ACM Template

zhsl

Last build at October 9, 2013

Contents

1 To Do List

1.1 Header file

```
#include <functional>
#include <algorithm>
#include <iostream>
//#include <ext/rope>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <numeric>
#include <cstring>
#include <cassert>
#include <cstdio>
#include <string>
#include <vector>
#include <bitset>
#include <queue>
#include <stack>
#include <cmath>
#include <ctime>
#include <list>
#include <set>
#include <map>
using namespace std;
//#pragma comment(linker,"/STACK:102400000,102400000")
//using namespace __gnu_cxx;
//define
#define pii pair<int,int>
#define mem(a,b) memset(a,b,sizeof(a))
#define lson l,mid,rt<<1
#define rson mid+1,r,rt<<1|1
#define PI acos(-1.0)
//typedef
typedef __int64 LL;
typedef unsigned __int64 ULL;
//const
const int N=1010;
const int INF=0x3f3f3f3f;
const int MOD=100000,STA=8000010;
const LL LNF=1LL<<60;
const double EPS=1e-8;
const double OO=1e15;
const int dx[4]={-1,0,1,0};
const int dy[4]={0,1,0,-1};
const int day[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
//Daily Use ...
inline int sign(double x){return (x>EPS)-(x<-EPS);}
template<class T> T gcd(T a,T b){return b?gcd(b,a%b):a;}
template<class T> T lcm(T a,T b){return a/gcd(a,b)*b;}
template<class T> inline T lcm(T a,T b,T d){return a/d*b;}
template<class T> inline T Min(T a,T b){return a<b?a:b;}
template<class T> inline T Max(T a,T b){return a>b?a:b;}
template<class T> inline T Min(T a,T b,T c){return min(min(a, b),c);}
template<class T> inline T Max(T a,T b,T c){return max(max(a, b),c);}
template<class T> inline T Min(T a,T b,T c,T d){return min(min(a, b),min(c,d));}
template<class T> inline T Max(T a,T b,T c,T d){return max(max(a, b),max(c,d));}
//End

int main()
{
```

```
//   freopen("in.txt","r",stdin);

    return 0;
}
```

1.2 数据输入加速

```
/*  数据输入加速 - from 风神
调用 scan_d(n) 对输入数据比较多的题目，有非常明显的加速效果!!!  */

//适用于正整数
template <class T>
inline void scan_d(T &ret) {
    char c; ret=0;
    while((c=getchar())<'0'||c>'9');
    while(c>='0'&&c<='9') ret=ret*10+(c-'0'),c=getchar();
}

//适用于正整数
template <class T>
inline void scan_d(T &ret) {
    char c; ret=0;
    while((c=getchar())<'0'||c>'9');
    while(c>='0'&&c<='9') ret=ret*10+(c-'0'),c=getchar();
}

//适用于正负数,(int,long long,float,double)
template <class T>
bool scan_d(T &ret){
    char c; int sgn; T bit=0.1;
    if(c=getchar(),c==EOF) return 0;
    while(c!='-'&&c!='.'&&(c<'0'||c>'9')) c=getchar();
    sgn=(c=='-')?-1:1;
    ret=(c=='-')?0:(c-'0');
    while(c=getchar(),c>='0'&&c<='9') ret=ret*10+(c-'0');
    if(c==' '||c=='\n'){ ret*=sgn; return 1; }
    while(c=getchar(),c>='0'&&c<='9') ret+=(c-'0')*bit,bit/=10;
    ret*=sgn;
    return 1;
}

inline void out(int x) {
   if(x>9) out(x/10);
   putchar(x%10+'0');
}
```

2 数学

2.1 数学基础

```
/*  number theory basic
  -gcd
  -external gcd
  -inverse
  -Chinese Remainder Theorem
  -Modline
  -primetable
  -eulerphi
  -eulerphi table        */

LL a[N],m[N],phi[N];
int n;

LL gcd(LL a,LL b){return b?gcd(b,a%b):a;}

//求出来的 (x,y), 有 |x|+|y| 最小
void exgcd(LL a,LL b,LL &d,LL &x,LL &y)
{
    if(!b){d=a;x=1;y=0;}
    else {exgcd(b,a%b,d,y,x);y-=x*(a/b);}
}

LL inv(LL a,LL n)
{
    LL d,x,y;
    exgcd(a,n,d,x,y);
    return d==1?(x+n)%n:-1;
}

//求 ax = 1( mod m) 的 x 值，就是逆元 (0<a<m)
LL inv(LL a,LL m)
{
    if(a == 1)return 1;
    return inv(m%a,m)*(m-m/a)%m;
}

LL china()
{
    int i;
    LL M=1,w,d,y,x=0;
    for(i=0;i<n;i++)M*=m[i];
    for(i=0;i<n;i++){
        w=M/m[i];
        exgcd(m[i],w,d,d,y);
        x=(x+y*a[i])%M;
    }
    return (x+M)%M;
}

LL Modline(int n)
{
    LL d,x,y,A,M,Mod;
    A=a[n-1],M=m[n-1];
    n--;
    // m1*x-m2*y=a2-a1
    while(n--){
        exgcd(M,m[n],d,x,y);
        if((A-a[n])%d!=0){
```

```
            return -1;
        }
        Mod=m[n]/d;
        x=(x*((a[n]-A)/d)%Mod+Mod)%Mod;
        A+=M*x;
        M=M/d*m[n];
    }
    return A;
}

/*  primetable  O(n)
 isprime[i]=0 为素数
 prime 存储素数
 cnt 为素数个数     */
int isprime[N],prime[N];
int cnt;
void primetable(int n)
{
    int i,j;
    //Init isprime[N],prime[N], 全局变量初始为 0
    cnt=0;isprime[1]=1;
    for(i=2;i<=n;i++){
        if(!isprime[i])prime[cnt++]=i;
        for(j=0;j<cnt && i*prime[j]<=n;j++){
            isprime[i*prime[j]]=1;
            if(i%prime[j]==0)break;
        }
    }
}

LL eulerphi(LL n)
{
    int i,j;
    LL m,ans=n;
    m=(LL)sqrt(n+0.5);
    for(i=2;i<=m && i<n;i++){
        if(n%i==0){
            ans=ans/i*(i-1);
            while(n%i==0)n/=i;
        }
    }
    if(n>1)ans=ans/n*(n-1);
    return ans;
}

/*   eulerphi table O(n) 算法
 效率是下面那个 phitable 的 3-4 倍!   O(n)
 主要是递推优化:
    如果 i%prime[j], 那么 phi[i*prime[j]]=
n(p1-1)/p1*...(pn-1)/pn*(prime[j]-1)/prime[j]*prime[j]=phi[i]*(prime[j]-1)
    否则: phi[i*prime[j]]=n(p1-1)/p1*...(pn-1)/pn*prime[j]=phi[i]*(prime[j]-1)
prime 存储素数
cnt 为 1-n 之间的素数个数    */
int phi[N],prime[N];
int cnt;
void phitable(int n)
{
    int i,j;
    //Init phi[N],prime[N], 全局变量初始为 0
    cnt=0;phi[1]=1;
    for(i=2;i<=n;i++){
        if(!phi[i]){
```

```
            prime[cnt++]=i;  //prime[i]=1; 为素数表
            phi[i]=i-1;
        }
        for(j=0;j<cnt && i*prime[j]<=n;j++){
            if(i%prime[j])
                phi[i*prime[j]]=phi[i]*(prime[j]-1);
            else {phi[i*prime[j]]=phi[i]*prime[j];break;}
        }
    }
}

/*  eulerphi table  朴素算法 O(n*loglogn)   */
int phi[N],isprime[N];
void phitable(int n)
{
    int i,j;
    //Init phi[N],isprime[N], 全局变量初始为 0
    phi[1]=1;
    for(i=2;i<=n;i++)if(!phi[i]){
      //  isprime[i]=1;  //筛质数, isprime[i]=1 为质数
        for(j=i;j<=n;j+=i){
            if(!phi[j])phi[j]=j;
            phi[j]=phi[j]/i*(i-1);
        }
    }
}
```

2.2 factorial 相关

```
/*   factorial 相关
  -求 n! 某个因子 k 的个数  n!=(k^m)*(m!)*a
  推导:
    n!=n*(n-1)*(n-2)*......3*2*1
      =(k*2k*3k.....*mk)*a      a 是不含因子 k 的数的乘积, 显然 m=n/k;
      =(k^m)*(1*2*3...*m)*a
      =k^m*m!*a             */

LL ncount(LL n,LL k){
  LL cou=0;
while(n)
cou+=n/=k;
return cou;
}
```

2.3 Lucas

```
/* Lucas 定理求 C(n, m)%p
 Lucas(n,m,p)=C(n%p,m%p)* Lucas(n/p,m/p,p)
 Lucas(n,0,p)=1;      */

#define LL __int64
LL fac[MAX];   //MAX<=p,p 为取模数

void init(LL p)
{
    int i;
    fac[0]=1;
    for(i=1;i<=p;i++)
        fac[i]=fac[i-1]*i%p;
}
```

```
LL pow(LL a, LL b,LL p)
{
    LL tmp=a%p,ans=1;
    while(b)
    {
        if(b&1)ans=ans*tmp%p;
        tmp=tmp*tmp%p;
        b>>=1;
    }
    return  ans;
}

LL C(LL n, LL m,LL p)
{
    return  m>n?0:fac[n]*pow(fac[m]*fac[n-m],p-2,p)%p;
}

LL lucas(LL n,LL m,LL p)
{
    return  m?(C(n%p,m%p,p)*lucas(n/p,m/p,p))%p:1;
}
```

2.4 矩阵乘法

```
/*  Matrix multiplication
  size 为矩阵大小
  A 为初始矩阵   */

const int 1000000007;
const int size=7;

struct Matrix{
    LL ma[size][size];
    Matrix friend operator * (const Matrix a,const Matrix b){
        Matrix ret;
        mem(ret.ma,0);
        int i,j,k;
        for(i=0;i<size;i++)
            for(j=0;j<size;j++)
                for(k=0;k<size;k++)
                    ret.ma[i][j]=(ret.ma[i][j]+a.ma[i][k]*b.ma[k][j]%MOD)%MOD;
        return ret;
    }
}A;

Matrix mutilpow(LL k)
{
    int i;
    Matrix ret;
    mem(ret.ma,0);
    for(i=0;i<size;i++)
        ret.ma[i][i]=1;
    for(;k;k>>=1){
        if(k&1)ret=ret*A;
        A=A*A;
    }
    return ret;
}
```

2.5 康拓展开

```
/*   Cantor 展开
  Cantor() 返回的值从下标 0 开始
  len 为数组长度        */

const int PermSize = 12;
int factory[PermSize] =
{1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800};
int Cantor(int a[],int len)
{
    int i, j, counted;
    int result = 0;
    for (i = 0; i < len; ++i)
    {
        counted = 0;
        for (j = i + 1; j < len; ++j)
            if (a[i] > a[j])
                ++counted;
        result = result + counted * factory[len - i - 1];
    }
    return result;
}

bool h[13];

void UnCantor(int x, int res[],int len)
{
    int i,j,l,t;
    for (i = 1; i <= len; i++)
        h[i] = false;
    for (i = 1; i <= len; i++)
    {
        t = x / factory[len - i];
        x -= t * factory[len - i];
        for (j = 1, l = 0; l <= t; j++)
            if (!h[j])l++;
        j--;
        h[j] = true;
        res[i - 1] = j;
    }
}
```

2.6 pollard rho 质因数分解

```
/*   pollard rho
  pollard_rho 算法进行质因数分解  */

LL factor[100];   //质因数分解结果（刚返回时是无序的）
int tol;   //质因数的个数。数组小标从 0 开始

LL gcd(LL a,LL b)
{
    if(a==0)return 1;
    if(a<0) return gcd(-a,b);
    while(b)
    {
        LL t=a%b;
        a=b;
        b=t;
    }
    return a;
```

```
}

LL Pollard_rho(LL x,LL c)
{
    LL i=1,k=2;
    LL x0=rand()%x;
    LL y=x0;
    while(1)
    {
        i++;
        x0=(mult_mod(x0,x0,x)+c)%x;
        LL d=gcd(y-x0,x);
        if(d!=1&&d!=x) return d;
        if(y==x0) return x;
        if(i==k){y=x0;k+=k;}
    }
}
//对 n 进行素因子分解
void findfac(LL n)
{
    if(Miller_Rabin(n))//素数
    {
        factor[tol++]=n;
        return;
    }
    LL p=n;
    while(p>=n)p=Pollard_rho(p,rand()%(n-1)+1);
    findfac(p);
    findfac(n/p);
}
```

