A. Convolution

$\sum_{i=1}^{n}\sum_{j=1}^{n}2^{a_ia_j}$

多点求值做法

令 $f(x)=\sum_{i=1}^{n}x^{a_i}$

那么答案就是 $\sum_{i=1}^{n}f(2^{a_i})$

利用经典的多项式多点求值可以在 $O(n\log n)$ 时间内算出

平方根做法

$2^{a_ia_j}=2^{\frac{1}{2}((a_i+a_j)^2-a_i^2-a_j^2)}$

设 $p^2=2$，在模 998244353 下这样的 p 是存在的，我们可以暴力找出

则答案是 $\sum_{i=1}^{n}\sum_{j=1}^{n}p^{(a_i+a_j)^2-a_i^2-a_j^2}$

等于 $\sum_k p^{k^2}\sum_{a_i+a_j=k}p^{-a_i^2-a_j^2}$

设多项式 $f(x)=\sum_{i=1}^{n}p^{-a_i^2}x^{a_i}$

则答案就是 $\sum_k p^{k^2}[x^k]f(x)^2$（其中 $[x^k]g(x)$ 表示 $g(x)$ 的 x^k 项的系数）

可以用 NTT 在 $O(n\log n)$ 的时间内求出

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
```

```
typedef double db;
const int N=1<<18;
const int P=998244353;
const int S2=116195171;
typedef long long ll;
#define upmo(a,b) (((a)=((a)+(b))%mo)<0?(a)+=mo:(a))
const db pi=3.14159265358979323846264338327L;const int FFT_MAXN=262144;int
mo=998244353;
struct cp{
  db a,b;
  cp operator +(const cp&y)const{return (cp){a+y.a,b+y.b};}
  cp operator -(const cp&y)const{return (cp){a-y.a,b-y.b};}
  cp operator *(const cp&y)const{return (cp){a*y.a-b*y.b,a*y.b+b*y.a};}
  cp operator !()const{return cp{a,-b};};
}nw[FFT_MAXN+1];
int bitrev[FFT_MAXN];
void dft(cp*a,int n,int flag=1){
  int d=0;while((1<<d)*n!=FFT_MAXN)d++;
  rep(i,0,n-1)if(i<(bitrev[i]>>d))swap(a[i],a[bitrev[i]>>d]);
  for(int l=2;l<=n;l<<=1){
    int del=FFT_MAXN/l*flag;
    for(int i=0;i<n;i+=l){
      cp *le=a+i;cp *ri=a+i+(l>>1);
      cp *w=flag==1?nw:nw+FFT_MAXN;
      rep(k,0,(l>>1)-1){
        cp ne=*ri**w;*ri=*le-ne,*le=*le+ne;le++,ri++,w+=del;
      }
    }
  }
  if(flag!=1)rep(i,0,n-1)a[i].a/=n,a[i].b/=n;
}
void fft_init(){
  int L=0;while((1<<L)!=FFT_MAXN)L++;
  bitrev[0]=0;rep(i,1,FFT_MAXN-1)bitrev[i]=bitrev[i>>1]>>1|((i&1)<<(L-1));
  rep(i,0,FFT_MAXN)nw[i]=(cp){(db)cosl(2*pi/FFT_MAXN*i),
(db)sinl(2*pi/FFT_MAXN*i)};
}
void convoP(int *a,int n,int *b,int m,int *c){ // 任意模数 fft, 需要提前设定 mo
  rep(i,0,n+m)c[i]=0;
  static cp f[FFT_MAXN],g[FFT_MAXN],t[FFT_MAXN];int N=2;while(N<=n+m)N<<=1;
  rep(i,0,N-1){
    int aa=i<=n?a[i]:0;int bb=i<=m?b[i]:0;
    upmo(aa,0);upmo(bb,0);
    f[i]=(cp){db(aa>>15),db(aa&32767)};
    g[i]=(cp){db(bb>>15),db(bb&32767)};
  }
  dft(f,N);dft(g,N);
  rep(i,0,N-1){int j=i?N-i:0;t[i]=((f[i]+!f[j])*(!g[j]-g[i])+(!f[j]-f[i])*
(g[i]+!g[j]))*(cp){0,0.25};}
```

