

泡泡猿 ACM 模板

Rand0w & REXWIND & Dallby

2021 年 11 月 24 日



目录

1	头文件	1
1.1	头文件 (Rand0w)	1
1.2	头文件 (REXWind)	3
1.3	头文件 (Dallby)	4
2	博弈	5
2.1	SG 函数	5
2.2	阶梯 Nim	5
2.3	威佐夫博弈	5
3	数学	7
3.1	欧拉筛	7
3.2	Exgcd	7
3.3	ExCRT 扩展中国剩余定理	7
3.4	线性求逆元	8
3.5	多项式	8
3.5.1	FFT 快速傅里叶变换	8
3.5.2	NTT 快速数论变换	9
3.5.3	MTT 任意模数 FFT	9
3.5.4	FWT 快速沃尔什变换	10
3.6	组合数	11
3.7	矩阵快速幂	11
3.8	高斯消元	12
3.9	欧拉降幂	13
3.10	求质数的原根	13
4	图论	13
4.1	Tarjan	13
4.2	拓扑排序	15
4.3	最大流 Dinic	15
4.4	费用流	16
4.5	二分图最大匹配	18
4.6	dijkstra	18
4.7	2-Sat	18
4.8	最大团	19
4.9	稳定婚姻	20
4.10	点双连通分量	20
4.11	强连通分量	21
4.12	最小树形图	22
4.13	KM	22
4.14	差分约束	23
4.15	一般图最大权匹配带花树 (hjt)	24
5	计算几何	25
5.1	基础点和线	25
5.2	三点求圆心	26
5.3	拉格朗日插值	26
6	数据结构	28
6.1	树背包	28
6.2	树上启发式合并	29
6.3	扫描线	30
6.4	并查集系列	31
6.4.1	普通并查集	31

6.4.2	按秩合并并查集	32
6.4.3	可持久化并查集	32
6.4.4	可撤销并查集	33
6.4.5	ETT 维护动态图连通性	34
6.5	平衡树系列	34
6.5.1	fhq_treap	34
6.5.2	替罪羊树	36
6.5.3	文艺平衡树	38
6.5.4	二逼平衡树	40
6.6	KD tree	41
7	字符串	43
7.1	KMP	43
7.2	AC 自动机	44
7.3	FFT 解决字符串匹配问题	45
7.4	字符串哈希	46
7.4.1	快速取子串哈希值	46
7.4.2	双哈希	46
7.5	后缀数组 SA+LCP	47
7.6	后缀自动机 SAM	48
7.7	广义 SAM	49
8	其他	50
8.1	FFT	50
8.2	基础母函数求方案数	51
8.3	ST 表求 RMQ	52
8.4	CDQ 分治三维偏序	52
8.5	三分	54
8.6	数位 dp	54
8.7	线性基	55
8.8	莫队	55
8.9	带修莫队	56
9	STL 等小技巧	57
9.1	集合 set	57
9.2	快读快写 (短)	58
9.3	GCD(压行)	58
9.4	计时	58
9.5	替换 unorderedset 的 hash 函数	58
9.6	对拍	58
9.6.1	linux&mac 版	58
9.6.2	windows 版	59
9.7	火车头	59

1 头文件

1.1 头文件 (Rand0w)

```
1 #include <bits/stdc++.h>
2 // #include <bits/extc++.h>
3 // using namespace __gnu_pbds;
4 // using namespace __gnu_cxx;
5 using namespace std;
6 #pragma optimize(2)
7 // #pragma GCC optimize("Ofast,no-stack-protector")
8 // #pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,tune=native")
9 #define rbset(T) tree<T,null_type,less<T>,rb_tree_tag,tree_order_statistics_node_update>
10 const int inf = 0x7FFFFFFF;
11 typedef long long ll;
12 typedef double db;
13 typedef long double ld;
14 template<class T>inline void MAX(T &x,T y){if(y>x)x=y;}
15 template<class T>inline void MIN(T &x,T y){if(y<x)x=y;}
16 namespace FastIO
17 {
18 char buf[1 << 21], buf2[1 << 21], a[20], *p1 = buf, *p2 = buf, hh = '\n';
19 int p, p3 = -1;
20 void read() {}
21 void print() {}
22 inline int getc()
23 {
24 return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 1 << 21, stdin), p1 == p2) ? EOF : *p1++;
25 }
26 inline void flush()
27 {
28 fwrite(buf2, 1, p3 + 1, stdout), p3 = -1;
29 }
30 template <typename T, typename... T2>
31 inline void read(T &x, T2 &... oth)
32 {
33 int f = 0; x = 0; char ch = getc();
34 while (!isdigit(ch)){if (ch == '-')f = 1; ch = getc();}
35 while (isdigit(ch)){x = x * 10 + ch - 48; ch = getc();}
36 x = f ? -x : x; read(oth...);
37 }
38 template <typename T, typename... T2>
39 inline void print(T x, T2... oth)
40 {
41 if (p3 > 1 << 20) flush();
42 if (x < 0) buf2[++p3] = 45, x = -x;
43 do{a[++p] = x % 10 + 48;} while (x /= 10);
44 do{buf2[++p3] = a[p];} while (--p);
45 buf2[++p3] = hh;
46 print(oth...);
47 }
48 } // namespace FastIO
49 #define read FastIO::read
50 #define print FastIO::print
```

```
51 #define flush FastIO::flush
52 #define spt fixed<<setprecision
53 #define endl1 '\n'
54 #define mul(a,b,mod) (__int128)(a)*(b)%(mod)
55 #define pii(a,b) pair<a,b>
56 #define pow powmod
57 #define X first
58 #define Y second
59 #define lowbit(x) (x&-x)
60 #define MP make_pair
61 #define pb push_back
62 #define pt putchar
63 #define yx_queue priority_queue
64 #define lson(pos) (pos<<1)
65 #define rson(pos) (pos<<1|1)
66 #define y1 code_by_Rand0w
67 #define yn A_muban_for_ACM
68 #define j1 it_is_just_an_eastegg
69 #define lr hope_you_will_be_happy_to_see_this
70 #define int long long
71 #define rep(i, a, n) for (register int i = a; i <= n; ++i)
72 #define per(i, a, n) for (register int i = n; i >= a; --i)
73 const ll llinf = 4223372036854775851;
74 const ll mod = (0 ? 1000000007 : 998244353);
75 ll pow(ll a,ll b,ll md=mod) {ll res=1;a%=md; assert(b>=0); for(;b>=>1){if(b&1)res=mul(res,a,md);a=mul(a,a,md);}return res;}
76 const ll mod2 = 999998639;
77 const int m1 = 998244353;
78 const int m2 = 1000001011;
79 const int pr=233;
80 const double eps = 1e-7;
81 const int maxm= 1;
82 const int maxn = 510000;
83 void work()
84 {
85 }
86 signed main()
87 {
88     #ifndef ONLINE_JUDGE
89         //freopen("in.txt","r",stdin);
90         //freopen("out.txt","w",stdout);
91     #endif
92     //std::ios::sync_with_stdio(false);
93     //cin.tie(NULL);
94     int t = 1;
95     //cin>>t;
96     for(int i=1;i<=t;i++){
97         //cout<<"Case #"<<i<<": "<<endl1;
98         work();
99     }
100     return 0;
101 }
```

1.2 头文件 (REXWind)

```
1  #include<iostream>
2  #include<cstring>
3  #include<cstdio>
4  #include<algorithm>
5  #include<vector>
6  #include<map>
7  #include<queue>
8  #include<cmath>
9  using namespace std;
10
11 template<class T>inline void read(T &x){
12     x=0;char o,f=1;
13     while(o=getchar(),o<48)if(o==45)f=-f;
14     do x=(x<<3)+(x<<1)+(o^48);while(o=getchar(),o>47);x*=f;}
15 int cancel_sync=(ios::sync_with_stdio(0),cin.tie(0),0);
16 #define ll long long
17 #define ull unsigned long long
18 #define rep(i,a,b) for(int i=(a);i<=(b);i++)
19 #define repb(i,a,b) for(int i=(a);i>=b;i--)
20 #define mkp make_pair
21 #define ft first
22 #define sd second
23 #define log(x) (31-__builtin_clz(x))
24 #define INF 0x3f3f3f3f
25 typedef pair<int,int> pii;
26 typedef pair<ll,ll> pll;
27 ll gcd(ll a,ll b){ while(b^=a^=b^=a%=b); return a; }
28 // #define INF 0x7fffffff
29
30 void solve(){
31
32 }
33
34 int main(){
35     int z;
36     cin>>z;
37     while(z--) solve();
38 }
```

1.3 头文件 (Dallby)

```
1 #include<bits/stdc++.h>
2 // #pragma GCC optimize(3)
3 #include<bits/stdc++.h>
4 using namespace std;
5 #define rep(i,x,y) for(int i=(x);i<=(y);++i)
6 #define dep(i,x,y) for(int i=(x);i>=(y);--i)
7 #define mst(a,x) memset(a,x,sizeof(a))
8 #define endl "\n"
9 #define fr first
10 #define sc second
11 #define debug cout<<"DEBUG\n";
12 #define OMG(a,n) rep(i,1,n) cout<<a[i]<<" "; cout<<endl;
13 #define OMG2(a,n,m) rep(i,1,n) {rep(i,1,m) cout<<a[i][j]<<" "; cout<<endl;}
14 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...);}
15 mt19937 rnd(chrono::high_resolution_clock::now().time_since_epoch().count());
16 typedef long long ll; typedef unsigned long long ull; typedef pair<int,ll>pil; typedef pair<int,int>pii;
    typedef pair<ll,ll>p11;
17 const int N=1e6+10; const double eps=1e-9;
18 const int inf=0x3f3f3f3f; const ll INF=0x3f3f3f3f3f3f3f3f;
19 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;}
20 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;}
21 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;}
22 ll a[N],b[N];
23 void Solve(){
24
25 }
26 int main(){
27     //freopen("D:\\in.txt","r",stdin);
28     //freopen("D:\\out.txt","w",stdout);
29     //ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
30     int T=1; //T=read();
31     rep(kase,1,T){
32         Solve();
33     }
34     return 0;
35 }
```

2 博弈

2.1 SG 函数

```

1 //sg板子
2 int sg[MAXN];
3 bool vis[MAXN];
4 inline void get_sg(){
5     int j;
6     rep(i,0,MAXN-1){
7         memset(vis,0,sizeof(vis));
8         rep(j,1,cnt){//转移
9             if(i-fb[j]<0) break;
10            vis[sg[i-fb[j]]] = 1;
11        }
12        //找mex
13        rep(j,0,MAXN-1)//找后继中最小未出现的数字
14            if(!vis[j]){sg[i]=j;break;}
15    }
16 }

```

2.2 阶梯 Nim

n 个位置，每个位置上有 a_i 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 a_i 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。

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

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 $|x-y| \leq k$ 。

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 。

```

1 void solve(){
2     cin>>a>>b>>k;
3     if(a>b) swap(a,b);
4     double l = 1,r = 2;//alpha的区间应该是在1和2之间的
5     double now=0,mid;
6     while(abs(now-1.0)>eps){//第一次二分寻找合适的alpha
7         mid = (l+r)/2;
8         now = cal(k,mid);
9         if(now>1.0) l = mid;
10        else r = mid;
11    }
12    double alpha = mid;//通过二分计算得到alpha
13    int lb = 0,rb = 5e7+5;

```



```
14     int midd;
15     while(lb+1<rb){//第二次二分求a对应第几个二元组
16         midd = (lb+rb)/2;
17         if(floor(midd*alpha)>a) rb = midd;
18         else lb = midd;
19     }
20     if(floor(lb*alpha)==a&&a+(k+1)*lb==b) cout<<0<<endl;
21     else cout<<1<<endl;
22 }
```

3 数学

3.1 欧拉筛

$O(n)$ 筛素数

```

1 int primes[maxn+5],tail;
2 bool is_prime[maxn+5];
3 void euler(){
4     is_prime[1] = 1;
5     for (int i = 2; i < maxn; i++)
6     {
7         if (!is_prime[i])
8             primes[++tail]=i;
9         for (int j = 0; j < primes.size() && i * primes[j] < maxn; j++)
10        {
11            is_prime[i * primes[j]] = 1;
12            if ((i % primes[j]) == 0)
13                break;
14        }
15    }
16 }
```

3.2 Exgcd

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

```

1 void Exgcd(ll a,ll b,ll &d,ll &x,ll &y){
2     if(!b){d=a;x=1;y=0;}
3     else{Exgcd(b,a%b,d,y,x);y-=x*(a/b);}
4 }
```

3.3 ExCRT 扩展中国剩余定理

求解同余方程组

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

```

1 int excrt(int a[],int b[],int n){
2     int lc=1;
3     for(int i=1;i<=n;i++){
4         lc=lcm(lc,a[i]);
5     }
6     for(int i=1;i<n;i++){
7         int p,q,g;
8         g=exgcd(a[i],a[i+1],p,q);
9         int k=(b[i+1]-b[i])/g;
10        q=-q;p*=k;q*=k;
11        b[i+1]=a[i]*p%lc+b[i];
12        b[i+1]%=lc;
13        a[i+1]=lcm(a[i],a[i+1]);
14    }
```

```
14     return (b[n]%lc+lc)%lc;
15 }
```

3.4 线性求逆元

```
1 void init(int p){
2     inv[1] = 1;
3     for(int i=2;i<=n;i++){
4         inv[i] = (ll)(p-p/i)*inv[p%i]%p;
5     }
6 }
```

