();
    s=" "+s;
    get_SA(s,n);
    get_lcp(s,n);
    rep(i,1,n){
        cout<<sa[i]<<" \n"[i==n];
    }
    rep(i,1,n){
```

```
        cout<<lcp[i]<<" \n"[i==n];
    }
}
```

动态规划

背包DP

01背包

```
int v,w; //v体积,w价值
int f[N];

void solve(){
    int n,m;cin>>n>>m;
    for(int i=1;i<=n;i++){
        scanf("%d%d",&v,&w);
        for(int j=m;j>=v;j--){
            f[j]=max(f[j], f[j-v]+w);
        }
    }
    cout<<f[m]<<"\n";
}
```

完全背包

```
int v,w; //v体积,w价值
int f[N];

void solve(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        scanf("%d%d",&v,&w);
        for(int j=v;j<=m;j++){
            f[j]=max(f[j],f[j-v]+w);
        }
    }
    cout<<f[m]<<"\n";
}
```

多重背包

朴素

$O(nV\sum_{i=1}^{n} s_i)$

```
int v,w,s; //v体积 w价值 s数量
int f[N];

void solve(){
    int n,m;cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>v>>w>>s;
        for(int j=m;j>=v;j--){
            for(int k=1;k<=s&&k*v<=j;k++){
                f[j]=max(f[j],f[j-k*v]+w*k);
            }
```

```
        }
    }
    cout<<f[m]<<"\n";
}
```

二进制优化

$O(nV\sum_{i=1}^{n}log_2s_i)$

```
int v,w,s;
int dp[N];
vector<pii> good;

void solve() {
    int n,V;
    cin>>n>>V;
    for(int i=1;i<=n;i++){
        cin>>v>>w>>s;
        for(int k=1;k<=s;k<<=1){
            s-=k;
            good.push_back({v*k,w*k});
        }
        if(s>0) good.push_back({v*s,w*s});
    }
    for(auto t:good) {
        for(int j=V; j>=t.X; j--) {
            dp[j]=max(dp[j],dp[j-t.X]+t.Y);
        }
    }
    cout<<dp[V]<<"\n";
}
```

单调队列优化

$O(nV)$

```
int v[N],w[N],s[N];//体积、价值和数量
int f[N],g[N];//g[]为f[i-1][],f[]为f[i][]

void solve(){
    int n,V;cin>>n>>V;
    for(int i=1;i<=n;i++) cin>>v[i]>>w[i]>>s[i];
    for(int i=1;i<=n;i++) {
        memcpy(g,f,sizeof f);
        for(int j=0;j<v[i];j++){ //枚举余数
            deque<int> q;
            for (int k=j;k<=V;k+=v[i]){
                f[k]=g[k];
                if(!q.empty()&&k-s[i]*v[i]>q.front()){
                    q.pop_front();
                }
                if(!q.empty()){
                    f[k]=max(f[k],g[q.front()]+(k-q.front())/v[i]*w[i]);
                }
                while(!q.empty()&&g[q.back()]-(q.back()-j)/v[i]*w[i]<=g[k]-(k-
j)/v[i]*w[i]){
                    q.pop_back();
                }
                q.push_back(k);
            }
        }
    }
```

```
    cout<<f[V]<<"\n";
}
```

混合背包

- $si = -1$ 表示第 i 种物品只能用1次;
- $si = 0$ 表示第 i 种物品可以用无限次;
- $si > 0$ 表示第 i 种物品可以使用 si 次;