2.7 Miller Rabin

```
/*  Miller Rabin b  */

/* (吉大模板), poj 验证超时, 谨慎使用。
   CALL: bool res = miller(n);
   快速测试 n 是否满足素数的' 必要' 条件, 出错概率很小;
   对于任意奇数 n>2 和正整数 s, 算法出错概率 <= 2^(-s);
   初始化时间种子: srand(time(NULL));      */
LL witness(LL a, LL n)
{
    LL x,d=1,i=ceil(log(n-1.0)/log(2.0))-1;
    for (; i>=0; i--)
    {
        x=d;
        d=(d * d)%n;
        if (d==1 && x!=1 && x!=n-1) return 1;
        if (((n-1)&(1<<i))>0)d=(d*a)%n;
    }
    return d==1?0:1;
}

LL miller(LL n, LL s=50)
{
    if (n==2) return 1;
    if ((n%2)==0) return 0;
    int j,a;
    for (j=0; j<s; j++)
    {
        a=rand()*(n-2)/RAND_MAX+1;
```

```
        // rand() 只能随机产生 [0, RAND_MAX) 内的整数
        // 而且这个 RAND_MAX 只有 32768 直接%n 的话永远也产生不了
        // [RAND-MAX, n) 之间的数
        if (witness(a, n)) return 0;
    }
    return 1;
}

/*  网络模板，时间效率不错
  Miller_Rabin 算法进行素数测试
  速度快，而且可以判断 <2^63 的数
  srand(time(NULL)); 需要 time.h 头文件  POJ 上 G++ 要去掉这句话
  const int S=20;  随机算法判定次数，S 越大，判错概率越小
  计算 (a*b)%c.   a,b 都是 long long 的数，直接相乘可能溢出的
  a,b,c <2^63                                               */
LL mult_mod(LL a,LL b,LL c)
{
    a%=c;
    b%=c;
    LL ret=0;
    while(b)
    {
        if(b&1){ret+=a;ret%=c;}
        a<<=1;
        if(a>=c)a%=c;
        b>>=1;
    }
    return ret;
}
//计算  x^n %c
LL pow_mod(LL x,LL n,LL mod)//x^n%c
{
    if(n==1)return x%mod;
    x%=mod;
    LL tmp=x;
    LL ret=1;
    while(n)
    {
        if(n&1) ret=mult_mod(ret,tmp,mod);
        tmp=mult_mod(tmp,tmp,mod);
        n>>=1;
    }
    return ret;
}
//以 a 为基,n-1=x*2^t      a^(n-1)=1(mod n)  验证 n 是不是合数
//一定是合数返回 true, 不一定返回 false
bool check(LL a,LL n,LL x,LL t)
{
    LL ret=pow_mod(a,x,n);
    LL last=ret;
    for(int i=1;i<=t;i++)
    {
        ret=mult_mod(ret,ret,n);
        if(ret==1&&last!=1&&last!=n-1) return true;//合数
        last=ret;
    }
    if(ret!=1) return true;
    return false;
}

// Miller_Rabin() 算法素数判定
//是素数返回 true.(可能是伪素数，但概率极小)
```

```
//合数返回 false;
bool Miller_Rabin(LL n)
{
    if(n<2)return false;
    if(n==2)return true;
    if((n&1)==0) return false;//偶数
    LL x=n-1;
    LL t=0;
    while((x&1)==0){x>>=1;t++;}
    for(int i=0;i<S;i++)
    {
        LL a=rand()%(n-1)+1;//rand() 需要 stdlib.h 头文件
        if(check(a,n,x,t))
            return false;//合数
    }
    return true;
}
```

2.8 Mobius 函数

```
/*    Mobius 函数   O(n)
Mobius 反演定理
  已知 f(n) = sigma(d|n, g(d))
        那么 g(n) = sigma(d|n, mu(d)*f(n/d))
  还有另一种形式更常用:
      在某一范围内，已知 f(n) = sigma(n|d, g(d))
      那么 g(n) = sigma(n|d, mu(d/n)*f(d))

Mobius 函数:
          1        n=1
  mu(n)= (-1)^k    n=p1*p2*...*pk, pi 为不相同的质数
          0        others                           */

int isprime[N],mu[N],prime[N];
int cnt;
void Mobius(int n)
{
    int i,j;
    //Init isprime[N],mu[N],prime[N], 全局变量初始为 0
    cnt=0;mu[1]=1;
    for(i=2;i<=n;i++){
        if(!isprime[i]){
            prime[cnt++]=i;
            mu[i]=-1;
        }
        for(j=0;j<cnt && i*prime[j]<=n;j++){
            isprime[i*prime[j]]=1;
            if(i%prime[j])
                mu[i*prime[j]]=-mu[i];
            else {mu[i*prime[j]]=0;break;}
        }
    }
}
```

2.9 [1,n] 与 a 互素个数

```
/*   [1,n] 与 a 互素个数   O(sqrt n)
  先对 a 分解质因数
  然后用容斥原理             */
```

```
int fac[50];
int solve (int n, int a){
    int i,j,up,t,cnt=0,sum=0,flag;
    for(i=2;i*i<=a;i++)
        if(a%i==0){
            fac[cnt++]=i;
            while(a%i==0)a/=i;
        }
    if(a>1)fac[cnt++]=a;
    up=1<<cnt;
    for(i=1;i<up;i++){   //容斥原理，二进制枚举
        flag=0,t=1;
        for(j=0;j<cnt;j++){
            if(i&(1<<j)){
                flag^=1;
                t*=fac[j];
            }
        }
        sum+=flag?n/t:-(n/t);
    }
    return n-sum;
}
```

2.10 Bernoulli number

```
/*   bernoulli 方程
     Sn(m)=1^n+2^n+...+(m-1)^n
  => Sn(m)=1/(m+1)(0~m)ΣC(m+1,k)Bk(m^(n+1-k))
  其中:B0=1 , (0,m)ΣC(m+1,k)Bk=0     */

LL B[N][2],C[N][N],f[N][2];
int n,m;      //n 为幂大小

LL gcd(LL a,LL b){return b?gcd(b,a%b):a;}
LL lcm(LL a,LL b){return a/gcd(a,b)*b;}

void getC(int n)
{
    int i,j;
    n++;
    for(i=0;i<=n;i++)C[i][0]=C[i][i]=1;
    for(i=2;i<=n;i++){
        for(j=1;j<n;j++){
            C[i][j]=C[i-1][j-1]+C[i-1][j];
        }
    }
}

void bernoulli(int n)    //得到 B 数组
{
    int i,m;
    LL s[2],b[2],l,g;
    B[0][0]=1;B[0][1]=1;
    for(m=1;m<=n;m++){
        s[0]=1,s[1]=1;
        for(i=1;i<m;i++){
            b[0]=C[m+1][i]*B[i][0];
            b[1]=B[i][1];
            l=lcm(s[1],b[1]);
            s[0]=l/s[1]*s[0]+l/b[1]*b[0];
            s[1]=l;
```

```
        }
        s[0]=-s[0];
        if(s[0]){
            g=gcd(s[0],s[1]*C[m+1][m]);
            B[m][0]=s[0]/g;
            B[m][1]=s[1]*C[m+1][m]/g;
        }
        else B[m][0]=0,B[m][1]=1;
    }
}
```

2.11 大数幂取模

```
/*     大数幂取模   O(log n)
  公式: A^x = A^(x % Phi(C) + Phi(C)) (mod C)  &&  x >= Phi(C)   */

//  a^b%c   只有 b 为大数
#define nnum 1000005
#define nmax 31625
int flag[nmax], prime[nmax];
int plen;
void mkprime() {
    int i, j;
    memset(flag, -1, sizeof(flag));
    for (i = 2, plen = 0; i < nmax; i++) {
        if (flag[i]) {
            prime[plen++] = i;
        }
        for (j = 0; (j < plen) && (i * prime[j] < nmax); j++) {
            flag[i * prime[j]] = 0;
            if (i % prime[j] == 0) {
                break;
            }
        }
    }
}
int getPhi(int n) {
    int i, te, phi;
    te = (int) sqrt(n * 1.0);
    for (i = 0, phi = n; (i < plen) && (prime[i] <= te); i++) {
        if (n % prime[i] == 0) {
            phi = phi / prime[i] * (prime[i] - 1);
            while (n % prime[i] == 0) {
                n /= prime[i];
            }
        }
    }
    if (n > 1) {
        phi = phi / n * (n - 1);
    }
    return phi;
}
int cmpCphi(int p, char *ch) {
    int i, len;
    LL res;
    len = strlen(ch);
    for (i = 0, res = 0; i < len; i++) {
        res = (res * 10 + (ch[i] - '0'));
        if (res > p) {
            return 1;
        }
```

```
    }
    return 0;
}
int getCP(int p, char *ch) {
    int i, len;
    LL res;
    len = strlen(ch);
    for (i = 0, res = 0; i < len; i++) {
        res = (res * 10 + (ch[i] - '0')) % p;
    }
    return (int) res;
}
int modular_exp(int a, int b, int c) {
    LL res, temp;
    res = 1 % c, temp = a % c;
    while (b) {
        if (b & 1) {
            res = res * temp % c;
        }
        temp = temp * temp % c;
        b >>= 1;
    }
    return (int) res;
}
void solve(int a, int c, char *ch) {
    int phi, res, b;
    phi = getPhi(c);
    if (cmpCphi(phi, ch)) {
        b = getCP(phi, ch) + phi;
    } else {
        b = atoi(ch);
    }
    res = modular_exp(a, b, c);
    printf("%d\n", res);
}
int main() {
 //   freopen("data.in", "r", stdin);
    int a, c;
    char ch[nnum];
    mkprime();
    while (~scanf("%d %s %d", &a, ch, &c)) {
        solve(a % c, c, ch);
    }
    return 0;
}

//  a^b%c   a 和 b 都为大数
#define nnum 1000005
#define nmax 31625
int flag[nmax], prime[nmax];
int plen;
void mkprime() {
    int i, j;
    memset(flag, -1, sizeof(flag));
    for (i = 2, plen = 0; i < nmax; i++) {
        if (flag[i]) {
            prime[plen++] = i;
        }
        for (j = 0; (j < plen) && (i * prime[j] < nmax); j++) {
            flag[i * prime[j]] = 0;
            if (i % prime[j] == 0) {
                break;
```

```
            }
        }
    }
}
int getPhi(int n) {
    int i, te, phi;
    te = (int) sqrt(n * 1.0);
    for (i = 0, phi = n; (i < plen) && (prime[i] <= te); i++) {
        if (n % prime[i] == 0) {
            phi = phi / prime[i] * (prime[i] - 1);
            while (n % prime[i] == 0) {
                n /= prime[i];
            }
        }
    }
    if (n > 1) {
        phi = phi / n * (n - 1);
    }
    return phi;
}
int cmpBigNum(int p, char *ch) {
    int i, len;
    LL res;
    len = strlen(ch);
    for (i = 0, res = 0; i < len; i++) {
        res = (res * 10 + (ch[i] - '0'));
        if (res > p) {
            return 1;
        }
    }
    return 0;
}
int getModBigNum(int p, char *ch) {
    int i, len;
    LL res;
    len = strlen(ch);
    for (i = 0, res = 0; i < len; i++) {
        res = (res * 10 + (ch[i] - '0')) % p;
    }
    return (int) res;
}
int modular_exp(int a, int b, int c) {
    LL res, temp;
    res = 1 % c, temp = a % c;
    while (b) {
        if (b & 1) {
            res = res * temp % c;
        }
        temp = temp * temp % c;
        b >>= 1;
    }
    return (int) res;
}
void solve(int a, int c, char *ch) {
    int phi, res, b;
    phi = getPhi(c);
    if (cmpBigNum(phi, ch)) {
        b = getModBigNum(phi, ch) + phi;
    } else {
        b = atoi(ch);
    }
    res = modular_exp(a, b, c);
```

```
    printf("%d\n", res);
}
int main() {
 // freopen("data.in", "r", stdin);
    int a, c;
    char cha[nnum], chb[nnum];
    mkprime();
    while (~scanf("%s %s %d", cha, chb, &c)) {
        a = getModBigNum(c, cha);
        solve(a, c, chb);
    }
    return 0;
}
```