3.5 多项式

3.5.1 FFT 快速傅里叶变换

```
1 const int SIZE=(1<<21)+5;
2 const double PI=acos(-1);
3 struct CP{
4     double x,y;
5     CP(double x=0,double y=0):x(x),y(y){}
6     CP operator +(const CP &A)const{return CP(x+A.x,y+A.y);}
7     CP operator -(const CP &A)const{return CP(x-A.x,y-A.y);}
8     CP operator *(const CP &A)const{return CP(x*A.x-y*A.y,x*A.y+y*A.x);}
9 };
10 int limit,rev[SIZE];
11 void DFT(CP *F,int op){
12     for(int i=0;i<limit;i++)if(i<rev[i])swap(F[i],F[rev[i]]);
13     for(int mid=1;mid<limit;mid<=1){
14         CP wn(cos(PI/mid),op*sin(PI/mid));
15         for(int len=mid<<1,k=0;k<limit;k+=len){
16             CP w(1,0);
17             for(int i=k;i<k+mid;i++){
18                 CP tmp=w*F[i+mid];
19                 F[i+mid]=F[i]-tmp;
20                 F[i]=F[i]+tmp;
21                 w=w*wn;
22             }
23         }
24     }
25     if(op==1)for(int i=0;i<limit;i++)F[i].x/=limit;
26 }
27 void FFT(int n,int m,CP *F,CP *G){
28     for(limit=1;limit<=n+m;limit<=1);
29     for(int i=0;i<limit;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?limit>>1:0);
30     DFT(F,1),DFT(G,1);
31     for(int i=0;i<limit;i++)F[i]=F[i]*G[i];
32     DFT(F,-1);
33 }
```

3.5.2 NTT 快速数论变换

```

1  const int SIZE=(1<<21)+5;
2  int limit,rev[SIZE];
3  void DFT(ll *f, int op) {
4      const ll G = 3;
5      for(int i=0; i<limit; ++i) if(i<rev[i]) swap(f[i],f[rev[i]]);
6      for(int len=2; len<=limit; len<<=1) {
7          ll w1=pow(pow(G,(mod-1)/len),~op?1:mod-2);
8          for(int l=0, hf=len>>1; l<limit; l+=len) {
9              ll w=1;
10             for(int i=l; i<l+hf; ++i) {
11                 ll tp=w*f[i+hf]%mod;
12                 f[i+hf]=(f[i]-tp+mod)%mod;
13                 f[i]=(f[i]+tp)%mod;
14                 w=w*w1%mod;
15             }
16         }
17     }
18     if(op==-1) for(int i=0, inv=pow(limit,mod-2); i<limit; ++i) f[i]=f[i]*inv%mod;
19 }
20 void NTT(int n,int m,int *F,int *G){
21     for(limit=1;limit<=n+m;limit<<=1);
22     for(int i=0;i<limit;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?limit>>1:0);
23     DFT(F,1),DFT(G,1);
24     for(int i=0;i<limit;i++)F[i]=F[i]*G[i];
25     DFT(F,-1);
26 }

```

3.5.3 MTT 任意模数 FFT

FFT 版常数巨大，慎用。

```

1  struct MTT{
2      static const int N=1<<20;
3      struct cp{
4          long double a,b;
5          cp(){a=0,b=0;}
6          cp(const long double &a,const long double &b):a(a),b(b){}
7          cp operator+(const cp &t)const{return cp(a+t.a,b+t.b);}
8          cp operator-(const cp &t)const{return cp(a-t.a,b-t.b);}
9          cp operator*(const cp &t)const{return cp(a*t.a-b*t.b,a*t.b+b*t.a);}
10         cp conj()const{return cp(a,-b);}
11     };
12     cp wn(int n,int f){
13         static const long double pi=acos(-1.0);
14         return cp(cos(pi/n),f*sin(pi/n));
15     }
16     int g[N];
17     void dft(cp a[],int n,int f){
18         for(int i=0;i<n;i++)if(i>g[i])swap(a[i],a[g[i]]);
19         for(int i=1;i<n;i<<=1){
20             cp w=wn(i,f);
21             for(int j=0;j<n;j+=i<<1){
22                 cp e(1,0);

```

```

23         for(int k=0;k<i;e=e*w,k++){
24             cp x=a[j+k],y=a[j+k+i]*e;
25             a[j+k]=x+y,a[j+k+i]=x-y;
26         }
27     }
28 }
29 if(f==-1){
30     cp Inv(1.0/n,0);
31     for(int i=0;i<n;i++)a[i]=a[i]*Inv;
32 }
33 }
34 cp a[N],b[N],Aa[N],Ab[N],Ba[N],Bb[N];
35 vector<ll> conv_mod(const vector<ll> &u,const vector<ll> &v,ll mod){ // 任意模数fft
36     const int n=(int)u.size()-1,m=(int)v.size()-1,M=sqrt(mod)+1;
37     const int k=32-__builtin_clz(n+m+1),s=1<<k;
38     g[0]=0; for(int i=1;i<s;i++)g[i]=(g[i/2]/2)|(((i&1)<<(k-1)));
39     for(int i=0;i<s;i++){
40         a[i]=i<=n?cp(u[i]%mod%M,u[i]%mod/M):cp();
41         b[i]=i<=m?cp(v[i]%mod%M,v[i]%mod/M):cp();
42     }
43     dft(a,s,1); dft(b,s,1);
44     for(int i=0;i<s;i++){
45         int j=(s-i)%s;
46         cp t1=(a[i]+a[j].conj())*cp(0.5,0);
47         cp t2=(a[i]-a[j].conj())*cp(0,-0.5);
48         cp t3=(b[i]+b[j].conj())*cp(0.5,0);
49         cp t4=(b[i]-b[j].conj())*cp(0,-0.5);
50         Aa[i]=t1*t3,Ab[i]=t1*t4,Ba[i]=t2*t3,Bb[i]=t2*t4;
51     }
52     for(int i=0;i<s;i++){
53         a[i]=Aa[i]+Ab[i]*cp(0,1);
54         b[i]=Ba[i]+Bb[i]*cp(0,1);
55     }
56     dft(a,s,-1); dft(b,s,-1);
57     vector<ll> ans;
58     for(int i=0;i<n+m+1;i++){
59         ll t1=llround(a[i].a)%mod;
60         ll t2=llround(a[i].b)%mod;
61         ll t3=llround(b[i].a)%mod;
62         ll t4=llround(b[i].b)%mod;
63         ans.push_back((t1+(t2+t3)*M%mod+t4*M*M)%mod);
64     }
65     return ans;
66 }
67 }mtt;

```

3.5.4 FWT 快速沃尔什变换

计算

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

\oplus 可以是与、或、异或

```

1 void FWT(ll *f, int op) {
2     for(int len=2; len<=up; len<<=1) {
3         for(int l=0, hf=len>>1; l<up; l+=len) {
4             for(int i=l; i<l+hf; ++i) {
5                 ll x=f[i], y=f[i+hf];
6                 if(op>0) {
7                     if(op==1) f[i]=(x+y)%mod, f[i+hf]=(x-y+mod)%mod; //xor
8                     else if(op==2) f[i]=(x+y)%mod; //and
9                     else f[i+hf]=(x+y)%mod; //or
10                }
11                else {
12                    if(op==-1) f[i]=(x+y)*inv2%mod, f[i+hf]=(x-y+mod)*inv2%mod; //xor
13                    else if(op==-2) f[i]=(x-y+mod)%mod; //and
14                    else f[i+hf]=(y-x+mod)%mod; //or
15                }
16            }
17        }
18    }
19 }

```

3.6 组合数

预处理阶乘，并通过逆元实现相除

```

1 ll jc[MAXN];
2 ll qpow(ll d,ll c){//快速幂
3     ll res = 1;
4     while(c){
5         if(c&1) res=res*d%med;
6         d=d*d%med;c>>=1;
7     }return res;
8 }
9 inline ll niyuan(ll x){return qpow(x,med-2);}
10 void initjc(){//初始化阶乘
11     jc[0] = 1;
12     rep(i,1,MAXN-1) jc[i] = jc[i-1]*i%med;
13 }
14 inline int C(int n,int m){//n是下面的
15     if(n<m) return 0;
16     return jc[n]*niyuan(jc[n-m])%med*niyuan(jc[m])%med;
17 }
18 int main(){
19     initjc();
20     int n,m;
21     while(cin>>n>>m) cout<<C(n,m)<<endl;
22 }

```

3.7 矩阵快速幂

```

1 struct Matrix{
2     ll a[MAXN][MAXN];
3     Matrix(ll x=0){
4         for(int i=0;i<n;i++){

```

```

5         for(int j=0;j<n;j++){
6             a[i][j]=x*(i==j);
7         }
8     }
9 }
10 Matrix operator *(const Matrix &b)const{//通过重载运算符实现矩阵乘法
11     Matrix res(0);
12     for(int i=0;i<n;i++){
13         for(int j=0;j<n;j++){
14             for(int k=0;k<n;k++){
15                 ll &ma = res.a[i][j];
16                 ma = (ma+a[i][k]*b.a[k][j])%mod;
17             }
18         }
19     }
20     return res;
21 }
22 };
23 Matrix qpow(Matrix d,ll m){//底数和幂次数
24     Matrix res(1);//构造E单位矩阵
25     while(m){
26         if(m&1)
27             res=res*d;
28         d=d*d;
29         m>>=1;
30     }
31     return res;
32 }

```

3.8 高斯消元

$O(n^3)$ 复杂度，需要用 double 存储。

```

1 double date[110][110];
2 bool guass(int n){
3     for(int i=1;i<=n;i++){
4         int mix=-1;
5         for(int j=i;j<=n;j++){
6             if(date[j][i]!=0){
7                 mix=j;break;
8             }
9         }
10        if(mix==-1)
11            return false;
12        if(mix!=i)
13            swap(date[mix][j],date[i][j]);
14        double t=date[i][i];
15        for(int j=i;j<=n;j++){
16            date[i][j]=date[i][j]/t;
17        }
18        for(int j=1;j<=n;j++){
19            if(date[j][i]==0||j==i)
20                continue;
21            double g=date[j][i]/date[i][i];
22            for(int k=1;k<=n+1;k++)

```

```

23         date[j][k]-=date[i][k]*g;
24     }
25 }
26 return true;
27 }

```

3.9 欧拉降幂

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

3.10 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 $a^{\frac{\varphi(m)}{p}} \not\equiv 1 \pmod{m}$ 。

```

1 vector<int> zyz;
2 inline ll get_yg(ll p){
3     zyz.clear();
4     ll tp = p-1;//phi(p)
5     int sqr = sqrt(tp+1);
6     rep(i,2,sqr){
7         if(tp%i==0)
8             zyz.push_back(i);
9         while(tp%i==0) tp/=i;
10    }
11    if(tp!=1) zyz.push_back(tp);
12    rep(i,2,p){
13        bool flag = 1;
14        for(auto x:zyz)
15            if(qpow(i,(p-1)/x,p)==1){flag = 0;break;}
16        if(flag) return i;
17    }
18    return -1;
19 }

```

4 图论

4.1 Tarjan

```

1 int n;
2 vector<int> e[MAXN]; //存图
3 int dfn[MAXN], low[MAXN];
4 int dfncnt; //dfn自增下标
5 int s[MAXN], tp; //数组模拟栈, tp记录大小
6 bool in[MAXN]; //记录该点是否在栈中
7 int scc[MAXN], sc; //节点i所在scc的编号, sc记录有几个强连通
8 int sz[MAXN]; //强连通i的大小
9 int indg[MAXN]; //记录缩点后的入度（这题才有的）
10

```



```
11 void tarjan(int u){
12     low[u]=dfn[u]++;dfncnt; //low初始值为自身dfn
13     s[++tp]=u; //推u入栈, 从1开始
14     in[u]=1; //记录u点在栈中
15     for(auto v:e[u]){ //访问到新点的情况
16         if(!dfn[v]){
17             tarjan(v);
18             low[u] = min(low[u],low[v]); //用low[v]更新low[u]
19         }
20         else if(in[v]) //v被访问过, 但是在栈中
21             low[u] = min(low[u],dfn[v]);
22     }
23     if(dfn[u]==low[u]){ //u是连通分量的根节点
24         sc++; //强连通数量++
25         sz[sc] = 0;
26         while(s[tp]!=u){ //u和u之后的点全部出栈
27             scc[s[tp]] = sc; //这个点包含于第几个强连通
28             sz[sc]++; //u为根的这个强连通的大小
29             in[s[tp]] = 0; //出栈
30             tp--;
31         }
32         scc[u] = sc; //给根节点标, 属于第sc个强连通
33         sz[sc]++;
34         in[u] = 0;
35         tp--;
36     }
37 }
38
39 void reset(){
40     tp = sc = dfncnt = 0;
41     rep(i,1,n){
42         in[i] = dfn[i] = 0; //low不用清空, sz在之后用到再清空
43         e[i].clear();
44     }
45 }
46
47 int main(){
48     cin>>n;
49     reset();
50     int v;
51     rep(u,1,n){
52         while(cin>>v&&v!=0) e[u].push_back(v);
53     }
54     rep(u,1,n)
55         if(!dfn[u]) tarjan(u);
56     rep(i,1,sc) indg[i] = 0; //这个不包含在tarjan里面, 是这题记录入度的
57     rep(u,1,n){
58         for(auto v:e[u]){
59             if(scc[u]!=scc[v]) indg[scc[v]]++;
60         }
61     }
62     int res = 0;
63     rep(i,1,sc){
64         if(indg[i]==0) res++;
65     }
```

```
66     cout<<res<<endl;
67 }
```