```
int v[N],w[N],s[N],dp[N];
struct good{
    int kind;
    int v,w;
};
vector<good> g;

void solve() {
    int n,V;
    cin>>n>>V;
    for(int i=1;i<=n;i++) cin>>v[i]>>w[i]>>s[i];
    for(int i=1;i<=n;i++){
        if(s[i]==-1||s[i]==0) g.push_back({s[i],v[i],w[i]});
        else{
            for(int k=1;k<=s[i];k<<=1){
                s[i]-=k;
                g.push_back({-1,v[i]*k,w[i]*k});
            }
            if(s[i]>0) g.push_back({-1,v[i]*s[i],w[i]*s[i]});
        }
    }
    for(auto t:g){
        if(t.kind==-1){
            for(int j=V;j>=t.v;j--){
                dp[j]=max(dp[j],dp[j-t.v]+t.w);
            }
        }
        else{
            for(int j=t.v;j<=V;j++){
                dp[j]=max(dp[j],dp[j-t.v]+t.w);
            }
        }
    }
    cout<<dp[V]<<"\n";
}
```

二维费用的背包

```
int dp[N][N];

void solve(){
    int n,V,M;cin>>n>>V>>M;
    for(int i=1;i<=n;i++){
        int v,m,w;cin>>v>>m>>w;
        for(int j=V;j>=v;j--){
            for(int k=M;k>=m;k--){
                dp[j][k]=max(dp[j][k],dp[j-v][k-m]+w);
            }
        }
    }
    cout<<dp[V][M]<<"\n";
}
```

分组背包

```
int dp[N],v[N],w[N];
void solve(){
    int n,V;cin>>n>>V;
    for(int i=1;i<=n;i++){  //循环每一组
        int s;cin>>s;
        for(int j=1;j<=s;j++) cin>>v[j]>>w[j];
        for(int j=V;j>=0;j--){  //循环背包容量
            for(int k=1;k<=s;k++){  //循环该组的每一个物品
                if(j>=v[k]) dp[j]=max(dp[j],dp[j-v[k]]+w[k]);
            }
        }
    }
    cout<<dp[V]<<"\n";
}
```

有依赖的背包

```
int n,m,root;
int v[N],w[N],dp[N][N];
vector<int> g[N];

void dfs(int u){
    for(int i=v[u];i<=m;i++) dp[u][i]=w[u]; //x必须选
    for(int x:g[u]){
        dfs(x);
        for(int j=m;j>=v[u];j--){
            for(int k=0;k<=j-v[u];k++){
                dp[u][j]=max(dp[u][j],dp[u][j-k]+dp[x][k]);
            }
        }
    }
}

void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        int fa;
        cin>>v[i]>>w[i]>>fa;
        if(~fa) g[fa].push_back(i);
        else root=i;
    }
    dfs(root);
    cout<<dp[root][m]<<"\n";
}
```

背包问题求最大价值的方案数

```
int f[N];
ll cnt[N];

void solve(){
    int n,m;cin>>n>>m;
    for(int i=0;i<=m;i++) cnt[i]=1;
    for(int i=1;i<=n;i++){
        int v,w;cin>>v>>w;
        for(int j=m;j>=v;j--){
            int value=f[j-v]+w;
```

```
            if(value>f[j]){
                f[j]=value;
                cnt[j]=cnt[j-v];
            }
            else if(value==f[j]){
                cnt[j]+=cnt[j-v];
                cnt[j]%=mod;
            }
        }
    }
    cout<<cnt[m]<<"\n";
}
```

背包问题求最大价值字典序最小的具体方案

```
int n,m;
int v[N],w[N],f[N][N];

void print(int i,int j){
    if(i==n+1) return;
    if(j>=v[i]&&f[i][j]==f[i+1][j-v[i]]+w[i]) {
        cout<<i<<" ";
        print(i+1,j-v[i]);
    }
    else print(i+1,j);
}

void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>v[i]>>w[i];
    for(int i=n;i>=1;i--){
        for(int j=0;j<=m;j++){
            if(j>=v[i]) f[i][j]=max(f[i+1][j],f[i+1][j-v[i]]+w[i]);
            else f[i][j]=f[i+1][j];
        }
    }
    print(1,m);
}
```

区间DP

模板