```
  dft(t,N,-1);
  rep(i,0,n+m)upmo(c[i],(ll(t[i].a+0.5))%mo<<15);
  rep(i,0,N-1){int j=i?N-i:0;t[i]=(!f[j]-f[i])*(!g[j]-g[i])*(cp){-0.25,0}+(cp)
{0,0.25}*(f[i]+!f[j])*(g[i]+!g[j]);}
  dft(t,N,-1);
  rep(i,0,n+m)upmo(c[i],ll(t[i].a+0.5)+(ll(t[i].b+0.5)%mo<<30));
}
inline int Pow(int a,int b){
  int c=1;
  for(;b;b>>=1,a=a*1ll*a%P)if(b&1)c=c*1ll*a%P;
  return c;
}
int n,a[N];
int b[N+2],c[N+2],d[N+2];
int main(){
  fft_init();
  scanf("%d",&n);
  rep(i,1,n)scanf("%d",&a[i]);
  rep(i,1,n){
    b[a[i]]=(b[a[i]]+Pow(S2,P-1-(a[i]*1ll*a[i])%(P-1)))%P;
  }
  rep(i,0,100000)c[i]=b[i];
  convoP(b,100000,c,100000,d);
  int ans=0;
  rep(i,0,200000){
    int res=d[i];
    res=res*1ll*Pow(S2,i*1ll*i%(P-1))%P;
    ans=(ans+res)%P;
  }
  printf("%d\n",ans);
  return 0;
}
```

B. 双圈覆盖

规定：非环边表示不在哈密尔顿环上的边

将图转化为度数全等于 3 的图

若一个点 x 有非环边 $(x,y),(x,z)$

我们新建立一个点 p，假设 x 在哈密尔顿环上下一个点是 w

则把 (x,w) 拆成 $(x,p),(p,w)$，(x,z) 变成 (p,z)

我们对这个图求完圈覆盖后，只要把 x,p 变回同一个点，且缩掉 (x,p) 即可

只要这样不断地拆，最后每个点的度数一定为 3，所以接下来只要讨论每个点度数为 3 的情况

构造

现在每个点度数为 3 了，那么一定有偶数个点，每个点的边有两条环边和一条非环边，我们称非环边为匹配边

第一类圈：整个哈密尔顿回路

第二类圈：所有匹配边+$\{(2i,2i+1)\}$

第三类圈：所有匹配边+$\{(2i,2i-1)\}$

可以发现哈密尔顿回路上的环边都恰好出现两次，且匹配边也恰好出现两次，符合双圈覆盖的定义

而且在第二类圈和第三类圈中，每个点度数都为 2，可以轻松找到一系列欧拉回路

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=505;
int n,m;
pii go[N][N];
vector<vector<int> >way;
int num[N];
bool vis[N][N];
pii getaft(int x,int d){
  if(d!=num[x])return pii(x,d+1);
  if(x==n)return pii(1,1);
  return pii(x+1,1);
}
pii getpre(int x,int d){
  if(d!=1)return pii(x,d-1);
  if(x==1)return pii(n,num[n]);
```

```
  return pii(x-1,num[x-1]);
}
vector<int> d;
int kd[N][N];
void pbd(int x){
  if(d.size())if(d[d.size()-1]==x)return;
  d.pb(x);
}
pii FL;
void dfs(int x,int y,int md){
  vis[x][y]=1;
  pbd(x);
  if(kd[x][y]==md){
    //go right
    pii w=getaft(x,y);
    pbd(w.fi);
    while(w.se==1){
      w=getaft(w.fi,w.se);
      pbd(w.fi);
    }
    x=w.fi;
    y=w.se;
    pbd(x);
    assert(vis[x][y]==0);
    vis[x][y]=1;

    w=go[x][y];
    x=w.fi;
    y=w.se;
    if(vis[x][y]){
      FL=pii(x,y);
      return;
    }
    pbd(x);
    vis[x][y]=1;
    dfs(x,y,md);
  }
  else{
    pii w=getpre(x,y);
    pbd(w.fi);
    while(w.se==1){
      w=getpre(w.fi,w.se);
      pbd(w.fi);
    }
    x=w.fi;
    y=w.se;
    pbd(x);
    assert(vis[x][y]==0);
    vis[x][y]=1;
```

```
    w=go[x][y];
    x=w.fi;
    y=w.se;
    if(vis[x][y]){
      FL=pii(x,y);
      return;
    }
    pbd(x);
    vis[x][y]=1;
    dfs(x,y,md);
  }
}
void Main(){
  scanf("%d%d",&n,&m);
  assert(4<=n&&n<=50);
  assert(n<=m&&m<=500);
  rep(i,1,n)rep(j,1,m)go[i][j]=pii(0,0);
  rep(i,1,n)num[i]=1;
  way.clear();
  memset(vis,0,sizeof vis);
  rep(i,1,m){
    int a,b;scanf("%d%d",&a,&b);
    assert(a<b);
    assert(!vis[a][b]);
    vis[a][b]=1;

    if(a+1==b)continue;
    if(a==1&&b==n)continue;
    num[a]++;num[b]++;
    go[a][num[a]]=pii(b,num[b]);
    go[b][num[b]]=pii(a,num[a]);
  }
  d.clear();rep(i,1,n)d.pb(i);way.pb(d);
  d.clear();

  int now=0;
  rep(i,1,n)rep(j,2,num[i]){
    now^=1;kd[i][j]=now;
  }
  memset(vis,0,sizeof vis);

  rep(i,1,n)rep(j,2,num[i])if(!vis[i][j]){
    d.clear();
    dfs(i,j,1);
    while(d[0]==d[d.size()-1])d.pop_back();
    way.pb(d);
    assert(FL==pii(i,j));
  }
```

```
  memset(vis,0,sizeof vis);
  rep(i,1,n)rep(j,2,num[i])if(!vis[i][j]){
    d.clear();
    dfs(i,j,0);
    while(d[0]==d[d.size()-1])d.pop_back();
    way.pb(d);
    assert(FL==pii(i,j));
  }
  printf("%d\n",way.size());
  for(auto t:way){
    printf("%d ",t.size());
    for(int x:t)printf("%d ",x);puts("");
  }
}
int main(){
  int t;scanf("%d",&t);
  while(t--)Main();
  return 0;
}
```