4.2 拓扑排序

```
1  vector<int> e[MAXN];
2  int n,m;
3  int indg[MAXN];
4
5  void solve(){
6      for(int i=1;i<=n;i++) {indg[i]=0;e[i].clear();}
7      for(int i=1,u,v;i<=m;i++){
8          cin>>u>>v;
9          e[u].push_back(v);
10         indg[v]++;
11     }
12     queue<int> q;
13     for(int i=1;i<=n;i++)
14         if(indg[i]==0) q.push(i); //找到入度为0的点
15     vector<int> res; //储存结果的向量
16     int u;
17     while(!q.empty()){ //BFS
18         u = q.front();
19         q.pop();
20         res.push_back(u);
21         for(auto v:e[u]){
22             if(--indg[v]==0)q.push(v);
23             //删掉当前点u, 点v的入度--
24             //这一步的目的是找到新的入度为0的节点, 推入队列Q
25         }
26     }
27     if(res.size()==n){ //如果这个图无环的话, 说明可以进行拓扑排序
28         for(int i=0;i<n;i++){ //输出结果
29             cout<<res[i]<<' ';
30         }
31         cout<<endl;
32     }
33 }
```

4.3 最大流 Dinic

编号从 0 开始可用, 复杂度 $O(V^2E)$

```
1  struct FLOW{
2      struct edge{int to,w,nxt;};
3      vector<edge> a; int head[N],cur[N];
4      int n,s,t;
5      queue<int> q; bool inque[N];
6      int dep[N];
7      void ae(int x,int y,int w){ // add edge
8          a.push_back({y,w,head[x]});
9          head[x]=a.size()-1;
10     }
11     bool bfs(){ // get dep[]
```

```

12     fill(dep,dep+n,inf); dep[s]=0;
13     copy(head,head+n,cur);
14     q=queue<int>(); q.push(s);
15     while(!q.empty()){
16         int x=q.front(); q.pop(); inque[x]=0;
17         for(int i=head[x];i!=-1;i=a[i].nxt){
18             int p=a[i].to;
19             if(dep[p]>dep[x]+1 && a[i].w){
20                 dep[p]=dep[x]+1;
21                 if(inque[p]==0){
22                     inque[p]=1;
23                     q.push(p);
24                 }
25             }
26         }
27     }
28     return dep[t]!=inf;
29 }
30 int dfs(int x,int flow){ // extend
31     int now,ans=0;
32     if(x==t)return flow;
33     for(int &i=cur[x];i!=-1;i=a[i].nxt){
34         int p=a[i].to;
35         if(a[i].w && dep[p]==dep[x]+1)
36             if((now=dfs(p,min(flow,a[i].w)))){
37                 a[i].w-=now;
38                 a[i^1].w+=now;
39                 ans+=now,flow-=now;
40                 if(flow==0)break;
41             }
42     }
43     return ans;
44 }
45 void init(int _n){
46     n=_n+1; a.clear();
47     fill(head,head+n,-1);
48     fill(inque,inque+n,0);
49 }
50 int solve(int _s,int _t){ // return max flow
51     s=_s,t=_t;
52     int ans=0;
53     while(bfs())ans+=dfs(s,inf);
54     return ans;
55 }
56 }flow;
57 void add(int x,int y,int w){flow.ae(x,y,w),flow.ae(y,x,0);}
58 // 先flow.init(n), 再add添边, 最后flow.solve(s,t)
    
```

4.4 费用流

编号从 0 开始可用, 复杂度 $O(VE^2)$

```

1 struct FLOW{
2     struct edge{int to,w,cost,nxt;};
3     vector<edge> a; int head[N];
    
```

```
4   int n,s,t,totcost;
5   deque<int> q;
6   bool inque[N];
7   int dis[N];
8   struct{int to,e;}pre[N];
9   void ae(int x,int y,int w,int cost){
10      a.push_back((edge){y,w,cost,head[x]});
11      head[x]=a.size()-1;
12  }
13  bool spfa(){
14      fill(dis,dis+n,inf); dis[s]=0;
15      q.assign(1,s);
16      while(!q.empty()){
17          int x=q.front(); q.pop_front();
18          inque[x]=0;
19          for(int i=head[x];i!=-1;i=a[i].nxt){
20              int p=a[i].to;
21              if(dis[p]>dis[x]+a[i].cost && a[i].w){
22                  dis[p]=dis[x]+a[i].cost;
23                  pre[p]={x,i};
24                  if(inque[p]==0){
25                      inque[p]=1;
26                      if(!q.empty()
27                         && dis[q.front()]<=dis[p])
28                          q.push_back(p);
29                      else q.push_front(p);
30                  }
31              }
32          }
33      }
34      return dis[t]!=inf;
35  }
36  void init(int _n){
37      n=_n+1;
38      a.clear();
39      fill(head,head+n,-1);
40      fill(inque,inque+n,0);
41  }
42  int solve(int _s,int _t){ // 返回最大流，费用存totcost里
43      s=_s,t=_t;
44      int ans=0;
45      totcost=0;
46      while(spfa()){
47          int fl=inf;
48          for(int i=t;i!=s;i=pre[i].to)
49              fl=min(fl,a[pre[i].e].w);
50          for(int i=t;i!=s;i=pre[i].to){
51              a[pre[i].e].w-=fl;
52              a[pre[i].e^1].w+=fl;
53          }
54          totcost+=dis[t]*fl;
55          ans+=fl;
56      }
57      return ans;
58  }
```

```

59 }flow;
60 void add(int x,int y,int w,int cost){
61     flow.ae(x,y,w,cost),flow.ae(y,x,0,-cost);
62 }
63 // 先 flow.init(n), 再 add 添边, 最后 flow.solve(s,t)

```

4.5 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 $O(n\log n)$

4.6 dijkstra

```

1 vector<pii> G[N];
2 int dis[N];
3 bool vis[N];
4 void dijkstra(int n,int pos){
5     rep(i,1,n) dis[i]=inf,vis[i]=0;
6     priority_queue<pii,vector<pii>,greater<pii>> pq;
7     pq.push({0,pos});
8     dis[pos]=0;
9     while(!pq.empty()){
10         pii top=pq.top(); pq.pop();
11         int u=top.sc,d=top.fr;
12         if(vis[u]) continue;
13         for(auto e:G[u]){
14             int v=e.fr;
15             if(dis[v]>d+e.sc) {
16                 dis[v]=d+e.sc;
17                 pq.push({dis[v],v});
18             }
19         }
20     }
21 }

```

4.7 2-Sat

```

1 //
2 struct TwoSat{
3     //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
4     //考虑到不会撤回操作 复杂度 $O(N)$ 
5     //编号从0开始
6     int n;
7     vector<int> G[2*N];
8     int S[2*N],c=0;
9     bool mark[2*N],ok;
10    void init(int n){
11        this->n=n,ok=1;
12        rep(i,0,n*2) mark[i]=0;
13        rep(i,0,n*2) G[i].clear();
14    }
15    bool dfs(int u){
16        if(mark[u^1]) return false;

```

```
17     if(mark[u]) return true;
18     S[++c]=u;
19     mark[u]=1;
20     for(auto v:G[u]){
21         if(!dfs(v)) return false;
22     }
23     return true;
24 }
25 //x,y代表抉择编号,val代表选0/1
26 void add(int x,int xval,int y,int yval){
27     x=x*2+xval,y=y*2+yval;
28     G[x^1].push_back(y);
29     G[y^1].push_back(x);
30 }
31 void solve(){
32     rep(i,0,n-1){
33         c=0;
34         if(!dfs(i*2)){
35             while(c) mark[S[c--]]=0;
36             if(!dfs(i*2+1)) ok=0;
37         }
38     }
39 }
40 }solver;
```

4.8 最大团

```
1 int mp[60][60];
2 int n;
3 int mx=0;//ans
4 vector<int> tmp;
5 bool check(int now){
6     bool fl=true;
7     for(int i=0;i<tmp.size();i++){
8         if(mp[tmp[i]][now]!=1||mp[now][tmp[i]]!=1){
9             fl=false ;
10            break;
11        }
12    }
13    return fl;
14 }
15 void dfs(int step){
16     if(step==n+1){
17         if(tmp.size()>mx){
18             mx=tmp.size();
19         }
20         return;
21     }
22     if(tmp.size()+n-step+1<=mx){
23         return;
24     }
25     if(check(step)){
26         tmp.push_back(step);
27         dfs(step+1);
```

```
28     tmp.pop_back();
29 }
30     dfs(step+1);
31 }
```

4.9 稳定婚姻

```
1 //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
2 int B[N][N],G[N][N];/*x对x的好感
3 vector<int> vec[N];
4 int pos[N],px[N],py[N],cur;
5 queue<int> que;
6 bool cmp(const int x,const int y){
7     return B[cur][x]>B[cur][y];
8 }
9 void stable_marriage(int n){
10     memset(pos,0,sizeof(pos));
11     memset(py,0,sizeof(py));
12     rep(i,1,n) vec[i].clear();
13     rep(i,1,n) rep(j,1,n) vec[i].push_back(j);
14     rep(i,1,n) {
15         cur=i;
16         sort(vec[i].begin(),vec[i].end(),cmp);
17     }
18     rep(i,1,n) que.push(i);
19     while(!que.empty()){
20         queue<int> tq;
21         while(!que.empty()){
22             int b=que.front(); que.pop();
23             int p=pos[b]++,g=vec[b][p];
24             if(!py[g]||G[g][b]>G[g][py[g]]){
25                 if(py[g]) tq.push(py[g]);
26                 py[g]=b,px[b]=g;
27             }
28             else tq.push(b);
29         }
30         que=tq;
31     }
32 }
```

4.10 点双连通分量

```
1 vector<int> G[N],bcc[N];
2 int pre[N]={0},low[N];
3 int cur,bcnt,bccno[N]; /*=0;
4 bool iscut[N];
5 stack<pii> S;
6 void dfs(int u,int fa){
7     low[u]=pre[u]=++cur;
8     int child=0;
9     for(auto v:G[u]){
10         if(v==fa) continue;
11         if(pre[v]&&pre[v]<pre[u]) {
```

```

12     //S.push({u,v});
13     low[u]=min(low[u],pre[v]);
14 }
15 else if(!pre[v]){
16     S.push({u,v});
17     child++;
18     dfs(v,u);
19     low[u]=min(low[u],low[v]);
20     if(low[v]>=pre[u]){
21         iscut[u]=1;
22         bcnt++; bcc[bcnt].clear();
23         while(1){
24             pii x=S.top(); S.pop();
25             if(bccno[x.fr]!=bcnt){
26                 bcc[bcnt].push_back(x.fr);
27                 bccno[x.fr]=bcnt;
28             }
29             if(bccno[x.sc]!=bcnt){
30                 bcc[bcnt].push_back(x.sc);
31                 bccno[x.sc]=bcnt;
32             }
33             if(x.fr==u&& x.sc==v) break;
34         }
35     }
36 }
37 }
38 if(fa==-1&& child==1) iscut[u]=0;

```

4.11 强连通分量

```

1 vector<int> G[N];
2 int sccno[N],dfn[N],low[N],scnt,tot;
3 stack<int> S;
4 void tarjan(int u){
5     dfn[u]=low[u]=++tot;
6     S.push(u);
7     for(auto v:G[u]){
8         if(sccno[v]) continue;
9         if(!dfn[v]){
10             tarjan(v);
11             low[u]=min(low[u],low[v]);
12         }
13         else low[u]=min(low[u],dfn[v]); //low[v] is also ok.
14     }
15     if(low[u]==dfn[u]){
16         scnt++;
17         int top=-1;
18         while(top!=u){
19             top=S.top(); S.pop();
20             sccno[top]=scnt;
21         }
22     }
23 }

```


4.12 最小树形图

```

1 struct Edge{int u,v,w,b};
2 vector<Edge> eset,E;
3 int fa[N],minw[N],id[N],top[N];
4 ll solve(int n,int rt){
5     ll res=0;
6     while(1){
7         int cnt=0;
8         rep(i,1,n) minw[i]=inf,id[i]=top[i]=0;
9         for(auto &e:eset){
10             if(e.u!=e.v&&e.w<minw[e.v]){
11                 minw[e.v]=e.w;
12                 fa[e.v]=e.u;
13             }
14         }
15         minw[rt]=0;
16         rep(i,1,n){
17             if(minw[i]==inf) return -1;
18             res+=minw[i];
19             for(int u=i;!id[u]&&u!=rt;u=fa[u]){
20                 if(top[u]==i){
21                     id[u]++;cnt;
22                     for(int v=fa[u];v!=u;v=fa[v]){
23                         id[v]=cnt;
24                     }
25                     break;
26                 }
27                 else top[u]=i;
28             }
29         }
30         if(!cnt) return res;
31         rep(i,1,n) if(!id[i]) id[i]++;cnt;
32         for(auto &e:eset){
33             e.w-=minw[e.v];
34             e.u=id[e.u],e.v=id[e.v];
35         }
36         n=cnt;
37         rt=id[rt];

```

4.13 KM

```

1 int n;
2 ll G[N][N];
3 int px[N],py[N],vx[N],vy[N],pre[N];
4 ll lx[N],ly[N],slack[N],d;
5 queue<int> que;
6 void upd(int v){
7     int t;
8     while(v){
9         t=px[pre[v]];
10        py[v]=pre[v];
11        px[pre[v]]=v;
12        v=t;