```
for(int i=1;i<=n;i++){
    dp[i][i] = 初始值
}
for(int len=2;len<=n;len++){ //区间长度
    for (int l=1;l+len-1<=n;l++) { //枚举起点   环形(len-1<=n*2)
        int r=l+len-1; //区间终点
        for(int k=l;k<r;k++){ //枚举分割点，构造状态转移方程
            dp[l][r]=max(dp[l][r],dp[l][k]+dp[k+1][r]+w[l][r]);
        }
    }
}
```

石子合并

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N=1005,INF=0x3f3f3f3f;
int s[N];  //前缀和
int dp[N][N];  //dp[l][r]在[l,r]区间合并的代价
int w[N][N]; //[l,r]区间的决策点

int main(){
    int n;cin>>n;
    for(int i=1;i<=n;i++){
        cin>>s[i];
        s[i]+=s[i-1];
        w[i][i]=i;
    }
    for(int len=2;len<=n;len++){
        for(int l=1;l+len-1<=n;l++){
            int r=l+len-1;
            dp[l][r]=INF;
            for(int k=w[l][r-1];k<=w[l+1][r];k++){
                if(dp[l][r]>dp[l][k]+dp[k+1][r]+s[r]-s[l-1]){
                    dp[l][r]=dp[l][k]+dp[k+1][r]+s[r]-s[l-1];
                    w[l][r]=k;
                }
            }
        }
    }
    cout<<dp[1][n]<<endl;
    return 0;
}
```

数位DP

```cpp
//区间不包含所有含有 4 或 62 的号码

int a[10];
ll f[10][3];
//pos是我们处理到a数组第pos位
//st=0没有4和62  st=1有6  st=2有4和62
//flag表示前pos-1位和原数是否相同 也就是有没有卡着上界
ll dp(int pos,int st,bool flag){
    if(!pos) return st==2;
    if(!flag && f[pos][st]!=-1) return f[pos][st];
    int x=flag?a[pos]:9;
    ll ans=0;
    for(int i=0;i<=x;i++){
        if(i==4||st==2||(st==1&&i==2)) ans+=dp(pos-1,2,flag&&i==x);
        else if(i==6) ans+=dp(pos-1,1,flag&&i==x);
        else ans+=dp(pos-1,0,flag&&i==x);
    }
    if(!flag) f[pos][st]=ans;
    return ans;
}

ll calc(ll x){
    int pos=0;
    while(x){
        a[++pos]=x%10;
        x/=10;
```

```
    }
    return dp(pos,0,1);
}

void solve(){
    ll n,m;mst(f,-1);
    while(cin>>n>>m,n+m){
        cout<<(m-n+1)-(calc(m)-calc(n-1))<<"\n";
    }
}
```

换根DP

给定一个 n 个点的树，请求出一个结点，使得以这个结点为根时，所有结点的深度之和最大。

一个结点的深度之定义为该节点到根的简单路径上边的数量。

```
int n;
vector<int> g[N];
ll siz[N],depth[N];
ll f[N];//以i为根的树的深度和

void dfs1(int u,int fa){
    siz[u]=1;
    for(auto v:g[u]){
        if(v==fa) continue;
        depth[v]=depth[u]+1;
        dfs1(v,u);
        siz[u]+=siz[v];
    }
}

void dfs2(int u,int fa){
    for(auto v:g[u]){
        if(v==fa) continue;
        /*
        本来是以u为根的树，变成以儿子v为根的树，
        那么v的所有结点的深度都会减1，深度和就会减少siz[v]，
        同样地，所有不在v的子树上的结点的深度都会+1，深度和就会加上n - size[v]，
        */
        f[v]=f[u]+n-2*siz[v];
        dfs2(v,u);
    }
}

void solve(){
    cin>>n;
    for(int i=1;i<n;i++){
        int u,v;cin>>u>>v;
        g[u].push_back(v);g[v].push_back(u);
    }

    //首先以1为根dfs一遍求出以每个点为根的子树的结点数
    dfs1(1,0);
    for(int i=1;i<=n;i++) f[1]+=depth[i];

    dfs2(1,0);
    ll ans=0;
    int id=0;
    for(int i=1;i<=n;i++){
        if(ans<f[i]) ans=f[i],id=i;
    }
```

```
    cout<<id<<"\n";
}
```