C. 酒馆战棋

分析一下两种随从的作用：

0：只能用来破圣盾

1：既能用来破圣盾，也可以用来杀非圣盾随从

所以在最优情况下，因为只有 1 能杀非圣盾随从，所以尽量让 1 去杀非圣盾随从，0 尽量去破盾

在最劣情况下，尽量让 0 白给，尽量让 1 去破圣盾

时间复杂度：$O(n)$

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
```

```
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=1005;
char s[N];
int n,a,b,c,d;
int ma(int a,int b,int c,int d){
  int res=0;
  rep(i,1,n){
    if(s[i]=='1'){
      if(c+d){
        if(c)c--,res++;
        else d--,c++;
      }
      else{
        if(a)a--,res++;
        else if(b)b--,a++;
      }
    }
    else{
      if(c+d){
        if(d)d--,c++;
      }
      else{
        if(b)b--,a++;
      }
    }
  }
  assert(a>=0&&b>=0&&c>=0&&d>=0);
  return res;
}
int mi(int a,int b,int c,int d){
  int res=0;
  rep(i,1,n){
    if(s[i]=='1'){
      if(c+d){
        if(d)d--,c++;
        else c--,res++;
      }
      else{
        if(b)b--,a++;
        else if(a)a--,res++;
      }
    }
    else{
      if(c+d){
```

```
        if(!c)d--,c++;
      }
      else if(a+b){
        if(!a)b--,a++;
      }
    }
  }
  assert(a>=0&&b>=0&&c>=0&&d>=0);
  return res;
}
void Main(){
  scanf("%d%d%d%d%d",&n,&a,&b,&c,&d);
  scanf("%s",s+1);
  printf("%d %d\n",ma(a,b,c,d),mi(a,b,c,d));
}
int main(){
  int t;scanf("%d",&t);
  while(t--)Main();
  return 0;
}
```

D. 递增递增

做法 1（通用但是麻烦）

考虑最高位，那么一定存在一个 k，使得 $a[1...k]$ 最高位是 0，$a[k+1...n]$ 最高位是 1，那么可以分成左右两个子区间去做，然后子区间枚举一下次高位的分界线，再继续分下去，以此类推。

令 $Dp[i][L][R][eqL][eqR]$ 表示：在 $i+1...60$ 位已经定下来的情况下，$L...R$ 的区间的答案，eqL 表示第 $i+1...60$ 位是否等于 $L...R$ 中最大的 $L[x]$，eqR 表示第 $i+1...60$ 位是否等于 $L...R$ 中最小的 $R[x]$

设 $mL(L,R)$ 为 $L...R$ 中最大的 $L[x]$，$mR(L,R)$ 为最小的 $R[x]$

转移枚举第 i 位的分界线 K，答案啥的都好办，主要是 eqL 和 eqR 的转移

如果 $mL(L,R)$ 和 $mL(L,K)$ 在 $i+1...60$ 位不相同，那么在区间 $[L,K]$ 中，对下界无限制，所以 eqL 为 0

如果 $mR(L,R)$ 和 $mR(L,K)$ 在 $i+1...60$ 位不相同，那么在区间 $[L,K]$ 中，对上界无限制，所以 eqR 为 0

否则就跟普通的数位 DP 一样转移 eqL 和 eqR 即可

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
```

```
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int P=998244353;
const int N=55;
int n;LL l[N],r[N];
struct atom{
  int sumw,way;
  inline atom(int _sumw=0,int _way=0){
    sumw=_sumw;
    way=_way;
  }
};
inline atom operator +(const atom &a,const atom &b){
  return atom((a.sumw+b.sumw)%P,(a.way+b.way)%P);
}
inline atom operator *(const atom &a,const atom &b){
  return atom((a.sumw*1ll*b.way+a.way*1ll*b.sumw)%P,a.way*1ll*b.way%P);
}
atom f[62][N][N][2][2];
LL mir[N][N],mal[N][N];
int main(){
  scanf("%d",&n);
  rep(i,1,n)scanf("%lld",&l[i]);
  rep(i,1,n)scanf("%lld",&r[i]);
  rep(i,1,n){
    mir[i][i]=r[i];
    mal[i][i]=l[i];
    rep(j,i+1,n){
      mir[i][j]=min(mir[i][j-1],r[j]);
      mal[i][j]=max(mal[i][j-1],l[j]);
    }
  }
  rep(i,1,n)rep(j,i,n)rep(x,0,1)rep(y,0,1)f[0][i][j][x][y]=atom(0,1);
  rep(i,1,60){
```

```
    rep(l,1,n)rep(r,l,n)rep(er,0,1)rep(el,0,1){
      atom ans=atom(0,0);
      rep(k,l,r-1){
        //l...k put 0
        //k+1..r put 1
        int lel=(el&&(mal[l][k]>>i)==(mal[l][r]>>i));
        if(lel&&(mal[l][k]&(1ll<<(i-1))))continue;
        int ler=(er&&(mir[l][k]>>i)==(mir[l][r]>>i));

        int rel=(el&&(mal[k+1][r]>>i)==(mal[l][r]>>i));
        int rer=(er&&(mir[k+1][r]>>i)==(mir[l][r]>>i));
        if(rer&&(!(mir[k+1][r]&(1ll<<(i-1)))))continue;

        ler&=((mir[l][k]&(1ll<<(i-1)))==0);
        rel&=((mal[k+1][r]&(1ll<<(i-1)))>0);

        ans=(ans+f[i-1][l][k][lel][ler]*f[i-1][k+1][r][rel][rer]*atom((r-
k)*1ll*((1ll<<(i-1))%P)%P,1));
      }
      //all0
      if(el&&(mal[l][r]&(1ll<<(i-1)))){}
      else{
        int nl=el;
        int nr=er&&((mir[l][r]&(1ll<<(i-1)))==0);
        ans=(ans+f[i-1][l][r][nl][nr]*atom(0,1));
      }
      //all1
      if(er&&(!(mir[l][r]&(1ll<<(i-1))))){}
      else{
        int nl=el&&(mal[l][r]&(1ll<<(i-1)));
        int nr=er;
        ans=(ans+f[i-1][l][r][nl][nr]*atom((r-l+1)*1ll*((1ll<<(i-1))%P)%P,1));
      }
      f[i][l][r][el][er]=ans;
    }
  }
  printf("%d\n",f[60][1][n][1][1].sumw);
  return 0;
}
```