2.12 快速傅里叶

```
/*  FFT    O(n*logn)
 多项式转化为点值表示法，利用 n 次单位复根来分治运算
 例如做大数乘法：
    1. 得到向量 a,b  DFT
    2.a,b 作一次乘积得到向量 c
    3. 向量 c 作 FFT   IDFT
    4. 转化结果                        */

//定义复数结构体
struct complex
{
    double r,i;
    complex(double _r = 0.0,double _i = 0.0){r = _r; i = _i;}
    complex operator +(const complex &b){return complex(r+b.r,i+b.i);}
    complex operator -(const complex &b){return complex(r-b.r,i-b.i);}
    complex operator *(const complex &b){return complex(r*b.r-i*b.i,r*b.i+i*b.r);}
};
/*
 * 进行 FFT 和 IFFT 前的反转变换。
 * 位置 i 和 （i 二进制反转后位置）互换
 * len 必须去 2 的幂
 */
void change(complex y[],int len)
{
    int i,j,k;
    for(i = 1, j = len/2;i < len-1; i++)
    {
        if(i < j)swap(y[i],y[j]);
        //交换互为小标反转的元素，i<j 保证交换一次
        //i 做正常的 +1，j 左反转类型的 +1，始终保持 i 和 j 是反转的
        k = len/2;
        while( j >= k)
        {
            j -= k;
            k /= 2;
        }
        if(j < k) j += k;
    }
}
/*
 * 做 FFT
 * len 必须为 2^k 形式，
 * on==1 时是 DFT，on==-1 时是 IDFT
 */
void FFT(complex y[],int len,int on)
```

```
{
    change(y,len);
    for(int h = 2; h <= len; h <<= 1)
    {
        complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h));
        for(int j = 0;j < len;j+=h)
        {
            complex w(1,0);
            for(int k = j;k < j+h/2;k++)
            {
                complex u = y[k];
                complex t = w*y[k+h/2];
                y[k] = u+t;
                y[k+h/2] = u-t;
                w = w*wn;
            }
        }
    }
    if(on == -1)
        for(int i = 0;i < len;i++)
            y[i].r /= len;
}

/*  大数乘法  O(n*logn)  */

char s1[N],s2[N],s[N];
int ans[N];
complex a[N],b[N];

void mutil(char *s1,char *s2,char *s)
{
    int i,len1,len2,len;
    len1=strlen(s1);
    len2=strlen(s2);
    len=1;
   while(len<(len1<<1) || len<(len2<<1))len<<=1;  //len 必须为 2^k 形式且大于 2*max(len1,len2)
   for(i=0;i<len1;i++)a[i]=complex(s1[len1-i-1]-'0',0);
    for(;i<len;i++)a[i]=complex(0,0);
    for(i=0;i<len2;i++)b[i]=complex(s2[len2-i-1]-'0',0);
    for(;i<len;i++)b[i]=complex(0,0);
    //DFT
    FFT(a,len,1);
    FFT(b,len,1);
    for(i=0;i<len;i++)a[i]=a[i]*b[i];
    //IDFT
    FFT(a,len,-1);
    //作乘积
    for(i=0;i<len;i++)ans[i]=(int)(a[i].r+0.5);
    //进位转化
    len=len1+len2-1;
    for(i=0;i<len;i++){
        ans[i+1]+=ans[i]/10;
        ans[i]%=10;
    }

    for(i=len;ans[i]<=0 && i>0;i--);
    len=i;
    for(;i>=0;i--)s[i]=(ans[len-i]+'0');
    s[len+1]=0;
}

int main(){
```

```
//    freopen("in.txt","r",stdin);
    int i,j,len1,len2,len;
    while(~scanf("%s%s",s1,s2))
    {
        mutil(s1,s2,s);
        printf("%s\n",s);
    }
    return 0;
}
```

2.13 高斯消元

```
/*   gauss_elimination  O(n^3)
   return 1，有解，return 0，无解。。
   n 个方程 n 个变元
   要求系数矩阵可逆
   A[][] 是增广矩阵，即 A[i][n] 是第 i 个方程右边的常数 bi
   运行结束后 A[i][n] 是第 i 个未知数的值    */

double A[N][N];

int gauss(int n)
{
    int i,j,k,r;
    for(i=0;i<n;i++){
        //选一行与 r 与第 i 行交换，提高数据值的稳定性
        r=i;
        for(j=i+1;j<n;j++)
            if(fabs(A[j][i]) > fabs(A[r][i]))r=j;
        if(r!=i)for(j=0;j<=n;j++)swap(A[r][j],A[i][j]);
        //i 行与 i+1~n 行消元
      /*  for(k=i+1;k<n;k++){   //从小到大消元，中间变量 f 会有损失
            double f=A[k][i]/A[i][i];
            for(j=i;j<=n;j++)A[k][j]-=f*A[i][j];
        }*/
        for(j=n;j>=i;j--){   //从大到小消元，精度更高
            for(k=i+1;k<n;k++)
                A[k][j]-=A[k][i]/A[i][i]*A[i][j];
        }
    }
    //判断方程时候有解
    for(i=0;i<n;i++)if(sign(A[i][i])==0)return 0;
    //回代过程
    for(i=n-1;i>=0;i--){
        for(j=i+1;j<n;j++)
            A[i][n]-=A[j][n]*A[i][j];
        A[i][n]/=A[i][i];
    }
}


/*   整数矩阵   O( n^3 )
  高斯消元法解方程组 (Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，
  -1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
  有 equ 个方程，var 个变元。增广矩阵行数为 equ，分别为 0 到 equ-1，列数为 var+1，分别为 0 到
var.    */

int a[N][N];//增广矩阵
int x[N];//解集
bool free_x[N];//标记是否是不确定的变元
int n,m,k;
```

```
//
void Debug(int equ,int var)
{
    int i, j;
    for (i = 0; i < equ; i++)
    {
        for (j = 0; j < var + 1; j++)
        {
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}
//

inline int gcd(int a,int b)
{
    int t;
    while(b!=0)
    {
        t=b;
        b=a%b;
        a=t;
    }
    return a;
}
inline int lcm(int a,int b)
{
    return a/gcd(a,b)*b;//先除后乘防溢出
}

// 高斯消元法解方程组 (Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，
//-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
//有 equ 个方程，var 个变元。增广矩阵行数为 equ, 分别为 0 到 equ-1, 列数为 var+1, 分别为 0 到
var.
int Gauss(int equ,int var)
{
    int i,j,k;
    int max_r; // 当前这列绝对值最大的行.
    int col; //当前处理的列
    int ta,tb;
    int LCM;
    int temp;
    int free_x_num;
    int free_index;

    for(int i=0;i<=var;i++)
    {
        x[i]=0;
        free_x[i]=true;
    }

    //转换为阶梯阵.
    col=0; // 当前处理的列
    for(k = 0;k < equ && col < var;k++,col++)
    {// 枚举当前处理的行.
// 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
        max_r=k;
        for(i=k+1;i<equ;i++)
        {
```

```
            if(abs(a[i][col])>abs(a[max_r][col])) max_r=i;
        }
        if(max_r!=k)
        {// 与第 k 行交换.
            for(j=k;j<var+1;j++) swap(a[k][j],a[max_r][j]);
        }
        if(a[k][col]==0)
        {// 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
            k--;
            continue;
        }
        for(i=k+1;i<equ;i++)    //  i=0 高斯约当消元，才能在多解的情况下判断变元是否确定
        {// 枚举要删去的行.
            if(a[i][col]!=0 && i!=k)
            {
                LCM = lcm(abs(a[i][col]),abs(a[k][col]));
                ta = LCM/abs(a[i][col]);
                tb = LCM/abs(a[k][col]);
                if(a[i][col]*a[k][col]<0)tb=-tb;//异号的情况是相加
                for(j=0;j<var+1;j++)
                {
                    a[i][j] = a[i][j]*ta - a[k][j]*tb;
                }
            }
        }
    }

  //  Debug(equ,var);

    // 1. 无解的情况: 化简的增广阵中存在 (0, 0, ..., a) 这样的行 (a != 0).
    for (i = k; i < equ; i++)
    {
        if (a[i][col] != 0) return -1;
    }
    // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.
    // 2. 无穷解的情况: 在 var * (var + 1) 的增广阵中出现 (0, 0, ..., 0) 这样的行，即说明没有形
成严格的上三角阵.
    // 且出现的行数即为自由变元的个数.
    if (k < var)return 1;
    {
        // 首先，自由变元有 var - k 个，即不确定的变元至少有 var - k 个.
        for (i = k - 1; i >= 0; i--)
        {
            // 第 i 行一定不会是 (0, 0, ..., 0) 的情况，因为这样的行是在第 k 行到第 equ 行.
            // 同样，第 i 行一定不会是 (0, 0, ..., a), a != 0 的情况，这样的无解的.
           free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，它
们仍然为不确定的变元.
            for (j = 0; j < var; j++)
            {
                if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
            }
            if (free_x_num > 1) continue; // 无法求解出确定的变元.
            // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，且该变元是确定的.
            temp = a[i][var];
            for (j = 0; j < var; j++)
            {
                if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
            }
            x[free_index] = temp / a[i][free_index]; // 求出该变元.
            free_x[free_index] = 0; // 该变元是确定的.
        }
        return var - k; // 自由变元有 var - k 个.
```

```
    }
    // 3. 唯一解的情况: 在 var * (var + 1) 的增广阵中形成严格的上三角阵.
    // 计算出 Xn-1, Xn-2 ... X0.
    for (i = var - 1; i >= 0; i--)
    {
        temp = a[i][var];
        for (j = i + 1; j < var; j++)
        {
            if (a[i][j] != 0) temp -= a[i][j] * x[j];
        }
        if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
        x[i] = temp / a[i][i];
    }
    return 0;
}



/*  异或矩阵  O( n^3 )
   高斯-约当消元
   如果无解 return -1，如果有唯一解，返回变化个数
   如果方程有多个解，则二进制枚举自由变元 O(2^n)，返回最小变化个数!，
   n 个方程 n 个变元
   A[][] 增广矩阵
   B[] 结果矩阵
   is_free[] 在有多解的情况下，判断元素解是否唯一 */

int A[N][N],B[N],is_free[N],num[N];

void getA(int n)
{
    /*  得到增光矩阵 A[][]   */
}

int gauss(int n)
{
    int i,j,k,cnt,row,ok,ret,up,cnt_free;
    for(i=row=0;i<n;i++){
        if(!A[row][i]){
            for(j=row+1;j<n;j++){
                if(A[j][i]){
                    for(k=i;k<=n;k++)swap(A[row][k],A[j][k]);
                    break;
                }
            }
        }
        if(A[row][i]!=1)continue;    //保证为严格的阶梯矩阵
        for(j=0;j<n;j++){    //从 0 开始，高斯约当消元
            if(j!=row && A[j][i]){
                for(k=i;k<=n;k++)
                    A[j][k]^=A[row][k];
            }
        }
        row++;
    }
    for(i=n-1;i>=row;i--)
        if(A[i][n])return -1;   //无解
    if(row==n){    //唯一解
        for(i=ret=0;i<n;i++)if(A[i][n])ret++;
        return ret;
    }
    mem(is_free,0);
```

```
    for(i=k=j=0;i<n;i++,j++){
        while(!A[i][j] && j<n){
            is_free[j]=1;   //判断元素是否解唯一
            num[k++]=j++;
        }
    }
    ret=INF;cnt_free=n-row;   //自由变元个数
    up=1<<cnt_free;
    for(k=0;k<up;k++){    //枚举最小的变换个数
        for(i=0;i<cnt_free;i++)B[num[i]]=(k&(1<<i))?1:0;
        for(i=n-1;i>=0;i--){
            if(is_free[i])continue;
            B[i]=0;
            for(j=row;j<n;j++)B[i]^=B[j]*A[i][j];
            B[i]^=A[i][n];
        }
        for(i=cnt=0;i<n;i++)if(B[i])cnt++;
        ret=Min(ret,cnt);
    }
    return ret;    //返回最小的变换个数
}
```

3 数据结构

3.1 Hash

```
/*    HASH
   数字 HASH，开散列，邻接表   */

const int MOD=4001,STA=1000010;  //MOD 为表长,STA 为表大小

struct Hash{
    int first[MOD],next[STA],size;
    LL f[STA],sta[STA];   //sta[] 存放状态,f[] 为对应状态的权值
    void init(){
        size=0;
        mem(first,-1);
    }
    int find_add(LL st,LL ans){   //查找，如果未查找到则添加
        int i,u=st%MOD;
        for(i=first[u];i!=-1;i=next[i]){
            if(sta[i]==st){
                f[i]+=ans;   //状态累加，注意啦
                return 1;   //已存在状态
            }
        }
        sta[size]=st;
        f[size]=ans;
        next[size]=first[u];
        first[u]=size++;
    }
}hs;
```

