泡泡猿 ACM 模板

Rand0w & REXWIND & Dallby

2021 年 11 月 29 日

目录

1 头文件

1.1 头文件 (Rand0w)

```cpp
#include <bits/stdc++.h>
//#include <bits/extc++.h>
//using namespace __gnu_pbds;
//using namespace __gnu_cxx;
using namespace std;
#pragma optimize(2)
//#pragma GCC optimize("Ofast,no-stack-protector")
//#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,tune=native")
#define rbset(T) tree<T,null_type,less<T>,rb_tree_tag,tree_order_statistics_node_update>
const int inf = 0x7FFFFFFF;
typedef long long ll;
typedef double db;
typedef long double ld;
template<class T>inline void MAX(T &x,T y){if(y>x)x=y;}
template<class T>inline void MIN(T &x,T y){if(y<x)x=y;}
namespace FastIO
{
char buf[1 << 21], buf2[1 << 21], a[20], *p1 = buf, *p2 = buf, hh = '\n';
int p, p3 = -1;
void read() {}
void print() {}
inline int getc()
{
return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 1 << 21, stdin), p1 == p2) ? EOF : *p1++;
}
inline void flush()
{
fwrite(buf2, 1, p3 + 1, stdout), p3 = -1;
}
template <typename T, typename... T2>
inline void read(T &x, T2 &... oth)
{
int f = 0;x = 0;char ch = getc();
while (!isdigit(ch)){if (ch == '-')f = 1;ch = getc();}
while (isdigit(ch)){x = x * 10 + ch - 48;ch = getc();}
x = f ? -x : x;read(oth...);
}
template <typename T, typename... T2>
inline void print(T x, T2... oth)
{
if (p3 > 1 << 20)flush();
if (x < 0)buf2[++p3] = 45, x = -x;
do{a[++p] = x % 10 + 48;}while (x /= 10);
do{buf2[++p3] = a[p];}while (--p);
buf2[++p3] = hh;
print(oth...);
}
} // namespace FastIO
#define read FastIO::read
#define print FastIO::print
```

```
#define flush FastIO::flush
#define spt fixed<<setprecision
#define endll '\n'
#define mul(a,b,mod) (__int128)(a)*(b)%(mod)
#define pii(a,b) pair<a,b>
#define pow powmod
#define X first
#define Y second
#define lowbit(x) (x&-x)
#define MP make_pair
#define pb push_back
#define pt putchar
#define yx_queue priority_queue
#define lson(pos) (pos<<1)
#define rson(pos) (pos<<1|1)
#define y1 code_by_Rand0w
#define yn A_muban_for_ACM
#define j1 it_is just_an_eastegg
#define lr hope_you_will_be_happy_to_see_this
#define int long long
#define rep(i, a, n) for (register int i = a; i <= n; ++i)
#define per(i, a, n) for (register int i = n; i >= a; --i)
const ll llinf = 4223372036854775851;
const ll mod = (0 ? 1000000007 : 998244353);
ll pow(ll a,ll b,ll md=mod) {ll res=1;a%=md; assert(b>=0); for(;b;b>>=1){if(b&1)res=mul(res,a,md);a=mul(a,a,
    md);}return res;}
const ll mod2 = 999998639;
const int m1 = 998244353;
const int m2 = 1000001011;
const int pr=233;
const double eps = 1e-7;
const int maxm= 1;
const int maxn = 510000;
void work()
{
}
signed main()
{
  #ifndef ONLINE_JUDGE
  //freopen("in.txt","r",stdin);
   //freopen("out.txt","w",stdout);
#endif
   //std::ios::sync_with_stdio(false);
   //cin.tie(NULL);
   int t = 1;
   //cin>>t;
   for(int i=1;i<=t;i++){
      //cout<<"Case #"<<i<<":"<<endll;
      work();
   }
   return 0;
}
```

1.2 头文件 (REXWind)

```cpp
#include<iostream>
#include<cstring>
#include<cstdio>
#include<algorithm>
#include<vector>
#include<map>
#include<queue>
#include<cmath>
using namespace std;

template<class T>inline void read(T &x){
    x=0;char o,f=1;
    while(o=getchar(),o<48)if(o==45)f=-f;
    do x=(x<<3)+(x<<1)+(o^48);while(o=getchar(),o>47);x*=f;}
int cansel_sync=(ios::sync_with_stdio(0),cin.tie(0),0);
#define ll long long
#define ull unsigned long long
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define repb(i,a,b) for(int i=(a);i>=b;i--)
#define mkp make_pair
#define ft first
#define sd second
#define log(x) (31-__builtin_clz(x))
#define INF 0x3f3f3f3f
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
ll gcd(ll a,ll b){ while(b^=a^=b^=a%=b); return a; }
//#define INF 0x7fffffff

void solve(){

}

int main(){
    int z;
    cin>>z;
    while(z--) solve();
}
```

1.3 头文件 (Dallby)

```cpp
#include<bits/stdc++.h>
// #pragma GCC optimize(3)
#include<bits/stdc++.h>
using namespace std;
#define rep(i,x,y) for(int i=(x);i<=(y);++i)
#define dep(i,x,y) for(int i=(x);i>=(y);--i)
#define mst(a,x) memset(a,x,sizeof(a))
#define endl "\n"
#define fr first
#define sc second
#define debug cout<<"DEBUG\n";
#define OMG(a,n) rep(i,1,n) cout<<a[i]<<" "; cout<<endl;
#define OMG2(a,n,m) rep(i,1,n) {rep(i,1,m) cout<<a[i][j]<<" "; cout<<endl;}
template <typename Type> void RIP(Type x) {cout<<x<<endl;}template <typename Type, typename... Targs>void RIP(Type x, Targs... args) {cout<<x<<" ";RIP(args...);}
mt19937 rnd(chrono::high_resolution_clock::now().time_since_epoch().count());
typedef long long ll; typedef unsigned long long ull; typedef pair<int,ll>pil; typedef pair<int,int>pii; typedef pair<ll,ll>pll;
const int N=1e6+10; const double eps=1e-9;
const int inf=0x3f3f3f3f; const ll INF=0x3f3f3f3f3f3f3f3f;
const int mo=(1?998244353:1000000007); ll mul(ll a,ll b,ll m=mo){return a*b%m;} ll fpow(ll a,ll b,ll m=mo){ll ans=1; for(;b;a=mul(a,a,m),b>>=1)if(b&1)ans=mul(ans,a,m); return ans;}
inline ll read(){ll x=0,tag=1; char c=getchar();for(;!isdigit(c);c=getchar())if(c=='-')tag=-1;for(; isdigit(c);c=getchar())x=x*10+c-48;return x*tag;}
typedef double lf; const lf pi=acos(-1.0); lf readf(){lf x; if(scanf("%lf",&x)!=1)exit(0); return x;} template<typename T> T sqr(T x){return x*x;}
ll a[N],b[N];
void Solve(){

}
int main(){
   //freopen("D:\\in.txt","r",stdin);
   //freopen("D:\\out.txt","w",stdout);
   //ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
   int T=1; //T=read();
   rep(kase,1,T){
      Solve();
   }
   return 0;
}
```

2 博弈

2.1 SG 函数

```
//sg板子
int sg[MAXN];
bool vis[MAXN];
inline void get_sg(){
   int j;
   rep(i,0,MAXN-1){
      memset(vis,0,sizeof(vis));
      rep(j,1,cnt){//转移
         if(i-fb[j]<0) break;
         vis[sg[i-fb[j]]] = 1;
      }
      //找mex
      rep(j,0,MAXN-1)//找后继中最小未出现的数字
         if(!vis[j]){sg[i]=j;break;}
   }
}
```

2.2 阶梯 Nim

n 个位置，每个位置上有 ai 个石子，两个人轮流操作，操作步骤是：选择 i 位置上的任意数量石子，移动到下一级阶梯 (i-1) 位置。直到 0 级阶梯不能移动。谁最后无法操作谁输。

问题等价于，求位置为奇数的 ai 的异或和，若异或和等于 0，则先手必败，否则先手必胜。

2.3 威佐夫博弈

普通的威佐夫博弈：有 2 堆石子，每人每次可以拿走任意一堆中任意数量的石子或在两堆石子中拿走相同数量的石子，不能拿的人输。

$\frac{1}{\alpha}+\frac{1}{\alpha+1}=1$ 计算解得 $\alpha=\frac{\sqrt{5}+1}{2}$

第 n 个奇异局势 (必败态)

$$a[n]=\lfloor n*\frac{\sqrt{5}+1}{2}\rfloor, b[n]=a[n]+n;$$

扩展威佐夫博弈：从一堆拿走 x，另一堆拿走 y，满足 $|x-y|\le k$。

通过这条式子用 k 解 $\frac{1}{\alpha}+\frac{1}{\alpha+k+1}=1$ 因此第 n 个奇异局势为

$$a[n]=\lfloor n*\alpha\rfloor, b[n]=a[n]+n*(k+1);$$

当初是在多校写的这题，这个 α 可以通过二分求，之后也用二分去找 n。

```
void solve(){
   cin>>a>>b>>k;
   if(a>b) swap(a,b);
   double l = 1,r = 2;//alpha的区间应该是在1和2之间的
   double now=0,mid;
   while(abs(now-1.0)>eps){//第一次二分寻找合适的alpha
      mid = (l+r)/2;
      now = cal(k,mid);
      if(now>1.0) l = mid;
      else r = mid;
   }
   double alpha = mid;//通过二分计算得到alpha
   int lb = 0,rb = 5e7+5;
```

```
    int midd;
    while(lb+1<rb){//第二次二分求a对应第几个二元组
        midd = (lb+rb)/2;
        if(floor(midd*alpha)>a) rb = midd;
        else lb = midd;
    }
    if(floor(lb*alpha)==a&&a+(k+1)*lb==b) cout<<0<<endl;
    else cout<<1<<endl;
}
```

2.4 二分图博弈

给定一张二分图，有一个起始点，AliceBob 俩人轮流移动，不能经过重复的点，谁不能移动了谁输掉游戏

问题转化为每一次可以从 X 部的点走到 Y 部的点: 一开始的人在 X, 如果不能走就输了

考虑起点 S 在不在一个完美匹配中即可。先去掉 S 点跑一次二分图匹配，之后加入 X 点，看匹配数是否增加，如若一样多，则说明 S 是可有可无的，存在不包含 S 的最大匹配。否则，说明 S 是不可或缺的。

如果 S 不可或缺，那先手无论选择哪个点，它都一定是匹配点，则先手就输掉了。

3 数学

3.1 高次期望 DP

简言之，就是高次的期望值需要从低次的推得，如求 X^3 的期望，则需要拆分

$$x^2 = ((x-1)+1)^2 = (x-1)^2 + 2(x-1) + 1$$

转为递推式

```
b[i] = b[i-1] + 2*a[i] + 1;
c[i] = c[i-1] + 3*b[i-1] + 3*a[i-1] + 1
```

这题是有 n 次游戏，第 i 次成功概率为 p[x], 连续 X 次成功会得到 X^3 分，问期望分数

```
inline void solve(){
    cin>>n;
    rep(i,1,n) cin>>p[i];
    a[0]=b[0]=f[0]=0;
    rep(i,1,n)
        a[i] = (a[i-1]+1)*p[i];
    rep(i,1,n)
        b[i] = (b[i-1]+2*a[i-1]+1)*p[i];
    rep(i,1,n)
        f[i] = (f[i-1]+3*b[i-1]+3*a[i-1]+1)*p[i] + f[i-1]*(1-p[i]);
    printf("%.1lf\n",f[n]);
}
```

3.2 公式集合

3.2.1 排列组合相关公式

化简可能有用

$C_{n+1}^{i} = C_{n}^{i} + C_{n}^{i-1}$

$m \cdot C_{n}^{m} = n \cdot C_{n-1}^{m-1}$

$1C_{n}^{1} + 2C_{n}^{2} + 3C_{n}^{3} + \cdots + nC_{n}^{n} = n2^{n-1}$

$1^2C_{n}^{1} + 2^2C_{n}^{2} + 3^2C_{n}^{3} + \cdots + n^2C_{n}^{n} = n(n+1)2^{n-2}$

3.2.2 数列求和相关公式

$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

$\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$

3.2.3 错排公式

每个数字都不在自己位置上的排列数

D(n) = (n-1) [D(n-2) + D(n-1)]

特殊地，D(1) = 0, D(2) = 1.

3.3 欧拉筛

$O(n)$ 筛素数

```
int primes[maxn+5],tail;
bool is_prime[maxn+5];
void euler(){
  is_prime[1] = 1;
  for (int i = 2; i < maxn; i++)
  {
    if (!is_prime[i])
    primes[++tail]=i;
    for (int j = 0; j < primes.size() && i * primes[j] < maxn; j++)
    {
      is_prime[i * primes[j]] = 1;
      if ((i % primes[j]) == 0)
        break;
    }
  }
}
```

3.4 Exgcd

求出 $ax + by = gcd(a, b)$ 的一组可行解 $O(logn)$

```
void Exgcd(ll a,ll b,ll &d,ll &x,ll &y){
   if(!b){d=a;x=1;y=0;}
   else{Exgcd(b,a%b,d,y,x);y-=x*(a/b);}
}
```

非递归版

```
int gcd(int a, int b, int& x, int& y) {
   x = 1, y = 0;
   int x1 = 0, y1 = 1, a1 = a, b1 = b;
   while (b1) {
```

```
    int q = a1 / b1;
    tie(x, x1) = make_tuple(x1, x - q * x1);
    tie(y, y1) = make_tuple(y1, y - q * y1);
    tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}
```

3.5 Excrt 扩展中国剩余定理

求解同余方程组

$$\begin{cases} x \ \% \ b_1 \equiv \ a_1 \\ x \ \% \ b_2 \equiv \ a_2 \\ \quad\quad \vdots \\ x \ \% \ b_n \equiv \ a_n \end{cases}$$

```
int excrt(int a[],int b[],int n){
  int lc=1;
  for(int i=1;i<=n;i++)
    lc=lcm(lc,a[i]);
  for(int i=1;i<n;i++){
    int p,q,g;
    g=exgcd(a[i],a[i+1],p,q);
    int k=(b[i+1]-b[i])/g;
    q=-q;p*=k;q*=k;
    b[i+1]=a[i]*p%lc+b[i];
    b[i+1]%=lc;
    a[i+1]=lcm(a[i],a[i+1]);
  }
  return (b[n]%lc+lc)%lc;
}
```

3.6 线性求逆元

```
void init(int p){
  inv[1] = 1;
  for(int i=2;i<=n;i++){
    inv[i] = (ll)(p-p/i)*inv[p%i]%p;
  }
}
```

3.7 多项式

3.7.1 FFT 快速傅里叶变换

```
const int SIZE=(1<<21)+5;
const double PI=acos(-1);
struct CP{
   double x,y;
   CP(double x=0,double y=0):x(x),y(y){}
   CP operator +(const CP &A)const{return CP(x+A.x,y+A.y);}
   CP operator -(const CP &A)const{return CP(x-A.x,y-A.y);}
   CP operator *(const CP &A)const{return CP(x*A.x-y*A.y,x*A.y+y*A.x);}
};
int limit,rev[SIZE];
void DFT(CP *F,int op){
   for(int i=0;i<limit;i++)if(i<rev[i])swap(F[i],F[rev[i]]);
   for(int mid=1;mid<limit;mid<<=1){
      CP wn(cos(PI/mid),op*sin(PI/mid));
      for(int len=mid<<1,k=0;k<limit;k+=len){
         CP w(1,0);
         for(int i=k;i<k+mid;i++){
            CP tmp=w*F[i+mid];
            F[i+mid]=F[i]-tmp;
            F[i]=F[i]+tmp;
            w=w*wn;
         }
      }
   }
   if(op==-1)for(int i=0;i<limit;i++)F[i].x/=limit;
}
void FFT(int n,int m,CP *F,CP *G){
   for(limit=1;limit<=n+m;limit<<=1);
   for(int i=0;i<limit;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?limit>>1:0);
   DFT(F,1),DFT(G,1);
   for(int i=0;i<limit;i++)F[i]=F[i]*G[i];
   DFT(F,-1);
}
```

3.7.2 NTT 快速数论变换

```
const int SIZE=(1<<21)+5;
int limit,rev[SIZE];
void DFT(ll *f, int op) {
    const ll G = 3;
    for(int i=0; i<limit; ++i) if(i<rev[i]) swap(f[i],f[rev[i]]);
    for(int len=2; len<=limit; len<<=1) {
        ll w1=pow(pow(G,(mod-1)/len),~op?1:mod-2);
        for(int l=0, hf=len>>1; l<limit; l+=len) {
            ll w=1;
            for(int i=l; i<l+hf; ++i) {
                ll tp=w*f[i+hf]%mod;
                f[i+hf]=(f[i]-tp+mod)%mod;
                f[i]=(f[i]+tp)%mod;
                w=w*w1%mod;
            }
        }
    }
    if(op==-1) for(int i=0, inv=pow(limit,mod-2); i<limit; ++i) f[i]=f[i]*inv%mod;
}
void NTT(int n,int m,int *F,int *G){
    for(limit=1;limit<=n+m;limit<<=1);
    for(int i=0;i<limit;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?limit>>1:0);
    DFT(F,1),DFT(G,1);
    for(int i=0;i<limit;i++)F[i]=F[i]*G[i];
    DFT(F,-1);
}
```