做法 2

首先我们将读入的 $l[i], r[i]+1$ 扔进一个数组里进行离散化，最后我们可以得到 $O(n)$ 个不交的小区间 $[sl_i, sr_i]$，且每个 $[l[i], r[i]]$ 都是若干个这样的小区间的并。我们将这些小区间从小到大排序。

那么对于每个 a_i 它所属的区间的编号肯定也是不降的

令 $dp[i][j]$ 表示整完了 $a_{1...i}$，且 a_i 在第 j 个小区间内

我们可以枚举 $a_{i+1}\dots a_r$ 在第 k 个小区间内 $(k>j)$，然后转移到 $dp[r][k]$

那么关键就是这个转移系数了，相当于在 $L\dots R$ 中可重复地顺序无关地选出 t 个数。

有两个系数要求，一个是方案数，一个是总和

方案数：相当于有 $R-L+1$ 个箱子，有 t 个球要放进去，可以用经典的插板法，方案数就是 C_{R-L+t}^{t}，这个组合数可以在 $O(t)$ 的时间内求出

总和：就是 $t\frac{R+L}{2}C_{R-L+t}^{t}$

证明：考虑一种方案 $S=\{b_1\dots b_t\}$，设 $rev(S)=\{R+L-b_1\dots R+L-b_t\}$

S 与 $rev(S)$ 形成了两两对应关系，所以他们的和的和为 $t(R+L)$，平均下来每个的和就是 $t\frac{R+L}{2}$

若 $S=rev(S)$，它的和也是 $t\frac{R+L}{2}$

所以总和为 $t\frac{R+L}{2}C_{R-L+t}^{t}$

时间复杂度：$O(n^4)$

E. Access

首先，如果最终的方案中 Access 了 x 和 x 的某个祖先 y 的话，那必然是先 Access x 再 Access y，不然的话是无效的

我们令 $f[x][i]$ 表示以 x 为根的子树中，如果进行了恰好 i 次 Access，能有几种不同的形态。

首先考虑这么一个现实：一个子树 x 里有点被 Access 的话，那么如果没有其他干扰因素的话，$(x,fa(x))$ 需要是实边，但是比如 x 的儿子 y_1,y_2 的子树里都有点被 Access 的话，那就是最后 Access 的那个子树跟 x 相连的边是实边。

所以假设 x 有儿子 $y_1,y_2\dots y_m$，它们子树里 Access 的次数为 $i_1,i_2\dots i_m$，那么最后整个子树 Access 的次数就是 $\sum_{j=1}^{m}i_j$，之后我们还需要选择某个 y_j 使得 $i_j>0$，然后让 (x,y_j) 成为实边，其他的成为虚边，这些都是可以用树形 DP 计算的