3.2 LCA

```
/*   LCA    */

/*    DFS+ST    O( n*logn )
  DFS 处理：
     T=<V,E>，其中 V={A,B,C,D,E,F,G},E={AB,AC,BD,BE,EF,EG}, 且 A 为树根。
     则图 T 的 DFS 结果为：A->B->D->B->E->F->E->G->E->B->A->C->A
  ST 处理 d[] 数组
  dis 为深度遍历计数
  d[] 为树遍历节点的深度
  E[] 为树遍历的节点的标号
  R[i] 表示 E 数组中第一个值为 i 的元素下标
  f[i][j] 为深度遍历 [i,j] 区间的最小深度的深度遍历编号
  如果有 n 个节点，那么 E[] 和 R[] 下标范围为 2*n-1
  rmq 下标范围为 [1,n]
  注意他们的最近公共祖先为他们本身的情况!!!         */


struct Edge{
    int u,v;
}e[N];
int first[N],next[N],f[N][20],d[N],E[N],R[N],vis[N];
int T,n,m,mt,dis,root;

void adde(int a,int b)
{
    e[mt].u=a;e[mt].v=b;
    next[mt]=first[a],first[a]=mt++;
}
```

```
int Minele(int i,int j)   //比较距离来获取下标
{
    return d[i]<d[j]?i:j;
}

void rmq_init(int n)
{
    int i,j;
    for(i=1;i<=n;i++)f[i][0]=i;
    for(j=1;(1<<j)<=n;j++){
        for(i=1;i+(1<<j)-1<=n;i++){
            f[i][j]=Minele(f[i][j-1],f[i+(1<<(j-1))][j-1]);
        }
    }
}

int rmq(int l,int r)
{
    int k=0;
    while((1<<(k+1))<=r-l+1)k++;
    return Minele(f[l][k],f[r-(1<<k)+1][k]);
}

void dfs(int u,int deep)
{
    d[dis]=deep;
    E[dis]=u;
    R[u]=dis++;
    int i;
    for(i=first[u];i!=-1;i=next[i]){
        dfs(e[i].v,deep+1);
        d[dis]=deep;
        E[dis++]=u;
    }
}

void LCA_Init()
{
    dis=1;  //初始化为 1
    dfs(root,0);   //传递根节点和深度
    rmq_init(2*n-1);  //传递 E[] 和 R[] 的长度
}

int LCA(int a,int b)   //返回 a 和 b 节点的最近祖先节点标号，注意他们的最近公共祖先为他们本身的情况
{
    int left=R[a],right=R[b];
    if(left>right)swap(left,right);
    return E[rmq(left,right)];
}
```

3.3 Merge sort

```
/*  merge sort
   求逆序对数量 */

int num[N],temp[N];
int n;

LL sort(int l,int r)
```

```
{
    if(l==r)return 0;
    int i,j,k,mid=(l+r)>>1;
    LL ans;
    sort(l,mid);
    sort(mid+1,r);
    for(i=k=l,j=mid+1;i<=mid && j<=r;){
        if(num[i]<num[j]){
            ans+=j-mid-1;
            temp[k++]=num[i++];
        }
        else temp[k++]=num[j++];
    }
    while(i<=mid){
        ans+=j-mid-1;
        temp[k++]=num[i++];
    }
    while(j<=r)
        temp[k++]=num[j++];
    for(i=l;i<=r;i++)
        num[i]=temp[i];
    return ans;
}
```

3.4 ST RMQ

```
/*    ST   预处理 O( n*logn )   查询 O( 1 )
   d[i][j] 为区间 [i,i+2^j-1] 的最值
   d[i][j]=Min( d[i][j-1], d[i+(1<<(j-1))][j-1] )   */


struct ST
{
    int n;
    int num[N],d[N][20];

    void rmq_init(int n)
    {
        int i,j;
        for(i=1;i<=n;i++)d[i][0]=num[i];
        for(j=1;(1<<j)<=n;j++){
            for(i=1;i+(1<<j)-1<=n;i++){
                d[i][j]=Min(d[i][j-1],d[i+(1<<(j-1))][j-1]);
            }
        }
    }
    int rmq(int l,int r)
    {
        int k=0;
        while((1<<(k+1))<=r-l+1)k++;
        return Min(d[l][k],d[r-(1<<k)+1][k]);
    }
}st;
```

3.5 划分树

```
/*  划分树   O(n*logn)
  s[] 是数列的前缀和
  val[] 是输入的数列，然后 sort 排序
  num[u+1][] 存的是 u 层划分后的数字（类似快排 Partation （d-1）次后的结果)
```

```
  cnt[u][i] 是统计在当前区间第 i 个数之前（包括第 i 个数）放到左区间的数的个数
  sum[u][i] 是求在当前区间第 i 个数之前（包括第 i 个数）放到左区间的数和
  注意：每次询问前初始化 ksum=0               */


int val[N],num[20][N],cnt[20][N];
LL sum[20][N],s[N];
int knum;
LL ksum;

void build(int u,int l,int r)
{
    if(l==r){
        sum[u][l]=num[u][l];
        return;
    }
    int i,mid,midnum,kl,kr,lsame;
    LL s=0;
    mid=(l+r)>>1;
    kl=l;kr=mid+1;lsame=mid-l+1;
    midnum=val[mid];
    for(i=l;i<=mid;i++)   //注意这里需要统计等于中位数放在左儿子区间的个数
        if(val[i]<midnum)lsame--;
    for(i=l;i<=r;i++){
        if(num[u][i]<midnum || (num[u][i]==midnum && lsame)){  //注意等于中位数情况
            if(num[u][i]==midnum)lsame--;
            num[u+1][kl++]=num[u][i];
            sum[u][i]=s+(LL)num[u][i];
            s=sum[u][i];
        }
        else {
            num[u+1][kr++]=num[u][i];
            sum[u][i]=s;
        }
        cnt[u][i]=kl-l;
    }
    build(u+1,l,mid);
    build(u+1,mid+1,r);
}

void query(int u,int l,int r,int a,int b,int k)
{
    if(a==b){   //注意这里可能 l<r 啦
        knum=num[u][a];
        ksum+=num[u][a];
        return;
    }
    int i,t,mid,cnta;
    mid=(l+r)>>1;
    cnta=(a>l?cnt[u][a-1]:0);   //注意 l==a
    t=cnt[u][b]-cnta;
    if(k<=t){
        query(u+1,l,mid,l+cnta,l+cnt[u][b]-1,k);
    }
    else{
        ksum+=sum[u][b]-(a>l?sum[u][a-1]:0);
        query(u+1,mid+1,r,mid+a-l-cnta+1,mid+b-l-cnt[u][b]+1,k-t);
    }
}
```

3.6 三维 LIS

```
/*   三维 LIS  O(n*lognlogn)
  (x,y,z)<=(x,y,z)
  分治算法
  ans.first 为长度, ans.second 为方法数    */

const int N=100010;

struct Node{
    int x,y,z,id;
    bool operator <(const Node& a)const{
        return x!=a.x?x<a.x:(y!=a.y?y<a.y:z<a.z);
    }
}a[N],b[N];
pii f[N],bit[N];
int z[N];
int T,n,m;

#define lowbit(x) (x&-x)

void update(pii& a,pii& b)
{
    if(b.first>a.first)a=b;
    else if(b.first==a.first){
        a.second+=b.second;
    }
}

void add(int x,pii& b)
{
    for(;x<=m;x+=lowbit(x)){
        update(bit[x],b);
    }
}

pii query(int x)
{
    pii ret=make_pair(0,0);
    for(;x>0;x-=lowbit(x)){
        update(ret,bit[x]);
    }
    return ret;
}

void clear(int x)
{
    for(;x<=m;x+=lowbit(x)){
        bit[x]=make_pair(0,0);
    }
}

void solve(int l,int r)
{
    if(l==r)return;
    int i,j,k,mid,cnt=0;
    mid=(l+r)>>1;
    solve(l,mid);
    for(i=l;i<=r;i++){
        b[cnt]=a[i];
        b[cnt++].x=0;
    }
```

```
    sort(b,b+cnt);
    for(i=0;i<cnt;i++){
        if(b[i].id<=mid){
            add(b[i].z,f[b[i].id]);
        }
        else {
            pii t=query(b[i].z);
            t.first++;
            update(f[b[i].id],t);
        }
    }
    for(i=0;i<cnt;i++){
        if(b[i].id<=mid)
            clear(b[i].z);
    }
    solve(mid+1,r);
}

int main()
{
    freopen("in.txt","r",stdin);
    int i,j;
    pii ans;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        for(i=0;i<n;i++){
            scanf("%d%d%d",&a[i].x,&a[i].y,&a[i].z);
            z[i]=a[i].z;
        }

        sort(a,a+n);
        sort(z,z+n);
        m=unique(z,z+n)-z;
        for(i=0;i<n;i++){
            f[i]=make_pair(1,1);
            a[i].id=i;
            a[i].z=lower_bound(z,z+m,a[i].z)-z+1;
        }
        solve(0,n-1);
        ans=make_pair(0,0);
        for(i=0;i<n;i++){
            update(ans,f[i]);
        }

        printf("%d %d\n",ans.first,ans.second);
    }
    return 0;
}
```

3.7 动态连续上升子序列

```
/*  动态连续上升子序列   O( n*Logn )
   线段树维护
   update() 修改数列中位置为 a 的数为 b
   query() 访问 [a,b] 区间的 LCIS
   ans 为询问后答案,anst 为 query() 时，临时记录的最长长度，每次询问前要初始话 ans=anst=0   */


struct Node{
```

```
    int maxl,max,maxr;
}ret[N<<2];
int num[N];
int n,m,a,b,ans,anst;

void pushup(int rt,int lnum,int rnum,int lenl,int lenr)
{
    ret[rt].max=max(ret[rt<<1].max,ret[rt<<1|1].max);
    ret[rt].maxl=ret[rt<<1].maxl;
    ret[rt].maxr=ret[rt<<1|1].maxr;
    if(lnum<rnum){
        ret[rt].max=max(ret[rt].max,ret[rt<<1].maxr+ret[rt<<1|1].maxl);
        if(ret[rt<<1].max==lenl)
            ret[rt].maxl+=ret[rt<<1|1].maxl;
        if(ret[rt<<1|1].max==lenr)
            ret[rt].maxr+=ret[rt<<1].maxr;
    }
}

void build(int l,int r,int rt)
{
    if(l==r){
        ret[rt].max=ret[rt].maxl=ret[rt].maxr=1;
        return;
    }
    int mid=(l+r)>>1;
    build(lson);
    build(rson);
    pushup(rt,num[mid],num[mid+1],mid-l+1,r-mid);
}

void update(int l,int r,int rt)
{
    if(l==r){
        num[l]=b;
        return;
    }
    int mid=(l+r)>>1;
    if(a<=mid)update(lson);
    else update(rson);
    pushup(rt,num[mid],num[mid+1],mid-l+1,r-mid);
}

void query(int l,int r,int rt)
{
    if(a<=l && r<=b){
        if(ret[rt].max>ans)
            ans=ret[rt].max;
        if(anst){
            if(num[l]>num[l-1]){
                if(ret[rt].max==r-l+1)
                    anst+=ret[rt].max;
                else {
                    anst+=ret[rt].maxl;
                    if(anst>ans)ans=anst;
                    anst=ret[rt].maxr;
                }
            }
            if(anst>ans)
                ans=anst;
            if(num[l]<=num[l-1])
                anst=ret[rt].maxr;
```

```
        }
        else
            anst=ret[rt].maxr;
        return;
    }
    int mid=(l+r)>>1;
    if(a<=mid)query(lson);
    if(b>mid)query(rson);
}
```