```

```

13     }
14 }
15 void bfs(int x){
16     rep(i,1,n) vy[i]=vx[i]=0,slack[i]=inf;
17     while(!que.empty()) que.pop();
18     que.push(x);
19     while(1){
20         while(!que.empty()){
21             int u=que.front(); que.pop();
22             vx[u]=1;
23             rep(v,1,n) if(!vy[v]){
24                 if(1x[u]+1y[v]-G[u][v]<slack[v]){
25                     slack[v]=1x[u]+1y[v]-G[u][v];
26                     pre[v]=u;
27                     if(!slack[v]){
28                         vy[v]=1;
29                         if(!py[v]) {upd(v); return;}
30                         else que.push(py[v]);
31                     }
32                 }
33             }
34         }
35         d=inf;
36         rep(i,1,n) if(slack[i]) d=min(d,slack[i]);
37         rep(i,1,n) {
38             if(vx[i]) 1x[i]-=d;
39             if(vy[i]) 1y[i]+=d;
40             else slack[i]-=d;
41         }
42         rep(i,1,n){
43             if(!vy[i]&&!slack[i]) {
44                 vy[i]=1;
45                 if(!py[i]) {upd(i); return;}
46                 else que.push(py[i]);
47             }
48         }
49     }
50 }
51 ll KM(){
52     ll res=0;
53     fill(1x+1,1x+n+1,-inf); memset(1y,0,sizeof(1y));
54     memset(px,0,sizeof(px)); memset(py,0,sizeof(py));
55     memset(pre,0,sizeof(pre));
56     rep(i,1,n) rep(j,1,n) 1x[i]=max(1x[i],G[i][j]);
57     rep(i,1,n) bfs(i);
58     rep(i,1,n) res+=G[i][px[i]];
59     return res;
60 }

```

4.14 差分约束

```

1 //介绍WIP
2 vector<pair<int,ll>> G[N];
3 bool inque[N];

```

```
4 int incnt[N];
5 ll dis[N];
6 bool bellman_ford(ll x){
7     queue<int> que;
8     rep(i,1,n) {
9         dis[i]=0,inque[i]=1,incnt[i]=0;
10        que.push(i);
11    }
12    while(!que.empty()){
13        int u=que.front(); que.pop();
14        inque[u]=0;
15        for(auto e:G[u]){
16            int v=e.fr;
17            //cout<<dis[u]<<" "<<e.sc<<" "<<dis[v]<<endl;
18            if(dis[u]+e.sc<dis[v]){
19                dis[v]=dis[u]+e.sc;
20                if(!inque[v]){
21                    inque[v]=1;
22                    que.push(v);
23                    if(++incnt[v]>n) return true;
24                }
25            }
26        }
27    }
28    return false;
29 }
```

4.15 一般图最大权匹配带花树 (hjt)

```
1 int n;
2 vector<int> G[N];
3 struct DSU { // join: d[x] = d[y], query: d[x] == d[y]
4     int a[N];
5     void init(int n) { iota(a, a + n + 1, 0); }
6     int fa(int x) { return a[x] == x ? x : a[x] = fa(a[x]); }
7     int &operator[](int x) { return a[fa(x)]; }
8 } d;
9 deque<int> q;
10 int mch[N],vis[N],dfn[N],fa[N],dcnt=0;
11 int lca(int x,int y){
12     dcnt++;
13     while(1){
14         if(x==0)swap(x,y); x=d[x];
15         if(dfn[x]==dcnt)return x;
16         else dfn[x]=dcnt,x=fa[mch[x]];
17     }
18 }
19 void shrink(int x,int y,int p){
20     while(d[x]!=p){
21         fa[x]=y; y=mch[x];
22         if(vis[y]==2)vis[y]=1,q.push_back(y);
23         if(d[x]==x)d[x]=p;
24         if(d[y]==y)d[y]=p;
25         x=fa[y];
26     }
```

```

26     }
27 }
28 bool match(int s){
29     d.init(n); fill(fa,fa+n+1,0);
30     fill(vis,vis+n+1,0); vis[s]=1;
31     q.assign(1,s);
32     while(!q.empty()){
33         int x=q.front(); q.pop_front();
34         for(auto p:G[x]){
35             if(d[x]==d[p] || vis[p]==2)continue;
36             if(!vis[p]){
37                 vis[p]=2; fa[p]=x;
38                 if(!mch[p]){
39                     for(int now=p,last,tmp;now;now=last){
40                         last=mch[tmp=fa[now]];
41                         mch[now]=tmp,mch[tmp]=now;
42                     }
43                     return 1;
44                 }
45                 vis[mch[p]]=1; q.push_back(mch[p]);
46             }
47             else if(vis[p]==1){
48                 int l=lca(x,p);
49                 shrink(x,p,l);
50                 shrink(p,x,l);
51             }
52         }
53     }
54     return 0;
55 }

```

5 计算几何

5.1 基础点和线

```

1 struct point{
2     double x,y;
3     point(double _x=0,double _y=0){ x=_x,y=_y; }
4     point operator+ (const point& a) const{ return point(x+a.x,y+a.y); }
5     point operator- (const point& a) const{ return point(x-a.x,y-a.y); }
6     point operator* (double a) const{ return point(x*a,y*a); }
7 }; // 点
8 struct line{
9     point s,e;
10    line(point a,point b){ s=a,e=b; }
11    line(){ }
12 }; // 线
13 double getdis(point a, point b){
14     double xx=a.x-b.x,yy=a.y-b.y;
15     return sqrt(xx*xx+yy*yy);
16 } // 两点距离
17 double multi(point a,point b,point c){
18     double xa,ya,xb,yb;

```

```

19     xa=b.x-a.x; ya=b.y-a.y;
20     xb=c.x-b.x; yb=c.y-b.y;
21     return xa*xb+ya*yb;
22 } // 点乘
23 double cross(point a,point b,point c){
24     double xa, ya, xb, yb;
25     xa=b.x-a.x; ya=b.y-a.y;
26     xb=c.x-a.x; yb=c.y-a.y;
27     return xa*yb-xb*ya;
28 } // 叉乘
29 int judgec(line a,line b){
30     if (max(a.s.x,a.e.x)>=min(b.s.x,b.e.x) &&
31         max(a.s.y,a.e.y)>=min(b.s.y,b.e.y) &&
32         max(b.s.x,b.e.x)>=min(a.s.x,a.e.x) &&
33         max(b.s.y,b.e.y)>=min(a.s.y,a.e.y) &&
34         cross(a.s,b.s,b.e)*cross(a.e,b.s,b.e)<=0
35         && cross(b.s,a.s,a.e)*cross(b.e,a.s,a.e)<=0
36     ) return 1;
37     else return 0;
38 } // 判断线段是否相交
39 point getpoi(point a,point b,point c,point d){
40     double u=cross(a,b,c),v=cross(b,a,d);
41     return point((c.x*v+d.x*u)/(u+v),(c.y*v+d.y*u)/(u+v));
42 } // 求交点

```

5.2 三点求圆心

```

1 struct point{
2     double x;
3     double y;
4 };
5
6 point cal(point a,point b,point c){
7     double x1 = a.x;double y1 = a.y;
8     double x2 = b.x;double y2 = b.y;
9     double x3 = c.x; double y3 = c.y;
10    double a1 = 2*(x2-x1); double a2 = 2*(x3-x2);
11    double b1 = 2*(y2-y1); double b2 = 2*(y3-y2);
12    double c1 = x2*x2 + y2*y2 - x1*x1 - y1*y1;
13    double c2 = x3*x3 + y3*y3 - x2*x2 - y2*y2;
14    double rx = (c1*b2-c2*b1)/(a1*b2-a2*b1);
15    double ry = (c2*a1-c1*a2)/(a1*b2-a2*b1);
16    return point{rx,ry};
17 }

```

5.3 拉格朗日插值

```

1 namespace polysum {
2 #define rep(i,a,n) for (int i=a;i<n;i++)
3 #define per(i,a,n) for (int i=n-1;i>=a;i--)
4 const int D = 1010000; ///可能要用到的最高次
5 LL a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D];
6 LL powmod(LL a,LL b) {

```

```

7   LL res = 1;
8   a %= mod;
9   assert(b >= 0);
10
11  for (; b; b >>= 1) {
12      if (b & 1)
13          res = res * a % mod;
14
15      a = a * a % mod;
16  }
17
18  return res;
19 }
20
21 ///函数用途: 给出数列的 (d+1) 项, 其中d为最高次方项
22 ///求出数列的第n项, 数组下标从0开始
23 LL calcn(int d, LL *a, LL n) { /// a[0].. a[d] a[n]
24     if (n <= d)
25         return a[n];
26
27     p1[0] = p2[0] = 1;
28     rep(i, 0, d + 1) {
29         LL t = (n - i + mod) % mod;
30         p1[i + 1] = p1[i] * t % mod;
31     }
32     rep(i, 0, d + 1) {
33         LL t = (n - d + i + mod) % mod;
34         p2[i + 1] = p2[i] * t % mod;
35     }
36     LL ans = 0;
37     rep(i, 0, d + 1) {
38         LL t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
39
40         if ((d - i) & 1)
41             ans = (ans - t + mod) % mod;
42         else
43             ans = (ans + t) % mod;
44     }
45     return ans;
46 }
47 void init(int M) {///用到的最高次
48     f[0] = f[1] = g[0] = g[1] = 1;
49     rep(i, 2, M + 5) f[i] = f[i - 1] * i % mod;
50     g[M + 4] = powmod(f[M + 4], mod - 2);
51     per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod; ///费马小定理筛逆元
52 }
53
54 ///函数用途: 给出数列的 (m+1) 项, 其中m为最高次方
55 ///求出数列的前 (n-1) 项的和 (从第0项开始)
56 LL polysum(LL m, LL *a, LL n) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]
57     for (int i = 0; i <= m; i++)
58         b[i] = a[i];
59
60     ///前n项和, 其最高次幂加1
61     b[m + 1] = calcn(m, b, m + 1);

```

```

62     rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
63     return calcn(m + 1, b, n - 1);
64 }
65 LL qpolysum(LL R, LL n, LL *a, LL m) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
66     if (R == 1)
67         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儿子那里拿几个
10                 dp[now][j] = max(dp[now][j], dp[now][j - k] + dp[v][k]);
11                 //状态转移
12             }
13         }
14     }
    
```

15 }

6.2 树上启发式合并

通过保留重儿子贡献降低复杂度

```
1  const int MAXN = 1e5+5;
2  const int MAXM = 1e5+5;
3  int sz[MAXN],son[MAXN];//子树大小和重儿子
4  //int ans[MAXN];//存最多的颜色和编号和,这题没必要
5  int c[MAXN];//每个点的颜色
6  int cnt[MAXM];//存每种颜色的点数
7  ll bhh[MAXN],bsum;//题目要求颜色的编号和
8  int maxx;//临时变量,用来找最大颜色数量的
9  int n;
10 vector<int> e[MAXN];
11 void init(int now,int pre){//预处理找重儿子
12     sz[now] = 1;son[now] = -1;
13     for(auto v:e[now]){
14         if(v==pre) continue;
15         init(v,now);
16         sz[now]+=sz[v];
17         if(son[now]==-1||sz[v]>sz[son[now]]) son[now]=v;
18     }
19 }
20 void dfs1(int now,int pre,int val,int rt){//rt是dfs里面当前在统计的子树根
21 //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
22     cnt[c[now]]+=val;
23     if(val==1){
24         if(cnt[c[now]]>maxx) maxx=cnt[c[now]],bsum=c[now];
25         else if(cnt[c[now]]==maxx) bsum+=c[now];
26     }
27     //遍历子节点的时候看到什么颜色就检查这个颜色
28     for(auto v:e[now]) if(v!=pre)dfs1(v,now,val,rt);
29 }
30 void dfs(int now,int pre,int opt){//opt为1代表要清空
31     for(auto v:e[now]){
32         if(v!=son[now]&&v!=pre) dfs(v,now,1);//先轻儿子,需要删去
33     }
34     if(son[now]!=-1) dfs(son[now],now,0);//处理重儿子
35     //此时计算轻儿子的贡献,重儿子的在cnt里面祖传上来了
36     for(auto v:e[now]){
37         if(v!=pre&&v!=son[now]) dfs1(v,now,1,now);//统计轻儿子上的结果
38     }
39     cnt[c[now]]++;//也要加上当前节点
40     if(cnt[c[now]]>maxx) maxx=cnt[c[now]],bsum=c[now];
41     //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
42     else if(cnt[c[now]]==maxx) bsum+=c[now];
43     bhh[now] = bsum;//记录结果
44     //减去轻儿子的贡献
45     if(opt){
46         cnt[c[now]]--;
47         for(auto v:e[now])
48             if(v!=pre) dfs1(v,now,-1,now);
49         maxx = bsum = 0;//删去轻儿子这颗子树对bsum和maxx的贡献
```



```

50     }
51 }
52 int rt = 1; // 题目默认1为根节点
53 int main(){
54     cin>>n;
55     rep(i,1,n) cin>>c[i];
56     int u,v;
57     rep(i,1,n-1){
58         cin>>u>>v;
59         e[u].push_back(v);
60         e[v].push_back(u);
61     }
62     init(rt,-1);
63     dfs(rt,-1,1);
64     rep(i,1,n){
65         cout<<bhh[i];
66         if(i!=n) cout<<' ';
67     }
68     cout<<endl;
69 }
70 // 没有修改只有查询的树形问题
71 // 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）。
72 // 洛谷CF600E Lomsat gelral
73 // https://www.luogu.com.cn/problem/CF600E

```

6.3 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。

```

1 #define ls (x<<1)
2 #define rs (x<<1|1) // 这种方法感觉还挺好的
3
4 int cansel_sync=(ios::sync_with_stdio(0),cin.tie(0),0);
5 const int MAXN = 2e5+5; // 这里要开n的两倍
6 // 线结构体
7 struct Line{
8     ll l,r,h;
9     int qz; // 记录位置和权值
10     bool operator < (Line &rhs){
11         return h < rhs.h;
12     }
13 }line[MAXN];
14 int n;
15 ll x1,y1,x2,y2;
16 ll X[MAXN];
17 // 线段树
18 struct Segt{
19     int l,r; // 是X的下标,即离散化后的
20     int sum; // sum是被完全覆盖的次数
21     ll len; // len是区间内被盖住的长度
22     // 因为每次查询都是查询根节点,所以这边不需要懒惰标记
23 }t[MAXN<<3]; // 一个边有两个点,所以这里要开8倍
24 void build(int x,int l,int r){