然后还有一种情况是，所有边 (x,y_j) 都是虚边，这种情况相当于最后进行了 Access x，所以只要给 Access 次数加一即可。

在 DP 时，第二维的 i 的范围是 $min(size[x],k)$，而这样的树形 DP 的复杂度是 $O(nk)$ 的，这是个经典模型

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
```

```cpp
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=10005;
const int P=998244353;
int f[N][505];
int n,k;
vector<int>go[N];
int sz[N],q[N],fa[N];
int g[505][2];
int h[505][2];
void dp(int x){
  sz[x]=1;
  f[x][0]=1;
  memset(g,0,sizeof g);
  memset(h,0,sizeof h);
  h[0][0]=1;
  for(int y:go[x])if(fa[y]==x){
    memset(g,0,sizeof g);
    rep(i,0,min(sz[x],k))rep(j,0,min(sz[y],k-i)){
      rep(p1,0,1)rep(p2,0,(j>0))if(!(p1&p2)){
        g[i+j][p1|p2]=(g[i+j][p1|p2]+h[i][p1]*1ll*f[y][j])%P;
      }
    }
    rep(i,1,min(sz[x]+1,k)){
      g[i][1]=(g[i][1]+h[i-1][0])%P;
    }
    sz[x]+=sz[y];
    rep(i,0,min(sz[x]+1,k))rep(j,0,1)h[i][j]=g[i][j];
  }
  rep(i,0,k){
    f[x][i]=(f[x][i]+g[i][1])%P;
    if(i)//access x}
    f[x][i+1]=(f[x][i+1]+g[i][0])%P;
  }
  //for access 1
}
int main(){
  scanf("%d%d",&n,&k);
  rep(i,1,n-1){
    int a,b;scanf("%d%d",&a,&b);
    go[a].pb(b);go[b].pb(a);
```

```
  }
  q[q[0]=1]=1;
  rep(i,1,q[0]){
    int x=q[i];
    for(int y:go[x])if(fa[x]!=y){
      fa[y]=x;q[++q[0]]=y;
    }
  }
  per(i,q[0],1){
    int x=q[i];
    dp(x);
  }
  int ans=0;
  rep(i,0,k)ans=(ans+f[1][i])%P;
  printf("%d\n",ans);
  return 0;
}
```

F. 图与三角形

我们称一个两条边一个点的结构为角

对于每一个异色三元环，一定存在两个角，使得这两个角是异色角

对于每一个同色三元环，不存在任何异色角

所以假设异色角个数为 x，异色三元环个数为 y，则 $2y = x$

$x = \sum_{i=1}^{n} deg[i][0]deg[i][1]$

其中 $deg[i][0]$ 表示点 i 连出去的白边个数

之后解出 y 就好了，然后减一下得出同色三元环个数

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
```

```
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=5005;
int n;
int du[N][2];
int num[2];
bool col[N][N];
void addedge(int a,int b,int c){
  num[c]++;
  du[a][c]++;
  du[b][c]++;
  col[a][b]=col[b][a]=c;
}
void readGraph(){
  int a,b,c,p,d;scanf("%d%d%d%d%d",&a,&b,&c,&p,&d);
  rep(i,1,n)rep(j,i+1,n){
    LL res=a*1ll*(i+j)*(i+j)+b*1ll*(i-j)*(i-j)+c;
    if(res%p>d){
      addedge(i,j,1);
    }
    else addedge(i,j,0);
  }
}
LL bl(){
  LL ans=0;
  rep(i,1,n)rep(j,i+1,n)rep(k,j+1,n)if(col[i][j]==col[j][k]&&col[j][k]==col[i]
[k])ans++;
  return ans;
}
int main(){
  scanf("%d",&n);
  readGraph();
  LL ans=0;
  rep(i,1,n){
    ans=(ans+du[i][0]*du[i][1]);
  }
  LL res=n*1ll*(n-1)*(n-2)/6-ans/2;
  //assert(res==bl());
  printf("%lld\n",res);
  //cerr<<num[0]<<" "<<num[1]<<endl;
  return 0;
}
```

G. 单调栈

首先，一定有 $f[1]=1$

那么我们考虑所有满足 $f[x]=1$ 的位置，从左到右分别是 $b_1 \dots b_k$

那么显然一定有 $p_{b_1} > p_{b_2} \dots > p_{b_k}$

而我们知道 $b_1=1$，所以要让 p_1 最小的话肯定填 $p_{b_i}=k-i+1$ 是最优的

这样填固然是最优的，但是是否一定有解呢?

不难发现，因为我们填掉了最小的 k 个数，所以剩下的所有 $f[x]$ 都减去了 1

我们把 $b_1 \dots b_k$ 全删掉，然后递归下去做即可，因为能一直做下去，所以是一定有解的

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=105;
vector<pii> a;
int n;
int p[N];
bool use[N];
void work(vector<pii> a){
  if(!a.size())return;
  rep(i,0,a.size()-1)assert(a[i].fi<=i+1);
  vector<int> pos;
  pos.pb(0);
  rep(i,1,a.size()-1)if(a[i].fi==1)pos.pb(i);
  int t=pos.size()-1;
  if(pos.size()){
```