3.8 DLX

```
/*    DLX   O(?)   */

const int maxn=110;
const int maxnode=110;
const int maxr=110;

struct DLX{
    int n,sz;
    int S[maxn];

    int row[maxnode],col[maxnode];
    int L[maxnode],R[maxnode],U[maxnode],D[maxnode];

    int ansd,ans[maxr];

    void init(int n){
        this->n = n;
        for(int i =  0 ; i <= n ;i++){
            U[i] = i;
            D[i] = i;
            L[i] = i-1;
            R[i] = i+1;
        }
        R[n] = 0;
        L[0] = n;
        sz = n+1;
        mem(S,0);
    }

    void addRow(int r , vector<int> columns){
        int first = sz;
        for(int i = 0 ; i < columns.size() ; i++){
            int c= columns[i];
            L[sz] = sz - 1;
            R[sz] = sz +1;
            D[sz] = c;
            U[sz] = U[c];
            D[U[c]] = sz ;
            U[c] = sz;
            row[sz] = r;
            col[sz] = c;
            S[c]++;
            sz++;
        }
        R[sz-1] = first;
        L[first] = sz - 1;
    }
    #define FOR(i,A,s) for(int i = A[s] ; i != s; i = A[i])
    void remove(int c){
```

```
        L[R[c]] = L[c];
        R[L[c]] = R[c];
        FOR(i,D,c){
            FOR(j,R,i){
                  U[D[j]] = U[j];
                  D[U[j]] = U[j] ;
                  D[U[j]] = D[j] ;
                  --S[col[j]];
            }
        }
    }

    void restore(int c){
        FOR(i,U,c){
            FOR(j,L,i){
                ++S[col[j]];
                U[D[j]] = j;
                D[U[j]] = j;
            }
        }
        L[R[c]] = c;
        R[L[c]] = c;
    }

    bool dfs(int d){
        if(R[0] == 0){
            ansd = d;
            return true;
        }
        int c= R[0];
        FOR(i,R,0) if(S[i] < S[c]) c = i;
        remove(c);
        FOR(i,D,c){
            ans[d] = row[i];
            FOR(j,R,i) remove(col[j]);
            if(dfs(d+1)) return true;
            FOR(j,L,i) restore(col[j]);
        }
        restore(c);
        return false;
    }

    bool solve(vector<int> & v){
        v.clear();
        if(!dfs(0)) return false;
        for(int i = 0 ; i < ansd ; i++) v.push_back(ans[i]);
        return true;
    }
}solver;
```

3.9 Spaly tree

```
/*  Spaly tree   O(log n)
[NOI2005] 维修数列：http://www.cnblogs.com/zhsl/p/3227535.html （区间反转，求和，删除，改变，最
值） */

#define Key_value ch[ch[root][1]][0]
int pre[N],key[N],ch[N][2];  //分别表示父结点，键值，左右孩子 (0 为左孩子，1 为右孩子)，根结点，
结点数量
int sz[N],st[N];   //子树规模，内存池
```

```
int root,tot,top;   //根节点，根节点数量，内存池容量
//题目特定数据
int num[N];
int val[N];
int add[N];
LL sum[N];
int n,m;
//debug 部分 copy from hh
void Treaval(int x) {
    if(x) {
        Treaval(ch[x][0]);
       printf(" 结点%2d: 左儿子 %2d 右儿子 %2d 父结点 %2d size = %2d ,val = %2d , sum = %2d \n",
x,ch[x][0],ch[x][1],pre[x],sz[x],val[x],sum[x]);
        Treaval(ch[x][1]);
    }
}
void debug() {printf("%d\n",root);Treaval(root);}
//以上 Debug
//新建一个结点
void NewNode(int &x,int fa,int k)
{
    if(top)x=st[top++];
    else x=++tot;
    pre[x]=fa;
    sz[x]=1;
    val[x]=k;
    add[x]=0;
    sum[x]=k;
    ch[x][0]=ch[x][1]=0;  //左右孩子为空
}

void Push_Up(int x)
{
    sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+1;
    sum[x]=sum[ch[x][0]]+sum[ch[x][1]]+val[x]+add[x];
}

void Push_Down(int x)
{
    if(add[x]){
        val[x]+=add[x];
        add[ch[x][0]]+=add[x];
        add[ch[x][1]]+=add[x];
        sum[ch[x][0]]+=(LL)add[x]*sz[ch[x][0]];
        sum[ch[x][1]]+=(LL)add[x]*sz[ch[x][1]];
        add[x]=0;
    }
}

//旋转，kind 为 1 为右旋，kind 为 0 为左旋
void Rotate(int x,int kind)
{
    int y=pre[x],z=pre[y];
    Push_Down(y);
    Push_Down(x);  //先把 y 的标记向下传递，再把 x 的标记往下传递
    //类似 SBT，要把其中一个分支先给父节点
    ch[y][!kind]=ch[x][kind];
    pre[ch[x][kind]]=y;
    //如果父节点不是根结点，则要和父节点的父节点连接起来
    if(z)ch[z][ch[z][1]==y]=x;
    pre[x]=z;
    ch[x][kind]=y;
```

```
    pre[y]=x;
    Push_Up(y);  //维护 y 结点，不要维护 x 节点，x 节点会再次 Push_Down，最后维护一下 x 节点即可
}
//Splay 调整，将根为 r 的子树调整为 goal
void Splay(int x,int goal)
{
    int y,kind;
    while(pre[x]!=goal){
        //父节点即是目标位置，goal 为 0 表示，父节点就是根结点
        y=pre[x];
        Push_Down(pre[y]);Push_Down(y);Push_Down(x);   //涉及到反转操作，要先更新，然后在判断!!
        if(pre[y]==goal){
            Rotate(x,ch[y][0]==x);
        }
        else {
            kind=ch[pre[y]][0]==y;
            //两个方向不同，则先左旋再右旋
            if(ch[y][kind]==x){
                Rotate(x,!kind);
                Rotate(x,kind);
            }
            //两个方向相同，相同方向连续两次
            else {
                Rotate(y,kind);
                Rotate(x,kind);
            }
        }
    }
    //更新根结点
    Push_Up(x);
    if(goal==0)root=x;
}
/* 把第 k 个节点旋转到 goal 节点下面
注意：是加了两个虚拟节点后，虚拟节点不算，否则需要改为 while(sz[ch[x][0]]!=k-1)  */
void RotateTo(int k,int goal)
{
    int x=root;
    Push_Down(x);
    while(sz[ch[x][0]]!=k){
        if(sz[ch[x][0]]>k)
            x=ch[x][0];
        else {
            k-=sz[ch[x][0]]+1;
            x=ch[x][1];
        }
        Push_Down(x);
    }
    Splay(x,goal);
}

int Insert(int k)
{
    int x=root;
    while(ch[x][k>key[x]]){
        //不重复插入
        if(key[x]==k){
            Splay(x,0);
            return 0;
        }
        x=ch[x][k>key[x]];
    }
    NewNode(ch[x][k>key[x]],x,k);
```

```
    //将新插入的结点更新至根结点
    Splay(ch[x][k>key[x]],0);
    return 1;
}
//找前驱，即左子树的最右结点
int Get_Pre(int x)
{
    if(!ch[x][0])return -INF;
    x=ch[x][0];
    while(ch[x][1])x=ch[x][1];
    return key[x];
}
//找后继，即右子树的最左结点
int Get_Suf(int x)
{
    if(!ch[x][1])return INF;
    x=ch[x][1];
    while(ch[x][0])x=ch[x][0];
    return key[x];
}
//建树，中间结点先建立，然后分别对区间两端在左右子树建立
void BuildTree(int &x,int l,int r,int fa)
{
    if(l>r)return;
    int mid=(l+r)>>1;
    NewNode(x,fa,num[mid]);
    BuildTree(ch[x][0],l,mid-1,x);
    BuildTree(ch[x][1],mid+1,r,x);
    Push_Up(x);
}

void Init()
{
    ch[0][0]=ch[0][1]=pre[0]=sz[0]=0;
    add[0]=sum[0]=0;
    root=top=tot=0;
    NewNode(root,0,-1);
    NewNode(ch[root][1],root,-1);     //头尾各加入一个空位
    sz[root]=2;

    for(int i=0;i<n;i++)
        scanf("%d",&num[i]);
    BuildTree(Key_value,0,n-1,ch[root][1]);    //让所有数据夹在两个-1 之间
    Push_Up(ch[root][1]);
    Push_Up(root);
}

void Update(int a,int b,int c)
{
    RotateTo(a-1,0);
    RotateTo(b+1,root);
    add[Key_value]+=c;
    sum[Key_value]+=sz[Key_value]*c;
}

LL Query(int a,int b)
{
    RotateTo(a-1,0);
    RotateTo(b+1,root);
    return sum[Key_value];
}
//第 k 个数的节点编号
```

```
int Get_Kth(int r,int k)
{
    Push_Down(r);
    int t=size[ch[r][0]]+1;
    if(t==k)return r;
    if(t>k)return Get_Kth(ch[r][0],k);
    else return Get_Kth(ch[r][1],k-t);
}
//内存池，删除的节点入栈
void erase(int r){
    if(!r) return;
    st[++top]=r;
    erase(ch[r][0]);
    erase(ch[r][1]);
}
//删除区间 [a,b] 的数，加了虚拟节点之后
void Delete(int a,int b)
{
    RotateTo(a-1,0);
    RotateTo(b+1,root);
    erase(Key_value);
    pre[Key_value]=0;
    Key_value=0;
    Push_Up(ch[root][1]);
    Push_Up(root);
}
```

4 字符串

4.1 kmp

```
/*  KMP

        0 1 2 3 4 5 6 7 8
 s:     a b a b a b c d
 next: -1 0 0 1 2 3 4 0 0    */


int next[N];
int T,n,m,len;

void getnext(char *s,int len)
{
    int j=0,k=-1;
    next[0]=-1;
    while(j<len){
        if(k==-1 || s[k]==s[j])
            next[++j]=++k;
        else k=next[k];
    }
}
```

4.2 Trie 树

```
/*  Trie   */

/*  静态化，注意内存开辟大小    */

const int wide=26;  //每个节点的最大子节点数

struct Trie {
    int ch[1<<13][wide];    //内存开辟尽量最大
    int val[1<<13];
    int sz;

    void init(){sz=1;mem(ch[0],0);}
    inline int idx(char c){return c-'a';}   //字母映射
    void insert(char *s,int v){     //建立 Trie 树
        int i,len=strlen(s),c,u=0;
        for(i=0;i<len;i++){
            c=idx(s[i]);
            if(!ch[u][c]){
                mem(ch[sz],0);
                val[sz]=0;
                ch[u][c]=sz++;
            }
            u=ch[u][c];
        }
        val[u]=v;
    }
    int find(char *s){  //查找
        int i,len=strlen(s),c,u=0;
        for(i=0;i<len;i++){
            c=idx(s[i]);
            if(!ch[u][c])return 0;
            u=ch[u][c];
        }
        for(i=0;i<wide;i++)
```

```
            if(ch[u][c])return 1;   //s 串是模板串中某个串的 prefix
        return 2;  //存在
    }
}trie;

/*  动态建立，注意释放内存  */

const int wide=26;  //每个节点的最大子节点数

struct Trie {
    struct Node{
        Node(){mem(ch,0);val=0;}
        Node *ch[wide];
        int val;
    };
    Node *head;

    void init(){head=new Node;}
    inline int idx(char c){return c-'a';}
    void insert(char *s,int v){   //插入
        int i,len=strlen(s),c;
        Node *p=head,*q;
        for(i=0;i<len;i++){
            c=idx(s[i]);   //求对应编号
            if(!p->ch[c]){
                q=new Node;
                p->ch[c]=q;
            }
            p=p->ch[c];
        }
        p->val=v;
    }
    int find(char *s){    //查找
        int i,len=strlen(s),c;
        Node *p=head;
        for(i=0;i<len;i++){
            c=idx(s[i]);
            if(!p->ch[c])return 0;
            p=p->ch[c];
        }
        for(i=0;i<wide;i++)
            if(p->ch[i])return 1;   //s 串是模板串中某个串的 prefix
        return 2; //存在
    }
    void free(Node *p){   //释放内存
        int i;
        for(i=0;i<wide;i++)
            if(p->ch[i])free(p->ch[i]);
        delete p;
    }
}trie;
```

4.3 Manacher

```
/*  Manacher 求最长回文串 O(n)
  getstr 预处理: abc -> $#a#b#c#
  len=strlen(s) , n=2*len+2
  p[i] 为 str[] 第 i 个字符为中心的回文串长度 +1 */

char str[N<<1],s[N];
int p[N<<1];
```

```
int n,len;

void Manacher(char *str,int *p)
{
    int i,j,id,mx;
    id=1,mx=1;
    p[0]=p[1]=1;
    for(i=2;i<n;i++){
        p[i]=1;
        if(mx>i){
            p[i]=Min(p[(id<<1)-i],mx-i);
        }
        while(str[i+p[i]]==str[i-p[i]])p[i]++;
        if(i+p[i]>mx){
            id=i;
            mx=i+p[i];
        }
    }
}

void getstr(char *s)
{
    int i;
    str[0]='$';str[1]='#';
    for(i=0;i<len;i++){
        str[(i<<1)+2]=s[i];
        str[(i<<1)+3]='#';
    }
    str[n]=0;
}
```