```

```

25     t[x].l = l; t[x].r = r;
26     t[x].len = t[x].sum = 0;
27     if(l==r) return; //到了叶子节点
28     int mid = (l+r)>>1;
29     build(ls, l, mid);
30     build(rs, mid+1, r);
31 }
32 void push_up(int x){
33     int l = t[x].l, r = t[x].r;
34     if(t[x].sum) t[x].len = X[r+1]-X[l]; //x的区间是X[l]到X[r+1]-1
35     else t[x].len = t[ls].len + t[rs].len; //合并儿子的信息
36 }
37 void update(int x, int L, int R, int v){ //这里的LR存的是实际值
38     //这里如果是线段L,R,线段树上是L到R-1的部分维护
39     int l = t[x].l, r = t[x].r;
40     if(X[r+1]<=L || R<=X[l]) return; //加等于, 不然会搞到无辜的线
41     if(L<=X[l] && X[r+1]<=R){
42         t[x].sum += v; //修改覆盖次数
43         push_up(x);
44         return;
45     }
46     update(ls, L, R, v);
47     update(rs, L, R, v);
48     push_up(x);
49 }
50 int main(){
51     cin>>n;
52     rep(i, 1, n){
53         cin>>x1>>y1>>x2>>y2;
54         X[2*i-1] = x1, X[2*i] = x2; //一会儿离散化要用的, 这里存实际值
55         line[2*i-1] = Line{x1, x2, y1, 1}; //开始的线
56         line[2*i] = Line{x1, x2, y2, -1}; //结束的线
57     }
58     n<<=1; //line的数量是四边形数量的2倍
59     sort(line+1, line+1+n);
60     sort(X+1, X+1+n);
61     int tot = unique(X+1, X+1+n)-(X+1); //去除重复相邻元素, 并且tot记录总数
62     build(1, 1, tot-1); //为什么是tot-1?
63     //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
64     ll res = 0;
65     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
66         update(1, line[i].l, line[i].r, line[i].qz); //扫描线加入线段树
67         res += t[1].len*(line[i+1].h-line[i].h);
68     }
69     cout<<res<<endl;
70 }

```

6.4 并查集系列

6.4.1 普通并查集

带路径压缩, $O(1)$ 复杂度

```

1 int fa[maxn];
2 int find(int x){ if(fa[x]^x) return fa[x]=find(fa[x]); return x; }

```

```
3 void merge(int a,int b){fa[find(a)]=find(b);}
```

6.4.2 按秩合并并查集

```
1 int fa[maxn];
2 int dep[maxn];
3 int find(int x){int now=x; while(fa[now]^now)now=fa[now];return now;}
4 void merge(int a,int b){
5     int l=find(a),r=find(b);
6     if(l==r) return;
7     if(dep[l]>dep[r])swap(l,r);
8     fa[l]=r;
9     dep[r]+=dep[l]==dep[r];
10 }
```

6.4.3 可持久化并查集

```
1 struct chair_man_tree{
2     struct node{
3         int lson,rson;
4     }tree[maxn<<5];
5     int tail=0;
6     int tail2=0;
7     int fa[maxn<<2];
8     int depth[maxn<<2];
9     inline int getnew(int pos){
10         tree[++tail]=tree[pos];
11         return tail;
12     }
13     int build(int l,int r){
14
15         if(l==r){
16             fa[++tail2]=l;
17             depth[tail2]=1;
18             return tail2;
19         }
20         int now=tail++;
21         int mid=(l+r)>>1;
22         tree[now].lson=build(l,mid);
23         tree[now].rson=build(mid+1,r);
24         return now;
25     }
26     int query(int pos,int l,int r,int qr){
27         if(l==r)
28             return pos;
29         int mid=(l+r)>>1;
30         if(qr<=mid)
31             return query(tree[pos].lson,l,mid,qr);
32         else return query(tree[pos].rson,mid+1,r,qr);
33     }
34     int update(int pos,int l,int r,int qr,int val){
35         if(l==r){
36             depth[++tail2]=depth[pos];
```

```

37         fa[tail2]=val;
38         return tail2;
39     }
40     int now=getnew(pos);
41     int mid=(l+r)>>1;
42     if(mid>=qr)
43         tree[now].lson=update(tree[now].lson,l,mid,qr,val);
44     else tree[now].rson=update(tree[now].rson,mid+1,r,qr,val);
45     return now;
46 }
47 int add(int pos,int l,int r,int qr){
48     if(l==r){
49         depth[++tail2]=depth[pos]+1;
50         fa[tail2]=fa[pos];
51         return tail2;
52     }
53     int now=getnew(pos);
54     int mid=(l+r)>>1;
55     if(mid>=qr)
56         tree[now].lson=add(tree[now].lson,l,mid,qr);
57     else tree[now].rson=add(tree[now].rson,mid+1,r,qr);
58     return now;
59 }
60 int getfa(int root,int qr){
61     int t=fa[query(root,1,n,qr)];
62     if(qr==t)
63         return qr;
64     else return getfa(root,t);
65 }
66 }t;

```

6.4.4 可撤销并查集

```

1 struct DSU{
2     int *rnk,*f,top;
3     pii *stk;
4     DSU(int n=N){
5         rnk=new int[n],f=new int[n];
6         stk=new pii[n];
7     }
8     void init(int n){
9         rep(i,1,n) rnk[i]=0,f[i]=i;
10        top=0;
11    }
12    int getf(int u){return f[u]==u?u:getf(f[u]);}
13    void unite(int u,int v){
14        u=getf(u),v=getf(v);
15        if(u==v) return;
16        if(rnk[u]>rnk[v]){
17            stk[++top]={v,rnk[v]};
18            f[v]=u;
19        }
20        else{
21            stk[++top]={u,rnk[u]};

```

```
22         f[u]=v;
23         if(rnk[u]==rnk[v]) rnk[v]++;
24     }
25 }
26 void undo(int pos){
27     while(top>pos){
28         int u=stk[top].fr,r=stk[top].sc;
29         f[u]=u,rnk[u]=r;
30         --top;
31     }
32 }
33 }dsu;
```

6.4.5 ETT 维护动态图连通性

待补

6.5 平衡树系列

6.5.1 fhq_treap

无旋 treap, 可持久化, 常数大

```
1 mt19937 rnd(514114);
2 struct fhq_treap{
3     struct node{
4         int l, r;
5         int val, key;
6         int size;
7     } fhq[maxn];
8     int cnt, root;
9     inline int newnode(int val){
10         fhq[++cnt].val = val;
11         fhq[cnt].key = rnd();
12         fhq[cnt].size = 1;
13         fhq[cnt].l = fhq[cnt].r = 0;
14         return cnt;
15     }
16     inline void pushup(int now){
17         fhq[now].size = fhq[fhq[now].l].size + fhq[fhq[now].r].size + 1;
18     }
19     void split(int now, int val, int &x, int &y){
20         if (!now){
21             x = y = 0;
22             return;
23         }
24         else if (fhq[now].val <= val){
25             x = now;
26             split(fhq[now].r, val, fhq[now].r, y);
27         }
28         else{
29             y = now;
30             split(fhq[now].l, val, x, fhq[now].l);
31         }
32         pushup(now);
33     }
```

```
33 }
34 int merge(int x, int y){
35     if (!x || !y)
36         return x + y;
37     if (fhq[x].key > fhq[y].key){
38         fhq[x].r = merge(fhq[x].r, y);
39         pushup(x);
40         return x;
41     }else{
42         fhq[y].l = merge(x, fhq[y].l);
43         pushup(y);
44         return y;
45     }
46 }
47 inline void insert(int val){
48     int x, y;
49     split(root, val, x, y);
50     root = merge(merge(x, newnode(val)), y);
51 }
52 inline void del(int val){
53     int x, y, z;
54     split(root, val - 1, x, y);
55     split(y, val, y, z);
56     y = merge(fhq[y].l, fhq[y].r);
57     root = merge(merge(x, y), z);
58 }
59 inline int getrk(int num){
60     int x, y;
61     split(root, num - 1, x, y);
62     int ans = fhq[x].size + 1;
63     root = merge(x, y);
64     return ans;
65 }
66 inline int getnum(int rank){
67     int now=root;
68     while(now)
69     {
70         if(fhq[fhq[now].l].size+1==rank)
71             break;
72         else if(fhq[fhq[now].l].size>=rank)
73             now=fhq[now].l;
74         else{
75             rank-=fhq[fhq[now].l].size+1;
76             now=fhq[now].r;
77         }
78     }
79     return fhq[now].val;
80 }
81 inline int pre(int val){
82     int x, y, ans;
83     split(root, val - 1, x, y);
84     int t = x;
85     while (fhq[t].r)
86         t = fhq[t].r;
87     ans = fhq[t].val;
```

```
88     root = merge(x, y);
89     return ans;
90 }
91 inline int aft(int val){
92     int x, y, ans;
93     split(root, val, x, y);
94     int t = y;
95     while (fhq[t].l)
96         t = fhq[t].l;
97     ans = fhq[t].val;
98     root = merge(x, y);
99     return ans;
100 }
101 } tree;
```

6.5.2 替罪羊树

```
1 struct node
2 {
3     int l, r, val;
4     int size, fact;
5     bool exsit;
6 };
7 class tzy_tree
8 {
9 public:
10     double alpha=0.75;
11     node tzy[1100000];
12     int root=0,cnt=0;
13     vector<int> tt;
14     inline void newnode(int &now,int v){
15         now=++cnt;
16         tzy[now].val=v;
17         tzy[now].l=tzy[now].r=0;
18         tzy[now].size=tzy[now].fact=tzy[now].exsit=1;
19     }
20     inline bool imbalanced(int now){
21         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)
22             return true;
23         return false;
24     }
25     void zhongxu(int now){
26         if(!now)
27             return;
28         zhongxu(tzy[now].l);
29         if(tzy[now].exsit)
30             tt.push_back(now);
31         zhongxu(tzy[now].r);
32     }
33     void lift(int l,int r,int &now){
34         if(l==r){
35             now=tt[l];
36             tzy[now].l=tzy[now].r=0;
```

```

37         tzy[now].fact=tzy[now].size=1;
38         return;
39     }
40     int m=(l+r)>>1;
41     while(1<m&&tzy[tt[m]].val==tzy[tt[m-1]].val)
42         m--;
43     now=tt[m];
44     if(1<m) lift(1,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--;
100        check(root,now);
101        return;
102    }
103    tzy[now].fact--;
104    if(val<tzy[now].val)
105        erase(tzy[now].l,val);
106    else erase(tzy[now].r,val);
107 }
108 int getrank(int val){
109     int now=root,rk=1;
110     while(now){
111         if(val<=tzy[now].val)
112             now=tzy[now].l;
113         else rk+=tzy[now].exsit+tzy[tzy[now].l].fact,now=tzy[now].r;
114     }
115     return rk;
116 }
117 int getnum(int rk){
118     int now=root;
119     while(now){
120         if(tzy[now].exsit&& tzy[tzy[now].l].fact+tzy[now].exsit==rk)
121             break;
122         else if(tzy[tzy[now].l].fact>=rk){
123             now=tzy[now].l;
124         }else{
125             rk-=tzy[tzy[now].l].fact+tzy[now].exsit;
126             now=tzy[now].r;
127         }
128     }
129     return tzy[now].val;
130 }
131 }tree;

```

6.5.3 文艺平衡树

处理区间翻转问题

```

1 struct Node{
2     int l,r;
3     int val,key;
4     int size;
5     bool reverse;//旋转的懒惰标记
6 }fhq[MAXN];
7 int cnt,root;
8 mt19937 rnd(233);
9 vector<int>res;//因为行的末尾不能有空格所以我先用一个vector存着
10 inline int newnode(int val){

```

```
11     fhq[++cnt].val = val;
12     fhq[cnt].key = rnd();
13     fhq[cnt].size=1;
14     return cnt;
15 }
16 inline void update(int now){
17     fhq[now].size = fhq[fhq[now].l].size + fhq[fhq[now].r].size+1;
18 }
19 inline void pushdown(int now){
20     //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
21     swap(fhq[now].l,fhq[now].r);
22     fhq[fhq[now].l].reverse^=1;//儿子的懒惰标记0变1,1变0
23     fhq[fhq[now].r].reverse^=1;
24     fhq[now].reverse = 0;//之前疑惑这里为什么不用取反^1的
25     //其实是因为如果now没有标记reverse,就不用执行pushdown
26 }
27 void split(int now,int siz,int &x,int &y){//按照大小分裂
28     //有点类似取第k大getnum的操作
29     if(!now){x=y=0;return;}
30     if(fhq[now].reverse) pushdown(now);//之后分裂操作可能碰到儿子,所以要pushdown
31     if(fhq[fhq[now].l].size<siz){
32         x = now;
33         split(fhq[now].r,siz-fhq[fhq[now].l].size-1,fhq[now].r,y);
34     }
35     else{
36         y = now;
37         split(fhq[now].l,siz,x,fhq[now].l);
38     }
39     update(now);
40 }
41 int merge(int x,int y){
42     if(!x||!y) return x+y;
43     if(fhq[x].key<fhq[y].key){//这里其实用什么符号都行
44         if(fhq[x].reverse) pushdown(x);
45         fhq[x].r = merge(fhq[x].r,y);
46         update(x);
47         return x;
48     }
49     else{
50         if(fhq[y].reverse) pushdown(y);
51         fhq[y].l = merge(x,fhq[y].l);
52         update(y);
53         return y;
54     }
55 }
56 void reverse(int l,int r){//区间反转
57     //拆成三段,(1,l-1),(l,r),(r+1,n);
58     int x,y,z;
59     split(root,l-1,x,y);
60     split(y,r-l+1,y,z);
61     fhq[y].reverse^=1;
62     root=merge(merge(x,y),z);
63 }
64 void ldr(int now){//最后用中序遍历找到结果
65     if(!now) return;
```

```

66     if(fhq[now].reverse) pushdown(now);
67     ldr(fhq[now].l);
68     res.push_back(fhq[now].val); // 推入结果的序列
69     ldr(fhq[now].r);
70 }
71 int main(){
72     int n,m;
73     cin>>n>>m;
74     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
75     int l,r;
76     while(m--){
77         cin>>l>>r;
78         reverse(l,r);
79     }
80     res.clear();
81     ldr(root);
82     int siz = res.size();
83     rep(i,0,siz-1){
84         cout<<res[i];
85         if(i!=siz-1) cout<<' ';
86     }
87     cout<<endl;
88 }