```
    rep(i,1,n)if(!use[i]){
      use[i]=1;
      p[a[pos[t--]].se]=i;
      if(t<0)break;
    }
    vector<pii> d;
    int pre=0;
    //这里是从一开始错误的思路改过来的，所以代码逻辑很混乱
    rep(i,0,a.size()-1)if(a[i].fi==1||i==0)pre=1;
    else d.pb(pii(a[i].fi-pre,a[i].se));
    work(d);
  }
}
int f[N];int d[N];
void check(){
  f[0]=0;
  rep(i,1,n){
    int k=0;
    rep(j,1,i-1)if(p[j]<p[i])if((!k)||(p[j]>p[k]))k=j;
    f[i]=f[k]+1;
  }
  rep(i,1,n)if(d[i]!=-1)assert(d[i]==f[i]);
}
void Main(){
  scanf("%d",&n);
  a.clear();
  memset(use,0,sizeof use);
  rep(i,1,n){
    int x;scanf("%d",&x);d[i]=x;
    a.pb(pii(x,i));
  }
  work(a);
  check();
  rep(i,1,n)printf("%d%c",p[i],i==n?'\n':' ');
}
int main(){
  int t;scanf("%d",&t);
  while(t--)Main();
  return 0;
}
```

H. 异或询问

我们规定一个符号 $S \oplus x = \{y\ xor\ x | y \in S\}$

$bit(x,i)$ 表示 x 的第 i 位的值

那么我们要求的就是 $\sum_{y \in [L,R] \oplus x} f(y)^2$

做法1

我们将区间分成两个前缀的差，现在只需要考虑 $[0,R]\oplus x$

考虑最高位，假设是 w 位

如果 $R<2^w$，那么考虑一下 $bit(x,w)$，可以变成 $[0,R]\oplus x'$ 或者 $[2^w,2^w+R]\oplus x'$

其中 x' 是 x 去掉第 w 位的值

如果 $R\geq 2^w$，那么考虑 $[0,R]=[0,2^w-1]+[2^w,R]$

假设 $bit(x,w)=0$，那么 $[0,2^w-1]\oplus x=[0,2^w-1]$

而 $[2^w,R]\oplus x$ 就递归下去做就好了

这样就能把 $[0,R]\oplus x$ 分成 $O(\log n)$ 个连续的区间

现在考虑对于区间 $[0,R]$，如何求 $\sum_{x\leq R}f(x)^2$

我们将 a_i 排序，那么答案就是 $\sum_{i=1}^{n}i^2(min(a_{i+1}-1,x)-a_i+1)$

我们可以对 $x=a_i$ 预处理出一个答案，然后询问就二分出小于等于 x 的最大的 a_y，然后计算答案即可

时间复杂度：$O(n\log^2 n)$

做法2

做法1主要的问题在于他要先拆完所有区间，然后每个区间去进行一次询问

但实际上我们可以在线段树上边拆边询问，这样的好处就是最后拆出来的每个区间都刚好是线段树上的某个区间，只要预处理好答案询问就是 $O(1)$ 的了，这个只要在线段树上询问的时候进行讨论就好了（细节可见标程）

另一种想法就是，因为做法 1 拆出来的每个区间的长度都是 2^i 的形式，所以可以倍增预处理出每个长度为 2^i 的区间的答案，这个贡献的形式还是比较好合并的

时间复杂度：$O(n\log n)$

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
```

```cpp
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int P=998244353;
const int N=110000;
int n,Q,a[N];
vector<int> v[N*33];
int go[N*33][2],s1[N*33],s2[N*33],tot=1;
void insert(int x){
  int now=1;v[now].pb(x);
  per(i,29,0){
    int w=((x&(1<<i))>0);
    if(!go[now][w])go[now][w]=++tot;
    now=go[now][w];
    v[now].pb(x&((1<<i)-1));
  }
}
void work(int x,int bit){
  //因为有这个 sort 所以 std 实际上是 2 个 log 的，但只要在外面把 a[i] sort 好再插入就可以消除掉这个 log
  sort(v[x].begin(),v[x].end());
  rep(i,0,v[x].size()-1){
    int tt;
    if(i+1!=v[x].size())tt=v[x][i+1]-v[x][i];
    else tt=(1<<(bit+1))-v[x][i];
    s1[x]=(s1[x]+tt*1ll*(i+1))%P;
    s2[x]=(s2[x]+tt*1ll*((i+1)*1ll*(i+1)%P))%P;
  }
}
void dfs(int x,int bit){
  work(x,bit);
  rep(i,0,1)if(go[x][i]){
    dfs(go[x][i],bit-1);
  }
}
int ans=0;
void evaluate(int now,int bit,int pre){
  //0...(1<<bit)-1
  //(x+pre)^2=x^2+2prex+pre^2
  int total=1<<bit;
  //printf("ev %d %d %d %d %d\n",now,bit,pre,s1[now],s2[now]);
  ans=(ans+s2[now])%P;
  ans=(ans+2ll*pre*s1[now])%P;
  ans=(ans+(pre*1ll*pre%P)*total)%P;
}
```

```
void gkd(int now,int l,int r,int x,int bit,int pre){
  if(!now){
    ans=(ans+(r-l+1)*1ll*(pre*1ll*pre%P))%P;
    return;
  }
  int w=((x&(1<<bit))>0);
  if(l==0&&r==(1<<(bit+1))-1){
    //printf("%d %d %d %d\n",now,l,r,bit);
    evaluate(now,bit+1,pre);
    return;
  }
  if(!w){
    if(l<(1<<bit))gkd(go[now][0],l,min(r,(1<<bit)-1),x,bit-1,pre);
    if(r>=(1<<bit))gkd(go[now][1],max(l,(1<<bit))-(1<<bit),r-(1<<bit),x,bit-
1,pre+v[go[now][0]].size());
  }
  else{
    if(l<(1<<bit))gkd(go[now][1],l,min(r,(1<<bit)-1),x,bit-1,pre+v[go[now]
[0]].size());
    if(r>=(1<<bit))gkd(go[now][0],max(l,(1<<bit))-(1<<bit),r-(1<<bit),x,bit-
1,pre);
  }
}
//l.....r
//<= i^x
int main(){
  scanf("%d%d",&n,&Q);
  rep(i,1,n)scanf("%d",&a[i]);
  rep(i,1,n)insert(a[i]);
  dfs(1,29);
  while(Q--){
    int l,r,x;scanf("%d%d%d",&l,&r,&x);
    ans=0;
    gkd(1,l,r,x,29,0);
    printf("%d\n",ans);
  }
  return 0;
}
```