单调队列优化DP

```
ll f[N];//表示不选第i个的最小不选效率
ll ans;
int q[N];

//每k+1个里必须不选1个，使不选的总效率最小
void solve(){
    int n,k;cin>>n>>k;
    rep(i,1,n) cin>>f[i],ans+=f[i];
    int hh=1,tt=0;
    for(int i=0;i<=n;i++){
        while(hh<=tt && i-q[hh]>k+1) hh++;
        f[i]+=f[q[hh]];
        while(hh<=tt && f[i]<=f[q[tt]]) tt--;
        q[++tt]=i;
    }
    cout<<ans-*min_element(f+max(n-k,1),f+n+1)<<"\n";
}
```

斜率优化DP

```
int n,s;
int q[N];
ll A[N],B[N],dp[N];


double X(int j){
    return B[j];
}

double Y(int j){
    return dp[j];
}

double slope(int a,int b){
    return (Y(a)-Y(b))/(X(a)-X(b));
}

int find(int l,int r,ll k){
    int ans=r;
    while(l<=r){
        int mid=(l+r)>>1;
        if((Y(q[mid+1])-Y(q[mid]))>k*(X(q[mid+1])-X(q[mid]))) r=mid-1,ans=mid;
        else l=mid+1;
    }
    return q[ans];
}

void solve(){
    cin>>n>>s;
    vector<ll> t(n+1),c(n+1),sumt(n+1),sumc(n+1);
    rep(i,1,n){
        cin>>t[i]>>c[i];
        sumt[i]=sumt[i-1]+t[i];
        sumc[i]=sumc[i-1]+c[i];
    }
```

```
    rep(i,1,n){
        A[i]=sumc[i]*sumt[i]+sumc[n]*s;
        B[i]=sumc[i];
    }
    //dp[i] = dp[j] + (sumc[i]-sumc[j])*sumt[i] + (sumc[n]-sumc[j])*s
    //dp[i] = dp[j] + sumc[i]*sumt[i]+sumc[n]*s - (sumt[i]+s)*sumc[j]
    //设 A[i]=sumc[i]*sumt[i]+sumc[n]*s   B[j]=sumc[j]
    //dp[j] = (sumt[i]+s)*sumc[j] + (dp[i]-sumc[i]*sumt[i]-sumc[n]*s)
    //dp[j] = (sumt[i]+s)*B[j] + A[i]
    //  y  =       k      * x   +  b

    int hh=1,tt=1; //向加入(0,0)点(即dp[0])
    for(int i=1;i<=n;i++){
        int j=find(hh,tt,sumt[i]+s);//当前斜率
        dp[i]=dp[j]+A[i]-(sumt[i]+s)*B[j];
        // while(hh<tt && slope(i,q[tt])<=slope(q[tt],q[tt-1])) tt--;//队尾去不在凸包上的
        while(hh<tt && (Y(i)-Y(q[tt]))*(X(q[tt])-X(q[tt-1]))<=(Y(q[tt])-Y(q[tt-1]))*
(X(i)-X(q[tt]))) tt--;
        q[++tt]=i;
    }

    cout<<dp[n]<<"\n";
}
```

SOSDP(高维前缀和)

> 对于所有的$i, 0 \le i \le 2^n - 1$，求解$\sum_{j \subset i} a_j$。

```
for(int j=0;j<n;j++){
    for(int i=0;i<(1<<n);i++){
        if(i>>j&1) sum1[i]+=sum1[i^(1<<j)];//子集前缀和
        else sum2[i]+=sum2[i^(1<<j)];//超集前缀和
    }
}
```