3.7.3 MTT 任意模数 FFT

FFT 版常数巨大，慎用。

```
struct MTT{
    static const int N=1<<20;
    struct cp{
        long double a,b;
        cp(){a=0,b=0;}
        cp(const long double &a,const long double &b):a(a),b(b){}
        cp operator+(const cp &t)const{return cp(a+t.a,b+t.b);}
        cp operator-(const cp &t)const{return cp(a-t.a,b-t.b);}
        cp operator*(const cp &t)const{return cp(a*t.a-b*t.b,a*t.b+b*t.a);}
        cp conj()const{return cp(a,-b);}
    };
    cp wn(int n,int f){
        static const long double pi=acos(-1.0);
        return cp(cos(pi/n),f*sin(pi/n));
    }
    int g[N];
    void dft(cp a[],int n,int f){
        for(int i=0;i<n;i++)if(i>g[i])swap(a[i],a[g[i]]);
        for(int i=1;i<n;i<<=1){
            cp w=wn(i,f);
            for(int j=0;j<n;j+=i<<1){
                cp e(1,0);
```

```
            for(int k=0;k<i;e=e*w,k++){
                cp x=a[j+k],y=a[j+k+i]*e;
                a[j+k]=x+y,a[j+k+i]=x-y;
            }
        }
    }
    if(f==-1){
        cp Inv(1.0/n,0);
        for(int i=0;i<n;i++)a[i]=a[i]*Inv;
    }
}
cp a[N],b[N],Aa[N],Ab[N],Ba[N],Bb[N];
vector<ll> conv_mod(const vector<ll> &u,const vector<ll> &v,ll mod){ // 任意模数fft
    const int n=(int)u.size()-1,m=(int)v.size()-1,M=sqrt(mod)+1;
    const int k=32-__builtin_clz(n+m+1),s=1<<k;
    g[0]=0; for(int i=1;i<s;i++)g[i]=(g[i/2]/2)|((i&1)<<(k-1));
    for(int i=0;i<s;i++){
        a[i]=i<=n?cp(u[i]%mod%M,u[i]%mod/M):cp();
        b[i]=i<=m?cp(v[i]%mod%M,v[i]%mod/M):cp();
    }
    dft(a,s,1); dft(b,s,1);
    for(int i=0;i<s;i++){
        int j=(s-i)%s;
        cp t1=(a[i]+a[j].conj())*cp(0.5,0);
        cp t2=(a[i]-a[j].conj())*cp(0,-0.5);
        cp t3=(b[i]+b[j].conj())*cp(0.5,0);
        cp t4=(b[i]-b[j].conj())*cp(0,-0.5);
        Aa[i]=t1*t3,Ab[i]=t1*t4,Ba[i]=t2*t3,Bb[i]=t2*t4;
    }
    for(int i=0;i<s;i++){
        a[i]=Aa[i]+Ab[i]*cp(0,1);
        b[i]=Ba[i]+Bb[i]*cp(0,1);
    }
    dft(a,s,-1); dft(b,s,-1);
    vector<ll> ans;
    for(int i=0;i<n+m+1;i++){
        ll t1=llround(a[i].a)%mod;
        ll t2=llround(a[i].b)%mod;
        ll t3=llround(b[i].a)%mod;
        ll t4=llround(b[i].b)%mod;
        ans.push_back((t1+(t2+t3)*M%mod+t4*M*M)%mod);
    }
    return ans;
}
}mtt;
```

3.7.4 FWT 快速沃尔什变换

计算

$$C_i = \sum_{j \bigoplus k=i}^{n} A_j \times B_k$$

$\bigoplus$ 可以是与、或、异或

```
void FWT(ll *f, int op) {
   for(int len=2; len<=up; len<<=1) {
      for(int l=0, hf=len>>1; l<up; l+=len) {
         for(int i=l; i<l+hf; ++i) {
            ll x=f[i], y=f[i+hf];
            if(op>0) {
               if(op==1) f[i]=(x+y)%mod, f[i+hf]=(x-y+mod)%mod; //xor
               else if(op==2) f[i]=(x+y)%mod; //and
               else f[i+hf]=(x+y)%mod; //or
            }
            else {
               if(op==-1) f[i]=(x+y)*inv2%mod, f[i+hf]=(x-y+mod)*inv2%mod; //xor
               else if(op==-2) f[i]=(x-y+mod)%mod; //and
               else f[i+hf]=(y-x+mod)%mod; //or
            }
         }
      }
   }
}
```

3.8 组合数

预处理阶乘，并通过逆元实现相除

```
ll jc[MAXN];
ll qpow(ll d,ll c){//快速幂
   ll res = 1;
   while(c){
      if(c&1) res=res*d%med;
      d=d*d%med;c>>=1;
   }return res;
}
inline ll niyuan(ll x){return qpow(x,med-2);}
void initjc(){//初始化阶乘
   jc[0] = 1;
   rep(i,1,MAXN-1) jc[i] = jc[i-1]*i%med;
}
inline int C(int n,int m){//n是下面的
   if(n<m) return 0;
   return jc[n]*niyuan(jc[n-m])%med*niyuan(jc[m])%med;
}
int main(){
   initjc();
   int n,m;
   while(cin>>n>>m) cout<<C(n,m)<<endl;
}
```

3.9 矩阵快速幂

```
struct Matrix{
    ll a[MAXN][MAXN];
    Matrix(ll x=0){
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                a[i][j]=x*(i==j);
            }
        }
    }
    Matrix operator *(const Matrix &b)const{//通过重载运算符实现矩阵乘法
        Matrix res(0);
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                for(int k=0;k<n;k++){
                    ll &ma = res.a[i][j];
                    ma = (ma+a[i][k]*b.a[k][j])%mod;
                }
            }
        }
        return res;
    }
};
Matrix qpow(Matrix d,ll m){//底数和幂次数
    Matrix res(1);//构造E单位矩阵
    while(m){
        if(m&1)
            res=res*d;
        d=d*d;
        m>>=1;
    }
    return res;
}
```

3.10 高斯消元

3.10.1 浮点

$O(n^3)$ 复杂度，需要用 double 存储。

```
double date[110][110];
bool guass(int n){
    for(int i=1;i<=n;i++){
        int mix=-1;
        for(int j=i;j<=n;j++)
            if(date[j][i]!=0){
                mix=j;break;
            }
        if(mix==-1)
            return false;
        if(mix!=i)
            for(int j=1;j<=n+1;j++)
                swap(date[mix][j],date[i][j]);
        double t=date[i][i];
```

```
        for(int j=i;j<=n+1;j++){
            date[i][j]=date[i][j]/t;
        }
        for(int j=1;j<=n;j++){
            if(date[j][i]==0||j==i)
                continue;
            double g=date[j][i]/date[i][i];
            for(int k=1;k<=n+1;k++)
                date[j][k]-=date[i][k]*g;
        }
    }
    return true;
}
```

3.10.2 质数取模

复杂度 $O(n^3)$

```
struct mat{
    static const int N=410;
    vector<ll> a[N];
    mat(){for(auto &i:a)i.assign(N*2,0);} // if get_inv is needed, N*2
    ll det;
    void r_div(int x,int m,ll k){ // a[x][]/=k
        ll r=qpow(k,mod-2);
        repeat(i,0,m)
            a[x][i]=a[x][i]*r%mod;
        det=det*k%mod;
    }
    void r_plus(int x,int y,int m,ll k){ // a[x][]+=a[y][]*k
        repeat(i,0,m)
            a[x][i]=(a[x][i]+a[y][i]*k)%mod;
    }
    bool gauss(int n,int m){ // n<=m, return whether succuss
        det=1;
        repeat(i,0,n){
            int t=-1;
            repeat(j,i,n)if(a[j][i]){t=j; break;}
            if(t==-1){det=0; return 0;}
            if(t!=i){a[i].swap(a[t]); det=-det;}
            r_div(i,m,a[i][i]);
            repeat(j,0,n)
            if(j!=i && a[j][i])
                r_plus(j,i,m,mod-a[j][i]);
        }
        return 1;
    }
    ll get_det(int n){gauss(n,n); return (det+mod)%mod;} // return det
    bool get_inv(int n){ // self=inv(self), return whether success
        repeat(i,0,n)
        repeat(j,0,n)
            a[i][j+n]=(i==j);
        bool t=gauss(n,n*2);
        repeat(i,0,n)
        repeat(j,0,n)
```

```
        a[i][j]=a[i][j+n];
    return t;
  }
  vector<ll> &operator[](int x){return a[x];}
  const vector<ll> &operator[](int x)const{return a[x];}
}a;
```

3.10.3 非质数取模

复杂度 $O(n^3logC)$

3.11 矩阵求逆

求 A 的逆矩阵，把 A 和单位矩阵 I 放在一个矩阵里
对 A 进行加减消元使 A 化成单位矩阵
此时原来单位矩阵转化成逆矩阵

3.12 欧拉降幂

$$a^b \equiv \begin{cases} a^{b\%\phi(p)}, & \gcd(a,p)=1 \\ a^b, & \gcd(a,p)\neq 1, b<\phi(p) \\ a^{b\%\phi(p)+\phi(p)}, & \gcd(a,p)\neq 1, b\geq\phi(p) \end{cases} (\mod p)$$

3.13 求质数的原根

模 m 乘法群中找到一个数 a，以 a 为生成元造出来的群的大小为 $\phi(m)$（对于指数 m 来说 =m-1），则 a 称为模数 m 的原根。
原根判定定理:
设 $m\geq 3, \gcd(a,m)=1$，则 a 是模 m 的原根的充要条件是，对于 $\varphi(m)$ 的每个素因数 p，都有 $a^{\frac{\varphi(m)}{p}}\not\equiv 1 \pmod m$。

```
vector<int> zyz;
inline ll get_yg(ll p){
  zyz.clear();;
  ll tp = p-1;//phi(p)
  int sqr = sqrt(tp+1);
  rep(i,2,sqr){
    if(tp%i==0)
      zyz.push_back(i);
    while(tp%i==0) tp/=i;
  }
  if(tp!=1) zyz.push_back(tp);
  rep(i,2,p){
    bool flag = 1;
    for(auto x:zyz)
      if(qpow(i,(p-1)/x,p)==1){flag = 0;break;}
    if(flag) return i;
  }
  return -1;
}
```

4 图论

4.1 树上倍增求 LCA

```cpp
const int maxn=510000;
const int lmaxn=log2(maxn)+3;
vector<int> mp[maxn];
int fa[maxn][lmaxn];
int n,m,s;
int dep[maxn];
void dfs(int pos,int f){
   fa[pos][0]=f;
   for(int i:mp[pos]){
      if(i==f)continue;
      dep[i]=dep[pos]+1;
      dfs(i,pos);
   }
}
void init(int n){
   for(int i=1;i<=log2(n)+1;i++)
      for(int j=1;j<=n;j++)
         fa[j][i]=fa[fa[j][i-1]][i-1];
}
int getlca(int u,int v){
   if(dep[u]<dep[v])swap(u,v);
   for(int i=log2(n)+1;i>=0;i--)
      if(dep[fa[u][i]]>=dep[v])u=fa[u][i];
   if(u==v)return u;
   for(int i=log2(n)+1;i>=0;i--)
      if(fa[u][i]!=fa[v][i]){
         u=fa[u][i];
         v=fa[v][i];
      }
   return fa[u][0];
}
int main(){
   ios::sync_with_stdio(false);
   cin.tie(0);
   cout.tie(0);
   cin>>n>>m>>s;
   int u,v;
   for(int i=1;i<n;i++){
      cin>>u>>v;
      mp[u].push_back(v);
      mp[v].push_back(u);
   }
   dep[s]=1;
   dfs(s,s);
   init(n);
   for(int i=1;i<=m;i++){
      cin>>u>>v;
      cout<<getlca(u,v)<<"\n";
   }
}
```

4.2 Tarjan

```cpp
int n;
vector<int> e[MAXN];//存图
int dfn[MAXN],low[MAXN];
int dfncnt;//dfn自增下标
int s[MAXN],tp;//数组模拟栈,tp记录大小
bool in[MAXN];//记录该点是否在栈中
int scc[MAXN],sc;//节点i所在scc的编号,sc记录有几个强连通
int sz[MAXN];//强连通i的大小
int indg[MAXN];//记录缩点后的入度（这题才有的

void tarjan(int u){
    low[u]=dfn[u]=++dfncnt;//low初始值为自身dfn
    s[++tp]=u;//推u入栈，从1开始
    in[u]=1;//记录u点在栈中
    for(auto v:e[u]){//访问到新点的情况
        if(!dfn[v]){
            tarjan(v);
            low[u] = min(low[u],low[v]);//用low[v}更新low[u]
        }
        else if(in[v])//v被访问过，但是在栈中
            low[u] = min(low[u],dfn[v]);
    }
    if(dfn[u]==low[u]){//u是连通分量的根节点
        sc++;//强连通数量++
        sz[sc] = 0;
        while(s[tp]!=u){//u和u之后的点全部出栈
            scc[s[tp]] = sc;//这个点包含于第几个强连通
            sz[sc]++;//u为根的这个强连通的大小
            in[s[tp]] = 0;//出栈
            tp--;
        }
        scc[u] = sc;//给根节点标，属于第sc个强连通
        sz[sc]++;
        in[u] = 0;
        tp--;
    }
}

void reset(){
    tp = sc = dfncnt =0;
    rep(i,1,n){
        in[i] = dfn[i] = 0;//low不用清空，sz在之后用到再清空
        e[i].clear();
    }
}

int main(){
    cin>>n;
    reset();
    int v;
    rep(u,1,n){
        while(cin>>v&&v!=0) e[u].push_back(v);
    }
```

```
    rep(u,1,n)
        if(!dfn[u]) tarjan(u);
    rep(i,1,sc) indg[i] = 0;//这个不包含在tarjan里面，是这题记录入度的
    rep(u,1,n){
        for(auto v:e[u]){
            if(scc[u]!=scc[v]) indg[scc[v]]++;
        }
    }
    int res = 0;
    rep(i,1,sc){
        if(indg[i]==0) res++;
    }
    cout<<res<<endl;
}
```

4.3 拓扑排序

```
vector<int> e[MAXN];
int n,m;
int indg[MAXN];

void solve(){
    for(int i=1;i<=n;i++) {indg[i]=0;e[i].clear();}
    for(int i=1,u,v;i<=m;i++){
        cin>>u>>v;
        e[u].push_back(v);
        indg[v]++;
    }
    queue<int> q;
    for(int i=1;i<=n;i++)
        if(indg[i]==0) q.push(i);//找到入度为0的点
    vector<int> res;//储存结果的向量
    int u;
    while(!q.empty()){//BFS
        u = q.front();
        q.pop();
        res.push_back(u);
        for(auto v:e[u]){
            if(--indg[v]==0)q.push(v);
            //删掉当前点u，点v的入度--
            //这一步的目的是找到新的入度为0的节点，推入队列Q
        }
    }
    if(res.size()==n){//如果这个图无环的话，说明可以进行拓扑排序
        for(int i=0;i<n;i++){//输出结果
            cout<<res[i]<<' ';
        }
        cout<<endl;
    }
}
```

4.4 最大流 Dinic

编号从 0 开始可用，复杂度 $O(V^2E)$

```
struct FLOW{
    struct edge{int to,w,nxt;};
    vector<edge> a; int head[N],cur[N];
    int n,s,t;
    queue<int> q; bool inque[N];
    int dep[N];
    void ae(int x,int y,int w){ // add edge
        a.push_back({y,w,head[x]});
        head[x]=a.size()-1;
    }
    bool bfs(){ // get dep[]
        fill(dep,dep+n,inf); dep[s]=0;
        copy(head,head+n,cur);
        q=queue<int>(); q.push(s);
        while(!q.empty()){
            int x=q.front(); q.pop(); inque[x]=0;
            for(int i=head[x];i!=-1;i=a[i].nxt){
                int p=a[i].to;
                if(dep[p]>dep[x]+1 && a[i].w){
                    dep[p]=dep[x]+1;
                    if(inque[p]==0){
                        inque[p]=1;
                        q.push(p);
                    }
                }
            }
        }
        return dep[t]!=inf;
    }
    int dfs(int x,int flow){ // extend
        int now,ans=0;
        if(x==t)return flow;
        for(int &i=cur[x];i!=-1;i=a[i].nxt){
            int p=a[i].to;
            if(a[i].w && dep[p]==dep[x]+1)
            if((now=dfs(p,min(flow,a[i].w)))){
                a[i].w-=now;
                a[i^1].w+=now;
                ans+=now,flow-=now;
                if(flow==0)break;
            }
        }
        return ans;
    }
    void init(int _n){
        n=_n+1; a.clear();
        fill(head,head+n,-1);
        fill(inque,inque+n,0);
    }
    int solve(int _s,int _t){ // return max flow
        s=_s,t=_t;
        int ans=0;
```

```
        while(bfs())ans+=dfs(s,inf);
        return ans;
    }
}flow;
void add(int x,int y,int w){flow.ae(x,y,w),flow.ae(y,x,0);}
// 先flow.init(n)，再add添边，最后flow.solve(s,t)
```