I. 变大!

考虑最后答案的形式，肯定是分成若干段，然后每段的值都是一样的（且显然都是那一段的最大值）

那么我们就有一个简单的思路，DP 一下怎么分段使得答案最大，以及在已知分段的情况下的最少操作次数

首先我们可以假设每一段里最大值是唯一的，不然我们可以分成若干个小段来满足这个条件

那么对于一段长度为 L 的段，每次操作可以把两个数变成最大值，所以最少操作次数就是 $L/2$

所以进行一个简单的 $O(n^3)$ DP 就行了

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=105;
int n,a[N];
int f[N][N];
pii g[N][N];
pii eval(int l,int r){
  //标程想的比较复杂，没有规约到每段最大值唯一的情况，但最大值不唯一时还是可以用贪心来算次数
  if(l==r)return pii(a[l],0);
  int ma=0;
  rep(i,l,r)ma=max(ma,a[i]);
  vector<int> blk;
  int rp=0;
  rep(i,l,r)if(a[i]==ma){
    blk.pb(rp);
    rp=0;
  }
  else rp++;
  blk.pb(rp);
  int res=0;
  int pre=0;
  for(int x:blk){
    int w=max(x-pre,0);
    res+=(w+1)/2;
    pre=(w&1);
  }
```

```
  return pii(ma*(r-l+1),res);
}
void Main(){
  scanf("%d",&n);
  rep(i,1,n)scanf("%d",&a[i]);
  memset(f,0,sizeof f);
  rep(i,1,n)rep(j,i+2,n)g[i][j]=eval(i,j);

  rep(i,0,n)rep(j,0,n){
    f[i+1][j]=max(f[i+1][j],f[i][j]+a[i+1]);
    rep(r,i+3,n)f[r][j+g[i+1][r].se]=max(f[r][j+g[i+1][r].se],f[i][j]+g[i+1]
[r].fi);
  }
  rep(i,1,n)printf("%d%c",f[n][i],i==n?'\n':' ');
}
int main(){
  int t;scanf("%d",&t);
  while(t--)Main();
  return 0;
}
```

J. K 重排列

对于一个排列 $p_{1...n}$，我们令 i 向 p_i 连边，可以得到一个有向图，且构成是若干个环

而 p_i^k 相当于点 i 在图上走 k 次到达的点

所以 K 如果要是一个周期的话，等价于 K 是每个环长的倍数

那么我们就可以整一个 DP 来算方案数了，思路就是每次枚举 1 所在的环的长度

$f_n = \sum_{i=1}^{n}[i|K]C_{n-1}^{i-1}(i-1)!f_{n-i}$

其中 C_{n-1}^{i-1} 表示选出这 i 个点，因为 1 一定在里面所以不用选

$(i-1)!$ 是长度为 i 的有向环的个数

时间复杂度：$O(n^2)$

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
```

```cpp
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int P=998244353;
const int N=55;
int n;LL k;
int C[N][N];
int fac[N];
int f[N];
int circle(int x){
  return fac[x-1];
}
void Main(){
  scanf("%d%lld",&n,&k);
  fac[0]=1;rep(i,1,n)fac[i]=fac[i-1]*1ll*i%P;
  rep(i,0,n){
    C[i][0]=C[i][i]=1;
    rep(j,1,i-1)C[i][j]=(C[i-1][j-1]+C[i-1][j])%P;
  }
  f[0]=1;
  rep(i,1,n){
    int res=0;
    rep(j,1,i)if(k%j==0){
      int w=C[i-1][j-1];
      w=w*1ll*circle(j)%P;
      res=(res+w*1ll*f[i-j])%P;
    }
    f[i]=res;
  }
  printf("%d\n",f[n]);
}
int main(){
  int t;scanf("%d",&t);
  while(t--)Main();
  return 0;
}
```