杂

单调队列

求m区间内的最小值

```
int q[N];

void solve(){
    int n,m;cin>>n>>m;
    vector<int> a(n+1);
    rep(i,1,n) cin>>a[i];
    cout<<"0\n";
    int hh=1,tt=0;
    rep(i,1,n-1){
        while(hh<=tt &&i-q[hh]+1>m) hh++;
        while(hh<=tt && a[i]<=a[q[tt]]) tt--;
        q[++tt]=i;
        cout<<a[q[hh]]<<"\n";
    }
}
```

单调栈

第 i个元素之后第一个大于 a_i 的元素的**下标**

```
void solve(){
    int n;cin>>n;
    vector<int> v(n+1);
    vector<int> ans(n+1);
    stack<int> s;
    rep(i,1,n){
        cin>>v[i];
    }
    per(i,n,1){
        while(!s.empty()&&v[s.top()]<=v[i]) s.pop();
        ans[i]=!s.empty()?s.top():0;
        s.push(i);
    }
    rep(i,1,n) cout<<ans[i]<<" \n"[i==n];
}
```

三分

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double eps=1e-6;
double a,b,c,d;
double xx[15];
int n;

double f(double x){
    double ans=0;
    for(int i=0;i<=n;i++){
        ans=ans*x+xx[i];
    }
    return ans;
}

double three_devide(double l,double r){
    while(fabs(r-l)>=eps){
        double m1,m2;
        m1=l+(r-l)/3;
        m2=r-(r-l)/3;
        if(f(m1)>f(m2)) r=m2;
        else l=m1;
    }
    return l;
}

int main(){
    double l,r;
    cin>>n>>l>>r;
    for(int i=0;i<=n;i++) cin>>xx[i];
    printf("%.5lf",three_devide(l,r));//输出答案
    return 0;
}
```

离散化

```
int a[N],b[N],x[N*2],p[N*2]; //乘几具体看题目

int n;cin>>n;
for(int i=0;i<n;i++){
    cin>>a[i]>>b[i];
    x[cnt++]=a[i];
    x[cnt++]=b[i];
}
sort(x,x+cnt);
cnt=unique(x,x+cnt)-x;
for(int i=0;i<n;i++){
    a[i]=lower_bound(x,x+cnt,a[i])-x;
    b[i]=lower_bound(x,x+cnt,b[i])-x;
}
```

```
sort(v.begin(),v.end());
v.erase(unique(v.begin(),v.end()),v.end());
t=lower_bound(v.begin(),v.end(),t)-v.begin();
```

模拟退火

时间允许可以多跑几次SA

```
const double eps=1e-12;

void SA(){
    double t=2000,delta=0.996;
    double nowx=50;
    while(t>eps){
        double nextx=nowx+((rand()<<1)-RAND_MAX)*t;
        if(nextx>=0&&nextx<=100){
            double nextans=calc(nextx);
            double Delta=nextans-ans;
            if(Delta<0){
                ans=nextans;
                nowx=nextx;
            }
            else if(exp(-Delta/t)*RAND_MAX>rand()) nowx=nextx;
        }
        t*=delta;
    }
}


while((double)clock()/CLOCKS_PER_SEC<0.8) calc(); //卡时
```

逆序对的数量

逆序对：满足$i<j$且$a[i]>a[j]$

```
#include <bits/stdc++.h>
using namespace std;
const int N=100010;
int a[N],b[N];
long long res=0;
void merge_sort(int a[],int l,int r){
    if(l>=r) return;
```

```
    int mid=l+r>>1;

    merge_sort(a,l,mid), merge_sort(a,mid+1,r);

    int k=0,i=l,j=mid+1;
    while(i<=mid&&j<=r){
        if(a[i]<=a[j]) b[k++]=a[i++];
        else{
            res+=mid-i+1;
            b[k++]=a[j++];
        }
    }
    while(i<=mid) b[k++]=a[i++];
    while(j<=r) b[k++]=a[j++];
    for(i=l,j=0;i<=r;i++,j++) a[i]=b[j];
}
int main(){
    int n;cin>>n;
    for(int i=0;i<n;i++) scanf("%d",&a[i]);
    merge_sort(a,0,n-1);
    cout<<res<<endl;
    return 0;
}
```

悬线法

矩形的数量