4.4 suffix array

```
/*  suffix array
   倍增算法   O(n*lgn)
   build_sa(num,n+1,m)   注意 n+1，每个字符的值为 0~m-1
   getHeight(num,n)
   rmq_init(height)  初始化 rmq，传递 height 数组
   rmq(a+1,b)   求排名分别为为 a 和 b 的最长公共前缀
   lcp(a,b)  求后缀 a 和后缀 b 的最长公共前缀

 n        =  8 ;
 num[]    = { 1, 1, 2, 1, 1, 1, 1, 2, $ }.   注意 num 数组最后一位值为 0，其它位须大于 0!
 rank[]   = { 4, 6, 8, 1, 2, 3, 5, 7, 0 }.   (rank[0~n-1] 为有效值)
 sa[]     = { 8, 3, 4, 5, 0, 6, 1, 7, 2 }.   (sa[1~n] 为有效值)
 height[] = { 0, 0, 3, 2, 3, 1, 2, 0, 1 }.   (height[2~n] 为有效值)    */

char s[N];
int d[N][20];
int num[N];
int sa[N],t1[N],t2[N],c[N],rank[N],height[N];
int n,m;

void build_sa(int s[],int n,int m)
{
    int i,k,p,*x=t1,*y=t2;
    //第一轮基数排序
    for(i=0;i<m;i++)c[i]=0;
    for(i=0;i<n;i++)c[x[i]=s[i]]++;
    for(i=1;i<m;i++)c[i]+=c[i-1];
    for(i=n-1;i>=0;i--)sa[--c[x[i]]]=i;
```

```
    for(k=1;k<=n;k<<=1){
        p=0;
        //直接利用 sa 数组排序第二关键字
        for(i=n-k;i<n;i++)y[p++]=i;
        for(i=0;i<n;i++)if(sa[i]>=k)y[p++]=sa[i]-k;
        //基数排序第一关键字
        for(i=0;i<m;i++)c[i]=0;
        for(i=0;i<n;i++)c[x[y[i]]]++;
        for(i=1;i<m;i++)c[i]+=c[i-1];
        for(i=n-1;i>=0;i--)sa[--c[x[y[i]]]]=y[i];
        //根据 sa 和 x 数组计算新的 x 数组
        swap(x,y);
        p=1;x[sa[0]]=0;
        for(i=1;i<n;i++)
            x[sa[i]]=y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+k]==y[sa[i]+k]?p-1:p++;
        if(p>=n)break;   //已经排好序，直接退出
        m=p;     //下次基数排序的最大值
    }
}

void getHeight(int s[],int n)
{
    int i,j,k=0;
    for(i=0;i<=n;i++)rank[sa[i]]=i;
    for(i=0;i<n;i++){
        if(k)k--;
        j=sa[rank[i]-1];
        while(s[i+k]==s[j+k])k++;
        height[rank[i]]=k;
    }
}

void rmq_init(int a[])
{
    int i,j;
    for(i=1;i<=n;i++)d[i][0]=a[i];
    for(j=1;(1<<j)<=n;j++){
        for(i=1;i+(1<<j)-1<=n;i++){
            d[i][j]=Min(d[i][j-1],d[i+(1<<(j-1))][j-1]);
        }
    }
}

int rmq(int l,int r)
{
    int k=0;
    while((1<<(k+1))<=r-l+1)k++;
    return Min(d[l][k],d[r-(1<<k)+1][k]);
}

int lcp(int a,int b)
{
    if(a==b)return n-a;      //a 和 b 为同一后缀，直接输出，字串串长度为 n
    int ra=rank[a],rb=rank[b];
    if(ra>rb)swap(ra,rb);
    return rmq(ra+1,rb);
}
```

4.5 String Hash

```
/*  String Hash
```

```
 -BKDRHash 建议使用 longlong，发生冲突概率很小，如果还是有冲突，那么用两个 hash

Hash 函数    数据 1 数据 2 数据 3 数据 4 数据 1 得分数据 2 得分数据 3 得分数据 4 得分平均分
BKDRHash   2 0 4774 481 96.55 100 90.95 82.05 92.64
APHash    2 3 4754 493 96.55 88.46 100 51.28 86.28
DJBHash    2 2 4975 474 96.55 92.31 0 100 83.43
JSHash     1 4 4761 506 100 84.62 96.83 17.95 81.94
RSHash     1 0 4861 505 100 100 51.58 20.51 75.96
SDBMHash   3 2 4849 504 93.1 92.31 57.01 23.08 72.41
PJWHash   30 26 4878 513 0 0 43.89 0 21.95
ELFHash   30 26 4878 513 0 0 43.89 0 21.95 */

// BKDR Hash Function
unsigned int BKDRHash(char *str)
{
    unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
    unsigned int hash = 0;

    while (*str)
    {
        hash = hash * seed + (*str++);
    }

    return (hash & 0x7FFFFFFF);
}

//SDBMHash
unsigned int SDBMHash(char *str)
{
    unsigned int hash = 0;

    while (*str)
    {
        // equivalent to: hash = 65599*hash + (*str++);
        hash = (*str++) + (hash << 6) + (hash << 16) - hash;
    }

    return (hash & 0x7FFFFFFF);
}

// RS Hash Function
unsigned int RSHash(char *str)
{
    unsigned int b = 378551;
    unsigned int a = 63689;
    unsigned int hash = 0;

    while (*str)
    {
        hash = hash * a + (*str++);
        a *= b;
    }

    return (hash & 0x7FFFFFFF);
}

// JS Hash Function
unsigned int JSHash(char *str)
{
    unsigned int hash = 1315423911;

    while (*str)
```

```
    {
        hash ^= ((hash << 5) + (*str++) + (hash >> 2));
    }

    return (hash & 0x7FFFFFFF);
}

// P. J. Weinberger Hash Function
unsigned int PJWHash(char *str)
{
    unsigned int BitsInUnignedInt = (unsigned int)(sizeof(unsigned int) * 8);
    unsigned int ThreeQuarters    = (unsigned int)((BitsInUnignedInt  * 3) / 4);
    unsigned int OneEighth        = (unsigned int)(BitsInUnignedInt / 8);
    unsigned int HighBits         = (unsigned int)(0xFFFFFFFF) << (BitsInUnignedInt - OneEighth);
    unsigned int hash             = 0;
    unsigned int test             = 0;

    while (*str)
    {
        hash = (hash << OneEighth) + (*str++);
        if ((test = hash & HighBits) != 0)
        {
            hash = ((hash ^ (test >> ThreeQuarters)) & (~HighBits));
        }
    }

    return (hash & 0x7FFFFFFF);
}

// ELF Hash Function
unsigned int ELFHash(char *str)
{
    unsigned int hash = 0;
    unsigned int x    = 0;

    while (*str)
    {
        hash = (hash << 4) + (*str++);
        if ((x = hash & 0xF0000000L) != 0)
        {
            hash ^= (x >> 24);
            hash &= ~x;
        }
    }

    return (hash & 0x7FFFFFFF);
}

// DJB Hash Function
unsigned int DJBHash(char *str)
{
    unsigned int hash = 5381;

    while (*str)
    {
        hash += (hash << 5) + (*str++);
    }

    return (hash & 0x7FFFFFFF);
}

// AP Hash Function
```

```
unsigned int APHash(char *str)
{
    unsigned int hash = 0;
    int i;

    for (i=0; *str; i++)
    {
        if ((i & 1) == 0)
        {
            hash ^= ((hash << 7) ^ (*str++) ^ (hash >> 3));
        }
        else
        {
            hash ^= (~((hash << 11) ^ (*str++) ^ (hash >> 5)));
        }
    }

    return (hash & 0x7FFFFFFF);
}
```

4.6 AhoCorasick

```
/*  Aho Corasick
   静态建立 Trie   */

const int wide=26;  //每个节点的最大子节点数

struct Aho_Corasick{
    int ch[N][wide];
    int val[N],f[N],last[N];
    int sz;

    void init(){sz=1;mem(ch[0],0);}
    inline int idx(char c){return c-'A';}   //字母映射
    void insert(char *s,int v){     //建立 Trie 树
        int i,len=strlen(s),c,u=0;
        for(i=0;i<len;i++){
            c=idx(s[i]);
            if(!ch[u][c]){
                mem(ch[sz],0);
                val[sz]=0;
                ch[u][c]=sz++;
            }
            u=ch[u][c];
        }
        val[u]=v;
    }

    void print(int u)   //输出找到的字符串，可修改为相应操作
    {
        if(u){
            printf("%d %d\n",u,val[u]);
            print(last[u]);
        }
    }

    void getFail()   //建立失配数组
    {
        int u,c,r;
        queue<int> q;
        f[0]=0;
```

```
        for(c=0;c<wide;c++){
            u=ch[0][c];
            if(u){f[u]=0;last[u]=0;q.push(u);}
        }
        while(!q.empty()){
            r=q.front();q.pop();
            for(c=0;c<wide;c++){
                u=ch[r][c];
                //改变了 ch[r][c]，使得空指针为非空，不改变 ch[r][c]
                //if(!u){continue;}
                if(!u){ch[r][c]=ch[f[r]][c];continue;}   //不修改 ch[][] 与上面替换
                q.push(u);
                //如果 ch[][] 空指针没有改变，应顺着失配边走
                /*
                int v=f[r];
                while(v && !ch[v][c])v=f[v];
                f[u]=ch[v][c];
                */
                f[u]=ch[f[r]][c];   //不修改 ch[][] 与上面替换
                last[u]=val[f[u]]?f[u]:last[f[u]];
            }
        }
    }

    void find(char *T)
    {
        int i,c,u=0,len=strlen(T);
        for(i=0;i<len;i++){
            c=idx(T[i]);
            //如果 getFail() 中 ch[][] 空指针没有改变，应顺着失配边走
            //while(u && !ch[u][c])u=f[u];
            u=ch[u][c];  //不修改 ch[][]，不注销上面一句
            if(val[u]){print(u);}
            else if(last[u]){print(last[u]);}
        }
    }
}ac;
```

5 图论

5.1 2-SAT

```
/*    Twosat    */

/*   dfs  O(VE)   */
struct Edge{
    int u,v;
}e[N*N*2];

int first[N],next[N*N*2],vis[N],S[N];
int n,mt,cnt;

void adde(int a,int b)
{
    e[mt].u=a,e[mt].v=b;
    next[mt]=first[a];first[a]=mt++;
}

int dfs(int u)
{
    if(vis[u^1])return 0;
    if(vis[u])return 1;
    int i;
    vis[u]=1;
    S[cnt++]=u;
    for(i=first[u];i!=-1;i=next[i]){
        if(!dfs(e[i].v))return 0;
    }
    return 1;
}

int Twosat()
{
    int i,j;
    for(i=0;i<n;i+=2){
        if(vis[i] || vis[i^1])continue;
        cnt=0;
        if(!dfs(i)){
            while(cnt)vis[S[--cnt]]=0;
            if(!dfs(i^1))return 0;
        }
    }
    return 1;
}

void init()
{
    int i,j;
    mt=0;mem(vis,0);
    mem(first,-1);
  /*
    建图
  */
}
```

5.2 Kruskal

```
/*   Kruskal   O( n*logn )
  求 MST 和次小生成树
```

```
  mst[][] 生成树中的边
  d[][] 生成树中两点的路径上的最长边  */

struct Edge{
    int u,v,w;
}e[N*N];
int p[N],vis[N],mst[N][N],d[N][N],w[N][N];
int n,m,mt;

int cmp(const Edge& a,const Edge& b)
{
    return a.w<b.w;
}

int find(int x){return p[x]==x?x:p[x]=find(p[x]);}

int Kruskal()
{
    int i,j,x,y,sum=0;
    for(i=1;i<=n;i++)p[i]=i;
    sort(e,e+m,cmp);
    mem(mst,0);
    for(i=0;i<m;i++){
        x=find(e[i].u);
        y=find(e[i].v);
        if(x!=y){
            sum+=e[i].w;
            p[y]=x;
        //    mst[e[i].u][e[i].v]=mst[e[i].v][e[i].u]=1;  //MST 中的边
        }
    }
    return sum;
}

int dfs(int& s,int u,int max)   //求任意两点路径的最长边
{
    int v;
    for(v=1;v<=n;v++){
        if(mst[u][v] && !vis[v]){
            vis[v]=1;
            d[s][v]=Max(max,w[u][v]);
            dfs(s,v,d[s][v]);
        }
    }
    return 0;
}

int Second_MST()    //求次小生成树  O(n^2)
{
    int i,j,ret,t;
    sort(e,e+m,cmp);
    ret=Kruskal();
    for(i=0;i<=n;i++){
        mem(vis,0);vis[i]=1;
        dfs(i,i,0);
    }
    t=ret;
    for(i=0;i<m;i++){
        if(d[e[i].u][e[i].v]==e[i].w && !mst[e[i].u][e[i].v])
            ret=Min(ret,t+e[i].w-d[e[i].u][e[i].v]);
    }
    return ret;
```

```
}
```