```

6.5.4 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）

```

1  rbset(pii(int, int)) seg[maxn << 2];
2  int date[maxn];
3  unsigned int cnt = 0;
4  void build(int pos, int l, int r)
5  {
6      for (int i = l; i <= r; i++)
7          seg[pos].insert(MP(date[i], ++cnt));
8      if (l == r)
9          return;
10     int mid = (l + r) >> 1;
11     build(lson(pos), l, mid);
12     build(rson(pos), mid + 1, r);
13 }
14 void update(int pos, int l, int r, int x, int v)
15 {
16     seg[pos].erase(seg[pos].lower_bound(MP(date[x], 0)));
17     seg[pos].insert(MP(v, ++cnt));
18     if (l == r)
19         return;
20     int mid = (l + r) >> 1;

```

```

21     if (x <= mid)
22         update(lson(pos), l, mid, x, v);
23     else
24         update(rson(pos), mid + 1, r, x, v);
25 }
26 int getrk(int pos, int l, int r, int ql, int qr, int k)
27 {
28     if (ql <= l && qr >= r)
29         return seg[pos].order_of_key(MP(k, 0));
30     int ans = 0;
31     int mid = (l + r) >> 1;
32     if (ql <= mid)
33         ans += getrk(lson(pos), l, mid, ql, qr, k);
34     if (qr >= mid+1)
35         ans += getrk(rson(pos), mid + 1, r, ql, qr, k);
36     return ans;
37 }
38 int pre(int pos, int l, int r, int ql, int qr, int k)
39 {
40     if (ql <= l && qr >= r)
41     {
42         auto tt = seg[pos].lower_bound(MP(k, 0));
43         if (tt == seg[pos].begin())
44             return -2147483647;
45         return (--tt)->first;
46     }
47     int ans = -2147483647;
48     int mid = l + r >> 1;
49     if (ql <= mid)
50         MAX(ans, pre(lson(pos), l, mid, ql, qr, k));
51     if (qr > mid)
52         MAX(ans, pre(rson(pos), mid + 1, r, ql, qr, k));
53     return ans;
54 }
55 int aft(int pos, int l, int r, int ql, int qr, int k){
56     if (ql <= l && qr >= r)
57     {
58         auto tt = seg[pos].lower_bound(MP(k + 1, 0));
59         if (tt == seg[pos].end())
60             return 2147483647;
61         return tt->first;
62     }
63     int ans = 2147483647;
64     int mid = (l + r) >> 1;
65     if (ql <= mid)
66         MIN(ans, aft(lson(pos), l, mid, ql, qr, k));
67     if (qr > mid)
68         MIN(ans, aft(rson(pos), mid + 1, r, ql, qr, k));
69     return ans;
70 }

```

6.6 KD tree

```

1 inline void updmin(int &x,int y){x=min(x,y);}

```

```
2 inline void updmax(int &x,int y){x=max(x,y);}
3 int Dim;
4 struct KDT{
5     int root;
6     int dim,top=0,tot=0,*rub;
7     struct point{
8         int x[2],w;
9         bool operator<(const point&b){
10             return x[Dim]<b.x[Dim];
11         }
12     }*p;
13     struct node{
14         int mi[2],mx[2],sum=0,ls,rs,sz=0;
15         point p;
16     }*tr;
17     int newnode(){
18         if(top) return rub[top--];
19         else return ++tot;
20     }
21     void up(int u){
22         int l=tr[u].ls,r=tr[u].rs;
23         rep(i,0,1){
24             tr[u].mi[i]=tr[u].mx[i]=tr[u].p.x[i];
25             if(l){
26                 updmin(tr[u].mi[i],tr[l].mi[i]);
27                 updmax(tr[u].mx[i],tr[l].mx[i]);
28             }
29             if(r){
30                 updmin(tr[u].mi[i],tr[r].mi[i]);
31                 updmax(tr[u].mx[i],tr[r].mx[i]);
32             }
33         }
34         tr[u].sum=tr[l].sum+tr[r].sum+tr[u].p.w;
35         tr[u].sz=tr[l].sz+tr[r].sz+1;
36     }
37     int build(int l,int r,int dim){
38         if(l>r) return 0;
39         int mid=l+r>>1,u=newnode();
40         Dim=dim; nth_element(p+l,p+mid,p+r+1);
41         tr[u].p=p[mid];
42         tr[u].ls=build(l,mid-1,dim^1),tr[u].rs=build(mid+1,r,dim^1);
43         up(u);
44         return u;
45     }
46     void pia(int u,int num){//传统?
47         int l=tr[u].ls,r=tr[u].rs;
48         if(l) pia(l,num);
49         p[tr[l].sz+1+num]=tr[u].p,rub[++top]=u;
50         if(r) pia(r,num+1+tr[l].sz);
51     }
52     void balance(int &u,int dim){
53         if(tr[u].sz*0.75<tr[tr[u].ls].sz||
54            tr[u].sz*0.75<tr[tr[u].rs].sz){
55             pia(u,0); u=build(1,tr[u].sz,dim);
56         }
```

```

57 }
58 void insert(int &u, point p, int dim){
59     if(!u) {
60         u=newnode(), tr[u].p=p;
61         tr[u].ls=tr[u].rs=0;
62         up(u); return; //待修改
63     }
64     if(p.x[dim]<=tr[u].p.x[dim]) insert(tr[u].ls, p, dim^1);
65     else insert(tr[u].rs, p, dim^1);
66     up(u); balance(u, dim);
67 }
68 int in(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2){
69     return x1>=X1&&x2<=X2&&y1>=Y1&&y2<=Y2;
70 } //左是否在右内
71 int out(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2){
72     return x2<X1 || x1>X2 || y2<Y1 || y1>Y2;
73 } //左是否在右外
74 int query(int u, int x1, int y1, int x2, int y2){
75     if(!u) return 0;
76     auto mx=tr[u].mx, mi=tr[u].mi, x=tr[u].p.x;
77     int res=0;
78     if(in(mi[0], mi[1], mx[0], mx[1], x1, y1, x2, y2)) res+=tr[u].sum;
79     if(out(mi[0], mi[1], mx[0], mx[1], x1, y1, x2, y2)) return 0;
80     if(in(x[0], x[1], x[0], x[1], x1, y1, x2, y2)) res+=tr[u].p.w;
81     res+=query(tr[u].ls, x1, y1, x2, y2)+query(tr[u].rs, x1, y1, x2, y2);
82     return res;
83 }
84 KDT(int maxn=1e6+10){
85     tr=new node[maxn], p=new point[maxn];
86     rub = new int[maxn], root=0;
87 }
88 void insert(int x, int y, int k){insert(root, (point){x, y, k}, 0);}
89 int query(int x1, int y1, int x2, int y2){return query(root, x1, y1, x2, y2);}
90 };

```

7 字符串

7.1 KMP

```

1  const int MAXN = 2e6+5;
2  int pi[MAXN]; //MAXN记得开大一点, 因为这里要存到m+n+1长度的
3  vector<int> res; //储存答案
4
5  void getpi(const string &s){ //求s的前缀函数
6      pi[0]=0;
7      int j=0;
8      rep(i, 1, s.length()-1){
9          while(j>0&&s[i]!=s[j]) j=pi[j-1]; //找到合适且最长的j
10         if(s[i]==s[j]) j++; //能成功匹配的情况
11         pi[i]=j;
12     }
13 }
14

```

```

15 void kmp(string s,string t){ //在主串t中找模式串s
16     getpi(s+'#'+t);
17     int n=(int)s.length(),m=(int)t.length();
18     rep(i,n+1,m+n+1-1)
19         if(pi[i]==n) res.push_back(i-2*s.size()); //i-2n计算得左端点
20 }

```

7.2 AC 自动机

```

1  const int MAXN = 1e5+5;
2  int jdbh[MAXN]; //记录第i个模式串对应的节点编号
3  int cntcx[MAXN]; //记录第i个模式串出现的次数
4  inline int idx(char c){return c-'a';}
5  struct Node{
6      int son[26],flag,fail; //cnt记录次数,flag记录编号
7      void clr(){
8          memset(son,0,sizeof(son));
9          flag=0;
10     }
11 }trie[MAXN*10];
12 int n,cntt; //cntt记录总点数
13 string s,ms[166];
14 int maxx;
15 queue<int>q;
16 inline void insert(string &s,int num){
17     int siz = s.size(),v,u=1;
18     rep(i,0,siz-1){
19         v = idx(s[i]);
20         if(!trie[u].son[v]){trie[u].son[v] = ++cntt;trie[cntt].clr();}
21         u = trie[u].son[v];
22     }
23     trie[u].flag = num; //标记为单词,flag记录编号
24     //保证每个模式串只出现一次
25     cntcx[num] = 0;
26     jdbh[num] = u; //记录当前单词对应的节点编号
27 }
28 inline void getfail(){
29     rep(i,0,25) trie[0].son[i] = 1;
30     trie[0].flag = 0;
31     q.push(1);
32     trie[1].fail = 0;
33     int u,v,ufail;
34     while(!q.empty()){
35         u = q.front();q.pop();
36         rep(i,0,25){
37             v = trie[u].son[i];
38             ufail = trie[u].fail;
39             if(!v){trie[u].son[i]=trie[ufail].son[i];continue;} //画好一条跳fail的路
40             trie[v].fail = trie[ufail].son[i];
41             q.push(v);
42         }
43     }
44 }
45 inline void query(string &s){

```

```

46  int siz = s.size(),u = 1,v,k;
47  rep(i,0,siz-1){
48      v = idx(s[i]);
49      k = trie[u].son[v];
50      while(k){
51          if(trie[k].flag){
52              cntcx[trie[k].flag]++;//计数
53              maxx = max(maxx,cntcx[trie[k].flag]);
54          }
55          k = trie[k].fail;//跳fail
56      }
57      u = trie[u].son[v];//这一句其实也有跳fail的功能，很精妙
58  }
59 }
60 inline void solve(){
61     cntt = 1;
62     trie[0].clr();
63     trie[1].clr();
64     rep(i,1,n){
65         cin>>ms[i];
66         insert(ms[i],i);
67     }
68     getfail();
69     cin>>s;
70     maxx = 0;
71     query(s);
72     cout<<maxx<<endl;
73     rep(i,1,n){
74         if(cntcx[i]==maxx) cout<<ms[i]<<endl;
75     }
76 }

```

7.3 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 $O(n \log n)$

```

1  void solve(){
2      limit = 1,l=0;
3      cin>>n>>m;
4      cin>>s1>>s2;

```



```

5   rep(i,0,n-1) B[i].x = s1[i]-'a'+1;
6   rep(i,0,m-1) A[i].x = s2[i]-'a'+1;
7   double T = 0;
8   //T = sigma A[i]^A[i] i=0~m-1
9   rep(i,0,m-1) T += A[i].x*A[i].x;
10  //f[x] = sigma B[i]^B[i] i=0~x
11  f[0] = B[0].x*B[0].x;
12  rep(i,1,n-1) f[i] = f[i-1]+B[i].x*B[i].x;
13  //g[x] = S[i]*B[j] i+j==x
14  reverse(A,A+m); //S = A.reverse
15  //FFT预处理
16  while(limit<=n+m-2) limit<<=1,l++;
17  rep(i,0,limit-1)
18      r[i]= ( r[i>>1]>>1 )| ( (i&1)<<(l-1) );
19
20  FFT(A,1);FFT(B,1);
21  rep(i,0,limit) A[i]=A[i]*B[i];
22  FFT(A,-1);
23  rep(i,0,n-1) g[i] = (int)(A[i].x/limit+0.5); //四舍五入
24
25  //T + f(x) - f(x-m) - 2g(x);
26  double tmp;
27  rep(x,m-1,n-1){
28      tmp = T+f[x]-2*g[x];
29      if(x!=m-1) tmp -= f[x-m];
30      //cout<<tmp<<' ';
31      if(fabs(tmp)<eps) cout<<x-(m-1)+1<<endl; //输出匹配上的位置
32  }
33  cout<<endl;
34  }

```

7.4 字符串哈希

7.4.1 快速取子串哈希值

```

1  const int b = 131; //推荐的base, 可以选其他质数
2  void init(int n){ //初始化
3      pw[0] = 1;
4      for (int i = 1; i <= n; i ++ ) {
5          h[i] = h[i-1]*b + str[i]; //做每个前缀的哈希值
6          pw[i] = pw[i-1]*b; //预处理b^k的值
7      }
8  }
9  // 计算子串 str[l ~ r] 的哈希值
10 ull get(int l, int r) {
11     return h[r] - h[l-1]*pw[r-l+1];
12 }

```

7.4.2 双哈希

```

1  #define ull unsigned long long
2  #define rep(i,a,b) for(int i=(a);i<=(b);i++)
3