4.5 费用流

编号从 0 开始可用，复杂度 $O(VE^2)$

```
struct FLOW{
    struct edge{int to,w,cost,nxt;};
    vector<edge> a; int head[N];
    int n,s,t,totcost;
    deque<int> q;
    bool inque[N];
    int dis[N];
    struct{int to,e;}pre[N];
    void ae(int x,int y,int w,int cost){
        a.push_back((edge){y,w,cost,head[x]});
        head[x]=a.size()-1;
    }
    bool spfa(){
        fill(dis,dis+n,inf); dis[s]=0;
        q.assign(1,s);
        while(!q.empty()){
            int x=q.front(); q.pop_front();
            inque[x]=0;
            for(int i=head[x];i!=-1;i=a[i].nxt){
                int p=a[i].to;
                if(dis[p]>dis[x]+a[i].cost && a[i].w){
                    dis[p]=dis[x]+a[i].cost;
                    pre[p]={x,i};
                    if(inque[p]==0){
                        inque[p]=1;
                        if(!q.empty()
                        && dis[q.front()]<=dis[p])
                            q.push_back(p);
                        else q.push_front(p);
                    }
                }
            }
        }
        return dis[t]!=inf;
    }
    void init(int _n){
        n=_n+1;
        a.clear();
        fill(head,head+n,-1);
        fill(inque,inque+n,0);
    }
    int solve(int _s,int _t){ // 返回最大流，费用存totcost里
        s=_s,t=_t;
        int ans=0;
```

```
        totcost=0;
        while(spfa()){
            int fl=inf;
            for(int i=t;i!=s;i=pre[i].to)
                fl=min(fl,a[pre[i].e].w);
            for(int i=t;i!=s;i=pre[i].to){
                a[pre[i].e].w-=fl;
                a[pre[i].e^1].w+=fl;
            }
            totcost+=dis[t]*fl;
            ans+=fl;
        }
        return ans;
    }
}flow;
void add(int x,int y,int w,int cost){
    flow.ae(x,y,w,cost),flow.ae(y,x,0,-cost);
}
// 先 flow.init(n)，再 add 添边，最后 flow.solve(s,t)
```

4.6 二分图最大匹配

用最大流求解，一边连源点流量 1，一遍连汇点流量 1，连上匹配边流量 1，最大流即为最大匹配数
时间复杂度趋近 $O(n\sqrt{n})$

4.7 最短路

4.7.1 SPFA

```
//前向星
struct Edge{
    int to,w,nxt;
}edges[MAXM];
int head[MAXN];
int n,m,s;
int cnt;
inline void add_edge(int u,int v,int w){
    edges[cnt].to = v;
    edges[cnt].w = w;
    edges[cnt].nxt = head[u];
    head[u] = cnt++;
}
//SPFA
queue<int> q;
ll dis[MAXN];//每个点当前最小距离
bool vis[MAXN];//记录队列中是否有该点
inline void init(){//初始化前向星和SPFA的dis数组
    rep(i,1,n) dis[i]=LINF,head[i]=-1;
    cnt = 0;
}
inline void solve(){
    cin>>n>>m>>s;
    init();//记得要初始化
    int u,v,w;
```

```
    rep(i,1,m){
        cin>>u>>v>>w;
        add_edge(u,v,w);
    }
    dis[s] = 0;
    q.push(s);
    int now;
    while(!q.empty()){
        now = q.front();
        q.pop();
        vis[now] = 0;//vis表示队中有无该点
        //每次取出队首,并检查关联点是否可以进行松弛操作
        for(int i=head[now];i!=-1;i=edges[i].nxt){
            v = edges[i].to;
            if(dis[v]>dis[now]+edges[i].w){//松弛操作
                dis[v] = dis[now]+edges[i].w;
                if(!vis[v]){//如果队列中没有点v,则将v入队
                    vis[v] = 1;
                    q.push(v);
                }
            }
        }
    }
    rep(i,1,n) cout<<dis[i]<<' ';
    //我这里还没有对到达不了的特殊情况进行处理
    cout<<endl;
}
```

4.7.2 dijkstra

```
vector<pii> G[N];
int dis[N];
bool vis[N];
void dijkstra(int n,int pos){
    rep(i,1,n) dis[i]=inf,vis[i]=0;
    priority_queue<pii,vector<pii>,greater<pii>> pq;
    pq.push({0,pos});
    dis[pos]=0;
    while(!pq.empty()){
        pii top=pq.top(); pq.pop();
        int u=top.sc,d=top.fr;
        if(vis[u]) continue;
        for(auto e:G[u]){
            int v=e.fr;
            if(dis[v]>d+e.sc) {
                dis[v]=d+e.sc;
                pq.push({dis[v],v});
            }
        }
    }
}
```

4.8 2-Sat

```
//
struct TwoSat{
   //适用于存在多个二选一抉择(互斥)，且不同编号抉择间存在互斥关系的问题
   //考虑到不会撤回操作 复杂度O(N)
   //编号从0开始
   int n;
   vector<int> G[2*N];
   int S[2*N],c=0;
   bool mark[2*N],ok;
   void init(int n){
      this->n=n,ok=1;
      rep(i,0,n*2) mark[i]=0;
      rep(i,0,n*2) G[i].clear();
   }
   bool dfs(int u){
      if(mark[u^1]) return false;
      if(mark[u]) return true;
      S[++c]=u;
      mark[u]=1;
      for(auto v:G[u]){
         if(!dfs(v)) return false;
      }
      return true;
   }
   //x,y代表抉择编号,val代表选0/1
   void add(int x,int xval,int y,int yval){
      x=x*2+xval,y=y*2+yval;
      G[x^1].push_back(y);
      G[y^1].push_back(x);
   }
   void solve(){
      rep(i,0,n-1){
         c=0;
         if(!dfs(i*2)){
            while(c) mark[S[c--]]=0;
            if(!dfs(i*2+1)) ok=0;
         }
      }
   }
}solver;
```

4.9 最大团

```
int mp[60][60];
int n;
int mx=0;//ans
vector<int> tmp;
bool check(int now){
   bool fl=true;
   for(int i=0;i<tmp.size();i++){
      if(mp[tmp[i]][now]!=1||mp[now][tmp[i]]!=1){
         fl=false ;
```

```
            break;
        }
    }
    return fl;
}
void dfs(int step){
    if(step==n+1){
        if(tmp.size()>mx){
            mx=tmp.size();
        }
        return;
    }
    if(tmp.size()+n-step+1<=mx){
        return;
    }
    if(check(step)){
        tmp.push_back(step);
        dfs(step+1);
        tmp.pop_back();
    }
        dfs(step+1);
}
```

4.10 稳定婚姻

```
//构造一种匹配方式，使得不存在一组未被选中且双方好感更高的匹配
int B[N][N],G[N][N];//x对x的好感
vector<int> vec[N];
int pos[N],px[N],py[N],cur;
queue<int> que;
bool cmp(const int x,const int y){
    return B[cur][x]>B[cur][y];
}
void stable_marriage(int n){
    memset(pos,0,sizeof(pos));
    memset(py,0,sizeof(py));
    rep(i,1,n) vec[i].clear();
    rep(i,1,n) rep(j,1,n) vec[i].push_back(j);
    rep(i,1,n) {
        cur=i;
        sort(vec[i].begin(),vec[i].end(),cmp);
    }
    rep(i,1,n) que.push(i);
    while(!que.empty()){
        queue<int> tq;
        while(!que.empty()){
            int b=que.front(); que.pop();
            int p=pos[b]++,g=vec[b][p];
            if(!py[g]||G[g][b]>G[g][py[g]]){
                if(py[g]) tq.push(py[g]);
                py[g]=b,px[b]=g;
            }
            else tq.push(b);
        }
```

```
        que=tq;
    }
}
```

4.11 点双连通分量

```
vector<int> G[N],bcc[N];
int pre[N]={0},low[N];
int cur,bcnt,bccno[N]; //=0;
bool iscut[N];
stack<pii> S;
void dfs(int u,int fa){
   low[u]=pre[u]=++cur;
   int child=0;
   for(auto v:G[u]){
      if(v==fa) continue;
      if(pre[v]&&pre[v]<pre[u]) {
         //S.push({u,v});
         low[u]=min(low[u],pre[v]);
      }
      else if(!pre[v]){
         S.push({u,v});
         child++;
         dfs(v,u);
         low[u]=min(low[u],low[v]);
         if(low[v]>=pre[u]){
            iscut[u]=1;
            bcnt++; bcc[bcnt].clear();
            while(1){
               pii x=S.top(); S.pop();
               if(bccno[x.fr]!=bcnt){
                  bcc[bcnt].push_back(x.fr);
                  bccno[x.fr]=bcnt;
               }
               if(bccno[x.sc]!=bcnt){
                  bcc[bcnt].push_back(x.sc);
                     bccno[x.sc]=bcnt;
               }
               if(x.fr==u&&x.sc==v) break;
            }
         }
      }
   }
   if(fa==-1&&child==1) iscut[u]=0;
```

4.12 强连通分量

```
vector<int> G[N];
int sccno[N],dfn[N],low[N],scnt,tot;
stack<int> S;
void tarjan(int u){
   dfn[u]=low[u]=++tot;
   S.push(u);
```

```
    for(auto v:G[u]){
        if(sccno[v]) continue;
        if(!dfn[v]){
            tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else low[u]=min(low[u],dfn[v]); //low[v] is also ok.
    }
    if(low[u]==dfn[u]){
        scnt++;
        int top=-1;
        while(top!=u){
            top=S.top(); S.pop();
            sccno[top]=scnt;
        }
    }
}
```

4.13 边双/割边

```
//只需要把scc改成不能跳fa即可
vector<int> G[N];
int dccno[N],dfn[N],low[N],dcnt,tot;
vector<pii> bridge;
stack<int> S;
void tarjan(int u,int fa){
    dfn[u]=low[u]=++tot;
    S.push(u);
    for(auto v:G[u]){
        if(dccno[v]||v==fa) continue;
        if(!dfn[v]){
            tarjan(v,u);
            low[u]=min(low[u],low[v]);
        }
        else low[u]=min(low[u],dfn[v]); //low[v] is also ok.
    }
    if(low[u]==dfn[u]){
        if(fa) bridge.push_back({u,fa});
        dcnt++;
        int top=-1;
        while(top!=u){
            top=S.top(); S.pop();
            dccno[top]=dcnt;
        }
    }
}
```

4.14 最小树形图

```
struct Edge{int u,v,w,b;};
vector<Edge> eset,E;
int fa[N],minw[N],id[N],top[N];
ll solve(int n,int rt){
```

```
    ll res=0;
    while(1){
        int cnt=0;
        rep(i,1,n) minw[i]=inf,id[i]=top[i]=0;
        for(auto &e:eset){
            if(e.u!=e.v&&e.w<minw[e.v]){
                minw[e.v]=e.w;
                fa[e.v]=e.u;
            }
        }
        minw[rt]=0;
        rep(i,1,n){
            if(minw[i]==inf) return -1;
            res+=minw[i];
            for(int u=i;!id[u]&&u!=rt;u=fa[u]){
                if(top[u]==i){
                    id[u]=++cnt;
                    for(int v=fa[u];v!=u;v=fa[v]){
                        id[v]=cnt;
                    }
                    break;
                }
                else top[u]=i;
            }
        }
        if(!cnt) return res;
        rep(i,1,n) if(!id[i]) id[i]=++cnt;
        for(auto &e:eset){
            e.w-=minw[e.v];
            e.u=id[e.u],e.v=id[e.v];
        }
        n=cnt;
        rt=id[rt];
```

4.15 KM

```
int n;
ll G[N][N];
int px[N],py[N],vx[N],vy[N],pre[N];
ll lx[N],ly[N],slack[N],d;
queue<int> que;
void upd(int v){
    int t;
    while(v){
        t=px[pre[v]];
        py[v]=pre[v];
        px[pre[v]]=v;
        v=t;
    }
}
void bfs(int x){
    rep(i,1,n) vy[i]=vx[i]=0,slack[i]=inf;
    while(!que.empty()) que.pop();
    que.push(x);
```

```
    while(1){
        while(!que.empty()){
            int u=que.front(); que.pop();
            vx[u]=1;
            rep(v,1,n) if(!vy[v]){
                if(lx[u]+ly[v]-G[u][v]<slack[v]){
                    slack[v]=lx[u]+ly[v]-G[u][v];
                    pre[v]=u;
                    if(!slack[v]){
                        vy[v]=1;
                        if(!py[v]) {upd(v); return;}
                        else que.push(py[v]);
                    }
                }
            }
        }
        d=inf;
        rep(i,1,n) if(slack[i]) d=min(d,slack[i]);
        rep(i,1,n) {
            if(vx[i]) lx[i]-=d;
            if(vy[i]) ly[i]+=d;
            else slack[i]-=d;
        }
        rep(i,1,n){
            if(!vy[i]&&!slack[i]) {
                vy[i]=1;
                if(!py[i]) {upd(i); return;}
                else que.push(py[i]);
            }
        }
    }
}
ll KM(){
    ll res=0;
    fill(lx+1,lx+n+1,-inf); memset(ly,0,sizeof(ly));
    memset(px,0,sizeof(px)); memset(py,0,sizeof(py));
    memset(pre,0,sizeof(pre));
    rep(i,1,n) rep(j,1,n) lx[i]=max(lx[i],G[i][j]);
    rep(i,1,n) bfs(i);
    rep(i,1,n) res+=G[i][px[i]];
    return res;
}
```

4.16 差分约束

```
//介绍WIP
vector<pair<int,ll>> G[N];
bool inque[N];
int incnt[N];
ll dis[N];
bool bellman_ford(ll x){
    queue<int> que;
    rep(i,1,n) {
        dis[i]=0,inque[i]=1,incnt[i]=0;
```

```
        que.push(i);
    }
    while(!que.empty()){
        int u=que.front(); que.pop();
        inque[u]=0;
        for(auto e:G[u]){
            int v=e.fr;
            //cout<<dis[u]<<" "<<e.sc<<" "<<dis[v]<<endl;
            if(dis[u]+e.sc<dis[v]){
                dis[v]=dis[u]+e.sc;
                if(!inque[v]){
                    inque[v]=1;
                    que.push(v);
                    if(++incnt[v]>n) return true;
                }
            }
        }
    }
    return false;
}
```

4.17 一般图最大权匹配带花树 (hjt)

```
int n;
vector<int> G[N];
struct DSU { // join: d[x] = d[y], query: d[x] == d[y]
    int a[N];
    void init(int n) { iota(a, a + n + 1, 0); }
    int fa(int x) { return a[x] == x ? x : a[x] = fa(a[x]); }
    int &operator[](int x) { return a[fa(x)]; }
} d;
deque<int> q;
int mch[N],vis[N],dfn[N],fa[N],dcnt=0;
int lca(int x,int y){
    dcnt++;
    while(1){
        if(x==0)swap(x,y); x=d[x];
        if(dfn[x]==dcnt)return x;
        else dfn[x]=dcnt,x=fa[mch[x]];
    }
}
void shrink(int x,int y,int p){
    while(d[x]!=p){
        fa[x]=y; y=mch[x];
        if(vis[y]==2)vis[y]=1,q.push_back(y);
        if(d[x]==x)d[x]=p;
        if(d[y]==y)d[y]=p;
        x=fa[y];
    }
}
bool match(int s){
    d.init(n); fill(fa,fa+n+1,0);
    fill(vis,vis+n+1,0); vis[s]=1;
    q.assign(1,s);
```

```
    while(!q.empty()){
        int x=q.front(); q.pop_front();
        for(auto p:G[x]){
            if(d[x]==d[p] || vis[p]==2)continue;
            if(!vis[p]){
                vis[p]=2; fa[p]=x;
                if(!mch[p]){
                    for(int now=p,last,tmp;now;now=last){
                        last=mch[tmp=fa[now]];
                        mch[now]=tmp,mch[tmp]=now;
                    }
                    return 1;
                }
                vis[mch[p]]=1; q.push_back(mch[p]);
            }
            else if(vis[p]==1){
                int l=lca(x,p);
                shrink(x,p,l);
                shrink(p,x,l);
            }
        }
    }
    return 0;
}
```