5.3 曼哈顿距离 MST

```
/*  曼哈顿距离 MST  O(n*logn)
  性质：对于某个点，以他为中心的区域分为 8 个象限，对于每一个象限，只会取距离最近的一个点连边。
建图方法：
我们把所有的点按照 x 从小到大排序：x1 ⩽ x2 ⩽…⩽ xn。
建立一个抽象数据结构 T。T 中的每个元素对应平面上的一个点 (x,y)，该元素的第一关键字等于 y-x，第二关键字等于 y+x。
从 Pn 到 P1 逐个处理每个点。处理 Pk 的时候，令 Pk+1，Pk+2，…，Pn 都已经存入到 T
中。某个点 Q(x,y) 如果落在 Pk 的 R1 区间内，必须满足：
1.      x ⩾ xk
2.      y-x>yk-xk
要满足第一个条件，Q 必须属于集合 {Pk+1，Pk+2，…，Pn}，即 Q 必然在 T 中。
要满足第二个条件，Q 在 T 中的第一关键字必须大于 yk-xk(定值)。
因为我们要使得 |PkQ| 最小，所以我们实际上就是：从 T 的第一关键字大于某常数的所有元素中，寻找第二关键字最小的元素。
很明显，T 可以用平衡二叉树来实现。按照第一关键字有序来建立平衡树，对于平衡树每个节点都记录以其为根的子树中第二关
键字最小的是哪个元素。查询、插入的时间复杂度都是 O(logn)。
平衡二叉树也可以用线段树代替。
这里的代码用的 BIT 维护！
坐标变化：
   R1->R2：关于 y=x 对称，swap(x,y)
   R2->R3：考虑到代码的方便性，我们考虑 R2->R7，x=-x。
   R7->R4：因为上面求的是 R2->R7，因此这里还是关于 y=x 对称。   */


const int INF=0x3f3f3f3f;

struct Point{
    int x,y,id;
    bool operator<(const Point p)const{
        return x!=p.x?x<p.x:y<p.y;
    }
}p[N];
struct BIT{
    int min_val,pos;
    void init(){
        min_val=INF;
        pos=-1;
    }
}bit[N];
struct Edge{
    int u,v,d;
    bool operator<(const Edge e)const{
        return d<e.d;
    }
}e[N<<2];
int T[N],hs[N];
int n,mt,pre[N];

void adde(int u,int v,int d)
{
    e[mt].u=u,e[mt].v=v;
    e[mt++].d=d;
}

int find(int x)
{
```

```
    return pre[x]=(x==pre[x]?x:find(pre[x]));
}
int dist(int i,int j)
{
    return abs(p[i].x-p[j].x)+abs(p[i].y-p[j].y);
}

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

void update(int x,int val,int pos)
{
    for(int i=x;i>=1;i-=lowbit(i))
        if(val<bit[i].min_val)
            bit[i].min_val=val,bit[i].pos=pos;
}

int query(int x,int m)
{
    int min_val=INF,pos=-1;
    for(int i=x;i<=m;i+=lowbit(i))
        if(bit[i].min_val<min_val)
            min_val=bit[i].min_val,pos=bit[i].pos;
    return pos;
}

int Manhattan_minimum_spanning_tree(int n,Point *p,int K)
{
    int i,w,dir,fa,fb,pos,m;
    //Build graph
    mt=0;
    for(dir=0;dir<4;dir++){
        //Coordinate transform - reflect by y=x and reflect by x=0
        if(dir==1||dir==3){
            for(i=0;i<n;i++)
                swap(p[i].x,p[i].y);
        }
        else if(dir==2){
            for(i=0;i<n;i++){
                p[i].x=-p[i].x;
            }
        }
        //Sort points according to x-coordinate
        sort(p,p+n);
        //Discretize
        for(i=0;i<n;i++){
            T[i]=hs[i]=p[i].y-p[i].x;
        }
        sort(hs,hs+n);
        m=unique(hs,hs+n)-hs;
        //Initialize BIT
        for(i=1;i<=m;i++)
            bit[i].init();
        //Find points and add edges
        for(i=n-1;i>=0;i--){
            pos=lower_bound(hs,hs+m,T[i])-hs+1;   //BIT 中从 1 开始'
            w=query(pos,m);
            if(w!=-1)
                adde(p[i].id,p[w].id,dist(i,w));
            update(pos,p[i].x+p[i].y,i);
```

```
        }
    }
    //Kruskal - 找到第 K 小的边
    sort(e,e+mt);
    for(i=0;i<n;i++)pre[i]=i;
    for(i=0;i<mt;i++){
        fa=find(e[i].u),fb=find(e[i].v);
        if(fa!=fb){
            K--;pre[fa]=fb;
            if(K==0)return e[i].d;
        }
    }
}
```

5.4 最小树形图 -朱刘算法

```
/*   最小树形图-朱刘算法  O( VE )
   mt 为边数
   题目:POJ3146
   http://www.cnblogs.com/zhsl/archive/2013/02/01/2888834.html  */


struct Edge{
    int u,v,w;
}e[N*N];
int pre[N],id[N],vis[N],minw[N];
int n,mt;

void adde(int a,int b,int c)
{
    e[mt].u=a,e[mt].v=b;e[mt].w=c;
    mt++;
}

int zhu_liu(int root)
{
    int i,cou,u,v,k;
    int ans=0;
    while(1)
    {
        //init
        mem(pre,-1);
        for(i=1;i<=n;i++)minw[i]=INF;
        for(i=0;i<mt;i++){
            u=e[i].u;
            v=e[i].v;
            if(e[i].w<minw[v] && u!=v){
                pre[v]=u;
                minw[v]=e[i].w;
            }
        }
        pre[root]=-1;minw[root]=0;
        for(cou=0,i=1;i<=n;i++)
            if(pre[i]==-1 && i!=root)cou++;
            else ans+=minw[i];
        if(cou)return -1;    //不存在最小树形图
        //cheack the circle
        mem(vis,0);
        mem(id,0);
        for(i=1,k=0;i<=n;i++){
            if(id[i])continue;
```

```
            u=i;
            while(u!=-1 && !id[u] && vis[u]!=i){
                vis[u]=i;
                u=pre[u];
            }
            if(u!=-1 && !id[u] && vis[u]==i){
                k++;
                while(id[u]!=k){
                    id[u]=k;
                    u=pre[u];
                }
            }
        }
        if(!k)break;
        for(i=1;i<=n;i++)if(!id[i])id[i]=++k;
        //eliminate circle
        for(i=0;i<mt;i++){
            e[i].w-=minw[e[i].v];
            e[i].u=id[e[i].u];
            e[i].v=id[e[i].v];
        }
        n=k;
        root=id[root];
    }
    return ans;
}
```

5.5 Dijkstra

```
/* Dijkstra O(E * log V)
  p[] 为路径
  不能处理负权图      */

struct Edge{
    int u,v,w;
}e[2*N];
int first[N],next[2*N],p[N],d[N];
int S,T,n,m,mt;

void adde(int a,int b,int c)
{
    e[mt].u=a,e[mt].v=b;e[mt].w=c;
    next[mt]=first[a],first[a]=mt++;
    e[mt].u=b,e[mt].v=a;e[mt].w=c;
    next[mt]=first[b],first[b]=mt++;
}

int dijkstra(int s)   //s is start
{
    int i,u;
    pii t;
    priority_queue<pii,vector<pii>,greater<pii> > q;
    mem(d,INF);d[s]=0;
    q.push(make_pair(d[s],s));
    while(!q.empty()){
        t=q.top();q.pop();
        u=t.second;
        if(t.first!=d[u])continue;
        for(i=first[u];i!=-1;i=next[i]){
            if(d[u]+e[i].w<d[e[i].v]){
                d[e[i].v]=d[u]+e[i].w;
```

```
                p[e[i].v]=i;
                q.push(make_pair(d[e[i].v],e[i].v));
            }
        }
    }
    return d[T];
}

/*   求最短路和次短路以及他们的路径数
   d[i][0] 为点 i 的最短路,d[i][1] 为次短路
   cnt[i][0] 为点 i 的最短路径数,cnt[i][0] 为次短路径数
   转移方程：
   1，如果 d[u][0]+w<d[u][0]，分别更新 v 点的最短路和次短路
   2，如果 d[u][0]+w==d[u][0]，那么 v 点的最短路数加上 u 点的最短路数
   3，如果 d[u][k]+w<d[u][1]，更新 v 点的次短路数（根据 k 来定）
   4，如果 d[u][k]+w==d[u][1]，那么 v 点的次短路数加上 u 点的最短路数或者次短路数（根据 k 来定）
*/


struct Node{
    int d,u,flag;
    bool operator < (const Node &oth) const{
        return d>oth.d;
    }
};
struct Edge{
    int u,v,w;
}e[N*N];
int first[N],next[N*N],d[N][2],cnt[N][2];
int s,T,n,m,mt;

void adde(int a,int b,int c)
{
    e[mt].u=a,e[mt].v=b;e[mt].w=c;
    next[mt]=first[a],first[a]=mt++;
}

int dijkstra(int s)
{
    int i,u,v,w,k,dis;
    Node t;
    priority_queue<Node> q;
    mem(d,0x3f);d[s][0]=d[s][1]=0;
    cnt[s][0]=cnt[s][1]=1;
    t.d=0,t.u=s,t.flag=0;
    q.push(t);
    while(!q.empty()){
        t=q.top();q.pop();
        u=t.u;dis=t.d;k=t.flag;
        if(dis!=d[u][k])continue;
        for(i=first[u];i!=-1;i=next[i]){
            v=e[i].v;w=e[i].w;
            if(dis+w<d[v][0]){
                cnt[v][1]=cnt[v][0];
                cnt[v][0]=cnt[u][0];
                d[v][1]=d[v][0];
                d[v][0]=dis+w;
                q.push(Node{d[v][0],v,0});
                q.push(Node{d[v][1],v,1});
            }
            else if(dis+w==d[v][0])cnt[v][0]+=cnt[u][0];
            else if(dis+w<d[v][1]){
```

```
                d[v][1]=dis+w;
                cnt[v][1]=cnt[u][k];
                q.push(Node{d[v][1],v,1});
            }
            else if(dis+w==d[v][1])cnt[v][1]+=cnt[u][k];
        }
    }
    return d[T][1];  //  返回次短路长度
}
```

5.6 SPFA

```
/*   SPFA  O(k*E)
   对于一般稀疏图,k=2~3. 稠密图最坏情况可达到 O(V^2)
   如果存在负权环返回 1, 否则返回 0    */

struct Edge{
    int u,v,w;
}e[N*N];

int first[N],next[N*N],inq[N],d[N],cnt[N];
int n,m,mt;

int spfa(int s)
{
    int i,u,v,t;
    queue<int> q;
    mem(d,INF);
    mem(cnt,0);
    q.push(s);
    d[s]=0;cnt[s]=1;
    while(!q.empty()){
     u=q.front();q.pop();
     inq[u]=0;
     for(i=first[u];i!=-1;i=next[i]){
         v=e[i].v;t=d[u]+e[i].w;
          if(t<d[v]){
              d[v]=t;
              if(!inq[v]){
               if(++cnt[v]>=n)return 1;   //存在负权环
               inq[v]=1;
               q.push(v);
              }
                }
    }
    }
    return 0;
}
```

5.7 K 短路

```
/*  K 短路  A*
  首先求出所有点到汇点的最短路距离, 然后 A* 搜索
  G1 和 edge1 是正向图,G2 和 edge2 是逆向图  */

struct Node{
    int d,u;
};
struct Edge{
    int from,to,dis;
};
```

```
int n,m,S,T,K;
int d[N],vis[N];
vector<Edge> edge1,edge2;
vector<int> G1[N],G2[N];

struct cmp1{
    bool operator()(const Node &a,const Node &b){
        return a.d>b.d;
    }
};
struct cmp2{
    bool operator()(const Node &a,const Node &b){
        return a.d+d[a.u]>b.d+d[a.u];
    }
};

void init(int n){
    edge1.clear();
    edge2.clear();
    for(int i=1;i<=n;i++)
        G1[i].clear();
    for(int i=1;i<=n;i++)
        G2[i].clear();
}

void dijkstra(int s,vector<Edge>& edge){
    int i,u;
    priority_queue<Node,vector<Node>,cmp1> q;
    for(i=1;i<=n;i++)d[i]=INF;d[s]=0;
    mem(vis,0);
    q.push((Node){0,s});
    while(!q.empty()){
        u=q.top().u;q.pop();
        if(vis[u])continue;
        vis[u]=1;
        for(i=0;i<G2[u].size();i++){
            Edge& e=edge[G2[u][i]];
            if(d[u]+e.dis<d[e.to]){
                d[e.to]=d[u]+e.dis;
                q.push((Node){d[e.to],e.to});
            }
        }
    }
}

int astar(int s,int t,int k,vector<Edge>& edge){
    int i;
    int cou[MAX];
    mem(cou,0);
    Node nod;
    priority_queue<Node,vector<Node>,cmp2> q;
    q.push((Node){d[s],s});
    if(s==t)k++;
    while(1){
        nod=q.top();q.pop();
        if(nod.u==t)cou[nod.u]++;
        if(cou[nod.u]==k)return nod.d;
        for(i=0;i<G1[nod.u].size();i++){  //从当前点到所有邻接点
            Edge& e=edge[G1[nod.u][i]];
            q.push((Node){nod.d-d[nod.u]+d[e.to]+e.dis,e.to});
        }
```

```
    }
    return 0;
}
```