```

```

4 typedef pair<ull,ull> pll;
5 const int MAXN = 1e4+5;
6 const int M1 = 1e9+7;//第一个模数
7 const int M2 = 1e9+9;//第二个模数
8 const int b = 131;
9 int n;
10 pll a[MAXN];
11
12 pll gethash(string s){
13     ull res1=0,res2=0;
14     int siz = s.length();
15     rep(i,0,siz-1){
16         res1=(res1*b%M1+s[i])%M1;//i位乘以b^i
17         res2=(res2*b%M2+s[i])%M2;//f(s)=Σ s[i] * b^i;
18     }
19     return make_pair(res1,res2);
20 }

```

7.5 后缀数组 SA+LCP

LCP(i,j) 后缀 i 和后缀 j 的最长公共前缀

```

1 int n,m;
2 string s;
3 int rk[MAXN],sa[MAXN],c[MAXN],rk2[MAXN];
4 //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
5 inline void get_SA(){
6     rep(i,1,n) ++c[rk[i]=s[i]];//基数排序
7     rep(i,2,m) c[i] += c[i-1];
8     //c做前缀和，可以知道每个关键字的排名最低在哪里
9     repb(i,n,1) sa[c[rk[i]]--] = i;//记录每个排名的原编号
10
11     for(int w=1;w<=n;w<=1){//倍增
12         int num = 0;
13         rep(i,n-w+1,n) rk2[++num] = i;//没有第二关键字的排在前面
14         rep(i,1,n) if(sa[i]>w) rk2[++num] = sa[i]-w;
15         //编号sa[i]大于w的才能作为编号sa[i]-w的第二关键字
16         rep(i,1,m) c[i] = 0;
17         rep(i,1,n) ++c[rk[i]];
18         rep(i,2,m) c[i]+=c[i-1];
19         repb(i,n,1) sa[c[rk[rk2[i]]]--]=rk2[i],rk2[i]=0;
20         //同一个桶中按照第二关键字排序
21         swap(rk,rk2);
22         //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
23
24         rk[sa[1]]=1,num=1;
25         rep(i,2,n)
26             rk[sa[i]] = (rk2[sa[i]]==rk2[sa[i-1]]&&rk2[sa[i]+w]==rk2[sa[i-1]+w])?num:++num;
27         //下一次排名的第一关键字,相同的两个元素排名也相同
28         if(num==n) break;//rk都唯一时，排序结束
29         m=num;
30     }
31 }
32 int height[MAXN];
33 inline void get_height(){

```

```

34  int k = 0,j;
35  rep(i,1,n) rk[sa[i]] = i;
36  rep(i,1,n){
37      if(rk[i]==1) continue;//第一名往前没有前缀
38      if(k) k--;//h[i]>=h[i-1]-1 即height[rk[i]]>=height[rk[i-1]]-1
39      j = sa[rk[i]-1];//找排在rk[i]前面的
40      while(j+k<=n&&i+k<=n&&s[i+k]==s[j+k]) ++k;//逐字符比较
41      //因为每次k只会-1, 故++k最多只会加2n次
42      height[rk[i]] = k;
43  }
44 }
45 inline void solve(){
46     cin>>s;
47     s = ' '+s;
48     n = s.size()-1,m = 122;//m为字符个数'z'=122
49     get_SA();
50     rep(i,1,n) cout<<sa[i]<<' ';
51     cout<<endl;
52 }

```

7.6 后缀自动机 SAM

```

1  struct state{
2      int len,link;
3      map<char,int> nxt;//也可以用数组,空间换时间
4  };
5  state sta[MAXN<<1];//状态数需要设定为两倍
6  int sz,last;//sz为自动机大小
7  inline void init_SAM(){
8      sta[0].len = 0;sta[0].link = -1;//虚拟状态t0
9      sz = 1;
10     last = 0;
11 }
12 int cnt[MAXN<<1];
13 void SAM_extend(char c){
14     int cur = sz++;
15     cnt[cur] = 1;
16     sta[cur].len = sta[last].len+1;
17     int p = last;
18     //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
19     while(p!=-1&&!sta[p].nxt.count(c)){
20         sta[p].nxt[c] = cur;
21         p = sta[p].link;
22     }
23     if(p==-1) sta[cur].link = 0;//情况1,没有符合的p
24     else{
25         int q = sta[p].nxt[c];
26         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)
27             sta[cur].link = q;
28         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
29             int clone = sz++;
30             cnt[clone] = 0;
31             sta[clone].len = sta[p].len+1;
32             sta[clone].nxt = sta[q].nxt;

```

```

33     sta[clone].link = sta[q].link;
34     while(p!=-1 && sta[p].nxt[c]==q){//把向q的转移指向clone
35         sta[p].nxt[c]=clone;
36         p=sta[p].link;
37     }
38     sta[q].link = sta[cur].link = clone;//clone是q的后缀,故linkq=clone
39 }
40 }
41 last = cur;//sta[last]包含目前处理的整个前缀!
42 }
43 string s;
44 vector<int> e[MAXN<<1];
45 void dfs(int now){
46     for(auto to:e[now]){
47         dfs(to);
48         cnt[now] += cnt[to];
49     }
50 }
51 inline void solve(){
52     cin>>s;
53     init_SAM();
54     int siz = s.size();
55     rep(i,0,siz-1) SAM_extend(s[i]);
56     rep(i,1,sz-1) e[sta[i].link].push_back(i);//link边反过来构造树
57     dfs(0);
58     ll maxx = 0;
59     rep(i,1,sz-1)
60         if(cnt[i]!=1) maxx = max(maxx,1ll*cnt[i]*sta[i].len);
61     cout<<maxx<<endl;
62 }
63 int main(){
64     solve();
65 }

```

7.7 广义 SAM

```

1 //广义sam只需要在插入新串时把last设为1
2 //其实就是存一下自己的sam
3 struct SAM{
4     static const int sigma=26,c0='a';
5     struct Node{
6         int fa=0,to[sigma],len=0;
7         int &operator [] (int x) {return to[x];}
8         Node(){
9             rep(i,0,sigma-1) to[i]=0;
10        }
11    }a[N*2];
12    int last=1,tot=1;
13    int newnode(){
14        a[++tot]=Node(); return tot;
15    }
16
17    ll sz[N*2],t[N*2],val[N*2];
18

```

```

19 void insert(int c){
20     bool fl=0; int z;
21     c-=c0;
22     //if(a[last][c]&&a[last].len+1==a[a[last][c]].len) return a[last][c];
23     int x=last,next=newnode();
24     sz[next]=1;
25     last=next;
26     a[next].len=a[x].len+1;
27     for(;x&&!a[x][c];x=a[x].fa) a[x][c]=next;
28     if(!x) a[next].fa=1;
29     else{
30         int y=a[x][c];
31         if(a[y].len==a[x].len+1){
32             a[next].fa=y;
33         }
34         else{
35             if(a[x].len+1==a[next].len) fl=1;
36             z=newnode();
37             a[z]=a[y],a[z].len=a[x].len+1;
38             //sz[z]=1;
39             for(;x&&a[x][c]==y;x=a[x].fa) a[x][c]=z;
40             a[y].fa=a[next].fa=z;
41         }
42     }
43 }
44 bool vis[N*2];
45 void dfs(int u){
46     vis[u]=1;
47     sz[u]=1;
48     rep(i,0,25) if(a[u][i]) {
49         if(!vis[a[u][i]]) dfs(a[u][i]);
50         sz[u]+=sz[a[u][i]];
51     }
52 }
53 void solve(){
54     int n=read();
55     rep(i,1,n){
56         last=1;
57         string s; cin>>s;
58         for(auto c:s) insert(c);
59     }
60     dfs(1);
61     cout<<sz[1]-1<<endl;
62 }
63
64 }sam;

```

8 其他

8.1 FFT

```

1 const double Pi = acos(-1.0);
2 struct cplx{//手写负数complex用起来会更快

```

```

3     double x,y;
4     cplx(double x=0,double y=0):x(x),y(y){}
5     friend cplx operator + (cplx a,cplx b){return cplx(a.x+b.x,a.y+b.y);}
6     friend cplx operator - (cplx a,cplx b){return cplx(a.x-b.x,a.y-b.y);}
7     friend cplx operator * (cplx a,cplx b){return cplx(a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x);}
8 }a[MAXN],b[MAXN];
9 void FFT(int limit,cplx *a,int type){//limit记录项数
10     if(limit==1) return;//只有一个常数项结束递归(即只剩下0次的)
11     cplx a1[limit>>1],a2[limit>>1];//按照奇偶分组
12     for(int i=0;i<limit;i+=2)
13         a1[i>>1] = a[i],a2[i>>1] = a[i+1];
14     FFT(limit>>1,a1,type);
15     FFT(limit>>1,a2,type);
16     cplx Wn(cos(2*Pi/limit),type*sin(2*Pi/limit));//单位根
17     //这里type挺重要的,反变换的时候用-1,因为是-k
18     cplx w(1,0);//一会儿算单位根幂的时候用w存
19     cplx tmp;
20     for(int i=0;i<(limit>>1);i++,w=w*Wn){//w相当于公式中的w_n^k
21         tmp = w*a2[i];//蝴蝶操作
22         a[i]=a1[i]+tmp;//偶
23         a[i+(limit>>1)]=a1[i]-tmp;//O(1)算另外一部分
24     }
25 }
26 int main(){
27     int n=read(),m=read();
28     rep(i,0,n) a[i].x=read();
29     rep(i,0,m) b[i].x=read();
30     int limit = 1;
31     while(limit<=n+m) limit<<=1;//这里非常精髓
32     //把长度补到2的幂,这样就不用考虑%2余数的情况
33     //而且不必担心高次项的系数,因为默认为0
34     FFT(limit,a,1);
35     FFT(limit,b,1);
36     //1表示FFT, -1则是反变换
37     rep(i,0,limit) a[i] = a[i]*b[i];//转换为点值后直接相乘
38     FFT(limit,a,-1);//现在变回去
39     rep(i,0,n+m) printf("%d ",(int)(a[i].x/limit+0.5));//还要除以n的
40     printf("\n");
41 }
42 //P3803 【模板】多项式乘法 (FFT)
43 //https://www.luogu.com.cn/problem/P3803

```

8.2 基础母函数求方案数

```

1 const int MAXN = 1e4;
2 int c1[MAXN+1];
3 int c2[MAXN+1];
4
5 int main(){
6     int n;
7     while(cin>>n){
8         for(int i=0;i<=n;i++){//初始化
9             c1[i]=0;
10            c2[i]=0;

```

```

11     }
12     for(int i=0;i<=n;i++){
13         c1[i]=1;//面值为1元的
14     }
15     for(int i=2;i<=n;i++){//枚举邮票的面值(一共有几组括号
16         for(int j=0;j<=n;j++){//枚举左边次数为0到次数为n的项
17             for(int k=0;j+k<=n;k+=i){//右边的乘过来,枚举放几枚
18                 //次数大于n的就不用管了
19                 c2[j+k]+=c1[j];
20             }
21         }
22         for(int i=0;i<=n;i++){//最左边两个括号完成处理
23             c1[i]=c2[i];//把c2算出来的值挪到左边作为下一次的左边
24             c2[i]=0;//清空c2记录下一次括号相乘的结果
25         }
26     }
27     cout<<c1[n]<<endl;
28 }
29 }

```

8.3 ST 表求 RMQ

$O(n\log n)$ 预处理, $O(1)$ 查询

```

1 #define log(x) (31-__builtin_clz(x))
2 const int MAXN = 1e5+10;
3 const int LOGN = log(MAXN)/log(2)+5;
4 int M[MAXN][LOGN];
5 int a[MAXN];
6 int z,m,n;
7 void init(){//初始化,复杂度O(nlogn)
8     for(int i=1;i<=n;i++) M[i][0]=i;//长度为1的区间最值是自己
9     for(int j=1;j<=LOGN;j++){
10         for(int i=1;i<=n-(1<<j)+1;i++){
11             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
12             else M[i][j] = M[i+(1<<j-1)][j-1];
13         }
14     }
15 }
16 int query(int l,int r){
17     int k = log(r-l+1)/log(2);//向下取整
18     if(a[M[l][k]]<a[M[r-(1<<k)+1][k]]) return M[l][k];
19     else return M[r-(1<<k)+1][k];
20 }

```

8.4 CDQ 分治三维偏序

```

1 #include<iostream>
2 #include<algorithm>
3 using namespace std;
4 #define rep(i,a,b) for(int i=(a);i<=(b);i++)
5 const int MAXN = 214514;
6 struct node{