5 计算几何

5.1 基础点和线

```
struct point{
    double x,y;
    point(double _x=0,double _y=0){ x=_x,y=_y; }
    point operator+ (const point& a) const{ return point(x+a.x,y+a.y); }
    point operator- (const point& a) const{ return point(x-a.x,y-a.y); }
    point operator* (double a) const{ return point(x*a,y*a); }
}; // 点
struct line{
    point s,e;
    line(point a,point b){ s=a,e=b; }
    line(){}
}; // 线
double getdis(point a, point b){
    double xx=a.x-b.x,yy=a.y-b.y;
    return sqrt(xx*xx+yy*yy);
} // 两点距离
double multi(point a,point b,point c){
    double xa,ya,xb,yb;
    xa=b.x-a.x; ya=b.y-a.y;
    xb=c.x-b.x; yb=c.y-b.y;
    return xa*xb+ya*yb;
} // 点乘
double cross(point a,point b,point c){
    double xa,ya,xb,yb;
```

```
    xa=b.x-a.x; ya=b.y-a.y;
    xb=c.x-a.x; yb=c.y-a.y;
    return xa*yb-xb*ya;
} // 叉乘
int judgec(line a,line b){
    if (max(a.s.x,a.e.x)>=min(b.s.x,b.e.x) &&
        max(a.s.y,a.e.y)>=min(b.s.y,b.e.y) &&
        max(b.s.x,b.e.x)>=min(a.s.x,a.e.x) &&
        max(b.s.y,b.e.y)>=min(a.s.y,a.e.y) &&
        cross(a.s,b.s,b.e)*cross(a.e,b.s,b.e)<=0
        && cross(b.s,a.s,a.e)*cross(b.e,a.s,a.e)<=0
        ) return 1;
    else return 0;
} // 判断线段是否相交
point getpoi(point a,point b,point c,point d){
    double u=cross(a,b,c),v=cross(b,a,d);
    return point((c.x*v+d.x*u)/(u+v),(c.y*v+d.y*u)/(u+v));
} // 求交点
```

5.2 三点求圆心

```
struct point{
    double x;
    double y;
};

point cal(point a,point b,point c){
    double x1 = a.x;double y1 = a.y;
    double x2 = b.x;double y2 = b.y;
    double x3 = c.x; double y3 = c.y;
    double a1 = 2*(x2-x1); double a2 = 2*(x3-x2);
    double b1 = 2*(y2-y1); double b2 = 2*(y3-y2);
    double c1 = x2*x2 + y2*y2 - x1*x1 - y1*y1;
    double c2 = x3*x3 + y3*y3 - x2*x2 - y2*y2;
    double rx = (c1*b2-c2*b1)/(a1*b2-a2*b1);
    double ry = (c2*a1-c1*a2)/(a1*b2-a2*b1);
    return point{rx,ry};
}
```

5.3 拉格朗日插值

```
namespace polysum {
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
const int D = 1010000; ///可能需要用到的最高次
LL a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D];
LL powmod(LL a, LL b) {
    LL res = 1;
    a %= mod;
    assert(b >= 0);

    for (; b; b >>= 1) {
        if (b & 1)
```

```
            res = res * a % mod;

        a = a * a % mod;
    }

    return res;
}

///函数用途：给出数列的（d+1）项，其中d为最高次方项
///求出数列的第n项，数组下标从0开始
LL calcn(int d, LL *a, LL n) { /// a[0].. a[d] a[n]
    if (n <= d)
        return a[n];

    p1[0] = p2[0] = 1;
    rep(i, 0, d + 1) {
        LL t = (n - i + mod) % mod;
        p1[i + 1] = p1[i] * t % mod;
    }
    rep(i, 0, d + 1) {
        LL t = (n - d + i + mod) % mod;
        p2[i + 1] = p2[i] * t % mod;
    }
    LL ans = 0;
    rep(i, 0, d + 1) {
        LL t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;

        if ((d - i) & 1)
            ans = (ans - t + mod) % mod;
        else
            ans = (ans + t) % mod;
    }
    return ans;
}
void init(int M) {///用到的最高次
    f[0] = f[1] = g[0] = g[1] = 1;
    rep(i, 2, M + 5) f[i] = f[i - 1] * i % mod;
    g[M + 4] = powmod(f[M + 4], mod - 2);
    per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod; ///费马小定理筛逆元
}

///函数用途：给出数列的（m+1）项，其中m为最高次方
///求出数列的前（n-1）项的和（从第0项开始）
LL polysum(LL m, LL *a, LL n) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]
    for (int i = 0; i <= m; i++)
        b[i] = a[i];

    ///前n项和，其最高次幂加1
    b[m + 1] = calcn(m, b, m + 1);
    rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
    return calcn(m + 1, b, n - 1);
}
LL qpolysum(LL R, LL n, LL *a, LL m) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
    if (R == 1)
        return polysum(n, a, m);
```

```
68
69      a[m + 1] = calcn(m, a, m + 1);
70      LL r = powmod(R, mod - 2), p3 = 0, p4 = 0, c, ans;
71      h[0][0] = 0;
72      h[0][1] = 1;
73      rep(i, 1, m + 2) {
74          h[i][0] = (h[i - 1][0] + a[i - 1]) * r % mod;
75          h[i][1] = h[i - 1][1] * r % mod;
76      }
77      rep(i, 0, m + 2) {
78          LL t = g[i] * g[m + 1 - i] % mod;
79
80          if (i & 1)
81              p3 = ((p3 - h[i][0] * t) % mod + mod) % mod, p4 = ((p4 - h[i][1] * t) % mod + mod) % mod;
82          else
83              p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
84      }
85      c = powmod(p4, mod - 2) * (mod - p3) % mod;
86      rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
87      rep(i, 0, m + 2) C[i] = h[i][0];
88      ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
89
90      if (ans < 0)
91          ans += mod;
92
93      return ans;
94  }
95  }
```

6 数据结构

6.1 树背包

```
1  int dp[MAXN][MAXN],val[MAXN];
2  void dfs(int now){
3      dp[now][1] = val[now];//肯定要选定自己
4      for(int i = head[now];i!=-1;i=edges[i].nxt){
5          int v = edges[i].v;
6          dfs(v);
7          //分组背包
8          repb(j,m,1){//当前u背包容量
9              rep(k,0,j-1){//从v儿子里拿几个
10                 dp[now][j]=max(dp[now][j],dp[now][j-k]+dp[v][k]);
11                 //状态转移
12             }
13         }
14     }
15 }
```

考虑根是否连边

```
1  void dfs(int pos,int pre){
2      siz[pos] = 1;//统计子树大小，子树最多有 siz[pos]/2 条选边
3      f[pos][0][0] = 1;
```

```
    for(auto to:e[pos]){
        if(to==pre) continue;
        dfs(to,pos);
        repb(i,siz[pos]/2,0){//父亲选i个
            rep(j,0,siz[to]/2){//儿子选j个
                if(j!=0){//不在当前儿子与根连
                    f[pos][i+j][0] += f[pos][i][0]*(f[to][j][1]+f[to][j][0]%med)%med;
                    f[pos][i+j][0] %= med;
                    f[pos][i+j][1] += f[pos][i][1]*(f[to][j][1]+f[to][j][0]%med)%med;
                    f[pos][i+j][1] %= med;
                    //不能让ij都等于0，不然f[pos][i+j][1]会丢失的（乘f[pos][j][0])变成0了
                }
                //考虑pos和to连边
                f[pos][i+j+1][1] += (f[pos][i][0]*f[to][j][0])%med;
                f[pos][i+j+1][1] %= med;
            }
        }
        siz[pos] += siz[to];
    }
}
```

6.2 树上启发式合并

通过保留重儿子贡献降低复杂度

```
const int MAXN = 1e5+5;
const int MAXM = 1e5+5;
int sz[MAXN],son[MAXN];//子树大小和重儿子
//int ans[MAXN];//存最多的颜色和编号和,这题没必要
int c[MAXN];//每个点的颜色
int cnt[MAXM];//存每种颜色的点数
ll bhh[MAXN],bsum;//题目要求颜色的编号和
int maxx;//临时变量,用来找最大颜色数量的
int n;
vector<int> e[MAXN];
void init(int now,int pre){//预处理找重儿子
    sz[now] = 1;son[now] = -1;
    for(auto v:e[now]){
        if(v==pre) continue;
        init(v,now);
        sz[now]+=sz[v];
        if(son[now]==-1||sz[v]>sz[son[now]]) son[now]=v;
    }
}
void dfs1(int now,int pre,int val,int rt){//rt是dfs里面当前在统计的子树根
//val=1的时候加上子节点的贡献,而-1时是dfs里面opt=1(代表轻儿子)时删去贡献
    cnt[c[now]]+=val;
    if(val==1){
        if(cnt[c[now]]>maxx) maxx=cnt[c[now]],bsum=c[now];
        else if(cnt[c[now]]==maxx) bsum+=c[now];
    }
    //遍历子节点的时候看到什么颜色就检查这个颜色
    for(auto v:e[now]) if(v!=pre)dfs1(v,now,val,rt);
}
void dfs(int now,int pre,int opt){//opt为1代表要清空
```

```
    for(auto v:e[now]){
        if(v!=son[now]&&v!=pre) dfs(v,now,1);//先轻儿子,需要删去
    }
    if(son[now]!=-1) dfs(son[now],now,0);//处理重儿子
    //此时计算轻儿子的贡献,重儿子的在cnt里面祖传上来了
    for(auto v:e[now]){
        if(v!=pre&&v!=son[now]) dfs1(v,now,1,now);//统计轻儿子上的结果
    }
    cnt[c[now]]++;//也要加上当前节点
    if(cnt[c[now]]>maxx) maxx=cnt[c[now]],bsum=c[now];
    //这里我一开始用的cnt[c[now]]>cnt[ans[now]],但是c[now]和ans[rt]相同时出问题
    else if(cnt[c[now]]==maxx) bsum+=c[now];
    bhh[now] = bsum;//记录结果
    //减去轻儿子的贡献
    if(opt){
        cnt[c[now]]--;
        for(auto v:e[now])
            if(v!=pre) dfs1(v,now,-1,now);
        maxx = bsum = 0;//删去轻儿子这颗子树对bsum和maxx的贡献
    }
}
int rt = 1;//题目默认1为根节点
int main(){
    cin>>n;
    rep(i,1,n) cin>>c[i];
    int u,v;
    rep(i,1,n-1){
        cin>>u>>v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    init(rt,-1);
    dfs(rt,-1,1);
    rep(i,1,n){
        cout<<bhh[i];
        if(i!=n) cout<<' ';
    }
    cout<<endl;
}
//没有修改只有查询的树形问题
//且询问以所有节点为根子树（即对每个节点为根的子树都要询问）。
//洛谷CF600E Lomsat gelral
//https://www.luogu.com.cn/problem/CF600E
```

6.3 ST 表求 RMQ

$O(nlog_n)$ 预处理，$O(1)$ 查询

```
#define log(x) (31-__builtin_clz(x))
const int MAXN = 1e5+10;
const int LOGN = log(MAXN)/log(2)+5;
int M[MAXN][LOGN];
int a[MAXN];
int z,m,n;
void init(){//初始化，复杂度O(nlogn)
```

```
    for(int i=1;i<=n;i++) M[i][0]=i;//长度为1的区间最值是自己
    for(int j=1;j<=LOGN;j++){
        for(int i=1;i<=n-(1<<j)+1;i++){
            if(a[M[i][j-1]]<a[M[i+(1<<(j-1))][j-1]]) M[i][j] = M[i][j-1];//这里以最小值为例
            else M[i][j] = M[i+(1<<j-1)][j-1];
        }
    }
}
int query(int l,int r){
    int k = log(r-l+1)/log(2);//向下取整
    if(a[M[l][k]]<a[M[r-(1<<k)+1][k]]) return M[l][k];
    else return M[r-(1<<k)+1][k];
}
```

6.4 扫描线

扫描线是离散化后，使用类似权值线段树来维护每个截面上的线段长度。
通过把二维平面上的四边形拆分成入边和出边两段，在遇到边的时候对对应的区间进行区间加/减即可。
每个节点上需要维护被完全覆盖的次数和实际长度。

```
#define ls (x<<1)
#define rs (x<<1|1)//这种方法感觉还挺好的

int cansel_sync=(ios::sync_with_stdio(0),cin.tie(0),0);
const int MAXN = 2e5+5;//这里要开n的两倍
//线结构体
struct Line{
    ll l,r,h;
    int qz;//记录位置和权值
    bool operator < (Line &rhs){
        return h < rhs.h;
    }
}line[MAXN];
int n;
ll x1,y1,x2,y2;
ll X[MAXN];
//线段树
struct Segt{
    int l,r;//是X的下标,即离散化后的
    int sum;//sum是被完全覆盖的次数
    ll len;//len是区间内被盖住的长度
    //因为每次查询都是查询根节点,所以这边不需要懒惰标记
}t[MAXN<<3];//一个边有两个点,所以这里要开8倍
void build(int x,int l,int r){
    t[x].l = l;t[x].r = r;
    t[x].len = t[x].sum = 0;
    if(l==r) return;//到了叶子节点
    int mid = (l+r)>>1;
    build(ls,l,mid);
    build(rs,mid+1,r);
}
void push_up(int x){
    int l = t[x].l,r = t[x].r;
    if(t[x].sum) t[x].len = X[r+1]-X[l];//x的区间是X[l]到X[r+1]-1
    else t[x].len = t[ls].len + t[rs].len;//合并儿子的信息
```

```
}
void update(int x,int L,int R,int v){//这里的LR存的是实际值
    //这里如果是线段L,R,线段树上是L到R-1的部分维护
    int l = t[x].l,r = t[x].r;
    if(X[r+1]<=L||R<=X[l]) return;//加等于,不然会搞到无辜的线
    if(L<=X[l]&&X[r+1]<=R){
        t[x].sum += v;//修改覆盖次数
        push_up(x);
        return;
    }
    update(ls,L,R,v);
    update(rs,L,R,v);
    push_up(x);
}
int main(){
    cin>>n;
    rep(i,1,n){
        cin>>x1>>y1>>x2>>y2;
        X[2*i-1] = x1,X[2*i] = x2;//一会儿离散化要用的,这里存实际值
        line[2*i-1] = Line{x1,x2,y1,1};//开始的线
        line[2*i] = Line{x1,x2,y2,-1};//结束的线
    }
    n<<=1;//line的数量是四边形数量的2倍
    sort(line+1,line+1+n);
    sort(X+1,X+1+n);
    int tot = unique(X+1,X+n+1)-(X+1);//去除重复相邻元素,并且tot记录总数
    build(1,1,tot-1);//为什么是tot-1?
    //因为线段树只需要维护X[1]到X[tot]-1这一段的,实际长度是向右贴的
    ll res = 0;
    rep(i,1,n-1){//每次高度是line[i+1].h-line[i].h,所以是到n-1就行
        update(1,line[i].l,line[i].r,line[i].qz);//扫描线加入线段树
        res += t[1].len*(line[i+1].h-line[i].h);
    }
    cout<<res<<endl;
}
```

6.5 树链剖分

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

ll ans;
int da[MAXN];//记录dfn序上的各点数值用来初始化线段树
int n,m,rt,p;//节点数，询问数，根，模数

struct tree{
    int sum;
    int lazy;
};

struct St{//线段树
    tree t[MAXN<<2];
```

```
17  void pushup(int pos){
18      t[pos].sum=(t[pos<<1].sum+t[pos<<1|1].sum) %p;
19      return;
20  }
21  void pushdown(int l,int r,int pos){
22      if(!t[pos].lazy) return;
23      int mid = (l+r)>>1;
24      t[pos<<1].sum += t[pos].lazy*(mid-l+1);
25      t[pos<<1].sum %= p;//取模
26      t[pos<<1|1].sum += t[pos].lazy*(r-(mid+1)+1);
27      t[pos<<1|1].sum %= p;//取模
28      t[pos<<1].lazy += t[pos].lazy;
29      t[pos<<1].lazy %= p;//取模
30      t[pos<<1|1].lazy += t[pos].lazy;
31      t[pos<<1|1].lazy %= p;//取模
32      t[pos].lazy = 0;
33  }
34  void build(int l,int r,int pos){
35      t[pos].sum = t[pos].lazy = 0;
36      if(l==r){
37          t[pos].sum = da[l];
38          return;
39      }
40      int mid = (l+r)>>1;
41      build(l,mid,pos<<1);
42      build(mid+1,r,pos<<1|1);
43      pushup(pos);
44  }
45  void update(int L,int R,int l,int r,int pos,int v){
46      if(L<=l&&r<=R){
47          t[pos].sum += v*(r-l+1);
48          t[pos].lazy += v;
49          t[pos].lazy %= p;//取模
50          return;
51      }
52      if(r<L||l>R) return;
53      pushdown(l,r,pos);
54      int mid = (l+r)>>1;
55      update(L,R,l,mid,pos<<1,v);
56      update(L,R,mid+1,r,pos<<1|1,v);
57      pushup(pos);
58  }
59  void query(int L,int R,int l,int r,int pos){
60      if(L<=l&&r<=R){
61          ans += t[pos].sum;
62          ans%=p;//取模
63          return;
64      }
65      if(r<L||R<l) return;
66      pushdown(l,r,pos);
67      int mid = (l+r)>>1;
68      query(L,R,l,mid,pos<<1);
69      query(L,R,mid+1,r,pos<<1|1);
70      return;
71  }
```