K. 最大权值排列

$\sum_{l=1}^{n}\sum_{r=l}^{n}\frac{1}{r-l+1}\sum_{i=l}^{r}A_i$

$=\sum_{i=1}^{n}A_i\sum_{l\le i}\sum_{r\ge i}\frac{1}{r-l+1}$

设 $w_i=\sum_{l\le i}\sum_{r\ge i}\frac{1}{r-l+1}$

则答案就是 $\sum_{i=1}^{n}A_iw_i$

所以求出 w_i 后贪心地去放即可

可以证明越往中间，则 w_i 越大

考虑 $w_i-w_{i-1}=\sum_{r=i}^{n}\frac{1}{r-i+1}-\sum_{l=1}^{i-1}\frac{1}{i-l}$

当 $i<n-i+1$ 时，这个值是大于 0 的

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
int a[110000];
int n;
inline bool cmp(const int &x,const int &y){
  if(min(x-1,n-x)!=min(y-1,n-y))return min(x-1,n-x)>min(y-1,n-y);
  return x>y;
}
int p[110000];
int main(){
  scanf("%d",&n);
  rep(i,1,n)p[i]=i;
  sort(p+1,p+1+n,cmp);
  rep(i,1,n)a[p[i]]=n-i+1;
  rep(i,1,n)printf("%d%c",a[i],i==n?' ':' ');
  return 0;
```

```
}
```

L. 你吓到我的马了.jpg

直接 BFS，没什么好说的，注意别把马的起点 M 当做障碍就行了

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=105;
int dx[8]={2,2,-2,-2,1,1,-1,-1};
int dy[8]={1,-1,1,-1,2,-2,2,-2};
int wx[8]={1,1,-1,-1,0,0,0,0};
int wy[8]={0,0,0,0,1,-1,1,-1};
int n,m;
char a[N][N];
int dis[N][N];
pii q[N*N];int t;
bool canGo(int x,int y){
  if(x<=0||x>n)return 0;
  if(y<=0||y>m)return 0;
  return a[x][y]=='.';
}
int main(){
  scanf("%d%d",&n,&m);
  rep(i,1,n)scanf("%s",a[i]+1);
  int Sx=0;int Sy=0;
  rep(i,1,n)rep(j,1,m)if(a[i][j]=='M')Sx=i,Sy=j,a[i][j]='.';
  rep(i,1,n)rep(j,1,m)dis[i][j]=-1;
  q[t=1]=pii(Sx,Sy);
```

```
  dis[Sx][Sy]=0;
  rep(ii,1,t){
    int x=q[ii].fi;
    int y=q[ii].se;
    rep(k,0,7)if(canGo(x+dx[k],y+dy[k]))
    if(a[x+wx[k]][y+wy[k]]=='.'){
      int x1=x+dx[k];
      int y1=y+dy[k];
      if(dis[x1][y1]==-1){
        dis[x1][y1]=dis[x][y]+1;
        q[++t]=pii(x1,y1);
      }
    }
  }
  rep(i,1,n){
    rep(j,1,m)printf("%d%c",dis[i][j],j==m?'\n':' ');
  }
  return 0;
}
```

M. 自闭

直接模拟，也没啥好说的

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=5005;
int n,m,k;
```

```
bool ac[N],st[N];
struct atom{
  bool ac;
  int lst;int ma;
  void upd(int x){
    ac|=x;
    if(x){
      lst=0;
    }
    else{
      lst++;
      ma=max(ma,lst);
    }
  }
}res[105][15];
int num[105];
int main(){
  memset(res,0,sizeof res);
  scanf("%d%d%d",&n,&m,&k);
  rep(i,1,k){
    int x,y,c;scanf("%d%d%d",&x,&y,&c);
    res[x][y].upd(c);
    ac[x]|=c;
    st[x]|=1;
  }
  rep(i,1,m)rep(j,1,n)if(res[j][i].ac)num[i]++;
  rep(i,1,n){
    if(!st[i]){
      printf("998244353\n");
    }
    else
    if(!ac[i]){
      printf("1000000\n");
    }
    else{
      bool ak=1;
      rep(j,1,m)ak&=res[i][j].ac;
      if(ak){
        printf("0\n");
      }
      else{
        int ans=0;
        rep(j,1,m){
          if(!res[i][j].ac)if(num[j])ans+=20;
          if(!res[i][j].ac)if(num[j]>=n/2)ans+=10;
          ans+=res[i][j].ma*res[i][j].ma;
          if(!res[i][j].ac)ans+=res[i][j].ma*res[i][j].ma;
        }
        printf("%d\n",ans);
```

```
      }
    }
  }
  return 0;
}
```

N. 合并!

证明：合并完 S 后得到的总收益一定是 S 中每两个数的乘积，无论怎么合并

考虑数学归纳法，根据最后一次合并，设 $S = A + B$

A, B 的代价和为：都在 A 中的两个数的乘积，都在 B 中的两个数的乘积

而 $sum(A)sum(B)$ 是一个在 A 中一个在 B 中的两个数的乘积，所以加起来刚好是 S 中每两个数的乘积

```
#include <bits/stdc++.h>
using namespace std;
int a[2010];

int main()
{
  int n; scanf("%d", &n);
  for (int i = 0; i < n; i++)
    scanf("%d", &a[i]);
  long long ans = 0;
  for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
      ans += a[i] * a[j];
  printf("%lld\n", ans);
  return 0;
}
```