```
//矩阵中白色(W)矩形的数量
char a[N][N];
ll h[N];
ll l[N],r[N];
ll ans;

void solve(){
    int n;cin>>n;
    rep(i,1,n) rep(j,1,n) cin>>a[i][j];
    rep(i,1,n){
        rep(j,1,n){
            h[j]++;
            if(a[i][j]=='B') h[j]=0;
        }

        stack<int> s1;
        rep(j,1,n){
            while(!s1.empty()&&h[j]<h[s1.top()]) s1.pop();
            l[j]=s1.size()?s1.top():0;
            s1.push(j);
        }

        stack<int> s2;
        per(j,n,1){
            while(!s2.empty()&&h[j]<=h[s2.top()]) s2.pop();
            r[j]=s2.size()?s2.top():(n+1);
            s2.push(j);
        }

        rep(j,1,n) ans+=(j-l[j])*(r[j]-j)*h[j];
    }
    cout<<ans<<"\n";
}
```

最大子矩阵

```
//最大 'F' 矩形土地面积
char a[N][N];
int h[N];

void solve(){
    int ans=0;
    int n,m;cin>>n>>m;
    rep(i,1,n){
        stack<int> s;
        rep(j,1,m){
            cin>>a[i][j];
            if(a[i][j]=='F') h[j]++;
            else h[j]=0;
            while(!s.empty()&&h[s.top()]>=h[j]){
                int hh=h[s.top()];s.pop();
                int left=!s.empty()?s.top():0;
                ans=max(ans,hh*(j-1-left));
            }
            s.push(j);
        }
        while(!s.empty()){
            int hh=h[s.top()];s.pop();
            int left=!s.empty()?s.top():0;
            ans=max(ans,hh*(m-left));
        }
    }
    cout<<ans<<"\n";
}
```

二维前缀和

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1010;

int a[N][N], s[N][N];

int main(){
    int n,m,q;
    scanf("%d%d%d",&n,&m,&q);
    for(int i=1;i<=n;i++){
        for (int j=1;j<=m;j++){
            scanf("%d",&a[i][j]);
            s[i][j]=a[i][j];
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            s[i][j]+=s[i-1][j];
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            s[i][j]+=s[i][j-1];
        }
    }
    while(q--){
        int x1,y1,x2,y2;
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
```

```
        printf("%d\n",s[x2][y2]-s[x1-1][y2]-s[x2][y1-1]+s[x1-1][y1-1]);
    }
    return 0;
}
```

差分

一维差分

```
#include <bits/stdc++.h>
using namespace std;
const int N=100010;
int a[N],b[N];

int main(){
    int m,n,l,r,c;
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        b[i]=a[i]-a[i-1];  //构造差分数组
    }

    while(m--){
        cin>>l>>r>>c;//表示将序列中[l,r]之间的每个数加上c
        b[l]+=c;
        b[r+1]-=c;
    }

    for(int i=1;i<=n;i++){
        a[i]=a[i-1]+b[i];
        cout<<a[i]<<" \n"[i==n];
    }
    return 0;
}
```

二维差分

```
#include <bits/stdc++.h>
using namespace std;
const int N=1010;
int a[N][N],b[N][N];

int main(){
    int m,n,q,x1,y1,x2,y2,c;
    cin>>n>>m>>q;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            cin>>a[i][j];
            b[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j-1];  //构造差分数组
        }
    }

    while(q--){
        cin>>x1>>y1>>x2>>y2>>c;
        b[x1][y1]+=c;
        b[x2+1][y1]-=c;
        b[x1][y2+1]-=c;
        b[x2+1][y2+1]+=c;
    }

    for(int i=1;i<=n;i++){
```

```
        for(int j=1;j<=m;j++){
            a[i][j]=a[i-1][j]+a[i][j-1]-a[i-1][j-1]+b[i][j];
            cout<<a[i][j]<<" \n"[j==m];
        }
    }
    return 0;
}
```

高阶等差数列

对于一个给定的数列，将连续两项之间的差$b_n=a_{n+1}-a_n$得到一个新的数列，那么b_n称为原数列的一阶等差数列，若$c_n=b_{n+1}-b_n$，那么c_n称为原数列的二阶等差数列，以此类推...