```
    //查询和修改，为了简化参数，我又写了两个
    ll tquery(int L,int R){
        ans = 0;
        query(L,R,1,n,1);
        return ans;
    }
    void tupdate(int L,int R,int v){
        update(L,R,1,n,1,v);
    }
};
//树结构
vector<int> e[MAXN];//记录边
int a[MAXN];//记录编号对应节点的初始数值

//树剖部分
St segt;
int si[MAXN],dep[MAXN],fa[MAXN],rem[MAXN],dfn[MAXN],top[MAXN];
int dfn_num;

void dfs1(int x,int faa){//预处理出fa,dep,si,rem
    int ma = 0;//用来x的重儿子，记录最大的size
    si[x] = 1;
    for(auto v:e[x]){
        if(v==faa) continue;//跳过父亲节点
        dep[v] = dep[x]+1;//更新儿子的dep
        dfs1(v,x);
        si[x] += si[v];//x的size加上当前儿子的size
        fa[v] = x;//标记v的父节点为x
        if(si[v]>ma){
            ma = si[v];
            rem[x] = v;//记录重儿子
        }
    }
}

void dfs2(int x,int faa){//预处理出dfn，top
    if(rem[faa]==x) top[x] = top[faa];//同一条重链同一个top
    else top[x] = x;//否则为重链头
    dfn[x] = ++dfn_num;//更新树剖序，同时下标自增
    da[dfn_num] = a[x];
    if(rem[x]) dfs2(rem[x],x);//优先遍历重儿子
    for(auto v:e[x]){
        if(v==faa) continue;
        if(v==rem[x]) continue;//重儿子之前已经遍历过了
        dfs2(v,x);
    }
}

inline ll cal(int L,int R){
    return segt.tquery(L,R);
}

void init(){//初始化
    dfn_num=0;
    dfs1(rt,0);
```

```
    dfs2(rt,0);
    segt.build(1,n,1);//这里通过da来初始化线段树
}

ll query(int x,int y){
    ll res=0;
    while(top[x]!=top[y]){//跳到同一条重链
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        //重链深度更大的点优先往上跳
        res += cal(dfn[top[x]],dfn[x]);//算上这条重链的
        res%=p;//取模
        x = fa[top[x]];//跳到重链头的父节点上
    }
    //跳出这个循环时，xy已经在同一个重链上了
    res += cal(min(dfn[x],dfn[y]),max(dfn[x],dfn[y]));
    res%=p;//取模
    //xy不确定顺序对不对，所以取minmax
    return res;
}

void update(int x,int y,int v){
    ll res=0;
    while(top[x]!=top[y]){//跳到同一条重链
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        //重链深度更大的点优先往上跳
        segt.tupdate(dfn[top[x]],dfn[x],v);//更新数值
        x = fa[top[x]];//跳到重链头的父节点上
    }
    //跳出这个循环时，xy已经在同一个重链上了
    segt.tupdate(min(dfn[x],dfn[y]),max(dfn[x],dfn[y]),v);
    //xy不确定顺序对不对，所以取minmax
}
int getlca(int u,int v){
    while(top[u]!=top[v]){
        if(dep[top[u]]<dep[top[v]])swap(u,v);
        u = fa[top[u]];
    }
    if(dep[u]>dep[v])swap(u,v);
    return u;
}
int main(){
    cin>>n>>m>>rt>>p;//节点数，操作数，根节点序号，模数
    for(int i=1;i<=n;i++){
        cin>>a[i];//记录初始数值
        a[i] = a[i]%p;
    }
    int x,y;
    for(int i=1;i<=n;i++) e[i].clear();
    for(int i=1;i<n;i++){
        cin>>x>>y;
        e[x].push_back(y);
        e[y].push_back(x);
    }
    init();
    int v;
```

```
    int typ;
    while(m--){
        cin>>typ;
        if(typ==1){//简单路径上修改
            cin>>x>>y>>v;
            update(x,y,v);
        }
        else if(typ==2){//简单路径上查询
            cin>>x>>y;
            cout<<query(x,y)<<endl;
        }
        else if(typ==3){//子树上修改
            cin>>x>>v;
            segt.tupdate(dfn[x],dfn[x]+si[x]-1,v);
        }
        else if(typ==4){//子树上查询
            cin>>x;
            cout<<segt.tquery(dfn[x],dfn[x]+si[x]-1)<<endl;
        }
    }
}
```

6.6 可持久化线段树

区间第 k 大

```
int n,m;
int date[200010];
int lsh[200010];
int root[200010];
int lk[200010];
map<int,int> rvs;
struct chair_man_tree{
    struct node{
        int v;
        int lson,rson;
    }tree[200010<<5];
    int tail=0;
    inline int addnode(int pos){
        tree[++tail]=tree[pos];
        return tail;
    }
    int build(int l,int r){
        int now=++tail;
          tree[now].v=0;
        if(l==r){
            return now;
        }
        int mid=(l+r)>>1;
        tree[now].lson=build(l,mid);
        tree[now].rson=build(mid+1,r);
        return now;
    }
    int update(int pos,int l,int r,int x){
        int now=addnode(pos);
```

```
        tree[now].v++;
        if(l<r){
            int mid=(l+r)>>1;
            if(x<=mid)tree[now].lson=update(tree[pos].lson,l,mid,x);
            else tree[now].rson=update(tree[pos].rson,mid+1,r,x);
        }
        return now;
    }
    int query(int u,int v,int l,int r,int k){
        if(l==r)
        return l;
        int x=tree[tree[v].lson].v-tree[tree[u].lson].v;
        int mid=(l+r)>>1;
        if(x>=k)
            return query(tree[u].lson,tree[v].lson,l,mid,k);
        else return query(tree[u].rson,tree[v].rson,mid+1,r,k-x);
    }
}a;
void work()
{
    rd(n),rd(m);
    for(int i=1;i<=n;i++)
    rd(date[i]),lsh[i]=date[i];
    sort(lsh+1,lsh+ n + 1);
    int cnt=0;
    cnt=unique(lsh+1,lsh+n+1)- lsh-1;
    for(int i=1;i<=cnt;i++)
        rvs[lsh[i]]=i;
    a.build(1,cnt);
    for(int i=1;i<=n;i++){
        root[i]=a.update(root[i-1],1,cnt,rvs[date[i]]);
    }
    int ll,rr,kk;
    for(int i=0;i<m;i++){
        rd(ll), rd(rr), rd(kk);
        printf("%d\n",lsh[a.query(root[ll-1],root[rr],1,cnt,kk)]);
    }
}
```

6.7 并查集系列

6.7.1 普通并查集

带路径压缩,$O(1)$ 复杂度

```
int fa[maxn];
int find(int x){if(fa[x]^x)return fa[x]=find(fa[x]);return x;}
void merge(int a,int b){fa[find(a)]=find(b);}
```

6.7.2 按秩合并并查集

```
int fa[maxn];
int dep[maxn];
int find(int x){int now=x; while(fa[now]^now)now=fa[now];return now;}
```

```
void merge(int a,int b){
    int l=find(a),r=find(b);
    if(l==r) return;
    if(dep[l]>dep[r])swap(l,r);
    fa[l]=r;
    dep[r]+=dep[l]==dep[r];
}
```

6.7.3 可持久化并查集

```
struct chair_man_tree{
    struct node{
        int lson,rson;
    }tree[maxn<<5];
    int tail=0;
    int tail2=0;
    int fa[maxn<<2];
    int depth[maxn<<2];
    inline int getnew(int pos){
        tree[++tail]=tree[pos];
        return tail;
    }
    int build(int l,int r){

        if(l==r){
            fa[++tail2]=l;
            depth[tail2]=1;
            return tail2;
        }
        int now=tail++;
        int mid=(l+r)>>1;
        tree[now].lson=build(l,mid);
        tree[now].rson=build(mid+1,r);
        return now;
    }
    int query(int pos,int l,int r,int qr){
        if(l==r)
            return pos;
        int mid=(l+r)>>1;
        if(qr<=mid)
            return query(tree[pos].lson,l,mid,qr);
        else return query(tree[pos].rson,mid+1,r,qr);
    }
    int update(int pos,int l,int r,int qr,int val){
        if(l==r){
            depth[++tail2]=depth[pos];
            fa[tail2]=val;
            return tail2;
        }
        int now=getnew(pos);
        int mid=(l+r)>>1;
        if(mid>=qr)
            tree[now].lson=update(tree[now].lson,l,mid,qr,val);
        else tree[now].rson=update(tree[now].rson,mid+1,r,qr,val);
```

```
        return now;
    }
    int add(int pos,int l,int r,int qr){
        if(l==r){
            depth[++tail2]=depth[pos]+1;
            fa[tail2]=fa[pos];
            return tail2;
        }
        int now=getnew(pos);
        int mid=(l+r)>>1;
        if(mid>=qr)
            tree[now].lson=add(tree[now].lson,l,mid,qr);
        else tree[now].rson=add(tree[now].rson,mid+1,r,qr);
        return now;
    }
    int getfa(int root,int qr){
        int t=fa[query(root,1,n,qr)];
        if(qr==t)
        return qr;
        else return getfa(root,t);
    }
}t;
```

6.7.4 可撤销并查集

```
struct DSU{
    int *rnk,*f,top;
    pii *stk;
    DSU(int n=N){
        rnk=new int[n],f=new int[n];
        stk=new pii[n];
    }
    void init(int n){
        rep(i,1,n) rnk[i]=0,f[i]=i;
        top=0;
    }
    int getf(int u){return f[u]==u?u:getf(f[u]);}
    void unite(int u,int v){
        u=getf(u),v=getf(v);
        if(u==v) return;
        if(rnk[u]>rnk[v]){
            stk[++top]={v,rnk[v]};
            f[v]=u;
        }
        else{
            stk[++top]={u,rnk[u]};
            f[u]=v;
            if(rnk[u]==rnk[v]) rnk[v]++;
        }
    }
    void undo(int pos){
        while(top>pos){
            int u=stk[top].fr,r=stk[top].sc;
            f[u]=u,rnk[u]=r;
```

```
        --top;
    }
  }
}dsu;
```

6.7.5 ETT 维护动态图连通性

待补

6.8 平衡树系列

6.8.1 fhq_treap

无旋 treap，可持久化，常数大

```
mt19937 rnd(514114);
struct fhq_treap{
  struct node{
    int l, r;
    int val, key;
    int size;
  } fhq[maxn];
  int cnt, root;
  inline int newnode(int val){
    fhq[++cnt].val = val;
    fhq[cnt].key = rnd();
    fhq[cnt].size = 1;
    fhq[cnt].l = fhq[cnt].r = 0;
    return cnt;
  }
  inline void pushup(int now){
  fhq[now].size = fhq[fhq[now].l].size + fhq[fhq[now].r].size + 1;
  }
  void split(int now, int val, int &x, int &y){
    if (!now){
      x = y = 0;
      return;
    }
    else if (fhq[now].val <= val){
    x = now;
    split(fhq[now].r, val, fhq[now].r, y);
    }
    else{
    y = now;
    split(fhq[now].l, val, x, fhq[now].l);
    }
  pushup(now);
  }
  int merge(int x, int y){
    if (!x || !y)
      return x + y;
    if (fhq[x].key > fhq[y].key){
      fhq[x].r = merge(fhq[x].r, y);
      pushup(x);
      return x;
```

```
    }else{
        fhq[y].l = merge(x, fhq[y].l);
        pushup(y);
        return y;
    }
}
inline void insert(int val){
    int x, y;
    split(root, val, x, y);
    root = merge(merge(x, newnode(val)), y);
}
inline void del(int val){
    int x, y, z;
    split(root, val - 1, x, y);
    split(y, val, y, z);
    y = merge(fhq[y].l, fhq[y].r);
    root = merge(merge(x, y), z);
}
inline int getrk(int num){
    int x, y;
    split(root, num - 1, x, y);
    int ans = fhq[x].size + 1;
    root = merge(x, y);
    return ans;
}
inline int getnum(int rank){
    int now=root;
    while(now)
    {
        if(fhq[fhq[now].l].size+1==rank)
          break;
        else if(fhq[fhq[now].l].size>=rank)
            now=fhq[now].l;
        else{
            rank-=fhq[fhq[now].l].size+1;
            now=fhq[now].r;
        }
    }
    return fhq[now].val;
}
inline int pre(int val){
    int x, y, ans;
    split(root, val - 1, x, y);
    int t = x;
    while (fhq[t].r)
        t = fhq[t].r;
    ans = fhq[t].val;
    root = merge(x, y);
    return ans;
}
inline int aft(int val){
    int x, y, ans;
    split(root, val, x, y);
    int t = y;
    while (fhq[t].l)
```

```
        t = fhq[t].l;
    ans = fhq[t].val;
    root = merge(x, y);
    return ans;
  }
} tree;
```

6.8.2 替罪羊树

```
struct node
{
    int l, r, val;
    int size, fact;
    bool exsit;
};
class tzy_tree
{
  public:
  double alpha=0.75;
  node tzy[1100000];
  int root=0,cnt=0;
  vector<int> tt;
  inline void newnode(int &now,int v){
    now=++cnt;
    tzy[now].val=v;
    tzy[now].l=tzy[now].r=0;
    tzy[now].size=tzy[now].fact=tzy[now].exsit=1;
  }
  inline bool imbalanced(int now){
    if(max(tzy[tzy[now].l].size,tzy[tzy[now].r].size)>tzy[now].size*alpha||tzy[now].size-tzy[now].fact>tzy
        [now].size*0.3)
      return true;
    return false;
  }
  void zhongxu(int now){
    if(!now)
      return;
    zhongxu(tzy[now].l);
    if(tzy[now].exsit)
      tt.push_back(now);
    zhongxu(tzy[now].r);
  }
  void lift(int l,int r,int &now){
    if(l==r){
      now=tt[l];
      tzy[now].l=tzy[now].r=0;
      tzy[now].fact=tzy[now].size=1;
      return;
    }
    int m=(l+r)>>1;
    while(l<m&&tzy[tt[m]].val==tzy[tt[m-1]].val)
      m--;
    now=tt[m];
    if(l<m) lift(l,m-1,tzy[now].l);
```

```
45      else tzy[now].l=0;
46      lift(m+1,r,tzy[now].r);
47      tzy[now].size=tzy[tzy[now].l].size+tzy[tzy[now].r].size+1;
48      tzy[now].fact=tzy[tzy[now].l].fact+tzy[tzy[now].r].fact+1;
49  }
50  void rebuild(int &now){
51      tt.clear();
52      zhongxu(now);
53      if(tt.empty()){
54          now=0;
55          return;
56      }
57      lift(0,tt.size()-1,now);
58  }
59  void update(int now,int end){
60      if(!now)return;
61      if(tzy[end].val<tzy[now].val)
62          update(tzy[now].l,end);
63      else update(tzy[now].r,end);
64      tzy[now].size=tzy[tzy[now].l].size+tzy[tzy[now].r].size+1;
65  }
66  void check(int &now,int end){
67      if(now==end)
68          return;
69      if(imbalanced(now)){
70          rebuild(now);
71          update(root,now);
72          return;
73      }
74      if(tzy[end].val<tzy[now].val)
75          check(tzy[now].l,end);
76      else
77          check(tzy[now].r,end);
78  }
79  inline void init(){
80      root=1;
81      cnt=0;
82  }
83  void insert(int &now,int val){
84
85      if(!now){
86          newnode(now,val);
87          check(root,now);
88          return;
89      }
90      tzy[now].fact++,tzy[now].size++;
91      if(val<tzy[now].val)
92          insert(tzy[now].l,val);
93      else
94          insert(tzy[now].r,val);
95  }
96  void erase(int now,int val){
97      if(tzy[now].exsit&&tzy[now].val==val){
98          tzy[now].exsit=false;
99          tzy[now].fact--;
```