```

```

7     int a,b,c,cnt,ans;
8 }s1[MAXN],s2[MAXN];
9 int ans[MAXN];//存对每个d,f[i]==d的i数量
10 int n,k,m;
11 //树状数组
12 int mx;
13 int treec[MAXN];
14 inline int lowbit(int x){return x&-x;}
15 inline void add(int x,int t){
16     while(x<=mx) treec[x]+=t,x+=lowbit(x);
17 }
18 inline int query(int x){
19     int ret = 0;
20     while(x) ret+=treec[x],x-=lowbit(x);
21     return ret;
22 }
23 //cdq分治
24 bool cmp1(node x,node y){//以a为关键字从小大到排序
25     if(x.a==y.a){
26         if(x.b==y.b) return x.c<y.c;
27         return x.b<y.b;
28     }
29     return x.a<y.a;
30 }
31 bool cmp2(node x,node y){//根据b排序,便只剩下c的顺序没有满足
32     if(x.b==y.b) return x.c<y.c;
33     return x.b<y.b;
34 }
35 void cdq(int l,int r){
36     if(l==r) return;
37     int mid = (l+r)>>1;
38     //处理前两种情况:ij同在左/右
39     cdq(l,mid);cdq(mid+1,r);
40     //处理第三种情况:i在左j在右
41     sort(s2+l,s2+mid+1,cmp2);
42     sort(s2+mid+1,s2+r+1,cmp2);
43     int pxj=1,pxi=mid+1;//双指针,对应pxi的移动把pxj插入树状数组中
44     for(pxi=mid+1;pxi<=r;pxi++){
45         while(s2[pxj].b<=s2[pxi].b&&pxj<=mid){
46             add(s2[pxj].c,s2[pxj].cnt);//带次数插入树状数组中
47             pxj++;
48         }
49         s2[pxi].ans+=query(s2[pxi].c);
50         //查询树状数组中c比他小的
51     }
52     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
53     //遍历pxj走过的区间,清空树状数组
54 }
55 int main(){
56     cin>>n>>k;
57     m=0,mx=k;//树状数组的最大数字(类似权值线段树)
58     rep(i,1,n) cin>>s1[i].a>>s1[i].b>>s1[i].c;
59     sort(s1+1,s1+1+n,cmp1);//以a为关键字从小到大排序
60     int tmp=0;//当前这个项出现了几次
61     rep(i,1,n){//合并相同的项并且记录出现次数进cnt

```



```

62     tmp++; //当前项出现了几次
63     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
64         s2[++m]=s1[i]; //m统计元素个数
65         s2[m].cnt = tmp;
66         tmp = 0;
67     }
68 }
69 cdq(1,m); //递归计算
70 //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
71 rep(i,1,m) ans[s2[i].ans+s2[i].cnt-1]+=s2[i].cnt;
72 //s2[i].cnt-1是这些数字自己之间满足的对数
73 rep(i,0,n-1) cout<<ans[i]<<endl;
74 }
75 //洛谷P3810陌上花开
76 //https://www.luogu.com.cn/problem/P3810

```

8.5 三分

```

1 while(l<r){ //类似求导的方式求极值
2     int x=(l+r)/2,y=x+1; //l+(r-l)/2
3     if(f(x)<f(y)) l=x+1; else r=y-1; //最大值
4     //if(f(x)<f(y)) r=y-1; else l=x+1; //最小值
5 }

```

8.6 数位 dp

```

1 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
2 ll dfs(int pos, ll s, bool lim=1, bool lz=1){
3     if(pos==-1) return !s; //返回该状态是否符合要求(0或1)
4     ll &x=dp[pos][s][0];
5     if(!lim && x!=-1) return x;
6     ll ans=0;
7     int maxi=lim?bit[pos]:9;
8     if(s){
9         if(maxi>=2) rep(i,0,1)
10             ans+=dfs(pos-1,i,lim&&2==maxi,0);
11     }
12     else{
13         rep(i,0,maxi){
14             if(i==2||i==4) continue;
15             //...//状态转移
16             //if(lz && i==0)...//可能要用lz, 其他地方都不用
17             ans+=dfs(pos-1,0,
18                 lim && i==maxi,
19                 lz && i==0);
20             if(i!=6) ans+=dfs(pos-1,1,lim&&i==maxi,lz&&i==0);
21         }
22     }
23     if(!lim)x=ans; //不限制的时候才做存储 yyds
24     return ans;
25 }
26 ll calc(ll n){
27     int len=0;

```

```
28 while(n)bit[len++]=n%10,n/=10;
29 return dfs(len-1,0)+dfs(len-1,1);
30 }
```

8.7 线性基

```
1 struct basis{//线性基超大常数版
2     ll p[64];
3     #define B(x,i) ((x>>i)&1)
4     //yyds
5     void init(){ memset(p,0,sizeof(p)); }
6     void add(ll x){
7         dep(i,62,0){
8             if(!B(x,i)) continue;
9             if(!p[i]) {p[i]=x; break;}
10            x^=p[i];
11        }
12    }
13    ll qmx(ll x){
14        dep(i,62,0) if(!B(x,i)&&p[i]) x^=p[i];
15        return x;
16    }
17    ll qmn(ll x){
18        dep(i,62,0) if(B(x,i)&&p[i]) x^=p[i];
19        return x;
20    }
21 }b;
```

8.8 莫队

```
1 int cnt[MAXN];//记录数字在区间[l,r]内出现的次数
2 int pos[MAXN],a[MAXN];
3 ll ans[MAXN];
4 int n,m,k,res;
5 struct Q{
6     int l,r,k;//k记录原来的编号
7     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
8         return pos[x.l]==pos[y.l]?x.r<y.r:pos[x.l]<pos[y.l];
9         //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);
10        //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
11    }
12 }q[MAXN];
13
14 void Add(int pos){
15     res -= cnt[a[pos]]*cnt[a[pos]];
16     cnt[a[pos]]++;
17     res += cnt[a[pos]]*cnt[a[pos]];
18 }
19 void Sub(int pos){
20     res -= cnt[a[pos]]*cnt[a[pos]];
21     cnt[a[pos]]--;
22     res += cnt[a[pos]]*cnt[a[pos]];
23 }
```

```

24 int main(){
25     cin>>n>>m>>k;//k为数字范围
26     memset(cnt,0,sizeof(cnt));
27     int siz = sqrt(n);//每个分块的大小
28     rep(i,1,n){
29         cin>>a[i];
30         pos[i] = i/siz;//分块
31     }
32     rep(i,1,m){
33         cin>>q[i].l>>q[i].r;
34         q[i].k = i;//记录原来的编号,用于打乱顺序后的还原
35     }
36     sort(q+1,q+1+m);
37     res = 0;//初始化res
38     int l = 1,r = 0;//当前知道的区间
39     //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
40     rep(i,1,m){//莫队的核心,注意加减的顺序
41         while(q[i].l<l) Add(--l);
42         while(q[i].l>l) Sub(l++);
43         while(q[i].r<r) Sub(r--);
44         while(q[i].r>r) Add(++r);
45         ans[q[i].k] = res;
46     }
47     rep(i,1,m) cout<<ans[i]<<endl;
48 }

```

8.9 带修莫队

```

1 int a[MAXN],b[MAXN];//a读入一开始的序列,b记录修改后的
2 int pos[MAXN];//分块
3 int cq,cr;//统计查询修改次数
4 int R[MAXN][3];//0记位置,1记原本的值,2记修改后的值
5 ll res;
6 int ans[MAXN];//记录结果
7 int n,m;
8 void Add(int x){if(cnt[x]==0)res++;cnt[x]++;};//带修莫队的add和sub有区别
9 void Sub(int x){if(cnt[x]==1)res--;cnt[x]--;}
10 struct Q{
11     int l,r,k,t;
12     friend bool operator < (Q a,Q b){
13         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);
14         //增加第三关键字,询问的先后顺序,用t或者k应该都行
15     }
16 }q[MAXN];
17 int main(){
18     cin>>n>>m;
19     cq = cr = 0;
20     int siz = pow(n,2.0/3.0);//这么分块最好,别问
21     rep(i,1,n){
22         cin>>a[i];
23         b[i]=a[i];
24         pos[i] = i/siz;
25     }
26     char hc;

```

```

27 rep(i,1,m){//读入修改和询问
28     cin>>hc;
29     if(hc=='Q'){
30         cin>>q[cq].l>>q[cq].r;
31         q[cq].k=cq;q[cq].t=cr;//注意这时候R[cr]还是有的,这次询问是在R[cr-1]之后的
32         cq++;
33     }
34     else{
35         cin>>R[cr][0]>>R[cr][2];
36         R[cr][1] = b[R[cr][0]];
37         b[R[cr][0]] = R[cr][2];//在b数组中记录更改
38         cr++;
39     }
40 }
41 sort(q,q+cq);
42 int l=1,r=0,sjc=0;//时间戳
43 res = 0;
44 rep(i,0,cq-1){
45     while(sjc<q[i].t){
46         if(l<=R[sjc][0]&&R[sjc][0]<=r)//判断修改是否在该区间内
47             Sub(R[sjc][1]),Add(R[sjc][2]);
48         a[R[sjc][0]] = R[sjc][2];//在a上也进行更改
49         sjc++;
50     }
51     while(sjc>q[i].t){
52         sjc--;
53         if(l<=R[sjc][0]&&R[sjc][0]<=r)//判断修改是否在该区间内
54             Sub(R[sjc][2]),Add(R[sjc][1]);
55         a[R[sjc][0]] = R[sjc][1];//在a上也进行更改
56     }
57     while(l>q[i].l) Add(a[--l]);
58     while(l<q[i].l) Sub(a[l++]);
59     while(r<q[i].r) Add(a[++r]);
60     while(r>q[i].r) Sub(a[r--]);
61     ans[q[i].k] = res;
62 }
63 rep(i,0,cq-1) cout<<ans[i]<<endl;
64 }

```

9 STL 等小技巧

9.1 集合 set

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

```

1 //并交集
2 vector<int> ANS;
3 set_union(s1.begin(),s1.end(),s2.begin(),s2.end(),inserter(ANS,ANS.begin()));//set_intersection()
4
5 //通过迭代器遍历集合
6 set<char>::iterator iter = temp1.begin();
7 while (iter!=temp1.end()){
8     cout<<*iter;
9     iter++;

```

```
10 }
```

9.2 快读快写 (短)

```
1 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;}
2 template<class T>
3 void wt(T x){//快写
4     if(x < 0) putchar('-'), x = -x;
5     if(x >= 10) wt(x / 10);
6     putchar('0' + x % 10);
7 }
```

9.3 GCD(压行)

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

9.4 计时

```
1 inline double run_time(){
2     return 1.0*clock()/CLOCKS_PER_SEC;
3 }
```

9.5 替换 unorderedset 的 hash 函数

```
1 struct VectorHash {
2     size_t operator()(const std::vector<int>& v) const {
3         std::hash<int> hasher;
4         size_t seed = 0;
5         for (int i : v) {
6             seed ^= hasher(i) + 0x9e3779b9 + (seed<<6) + (seed>>2);
7         }
8         return seed;
9     }
10 };
11 unordered_set<vector<int>,VectorHash> st;
```

9.6 对拍

9.6.1 linux&mac 版

```
1 while true; do
2     ./3.exe>tmp.in #出数据
3     ./2.exe<tmp.in>tmp.out #被测程序
4     ./4.exe<tmp.in>tmp2.out #正确（暴力）程序
5     if diff tmp.out tmp2.out; then #比较两个输出文件
6         printf AC #结果相同显示AC
7     else
8         echo WA #结果不同显示WA，并退出
```

```
9 cat tmp.in
10 cat tmp.out tmp2.out
11 exit 0
12 fi #if 的结束标志,与C语言相反,0为真
13 done # while 的结束标志
```

9.6.2 windows 版

```
1 @echo off
2 :loop
3     rand.exe > in.txt
4     my.exe < in.txt > myout.txt
5     std.exe < in.txt > stdout.txt
6     fc myout.txt stdout.txt
7 if not errorlevel 1 goto loop
8 pause
9 goto loop
```

9.7 火车头

```
1 #pragma GCC optimize(2)
2 #pragma GCC optimize(3)
3 #pragma GCC optimize("Ofast")
4 #pragma GCC optimize("inline")
5 #pragma GCC optimize("-fgcse")
6 #pragma GCC optimize("-fgcse-lm")
7 #pragma GCC optimize("-fipa-sra")
8 #pragma GCC optimize("-ftree-pre")
9 #pragma GCC optimize("-ftree-vrp")
10 #pragma GCC optimize("-fpeephole2")
11 #pragma GCC optimize("-ffast-math")
12 #pragma GCC optimize("-fsched-spec")
13 #pragma GCC optimize("unroll-loops")
14 #pragma GCC optimize("-falign-jumps")
15 #pragma GCC optimize("-falign-loops")
16 #pragma GCC optimize("-falign-labels")
17 #pragma GCC optimize("-fdevirtualize")
18 #pragma GCC optimize("-fcaller-saves")
19 #pragma GCC optimize("-fcrossjumping")
20 #pragma GCC optimize("-fthread-jumps")
21 #pragma GCC optimize("-funroll-loops")
22 #pragma GCC optimize("-fwhole-program")
23 #pragma GCC optimize("-freorder-blocks")
24 #pragma GCC optimize("-fschedule-insns")
25 #pragma GCC optimize("inline-functions")
26 #pragma GCC optimize("-ftree-tail-merge")
27 #pragma GCC optimize("-fschedule-insns2")
28 #pragma GCC optimize("-fstrict-aliasing")
29 #pragma GCC optimize("-fstrict-overflow")
30 #pragma GCC optimize("-falign-functions")
31 #pragma GCC optimize("-fcse-skip-blocks")
32 #pragma GCC optimize("-fcse-follow-jumps")
33 #pragma GCC optimize("-fsched-interblock")
```

```
34 #pragma GCC optimize("-fpartial-inlining")
35 #pragma GCC optimize("no-stack-protector")
36 #pragma GCC optimize("-freorder-functions")
37 #pragma GCC optimize("-findirect-inlining")
38 #pragma GCC optimize("-fhoist-adjacent-loads")
39 #pragma GCC optimize("-frerun-cse-after-loop")
40 #pragma GCC optimize("inline-small-functions")
41 #pragma GCC optimize("-finline-small-functions")
42 #pragma GCC optimize("-ftree-switch-conversion")
43 #pragma GCC optimize("-foptimize-sibling-calls")
44 #pragma GCC optimize("-fexpensive-optimizations")
45 #pragma GCC optimize("-funsafe-loop-optimizations")
46 #pragma GCC optimize("inline-functions-called-once")
47 #pragma GCC optimize("-fdelete-null-pointer-checks")
48 #pragma GCC optimize("Ofast,no-stack-protector")
49 #pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,tune=native")
```