高阶等差数列都有一个多项式的通项公式。

差分法

给定序列a，依次求出该序列的k阶等差序列，直到某个序列全为0为止，按照下列排列规则排列在纸上

$$
\begin{array}{llllll}
C_n^1 & a_1 & a_2 & a_3 & a_4 & a_5\dots \\
C_n^2 & & b_1 & b_2 & b_3 & b_4\dots \\
C_n^3 & & & c_1 & c_2 & c_3\dots \\
\dots & & & \dots & \dots & \\
C_n^m & & & & 0 & 0\dots
\end{array}
$$

上表称为序列a的差分表

定理一

若序列a的多项式$P(x)$的最高幂次为n，对于任何的$k\geq n$，k阶差分恒为0

定理二

序列的前缀和$S_n=a_1C_n^1+b_1C_n^2+c_1C_n^3+\dots+0C_n^m$，那么通项公式$a_n=S_n-S_{n-1}$

案例引入

设$a[i]=1,4,9,16,25,\dots$

差分表如下:

$$
\begin{array}{llllll}
C_n^1 & 1 & 4 & 9 & 16 & 25\dots \\
C_n^2 & & 3 & 5 & 7 & 9\dots \\
C_n^3 & & & 2 & 2 & 2\dots \\
C_n^4 & & & & 0 & 0\dots
\end{array}
$$

那么$S_n=C_n^1+3C_n^2+2C_n^3$，然后可得$S_n=\frac{n(n+1)(2n+1)}{6}$,$a_n=S_n-S_{n-1}=n^2$

生成函数

$1+x+x^2+\cdots+x^\infty=\frac{1}{1-x}$

$1+x=\frac{1-x^2}{1-x}$

$1+x+x^2+\cdots+x^n=\frac{1-x^{n+1}}{1-x}$

$(1+x)^\alpha=\sum_{k=0}^\infty C_\alpha^k x^k$

$\frac{1}{(1-x)^\alpha}=\sum_{k=0}^\infty C_{\alpha+k-1}^k x^k$ 第 x^n的系数即为$C_{n+\alpha-1}^n=C_{n+\alpha-1}^{\alpha-1}$

自适应辛普森积分

```
double f(double x){ //要求解积分的函数
    return sin(x)/x;
}

double simpson(double l, double r){ //辛普森积分的公式
    double mid = (l + r) / 2;
    return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6;
}

double calc(double L, double R, double ans) {
    double mid = (L + R) / 2, l = simpson(L, mid), r = simpson(mid, R);
    if (fabs(l + r - ans) <= eps) return ans;
    return calc(L, mid, l) + calc(mid, R, r);
}

void solve(){
    double a,b;cin>>a>>b;
    cout<<fixed<<setprecision(6)<<calc(a,b,simpson(a,b))<<"\n";
}
```