```
            check(root,now);
            return;
        }
        tzy[now].fact--;
        if(val<tzy[now].val)
            erase(tzy[now].l,val);
        else erase(tzy[now].r,val);
    }
    int getrank(int val){
        int now=root,rk=1;
        while(now){
            if(val<=tzy[now].val)
                now=tzy[now].l;
            else rk+=tzy[now].exsit+tzy[tzy[now].l].fact,now=tzy[now].r;
        }
        return rk;
    }
    int getnum(int rk){
        int now=root;
        while(now){
            if(tzy[now].exsit&&tzy[tzy[now].l].fact+tzy[now].exsit==rk)
                break;
            else if(tzy[tzy[now].l].fact>=rk){
                now=tzy[now].l;
            }else{
                rk-=tzy[tzy[now].l].fact+tzy[now].exsit;
                now=tzy[now].r;
            }
        }
        return tzy[now].val;
    }
}tree;
```

6.8.3 splay

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

template<class T>inline void read(T &x){x=0;char o,f=1;while(o=getchar(),o<48)if(o==45)f=-f;do x=(x<<3)+(x
    <<1)+(o^48);while(o=getchar(),o>47);x*=f;}
//int cansel_sync=(ios::sync_with_stdio(0),cin.tie(0),0);
#define ll long long
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define repb(i,a,b) for(int i=(a);i>=b;i--)
#define INF 0x3f3f3f3f
#define cendl printf("\n")
ll gcd(ll a,ll b){ while(b^=a^=b^=a%=b); return a; }
//#define INF 0x7fffffff

const int MAXN = 5e5+5;
int root,tot,n;

struct Node{
    int ch[2];//左右儿子
```

```
    int val;//值
    int fa;//父节点
    int size;//子树大小
    int cnt;//计数
}t[MAXN];

inline void pushup(int x){//维护节点的size
    t[x].size = t[x].cnt;//自己重复的次数先累计
    if(t[x].ch[0]) t[x].size+=t[t[x].ch[0]].size;
    if(t[x].ch[1]) t[x].size+=t[t[x].ch[1]].size;
    //如果有儿子,把儿子的size加到自己
    //同线段树的pushup必须先处理儿子再处理父节点
}

void rotate(int x){//旋转操作
    int y=t[x].fa, z=t[y].fa;
    int k=(t[y].ch[1]==x);//x是y的左还是右儿子
    t[z].ch[t[z].ch[1]==y] = x;//x换到y原来在的位置
    t[x].fa = z;
    t[y].ch[k] = t[x].ch[k^1];//y顶下来x的儿子，放在x原来对y的位置
    t[t[x].ch[k^1]].fa = y;
    t[x].ch[k^1] = y;//y换到 x原来在y的 相对的 位置
    t[y].fa = x;
    pushup(y),pushup(x);//更新size,因为y是儿子,所以先y
}//这里的异或是用来调整0和1(左右儿子)

inline void splay(int x,int goal){//找到x并把x旋转为goal的儿子根节点
    while(t[x].fa!=goal){//x一直旋转到成为goal的儿子
        int y=t[x].fa,z=t[y].fa;
        if(z!=goal)//如果y不是根节点,分两类讨论
            (t[z].ch[0]==x)^(t[y].ch[0]==y)?rotate(x):rotate(y);
            //x和y分别是y和z的同一侧儿子,先转y再转x;不同则先转x转两次
        rotate(x);//无论三种情况中的哪一种都要最后转x
    }
    if(goal==0) root=x;//若goal是0,根节点更新为x
}

inline void find(int x){//查找x的位置,并旋转x到根节点,类似二分,O(logn)
    int u = root;
    if(!u) return;//树是空的情况
    while(t[u].ch[x>t[u].val]&&x!=t[u].val)//找到x,不一定找得到
        u = t[u].ch[x>t[u].val];//左儿子小，右儿子大
    splay(u,0);//当前位置旋转到根节点.根节点的fa存0
}

inline void insert(int x){//类似find,如果插入的数已经存在,可以在找到的节点计数
    int u = root,fu = 0;//当前位置u,u的爸爸fu
    while(u&&t[u].val!=x){//找合适位置,u找到空的地方也会停止
        fu = u;
        u = t[u].ch[x>t[u].val];
    }
    if(u) t[u].cnt++;//已有这个数字的情况,计数
    else{
        u = ++tot;//节点总数tot+1
        if(fu)//这时候fu是上一个u即新插入u的父节点,如果父节点不是0
```

```
74          t[fu].ch[x>t[fu].val]=u;
75      t[u].ch[0] = t[u].ch[1] = 0;//这个新节点没儿子
76      t[u].fa = fu;//父亲
77      t[u].val = x;//数值
78      t[u].cnt = 1;//计数
79      t[u].size = 1;//大小
80   }
81   splay(u,0);
82}
83
84inline int Next(int x,int f){
85   find(x);
86   int u=root;//根节点，此时x的父节点(存在的话)就是根节点
87   //这里sls回答了我的疑问
88   //"splay中不一定有x这个节点,那么它splay到根的就直接可以满足了"
89   //"如果有这个点的话就要在splay上再找一波(因为当前的根就是x这个点)"
90   if(t[u].val>x&&f)return u;//如果当前节点的值大于x并且要查找的是后继
91   if(t[u].val<x&&!f)return u;//如果当前节点的值小于x并且要查找的是前驱
92   //上面两个是x不在splay中的情况
93   u = t[u].ch[f];//后继在根右边找,前驱在左边找
94   while(t[u].ch[f^1]) u = t[u].ch[f^1];//左半边要往右跳找最大的,右半边往左跳
95   return u;
96}
97
98inline void Delete(int x){//删除x
99   int last = Next(x,0);//查找x的前驱
100  int next = Next(x,1);//找x的后继
101  splay(last,0);splay(next,last);//前驱节点转到根节点,后继转到根的儿子上
102  //操作完之后.后继是前驱的右儿子,x是前驱的左儿子,而且x是叶子节点
103  int del = t[next].ch[0];//后继的左儿子x
104  if(t[del].cnt>1){
105     t[del].cnt--;//如果有多个x,则x的计数减少一个
106     splay(del,0);//这个splay还重新pushup计算了del的子树
107  }
108  else
109     t[next].ch[0]=0;//因为是左儿子是叶子节点,直接丢掉
110}
111
112inline int kth(int x){//找第k小,改一下也可以找第k大
113  int u=root;
114     return 0;
115  while(1){
116     int y = t[u].ch[0];//左儿子
117     if(x>t[y].size+t[u].cnt){//如果左儿子和当前点的size比要找的排名数小
118        x-=t[y].size+t[u].cnt;//数量减少,相当于把这个寻找排名的起点变成了当前节点
119        u=t[u].ch[1];//那么当前排名的数一定要往右儿子上找
120     }
121     else if(t[y].size>=x) u=y;//左儿子的size足够,儿子在左侧上找
122     else return t[u].val;//左儿子的size比x小,加上当前点u的size则比x大,说明第kA大的就是x
123  }
124}
125
126int main(){
127  tot=0;
128  read(n);
```

```
    insert(+2147483647);insert(-2147483647);//博客作者在这里先加了正负INF
    int typ,x;
    while(n--){
        read(typ);
        if(typ==1){read(x);insert(x);}//插入
        else if(typ==2){read(x);Delete(x);}//删除
        else if(typ==3){//查找
            read(x);find(x);
            printf("%d\n",t[t[root].ch[0]].size);
        }
        else if(typ==4){//第k小
            read(x);printf("%d\n",kth(x+1));//之前插进去正负INF,所以要+1;
        }
        else if(typ==5){//前驱
            read(x);printf("%d\n",t[Next(x,0)].val);
        }
        else if(typ==6){//后继
            read(x);printf("%d\n",t[Next(x,1)].val);
        }
    }
    return 0;
}
```

6.8.4 文艺平衡树

处理区间翻转问题

```
struct Node{
    int l,r;
    int val,key;
    int size;
    bool reverse;//旋转的懒惰标记
}fhq[MAXN];
int cnt,root;
mt19937 rnd(233);
vector<int>res;//因为行的末尾不能有空格所以我先用一个vector存着
inline int newnode(int val){
    fhq[++cnt].val = val;
    fhq[cnt].key = rnd();
    fhq[cnt].size=1;
    return cnt;
}
inline void update(int now){
    fhq[now].size = fhq[fhq[now].l].size + fhq[fhq[now].r].size+1;
}
inline void pushdown(int now){
//如果当前节点的儿子有可能之后被动到,就需要向下传递reverse
    swap(fhq[now].l,fhq[now].r);
    fhq[fhq[now].l].reverse^=1;//儿子的懒惰标记0变1,1变0
    fhq[fhq[now].r].reverse^=1;
    fhq[now].reverse = 0;//之前疑惑这里为什么不用取反^1的
    //其实是因为如果now没有标记reverse,就不用执行pushdown
}
void split(int now,int siz,int &x,int &y){//按照大小分裂
//有点类似取第k大getnum的操作
```

```
    if(!now){x=y=0;return;}
    if(fhq[now].reverse) pushdown(now);//之后分裂操作可能碰到儿子,所以要pushdown
    if(fhq[fhq[now].l].size<siz){
        x = now;
        split(fhq[now].r,siz-fhq[fhq[now].l].size-1,fhq[now].r,y);
    }
    else{
        y = now;
        split(fhq[now].l,siz,x,fhq[now].l);
    }
    update(now);
}
int merge(int x,int y){
    if(!x||!y) return x+y;
    if(fhq[x].key<fhq[y].key){//这里其实用什么符号都行
        if(fhq[x].reverse) pushdown(x);
        fhq[x].r = merge(fhq[x].r,y);
        update(x);
        return x;
    }
    else{
        if(fhq[y].reverse) pushdown(y);
        fhq[y].l = merge(x,fhq[y].l);
        update(y);
        return y;
    }
}
void reverse(int l,int r){//区间反转
//拆成三段,(1,l-1),(l,r),(r+1,n);
    int x,y,z;
    split(root,l-1,x,y);
    split(y,r-l+1,y,z);
    fhq[y].reverse^=1;
    root=merge(merge(x,y),z);
}
void ldr(int now){//最后用中序遍历找到结果
    if(!now) return;
    if(fhq[now].reverse) pushdown(now);
    ldr(fhq[now].l);
    res.push_back(fhq[now].val);//推入结果的序列
    ldr(fhq[now].r);
}
int main(){
    int n,m;
    cin>>n>>m;
    rep(i,1,n) root = merge(root,newnode(i));//因为i一定比之前出现的都大才可以这样
    int l,r;
    while(m--){
        cin>>l>>r;
        reverse(l,r);
    }
    res.clear();
    ldr(root);
    int siz = res.size();
    rep(i,0,siz-1){
```

```
        cout<<res[i];
        if(i!=siz-1) cout<<' ';
    }
    cout<<endl;
}
```

6.8.5 二逼平衡树

线段树套平衡树

- 查询 k 在区间内的排名
- 查询区间内排名为 k 的值
- 修改某一位值上的数值
- 查询 k 在区间内的前驱（前驱定义为严格小于 x，且最大的数，若不存在输出 -2147483647）
- 查询 k 在区间内的后继（后继定义为严格大于 x，且最小的数，若不存在输出 2147483647）

```
rbset(pii(int, int)) seg[maxn << 2];
int date[maxn];
unsigned int cnt = 0;
void build(int pos, int l, int r)
{
  for (int i = l; i <= r; i++)
    seg[pos].insert(MP(date[i], ++cnt));
  if (l == r)
    return;
  int mid = (l + r) >> 1;
  build(lson(pos), l, mid);
  build(rson(pos), mid + 1, r);
}
void update(int pos, int l, int r, int x, int v)
{
  seg[pos].erase(seg[pos].lower_bound(MP(date[x], 0)));
  seg[pos].insert(MP(v, ++cnt));
  if (l == r)
    return;
  int mid = (l + r) >> 1;
  if (x <= mid)
    update(lson(pos), l, mid, x, v);
  else
    update(rson(pos), mid + 1, r, x, v);
}
int getrk(int pos, int l, int r, int ql, int qr, int k)
{
  if (ql <= l && qr >= r)
    return seg[pos].order_of_key(MP(k, 0));
  int ans = 0;
  int mid = (l + r) >> 1;
  if (ql <= mid)
    ans += getrk(lson(pos), l, mid, ql, qr, k);
  if (qr >= mid+1)
    ans += getrk(rson(pos), mid + 1, r, ql, qr, k);
  return ans;
}
int pre(int pos, int l, int r, int ql, int qr, int k)
```

```
{
  if (ql <= l && qr >= r)
  {
    auto tt = seg[pos].lower_bound(MP(k, 0));
    if (tt == seg[pos].begin())
      return -2147483647;
    return (--tt)->first;
  }
  int ans = -2147483647;
  int mid = l + r >> 1;
  if (ql <= mid)
    MAX(ans, pre(lson(pos), l, mid, ql, qr, k));
  if (qr > mid)
    MAX(ans, pre(rson(pos), mid + 1, r, ql, qr, k));
  return ans;
}
int aft(int pos, int l, int r, int ql, int qr, int k){
  if (ql <= l && qr >= r)
  {
    auto tt = seg[pos].lower_bound(MP(k + 1, 0));
    if (tt == seg[pos].end())
      return 2147483647;
    return tt->first;
  }
  int ans = 2147483647;
  int mid = (l + r) >> 1;
  if (ql <= mid)
    MIN(ans, aft(lson(pos), l, mid, ql, qr, k));
  if (qr > mid)
    MIN(ans, aft(rson(pos), mid + 1, r, ql, qr, k));
  return ans;
}
```

6.9 KD tree

```
inline void updmin(int &x,int y){x=min(x,y);}
inline void updmax(int &x,int y){x=max(x,y);}
int Dim;
struct KDT{
   int root;
   int dim,top=0,tot=0,*rub;
   struct point{
      int x[2],w;
      bool operator<(const point&b){
         return x[Dim]<b.x[Dim];
      }
   }*p;
   struct node{
      int mi[2],mx[2],sum=0,ls,rs,sz=0;
      point p;
   }*tr;
   int newnode(){
      if(top) return rub[top--];
      else return ++tot;
```

```
}
void up(int u){
    int l=tr[u].ls,r=tr[u].rs;
    rep(i,0,1){
        tr[u].mi[i]=tr[u].mx[i]=tr[u].p.x[i];
        if(l){
            updmin(tr[u].mi[i],tr[l].mi[i]);
            updmax(tr[u].mx[i],tr[l].mx[i]);
        }
        if(r){
            updmin(tr[u].mi[i],tr[r].mi[i]);
            updmax(tr[u].mx[i],tr[r].mx[i]);
        }
    }
    tr[u].sum=tr[l].sum+tr[r].sum+tr[u].p.w;
    tr[u].sz=tr[l].sz+tr[r].sz+1;
}
int build(int l,int r,int dim){
    if(l>r) return 0;
    int mid=l+r>>1,u=newnode();
    Dim=dim; nth_element(p+l,p+mid,p+r+1);
    tr[u].p=p[mid];
    tr[u].ls=build(l,mid-1,dim^1),tr[u].rs=build(mid+1,r,dim^1);
    up(u);
    return u;
}
void pia(int u,int num){//传统?
    int l=tr[u].ls,r=tr[u].rs;
    if(l) pia(l,num);
    p[tr[l].sz+1+num]=tr[u].p,rub[++top]=u;
    if(r) pia(r,num+1+tr[l].sz);
}
void balance(int &u,int dim){
    if(tr[u].sz*0.75<tr[tr[u].ls].sz||
      tr[u].sz*0.75<tr[tr[u].rs].sz){
       pia(u,0); u=build(1,tr[u].sz,dim);
    }
}
void insert(int &u,point p,int dim){
    if(!u) {
        u=newnode(),tr[u].p=p;
        tr[u].ls=tr[u].rs=0;
        up(u); return; //待修改
    }
    if(p.x[dim]<=tr[u].p.x[dim]) insert(tr[u].ls,p,dim^1);
    else insert(tr[u].rs,p,dim^1);
    up(u); balance(u,dim);
}
int in(int x1,int y1,int x2,int y2,int X1,int Y1,int X2,int Y2){
    return x1>=X1&&x2<=X2&&y1>=Y1&&y2<=Y2;
}//左是否在右内
int out(int x1,int y1,int x2,int y2,int X1,int Y1,int X2,int Y2){
    return x2<X1||x1>X2||y2<Y1||y1>Y2;
}//左是否在右外
int query(int u,int x1,int y1,int x2,int y2){
```

```
        if(!u) return 0;
        auto mx=tr[u].mx,mi=tr[u].mi,x=tr[u].p.x;
        int res=0;
        if(in(mi[0],mi[1],mx[0],mx[1],x1,y1,x2,y2)) return tr[u].sum;
        if(out(mi[0],mi[1],mx[0],mx[1],x1,y1,x2,y2)) return 0;
        if(in(x[0],x[1],x[0],x[1],x1,y1,x2,y2)) res+=tr[u].p.w;
        res+=query(tr[u].ls,x1,y1,x2,y2)+query(tr[u].rs,x1,y1,x2,y2);
        return res;
    }
    KDT(int maxn=1e6+10){
        tr=new node[maxn],p=new point[maxn];
        rub = new int[maxn],root=0;
    }
    void insert(int x,int y,int k){insert(root,(point){x,y,k},0);}
    int query(int x1,int y1,int x2,int y2){return query(root,x1,y1,x2,y2);}
};
```

7 字符串

7.1 马拉车

求字符串的最长回文串的长度。

```
int len[N*2]; char s[N*2]; // 两倍内存
int manacher(char s1[]){ // s1可以是s
    int n=strlen(s1)*2+1;
    repeat_back(i,0,n)s[i+1]=(i%2==0?'*':s1[i/2]);
    n++; s[0]='#'; s[n++]=0;
    len[0]=0;
    int mx=0,id=0,ans=0;
    repeat(i,1,n-1){
        if(i<mx)len[i]=min(mx-i,len[2*id-i]);
        else len[i]=1;
        while(s[i-len[i]]==s[i+len[i]])len[i]++;
        if(len[i]+i>mx)mx=len[i]+i,id=i;
        ans=max(ans,len[i]-1); // 最长回文串长度
    }
    return ans;
}
```

7.2 KMP

```
const int MAXN = 2e6+5;
int pi[MAXN];//MAXN记得开大一点,因为这里要存到m+n+1长度的
vector<int> res;//储存答案

void getpi(const string &s){ //求s的前缀函数
    pi[0]=0;
    int j=0;
    rep(i,1,s.length()-1){
        while(j>0&&s[i]!=s[j]) j=pi[j-1];//找到合适且最长的j
        if(s[i]==s[j])j++;//能成功匹配的情况
        pi[i]=j;
```

```
    }
}

void kmp(string s,string t){ //在主串t中找模式串s
    getpi(s+'#'+t);
    int n=(int)s.length(),m=(int)t.length();
    rep(i,n+1,m+n+1-1)
        if(pi[i]==n) res.push_back(i-2*s.size()); //i-2n计算得左端点
}
```

7.3 AC 自动机

```
const int MAXN = 1e5+5;
int jdbh[MAXN];//记录第i个模式串对应的节点编号
int cntcx[MAXN];//记录第i个模式串出现的次数
inline int idx(char c){return c-'a';}
struct Node{
    int son[26],flag,fail;//cnt记录次数,flag记录编号
    void clr(){
        memset(son,0,sizeof(son));
        flag=0;
    }
}trie[MAXN*10];
int n,cntt;//cntt记录总点数
string s,ms[166];
int maxx;
queue<int>q;
inline void insert(string &s,int num){
    int siz = s.size(),v,u=1;
    rep(i,0,siz-1){
        v = idx(s[i]);
        if(!trie[u].son[v]){trie[u].son[v] = ++cntt;trie[cntt].clr();}
        u = trie[u].son[v];
    }
    trie[u].flag = num;//标记为单词,flag记录编号
    //保证每个模式串只出现一次
    cntcx[num] = 0;
    jdbh[num] = u;//记录当前单词对应的节点编号
}
inline void getfail(){
    rep(i,0,25) trie[0].son[i] = 1;
    trie[0].flag = 0;
    q.push(1);
    trie[1].fail = 0;
    int u,v,ufail;
    while(!q.empty()){
        u = q.front();q.pop();
        rep(i,0,25){
            v = trie[u].son[i];
            ufail = trie[u].fail;
            if(!v){trie[u].son[i]=trie[ufail].son[i];continue;}//画好一条跳fail的路
            trie[v].fail = trie[ufail].son[i];
            q.push(v);
        }
```

```
    }
}
inline void query(string &s){
    int siz = s.size(),u = 1,v,k;
    rep(i,0,siz-1){
        v = idx(s[i]);
        k = trie[u].son[v];
        while(k){
            if(trie[k].flag){
                cntcx[trie[k].flag]++;//计数
                maxx = max(maxx,cntcx[trie[k].flag]);
            }
            k = trie[k].fail;//跳fail
        }
        u = trie[u].son[v];//这一句其实也有跳fail的功能，很精妙
    }
}
inline void solve(){
    cntt = 1;
    trie[0].clr();
    trie[1].clr();
    rep(i,1,n){
        cin>>ms[i];
        insert(ms[i],i);
    }
    getfail();
    cin>>s;
    maxx = 0;
    query(s);
    cout<<maxx<<endl;
    rep(i,1,n){
        if(cntcx[i]==maxx) cout<<ms[i]<<endl;
    }
}
```

7.4 FFT 解决字符串匹配问题

可以用来解决含有通配符的字符串匹配问题定义匹配函数

$$(x,y) = (A_x - B_x)^2$$

如果两个字符相同，则满足 $C(x,y) = 0$
定义模式串和文本串 x 位置对齐时候的完全匹配函数为

$$P(x) = \sum C(i, x+i)$$

模式串在位置 x 上匹配时,$p(x) = 0$
通过将模式串 reverse 后卷积，可以快速处理每个位置 x 上的完全匹配函数 $P(x)$ 同理，如果包含通配符，则设通配符的值为 0，可以构造损失函数

$$C(x,y) = (A_x - B_x)^2 \cdot A_x \cdot B_x = A_x^3 B_x + A_x B_x^3 - 2A_x^2 B_x^2$$

通过三次 FFT 即可求得每个位置上的 P(x)
以下是用 FFT 解决普通字符串匹配问题的代码
即实现 KMP 的功能，复杂度较高，为 $O(nlog_n)$

```
void solve(){
```

```
    limit = 1,l=0;
    cin>>n>>m;
    cin>>s1>>s2;
    rep(i,0,n-1) B[i].x = s1[i]-'a'+1;
    rep(i,0,m-1) A[i].x = s2[i]-'a'+1;
    double T = 0;
    //T = sigma A[i]^A[i] i=0~m-1
    rep(i,0,m-1) T += A[i].x*A[i].x;
    //f[x] = sigma B[i]^B[i] i=0~x
    f[0] = B[0].x*B[0].x;
    rep(i,1,n-1) f[i] = f[i-1]+B[i].x*B[i].x;
    //g[x] = S[i]*B[j] i+j==x
    reverse(A,A+m);//S = A.reverse
    //FFT预处理
    while(limit<=n+m-2) limit<<=1,l++;
    rep(i,0,limit-1)
        r[i]= ( r[i>>1]>>1 )| ( (i&1)<<(l-1) );

    FFT(A,1);FFT(B,1);
    rep(i,0,limit) A[i]=A[i]*B[i];
    FFT(A,-1);
    rep(i,0,n-1) g[i] = (int)(A[i].x/limit+0.5);//四舍五入

    //T + f(x) - f(x-m) - 2g(x);
    double tmp;
    rep(x,m-1,n-1){
        tmp = T+f[x]-2*g[x];
        if(x!=m-1) tmp -= f[x-m];
        //cout<<tmp<<' ';
        if(fabs(tmp)<eps) cout<<x-(m-1)+1<<endl;//输出匹配上的位置
    }
    cout<<endl;
}
```

7.5 字符串哈希

7.5.1 快速取子串哈希值

```
const int b = 131;//推荐的base，可以选其他质数
void init(int n){//初始化
    pw[0] = 1;
    for (int i = 1; i <= n; i ++ ) {
        h[i] = h[i-1]*b + str[i];//做每个前缀的哈希值
        pw[i] = pw[i-1]*b;//预处理b^k的值
    }
}
// 计算子串 str[l ~ r] 的哈希值
ull get(int l, int r) {
    return h[r] - h[l-1]*pw[r-l+1];
}
```

7.5.2 双哈希

```
#define ull unsigned long long
#define rep(i,a,b) for(int i=(a);i<=(b);i++)

typedef pair<ull,ull> pll;
const int MAXN = 1e4+5;
const int M1 = 1e9+7;//第一个模数
const int M2 = 1e9+9;//第二个模数
const int b = 131;
int n;
pll a[MAXN];

pll gethash(string s){
   ull res1=0,res2=0;
   int siz = s.length();
   rep(i,0,siz-1){
      res1=(res1*b%M1+s[i])%M1;//i位乘以b^i
      res2=(res2*b%M2+s[i])%M2;//f(s)=Σ s[i] * b^i;
   }
   return make_pair(res1,res2);
}
```

7.6 后缀数组 SA+LCP

LCP(i,j) 后缀 i 和后缀 j 的最长公共前缀

```
int n,m;
string s;
int rk[MAXN],sa[MAXN],c[MAXN],rk2[MAXN];
//sa[i]存排名i的原始编号 rk[i]存编号i的排名 第二关键字rk2
inline void get_SA(){
   rep(i,1,n) ++c[rk[i]=s[i]];//基数排序
   rep(i,2,m) c[i] += c[i-1];
   //c做前缀和，可以知道每个关键字的排名最低在哪里
   repb(i,n,1) sa[c[rk[i]]--] = i;//记录每个排名的原编号

   for(int w=1;w<=n;w<<=1){//倍增
      int num = 0;
      rep(i,n-w+1,n) rk2[++num] = i;//没有第二关键字的排在前面
      rep(i,1,n) if(sa[i]>w) rk2[++num] = sa[i]-w;
      //编号sa[i]大于w的才能作为编号sa[i]-w的第二关键字
      rep(i,1,m) c[i] = 0;
      rep(i,1,n) ++c[rk[i]];
      rep(i,2,m) c[i]+=c[i-1];
      repb(i,n,1) sa[c[rk[rk2[i]]]--]=rk2[i],rk2[i]=0;
      //同一个桶中按照第二关键字排序
      swap(rk,rk2);
      //这时候的rk2时这次排序用到的上一轮的rk,要计算出新的rk给下一轮排序

      rk[sa[1]]=1,num=1;
      rep(i,2,n)
         rk[sa[i]] = (rk2[sa[i]]==rk2[sa[i-1]]&&rk2[sa[i]+w]==rk2[sa[i-1]+w])?num:++num;
      //下一次排名的第一关键字,相同的两个元素排名也相同
      if(num==n) break;//rk都唯一时，排序结束
      m=num;
```

```
    }
}
int height[MAXN];
inline void get_height(){
    int k = 0,j;
    rep(i,1,n) rk[sa[i]] = i;
    rep(i,1,n){
        if(rk[i]==1) continue;//第一名往前没有前缀
        if(k) k--;//h[i]>=h[i-1]-1 即height[rk[i]]>=height[rk[i-1]]-1
        j = sa[rk[i]-1];//找排在rk[i]前面的
        while(j+k<=n&&i+k<=n&&s[i+k]==s[j+k]) ++k;//逐字符比较
        //因为每次k只会-1，故++k最多只会加2n次
        height[rk[i]] = k;
    }
}
inline void solve(){
    cin>>s;
    s = ' '+s;
    n = s.size()-1,m = 122;//m为字符个数'z'=122
    get_SA();
    rep(i,1,n) cout<<sa[i]<<' ';
    cout<<endl;
}
```

7.7 后缀自动机 SAM

```
struct state{
    int len,link;
    map<char,int> nxt;//也可以用数组,空间换时间
};
state sta[MAXN<<1];//状态数需要设定为两倍
int sz,last;//sz为自动机大小
inline void init_SAM(){
    sta[0].len = 0;sta[0].link = -1;//虚拟状态t0
    sz = 1;
    last = 0;
}
int cnt[MAXN<<1];
void SAM_extend(char c){
    int cur = sz++;
    cnt[cur] = 1;
    sta[cur].len = sta[last].len+1;
    int p = last;
    //沿着last的link添加到c的转移，直到找到已经有c转移的状态p
    while(p!=-1&&!sta[p].nxt.count(c)){
        sta[p].nxt[c] = cur;
        p = sta[p].link;
    }
    if(p==-1) sta[cur].link = 0;//情况1,没有符合的p
    else{
        int q = sta[p].nxt[c];
        if(sta[q].len==sta[p].len+1)//情况2,稳定的转移(lenq=lenp+1,前面没有增加)
            sta[cur].link = q;
        else{//情况3,把q的lenp+1的部分拿出来(clone),p到clone的转移是稳定的
```

```
            int clone = sz++;
            cnt[clone] = 0;
            sta[clone].len = sta[p].len+1;
            sta[clone].nxt = sta[q].nxt;
            sta[clone].link = sta[q].link;
            while(p!=-1 && sta[p].nxt[c]==q){//把向q的转移指向clone
                sta[p].nxt[c]=clone;
                p=sta[p].link;
            }
            sta[q].link = sta[cur].link = clone;//clone是q的后缀,故linkq=clone
        }
    }
    last = cur;//sta[last]包含目前处理的整个前缀!
}
string s;
vector<int> e[MAXN<<1];
void dfs(int now){
    for(auto to:e[now]){
        dfs(to);
        cnt[now] += cnt[to];
    }
}
inline void solve(){
    cin>>s;
    init_SAM();
    int siz = s.size();
    rep(i,0,siz-1) SAM_extend(s[i]);
    rep(i,1,sz-1) e[sta[i].link].push_back(i);//link边反过来构造树
    dfs(0);
    ll maxx = 0;
    rep(i,1,sz-1)
        if(cnt[i]!=1) maxx = max(maxx,1ll*cnt[i]*sta[i].len);
    cout<<maxx<<endl;
}
int main(){
    solve();
}
```

7.8 广义 SAM

```
//广义sam只需要在插入新串时把last设为1
//其实就是存一下自己的sam
struct SAM{
    static const int sigma=26,c0='a';
    struct Node{
        int fa=0,to[sigma],len=0;
        int &operator [] (int x) {return to[x];}
        Node(){
            rep(i,0,sigma-1) to[i]=0;
        }
    }a[N*2];
    int last=1,tot=1;
    int newnode(){
        a[++tot]=Node(); return tot;
```

```
    }

    ll sz[N*2],t[N*2],val[N*2];

    void insert(int c){
        bool fl=0; int z;
        c-=c0;
        //if(a[last][c]&&a[last].len+1==a[a[last][c]].len) return a[last][c];
        int x=last,next=newnode();
        sz[next]=1;
        last=next;
        a[next].len=a[x].len+1;
        for(;x&&!a[x][c];x=a[x].fa) a[x][c]=next;
        if(!x) a[next].fa=1;
        else{
            int y=a[x][c];
            if(a[y].len==a[x].len+1){
                a[next].fa=y;
            }
            else{
                if(a[x].len+1==a[next].len) fl=1;
                z=newnode();
                a[z]=a[y],a[z].len=a[x].len+1;
                //sz[z]=1;
                for(;x&&a[x][c]==y;x=a[x].fa) a[x][c]=z;
                a[y].fa=a[next].fa=z;
            }
        }
    }
    bool vis[N*2];
    void dfs(int u){
        vis[u]=1;
        sz[u]=1;
        rep(i,0,25) if(a[u][i]) {
            if(!vis[a[u][i]]) dfs(a[u][i]);
            sz[u]+=sz[a[u][i]];
        }
    }
    void solve(){
        int n=read();
        rep(i,1,n){
            last=1;
            string s; cin>>s;
            for(auto c:s) insert(c);
        }
        dfs(1);
        cout<<sz[1]-1<<endl;
    }

}sam;
```

7.9 trie 离线构造广义 SAM

```
```

```
int SAM_extend(char c,int last){
    int cur = sz++;
    sta[cur].len = sta[last].len+1;
    int p = last;
    //沿着last的link添加到c的转移，直到找到已经有c转移的状态p
    while(p!=-1&&!sta[p].nxt.count(c)){
        sta[p].nxt[c] = cur;
        p = sta[p].link;
    }
    if(p==-1) sta[cur].link = 0;//情况1,没有符合的p
    else{
        int q = sta[p].nxt[c];
        if(sta[q].len==sta[p].len+1)//情况2,稳定的转移(lenq=lenp+1,前面没有增加)
            sta[cur].link = q;
        else{//情况3,把q的lenp+1的部分拿出来(clone),p到clone的转移是稳定的
            int clone = sz++;
            sta[clone].len = sta[p].len+1;
            sta[clone].nxt = sta[q].nxt;
            sta[clone].link = sta[q].link;
            while(p!=-1 && sta[p].nxt[c]==q){//把向q的转移指向clone
                sta[p].nxt[c]=clone;
                p=sta[p].link;
            }
            sta[q].link = sta[cur].link = clone;//clone是q的后缀,故linkq=clone
        }
    }
    return cur;//sta[last]包含目前处理的整个前缀!
}

//trie tree=================================
map<char,int> tnxt[MAXN];
int tcnt = 1;
int poss[MAXN];//trie上点在sam上的位置
inline void trie_ins(string ss){
    int siz = ss.size();
    int now = 1;//rt
    rep(i,0,siz-1){
        if(tnxt[now][ss[i]]) now = tnxt[now][ss[i]];
        else tnxt[now][ss[i]] = ++tcnt,now = tcnt;
    }
}
#define ft first
#define sd second
//dfs on trie==============================
void dfs_on_trie(int pos){
    for(auto px:tnxt[pos]){
        poss[px.sd] = SAM_extend(px.ft,poss[pos]);
        dfs_on_trie(px.sd);
    }
}

ll res = 0;

int n;
inline void solve(){
```

```
    cin>>n;//字符数量
    string ts;
    rep(i,1,n){
        cin>>ts;
        trie_ins(ts);
    }
    sta[0].len = 0;sta[0].link = -1;//虚拟状态t0
    res = 0;
    poss[1] = 0;
    dfs_on_trie(1);//trie树建SAM
    // cout<<"sz = "<<sz<<endl;
    rep(i,1,sz-1){
        // cout<<sta[i].len<<'-'<<sta[sta[i].link].len<<endl;
        // cout<<"link["<<i<<"] = "<<sta[i].link<<endl;
        res += sta[i].len - sta[sta[i].link].len;
    }
    cout<<res<<endl;
}
```