约数个数与约数和

$a = p_1^{k_1} * p_2^{k_2} * p_3^{k_3} * \ldots * p_n^{k_n}$

约数个数：$cnt = (1 + k_1) * (1 + k_2) * \ldots * (1 + k_n)$

约数个数：$d(n)$表示

$d(ij) = \sum_{x|i} \sum_{y|j} [\gcd(x, y) = 1]$

> 当i是质数的时候，它的约数就是1和它本身。
> 当i%prime[j]==0的时候，先除掉它原来最小质因子对约数的贡献，再乘上最小质因子个数加1，就是i×prime[j]的约数个数了。
> 当i%prime[j]!=0的时候，这个时候枚举到的这一个质数是原来i中没有的，那它的贡献一定是2，所以在i的约数上乘以二就可以了。

```
//d表示约数个数 pre表示最小质因子出现次数
ll d[N],pre[N];
int prime[N],cnt=0;
bool st[N]; //false为素数
void get_phi(int n){
    st[1]=1;d[1]=1;
    for(int i=2;i<=n;i++){
        if(!st[i]) prime[cnt++]=i,d[i]=2,pre[i]=1;
        for(int j=0;j<cnt&&i*prime[j]<=n;j++){
            st[i*prime[j]]=true;
            if(i%prime[j]==0){
                pre[i*prime[j]]=pre[i]+1;
                d[i*prime[j]]=d[i]/(pre[i]+1)*(pre[i]+2);
                // d[i*prime[j]]=d[i]/pre[i*prime[j]]*(pre[i*prime[j]]+1);
                break;
            }
            pre[i*prime[j]]=1;
            d[i*prime[j]]=d[i]*2;
        }
    }
}
```

约数和:

$sum=(1+p_1^1+p_1^2+p_1^3+...+p_1^{k_1})*(1+p_2^1+p_2^2+p_2^3+...+p_1^{k_2})*...*(1+p_n^1+p_n^2+p_n^3+...+p_n^{k_n})$
$=\frac{p_1^{k_1+1}-1}{p_1-1}*\frac{p_2^{k_2+1}-1}{p_2-1}*...*\frac{p_n^{k_n+1}-1}{p_n-1}$

约数和: $\sigma(n)$表示

$\sigma(ij)=\sum\limits_{x|i}\sum\limits_{y|j}[\gcd(x,y)=1]*x*\frac{j}{y}=\sum\limits_{x|i}\sum\limits_{y|j}[\gcd(x,y)=1]*\frac{i}{x}*y$

```
ll sig[N],sum[N],pre[N];
//约数的和/最小质因子指数次幂和/最小质因子的指数次幂
int prime[N],cnt=0;
bool st[N]; //false为素数
void get_phi(int n){
    st[1]=1;sig[1]=1;
    for(int i=2;i<=n;i++){
        if(!st[i]){
            prime[cnt++]=i;
            sig[i]=i+1;sum[i]=i+1;pre[i]=i;
        }
        for(int j=0;j<cnt&&i*prime[j]<=n;j++){
            st[i*prime[j]]=true;
            if(i%prime[j]==0){
                pre[i*prime[j]]=prime[j]*pre[i];
                sum[i*prime[j]]=sum[i]+pre[i*prime[j]];
                sig[i*prime[j]]=sig[i]/sum[i]*sum[i*prime[j]];
                break;
            }
            sig[i*prime[j]]=sig[i]*sig[prime[j]];
            sum[i*prime[j]]=prime[j]+1;
            pre[i*prime[j]]=prime[j];
        }
    }
}
```

莫比乌斯反演

$F(n)=\sum_{d|n}f(d)\iff f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$

$F(n)=\sum_{n|d}f(d)\iff f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$

$[gcd(i,j)=1]=\sum_{d|gcd(i,j)}\mu(d)$

- 对于这种与gcd有关的莫比乌斯反演，一般我们都是套路的去设$f(d)$为$gcd(i,j)=d$的个数，$F(n)$为$gcd(i,j)=n$和n的倍数的个数，即:
  $f(d)=\sum_{i=1}^{N}\sum_{j=1}^{M}[gcd(i,j)=d]$
  $F(n)=\sum_{n|d}f(d)=\lfloor\frac{N}{n}\rfloor\lfloor\frac{M}{n}\rfloor$
  $f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j)=1]=\sum_{d=1}^{n}\mu(d)\lfloor\frac{n}{d}\rfloor\lfloor\frac{m}{d}\rfloor$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j)=k]=\sum_{i=1}^{\lfloor\frac{n}{k}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{k}\rfloor}[gcd(i,j)=1]$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} gcd(i,j) = \sum_{d=1}^{n} \varphi(d)\lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$$