8 其他

8.1 模拟退火

```
double ans=0;
double stemp=1997;//初始温度
double delta=0.997;//退火常数
mt19937 rnd(233);
int l,r
double calc(double x){
}
void SA(){
  double now=ans;
  double temp=stemp;
  while(temp>1e-14){
    double tnow=now+((rnd()<<1)-RAND_MAX)*temp;
    if(tnow<l||tnow>r){
      temp*=delta;
      continue;
    }
    double new_ans=calc(tnow);
    double de=new_ans-mx;
    if(de>0){
      now=tnow;
      ans=tnow;
      mx=new_ans;
    }else if(exp(-de/temp)*RAND_MAX>rnd()){
    now=tnow;
    }
  temp*=delta;
  }
},
void repeat(double t){
  while((double)clock()/CLOCKS_PER_SEC<t){
```

```
    SA();
  }
}
```

8.2 高精度

```
struct INT{
   static const int base=1e4;
   vector<int>v;
   INT(ll x=0){
      while(x)v.push_back(x%base),x/=base;
      if(v.size()==0)v.push_back(0);
   }
   inline int size()const{return v.size();}
   inline void resize(int sz=1){v.clear();v.resize(sz,0);}
   inline void topzero(){while(v.size()>1&&v.back()==0)v.pop_back();}
   INT operator +(const INT &A)const{
      INT T;
      T.resize(max(size(),A.size())+1);
      for(int i=0;i<T.size()-1;i++){
         if(i<size())T.v[i]+=v[i];
         if(i<A.size())T.v[i]+=A.v[i];
         if(T.v[i]>=base)T.v[i+1]++,T.v[i]-=base;
      }
      T.topzero();
      return T;
   }
   INT operator *(const INT &A)const{
      INT T;
      T.resize(size()+A.size()+5);
      for(int i=0;i<size();i++)for(int j=0;j<A.size();j++){
         T.v[i+j]+=v[i]*A.v[j];
         if(T.v[i+j]>=2e9)T.v[i+j+1]+=T.v[i+j]/base,T.v[i+j]%=base;
      }
      for(int i=0;i<T.size();i++){
         if(T.v[i]>=base)T.v[i+1]+=T.v[i]/base,T.v[i]%=base;
      }
      T.topzero();
      return T;
   }
   INT operator /(const int x)const{
      INT res;
      res.resize(size());
      ll tmp=0;
      for(int i=size()-1;i>=0;i--){
         tmp=tmp*base+v[i];
         res.v[i]=tmp/x,tmp%=x;
      }
      res.topzero();
      return res;
   }
   int operator %(const int mod)const{
      int res=0;
      for(int i=size()-1;i>=0;i--)res=(1ll*res*base+v[i])%mod;
```

```
        return res;
    }
    friend INT fast(INT a,ll b){
        INT res(1);
        while(b){
            if(b&1)res=res*a;
            a=a*a;
            b>>=1;
        }
        res.topzero();
        return res;
    }
    void read(){
        string str;
        cin>>str;
        resize();
        for(int i=0;i<str.size();i++)*this=*this*INT(10)+INT(str[i]-'0');
    }
    void print(){
        printf("%d",v[size()-1]);
        for(int i=size()-2;i>=0;i--)printf("%04d",v[i]);
    }
};
```

8.3 基础母函数求方案数

```
const int MAXN = 1e4;
int c1[MAXN+1];
int c2[MAXN+1];

int main(){
    int n;
    while(cin>>n){
        for(int i=0;i<=n;i++){//初始化
            c1[i]=0;
            c2[i]=0;
        }
        for(int i=0;i<=n;i++){
            c1[i]=1;//面值为一元的
        }
        for(int i=2;i<=n;i++){//枚举邮票的面值(一共有几组括号
            for(int j=0;j<=n;j++){//枚举左边次数为0到次数为n的项
                for(int k=0;j+k<=n;k+=i){//右边的乘过来，枚举放几枚
                //次数大于n的就不用管了
                    c2[j+k]+=c1[j];
                }
            }
            for(int i=0;i<=n;i++){//最左边两个括号完成处理
                c1[i]=c2[i];//把c2算出来的值挪到左边作为下一次的左边
                c2[i]=0;//清空c2记录下一次括号相乘的结果
            }
        }
        cout<<c1[n]<<endl;
    }
```

```
}
```

8.4 CDQ 分治三维偏序

```
#include<iostream>
#include<algorithm>
using namespace std;
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
const int MAXN = 214514;
struct node{
    int a,b,c,cnt,ans;
}s1[MAXN],s2[MAXN];
int ans[MAXN];//存对每个d,f[i]==d的i数量
int n,k,m;
//树状数组
int mx;
int treec[MAXN];
inline int lowbit(int x){return x&-x;}
inline void add(int x,int t){
    while(x<=mx) treec[x]+=t,x+=lowbit(x);
}
inline int query(int x){
    int ret = 0;
    while(x) ret+=treec[x],x-=lowbit(x);
    return ret;
}
//cdq分治
bool cmp1(node x,node y){//以a为关键字从小打到排序
    if(x.a==y.a){
        if(x.b==y.b) return x.c<y.c;
        return x.b<y.b;
    }
    return x.a<y.a;
}
bool cmp2(node x,node y){//根据b排序,便只剩下c的顺序没有满足
    if(x.b==y.b) return x.c<y.c;
    return x.b<y.b;
}
void cdq(int l,int r){
    if(l==r) return;
    int mid = (l+r)>>1;
    //处理前两种情况:ij同在左/右
    cdq(l,mid);cdq(mid+1,r);
    //处理第三种情况:i在左j在右
    sort(s2+l,s2+mid+1,cmp2);
    sort(s2+mid+1,s2+r+1,cmp2);
    int pxj=l,pxi=mid+1;//双指针,对应pxi的移动把pxj插入树状数组中
    for(pxi=mid+1;pxi<=r;pxi++){
        while(s2[pxj].b<=s2[pxi].b&&pxj<=mid){
            add(s2[pxj].c,s2[pxj].cnt);//带次数插入树状数组中
            pxj++;
        }
        s2[pxi].ans+=query(s2[pxi].c);
        //查询树状数组中c比他小的
```

```
    }
    rep(j,l,pxj-1) add(s2[j].c,-s2[j].cnt);//之前错在这里,注意只有pxj前面的才被加过,pxj没有
    //遍历pxj走过的区间,清空树状数组
}
int main(){
    cin>>n>>k;
    m=0,mx=k;//树状数组的最大数字(类似权值线段树)
    rep(i,1,n) cin>>s1[i].a>>s1[i].b>>s1[i].c;
    sort(s1+1,s1+1+n,cmp1);//以a为关键字从小到大排序
    int tmp=0;//当前这个项出现了几次
    rep(i,1,n){//合并相同的项并且记录出现次数进cnt
        tmp++;//当前项出现了几次
        if(s1[i].a!=s1[i+1].a||s1[i].b!=s1[i+1].b||s1[i].c!=s1[i+1].c){//判断两个node是否不同
            s2[++m]=s1[i];//m统计元素个数
            s2[m].cnt = tmp;
            tmp = 0;
        }
    }
    cdq(1,m);//递归计算
    //题目问的是f(x)=满足条件ij对数,对应每个x有几个,把结果存到ans[x]中
    rep(i,1,m) ans[s2[i].ans+s2[i].cnt-1]+=s2[i].cnt;
    //s2[i].cnt-1是这些数字自己之间满足的对数
    rep(i,0,n-1) cout<<ans[i]<<endl;
}
//洛谷P3810陌上花开
//https://www.luogu.com.cn/problem/P3810
```

8.5 三分

```
while(l<r){//类似求导的方式求极值
    int x=(l+r)/2,y=x+1; //l+(r-l)/2
    if(f(x)<f(y))l=x+1; else r=y-1; //最大值
    //if(f(x)<f(y)) r-y-1; else l=x+1; //最小值
}
```

8.6 数位 dp

```
ll dp[20][2][2],bit[20]; //这个[2]表示lz状态，如果lz被使用了的话就需要记录
ll dfs(int pos,ll s,bool lim=1,bool lz=1){
    if(pos==-1)return !s; //返回该状态是否符合要求(0或1)
    ll &x=dp[pos][s][0];
    if(!lim && x!=-1)return x;
    ll ans=0;
    int maxi=lim?bit[pos]:9;
    if(s){
        if(maxi>=2) rep(i,0,1)
            ans+=dfs(pos-1,i,lim&&2==maxi,0);
    }
    else{
        rep(i,0,maxi){
            if(i==2||i==4) continue;
            //...//状态转移
            //if(lz && i==0)...//可能要用lz，其他地方都不用
```

```
            ans+=dfs(pos-1,0,
                lim && i==maxi,
                lz && i==0);
            if(i!=6) ans+=dfs(pos-1,1,lim&&i==maxi,lz&&i==0);
        }
    }
    if(!lim)x=ans; //不限制的时候才做存储 yyds
    return ans;
}
ll calc(ll n){
    int len=0;
    while(n)bit[len++]=n%10,n/=10;
    return dfs(len-1,0)+dfs(len-1,1);
}
```

8.7 线性基

```
struct basis{//线性基超大常数版
    ll p[64];
    #define B(x,i) ((x>>i)&1)
    //yyds
    void init(){ memset(p,0,sizeof(p)); }
    void add(ll x){
        dep(i,62,0){
            if(!B(x,i)) continue;
            if(!p[i]) {p[i]=x; break;}
            x^=p[i];
        }
    }
    ll qmx(ll x){
        dep(i,62,0) if(!B(x,i)&&p[i]) x^=p[i];
        return x;
    }
    ll qmn(ll x){
        dep(i,62,0) if(B(x,i)&&p[i]) x^=p[i];
        return x;
    }
}b;
```

8.8 莫队

```
int cnt[MAXN];//记录数字在区间[l,r]内出现的次数
int pos[MAXN],a[MAXN];
ll ans[MAXN];
int n,m,k,res;
struct Q{
    int l,r,k;//k记录原来的编号
    friend bool operator < (Q x,Q y){//同一个分块内r小的排前面;不同分块则按分块靠前的
        return pos[x.l]==pos[y.l]?x.r<y.r:pos[x.l]<pos[y.l];
        //return (pos[a.l]^pos[b.l])?pos[a.l]<pos[b.l]:((pos[a.l]&1)?a.r<b.r:a.r>b.r);
        //这条第一个和==是一样的,后面的是对于左端点在同一奇数块的区间，右端点按升序排列，反之降序
    }
}q[MAXN];
```

```

void Add(int pos){
    res -= cnt[a[pos]]*cnt[a[pos]];
    cnt[a[pos]]++;
    res += cnt[a[pos]]*cnt[a[pos]];
}
void Sub(int pos){
    res -= cnt[a[pos]]*cnt[a[pos]];
    cnt[a[pos]]--;
    res += cnt[a[pos]]*cnt[a[pos]];
}
int main(){
    cin>>n>>m>>k;//k为数字范围
    memset(cnt,0,sizeof(cnt));
    int siz = sqrt(n);//每个分块的大小
    rep(i,1,n){
        cin>>a[i];
        pos[i] = i/siz;//分块
    }
    rep(i,1,m){
        cin>>q[i].l>>q[i].r;
        q[i].k = i;//记录原来的编号,用于打乱顺序后的还原
    }
    sort(q+1,q+1+m);
    res = 0;//初始化res
    int l = 1,r = 0;//当前知道的区间
    //因为是闭区间,如果是[1,1]的话则一开始就包含一个元素了
    rep(i,1,m){//莫队的核心,注意加减的顺序
        while(q[i].l<l) Add(--l);
        while(q[i].l>l) Sub(l++);
        while(q[i].r<r) Sub(r--);
        while(q[i].r>r) Add(++r);
        ans[q[i].k] = res;
    }
    rep(i,1,m) cout<<ans[i]<<endl;
}
```

8.9 带修莫队

```
int a[MAXN],b[MAXN];//a读入一开始的序列,b记录修改后的
int pos[MAXN];//分块
int cq,cr;//统计查询修改次数
int R[MAXN][3];//0记位置,1记原本的值,2记修改后的值
ll res;
int ans[MAXN];//记录结果
int n,m;
void Add(int x){if(cnt[x]==0)res++;cnt[x]++;}//带修莫队的add和sub有区别
void Sub(int x){if(cnt[x]==1)res--;cnt[x]--;}
struct Q{
    int l,r,k,t;
    friend bool operator < (Q a,Q b){
        return (pos[a.l]^pos[b.l])?pos[a.l]<pos[b.l]:((pos[a.r]^pos[b.r])?a.r<b.r:a.t<b.t);
        //增加第三关键字,询问的先后顺序,用t或者k应该都行
    }
```

```
}q[MAXN];
int main(){
   cin>>n>>m;
   cq = cr = 0;
   int siz = pow(n,2.0/3.0);//这么分块最好,别问
   rep(i,1,n){
      cin>>a[i];
      b[i]=a[i];
      pos[i] = i/siz;
   }
   char hc;
   rep(i,1,m){//读入修改和询问
      cin>>hc;
      if(hc=='Q'){
         cin>>q[cq].l>>q[cq].r;
         q[cq].k=cq;q[cq].t=cr;//注意这时候R[cr]还是没有的,这次询问是在R[cr-1]之后的
         cq++;
      }
      else{
         cin>>R[cr][0]>>R[cr][2];
         R[cr][1] = b[R[cr][0]];
         b[R[cr][0]] = R[cr][2];//在b数组中记录更改
         cr++;
      }
   }
   sort(q,q+cq);
   int l=1,r=0,sjc=0;//时间戳
   res = 0;
   rep(i,0,cq-1){
      while(sjc<q[i].t){
         if(l<=R[sjc][0]&&R[sjc][0]<=r)//判断修改是否在该区间内
            Sub(R[sjc][1]),Add(R[sjc][2]);
         a[R[sjc][0]] = R[sjc][2];//在a上也进行更改
         sjc++;
      }
      while(sjc>q[i].t){
         sjc--;
         if(l<=R[sjc][0]&&R[sjc][0]<=r)//判断修改是否在该区间内
            Sub(R[sjc][2]),Add(R[sjc][1]);
         a[R[sjc][0]] = R[sjc][1];//在a上也进行更改
      }
      while(l>q[i].l) Add(a[--l]);
      while(l<q[i].l) Sub(a[l++]);
      while(r<q[i].r) Add(a[++r]);
      while(r>q[i].r) Sub(a[r--]);
      ans[q[i].k] = res;
   }
   rep(i,0,cq-1) cout<<ans[i]<<endl;
}
```

9 STL 等小技巧

9.1 位运算应用-claris

9.1.1 枚举 i 的非空子集 j

```
for(j=i;j;j=(j-1)&i);
```

9.1.2 求 1 的个数

```
int __builtin_popcount(unsigned int x);
```

9.1.3 前缀 1 个数

```
int __builtin_clz(unsigned int x);
```

9.1.4 后缀 1 个数

```
int __builtin_ctz(unsigned int x);
```

9.2 集合 set

还可以通过 lower_bound 和 upper_bound 返回迭代器来找前驱, 后继

```
//并交集
vector<int> ANS;
set_union(s1.begin(),s1.end(),s2.begin(),s2.end(),inserter(ANS,ANS.begin()));//set_intersection()

//通过迭代器遍历集合
set<char>::iterator iter = temp1.begin();
while (iter!=temp1.end()){
   cout<<*iter;
   iter++;
}
```

9.3 快读快写 (短)

```
template<class T>inline void read(T &x){x=0;char o,f=1;while(o=getchar(),o<48)if(o==45)f=-f;do x=(x<<3)+(x
    <<1)+(o^48);while(o=getchar(),o>47);x*=f;}
template<class T>
void wt(T x){//快写
  if(x < 0) putchar('-'), x = -x;
  if(x >= 10) wt(x / 10);
  putchar('0' + x % 10);
}
```

9.4 GCD(压行)

```
ll gcd(ll a,ll b){ while(b^=a^=b^=a%=b); return a; }
```

9.5 计时

```
inline double run_time(){
    return 1.0*clock()/CLOCKS_PER_SEC;
}
```

9.6 替换 unorderedset 的 hash 函数

```
struct VectorHash {
    size_t operator()(const std::vector<int>& v) const {
        std::hash<int> hasher;
        size_t seed = 0;
        for (int i : v) {
            seed ^= hasher(i) + 0x9e3779b9 + (seed<<6) + (seed>>2);
        }
        return seed;
    }
};
unordered_set<vector<int>,VectorHash> st;
```

9.7 对拍

9.7.1 linux&mac 版

```
while true; do
./3.exe>tmp.in #出数据
./2.exe<tmp.in>tmp.out #被测程序
./4.exe<tmp.in>tmp2.out #正确（暴力）程序
if diff tmp.out tmp2.out; then #比较两个输出文件
printf AC #结果相同显示AC
else
echo WA #结果不同显示WA，并退出
cat tmp.in
cat tmp.out tmp2.out
exit 0
fi #if的结束标志,与C语言相反，0为真
done # while的结束标志
```

9.7.2 windows 版

```
@echo off
:loop
    rand.exe > in.txt
    my.exe < in.txt > myout.txt
    std.exe < in.txt > stdout.txt
    fc myout.txt stdout.txt
if not errorlevel 1 goto loop
pause
goto loop
```

9.8 火车头

```
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("-funsafe-loop-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,tune=native")
```