5.8 双连通分量

```
/*   BCC  O(E)
  tarjan 算法
 -Node-Biconnected Component
 -Edge-Biconnected Component(可以处理重边)
 -Edge-Biconnected Component(不能处理重边)
 */


/* Node-Biconnected Component
  iscut[] 为割点集
  bcc[] 为双连通点集
  割顶的 bccno[] 无意义       */
struct Edge{
    int u,v;
}e[N*N];
bool iscut[N];
int first[N],next[N*N],low[N],pre[N],bccno[N];
int n,m,mt,dfs_clock,bcnt;
vector<int> bcc[N];
stack<Edge> s;

void adde(int a,int b)
{
    e[mt].u=a;e[mt].v=b;
    next[mt]=first[a];first[a]=mt++;
    e[mt].u=b;e[mt].v=a;
    next[mt]=first[b];first[b]=mt++;
}

void dfs(int u,int fa)
{
    int i,j,v,child=0;
    Edge t;
    pre[u]=low[u]=++dfs_clock;
    for(i=first[u];i!=-1;i=next[i]){
        child++;
        v=e[i].v;
        t.u=u;t.v=v;
        if(!pre[v]){   //没有访问过
            s.push(t);
            dfs(v,u);
            low[u]=Min(low[u],low[v]);
            if(low[v]>=pre[u]){   //点 u 为割点
                iscut[u]=true;
                Edge x;x.u=-1;
                bcnt++;bcc[bcnt].clear();
                while(x.u!=u || x.v!=v){
                    x=s.top();s.pop();
                    if(bccno[x.u]!=bcnt){bcc[bcnt].push_back(x.u);bccno[x.u]=bcnt;}
                    if(bccno[x.v]!=bcnt){bcc[bcnt].push_back(x.v);bccno[x.v]=bcnt;}
                }
            }
        }
        else if(v!=fa && pre[v]<pre[u]){   //存在反向边，更新 low[u]
            s.push(t);
```

```
            low[u]=Min(low[u],pre[v]);
        }
    }
    if(fa==-1 && child==1)iscut[u]=false;  //根节点特判
}

void find_bcc()
{
    int i,j;
    bcnt=dfs_clock=0;mem(pre,0);
    mem(bccno,0);mem(iscut,0);
    for(i=1;i<=n;i++){
        if(!pre[i])dfs(i,-1);
    }
}

/* Edge-Biconnected Component(可以处理重边)
  iscut[] 为割边集
  bccno[] 为双连通点集，保存为编号        */
struct Edge{
    int u,v;
}e[N*N];
bool iscut[N*N];
int first[N],next[N*N],pre[N],low[N],bccno[N];
int n,m,mt,bcnt,dfs_clock;
stack<int> s;

void dfs(int u,int fa)
{
    int i,v;
    pre[u]=low[u]=++dfs_clock;
    s.push(u);
    int cnt=0;
    for(i=first[u];i!=-1;i=next[i]){
        v=e[i].v;
        if(!pre[v]){
            dfs(v,u);
            low[u]=Min(low[u],low[v]);
            if(low[v]>pre[u])iscut[i]=true;   //存在割边
        }
        else if(fa==v){  //反向边更新
            if(cnt)low[u]=Min(low[u],pre[v]);
            cnt++;
        }
        else low[u]=Min(low[u],pre[v]);
    }
    if(low[u]==pre[u]){  //充分必要条件
        int x=-1;
        bcnt++;
        while(x!=u){
            x=s.top();s.pop();
            bccno[x]=bcnt;
        }
    }
}

void find_bcc()
{
    int i;
    bcnt=dfs_clock=0;
    mem(pre,0);mem(bccno,0);
    for(i=1;i<=n;i++){
```

```
        if(!pre[i])dfs(i,-1);
    }
}


/* Edge-Biconnected Component(不能处理重边)
  iscut[] 为割边集
  bccno[] 为双连通点集，保存为编号        */
struct Edge{
    int u,v;
}e[N*N];
bool iscut[N*N];
int first[N],next[N*N],pre[N],low[N],bccno[N];
int n,m,mt,bcnt,dfs_clock;
stack<int> s;

void adde(int a,int b)
{
    e[mt].u=a;e[mt].v=b;
    next[mt]=first[a];first[a]=mt++;
    e[mt].u=b;e[mt].v=a;
    next[mt]=first[b];first[b]=mt++;
}

void dfs(int u,int fa)
{
    int i,v;
    pre[u]=low[u]=++dfs_clock;
    s.push(u);
    for(i=first[u];i!=-1;i=next[i]){
        v=e[i].v;
        if(!pre[v]){
            dfs(v,u);
            low[u]=Min(low[u],low[v]);
            if(low[v]>pre[u])iscut[i]=true;   //存在割边
        }
        else if(v!=fa && pre[v]<pre[u]){  //反向边更新
            low[u]=Min(low[u],pre[v]);
        }
    }
    if(low[u]==pre[u]){  //充分必要条件
        int x=-1;
        bcnt++;
        while(x!=u){
            x=s.top();s.pop();
            bccno[x]=bcnt;
        }
    }
}

void find_bcc()
{
    int i;
    bcnt=dfs_clock=0;mem(iscut,0);
    mem(pre,0);mem(bccno,0);
    for(i=1;i<=n;i++){
        if(!pre[i])dfs(i,-1);
    }
}
```

5.9 SCC

```
/*   SCC   O(E)
   tarjan 算法
   sccno[] 强连通集合，用编号标示    */

struct Edge{
    int u,v;
}e[N*N];
int first[N],next[N*N],pre[N],sccno[N],low[N];
int n,mt,dfs_clock,scnt;
stack<int> s;

void adde(int a,int b)
{
    e[mt].u=a;e[mt].v=b;
    next[mt]=first[a],first[a]=mt++;
}

void dfs(int u)
{
    int i,j,v;
    pre[u]=low[u]=++dfs_clock;
    s.push(u);
    for(i=first[u];i!=-1;i=next[i]){
        v=e[i].v;
        if(!pre[v]){
            dfs(v);
            low[u]=Min(low[u],low[v]);
        }
        else if(!sccno[v]){    //反向边更新
            low[u]=Min(low[u],low[v]);
        }
    }
    if(low[u]==pre[u]){   //存在强连通分量
        int x=-1;
        scnt++;
        while(x!=u){
            x=s.top();s.pop();
            sccno[x]=scnt;
        }
    }
}

void find_scc()
{
    int i;
    mem(pre,0);mem(sccno,0);
    scnt=dfs_clock=0;
    for(i=1;i<=n;i++){
        if(!pre[i])dfs(i);
    }
}
```

5.10 二分匹配

```
/*    二分匹配-dfs   O(E)
  最小点集覆盖 = 最大匹配数
  最小路径覆盖 = n - 最大匹配数
  最大独立集  = n - 最大匹配数      */

/*   邻接表 - 对于稀疏图效果好   */
```

```
struct Edge{
    int u,v;
}e[N*N];

int vis[N],y[N],first[N],next[N*N];
int n,mt;

void adde(int a,int b)
{
    e[mt].u=a;e[mt].v=b;
    next[mt]=first[a];first[a]=mt++;
}

int dfs(int u)
{
    int i,v;
    for(i=first[u];i!=-1;i=next[i]){
        v=e[i].v;
        if(!vis[v]){
            vis[v]=1;
            if(y[v]==-1 || dfs(y[v])){
                y[v]=u;
                return 1;
            }
        }
    }
    return 0;
}

int match()
{
    int i,cnt=0;
    mem(y,-1);
    for(i=0;i<n;i++){
        mem(vis,0);
        if(dfs(i))cnt++;
    }
    return cnt;
}


/*   邻接矩阵   O(n^2)  */
int vis[N],y[N],g[N][N];
int n;

int dfs(int u)
{
    int v;
    for(v=0;v<n;v++){
        if(g[u][v] && !vis[v]){
            vis[v]=1;
            if(y[v]==-1 || dfs(y[v])){
                y[v]=u;
                return 1;
            }
        }
    }
    return 0;
}

int match()
{
```

```
    int i,cnt=0;
    mem(y,-1);
    for(i=0;i<n;i++){
        mem(vis,0);
        if(dfs(i))cnt++;
    }
    return cnt;
}
```

5.11 最大权匹配

```
/*   Kuhn-Munkers   O(n^3) | O(n^4)
   邻接矩阵实现
   维护可行顶标 l(x)+l(y)>=w(x,y)
   S[]=T[]=1，是匈牙利树的点集合
   如果求最小权匹配，维护 l(x)+l(y)<=w(x,y) 即可  */

/*     O(n^3)   最大权匹配   */
int w[N][N],S[N],T[N],lx[N],ly[N],y[N];
int n,slack;

int match(int u)
{
    int v,t;
    S[u]=1;
    for(v=1;v<=n;v++){
        t=lx[u]+ly[v]-w[u][v];  //最小:t=w[u][v]-lx[u]-ly[v];
        if(!t){
            if(!T[v]){
                T[v]=1;
                if(y[v]==-1 || match(y[v])){
                    y[v]=u;
                    return 1;
                }
            }
        }
        else if(t<slack)slack=t;
    }
    return 0;
}

void KM()
{
    int i,j,a;
    mem(y,-1);
    mem(ly,0);
    for(i=1;i<=n;i++){
        lx[i]=w[i][1];
        for(j=2;j<=n;j++)
            if(w[i][j]>lx[i])lx[i]=w[i][j]; //最小:if(w[i][j]<lx[i])lx[i]=w[i][j];
    }
    for(i=1;i<=n;i++){
        while(1){
            slack=INF;
            mem(S,0);mem(T,0);
            if(match(i))break;
            for(j=1;j<=n;j++){
                if(S[j])lx[j]-=slack;  //最小:if(S[j])lx[j]+=slack;
                if(T[j])ly[j]+=slack;  //最小:if(T[j])ly[j]-=slack;
            }
        }
```

```
    }
}

/*   O(n^4) 最大权匹配  实际效果没这么遭   */
int w[N][N],lx[N],ly[N],S[N],T[N],y[N];
int n,m;

int dfs(int u)
{
    S[u]=1;
    int v;
    for(v=1;v<=n;v++){
        if(w[u][v]==lx[u]+ly[v] && !T[v]){
            T[v]=1;
            if(y[v]==-1 || dfs(y[v])){
                y[v]=u;
                return 1;
            }
        }
    }
    return 0;
}

void update()
{
    int i,j,a;
    a=INF;
    for(i=1;i<=n;i++)if(S[i])
        for(j=1;j<=n;j++)if(!T[j])
            a=Min(a,lx[i]+ly[j]-w[i][j]);  //最小:a=Min(a,w[i][j]-lx[i]-ly[j]);
    for(i=1;i<=n;i++){
        if(S[i])lx[i]-=a;   //最小:if(S[j])lx[j]+=a;
        if(T[i])ly[i]+=a;   //最小:if(T[i])ly[i]-=a;
    }
}

void KM()
{
    mem(y,-1);
    mem(ly,0);
    int i,j;
    for(i=1;i<=n;i++){
        lx[i]=w[i][1];
        for(j=2;j<=n;j++)
            if(w[i][j]>lx[i])lx[i]=w[i][j];  //最小:if(w[i][j]<lx[i])lx[i]=w[i][j];
    }
    for(i=1;i<=n;i++){
        while(1){
            mem(S,0);mem(T,0);
            if(dfs(i))break;
            update();
        }
    }
}
```

5.12 带花树

```
/*    带花树    O(n^3)
  Edmonds's matching algorithm  Code by Amber
  邻接矩阵建图，n 为顶点个数，编号为 1-n
  Edmonds() 返回最大的匹配点数
```

```
  如果图要删除节点，直接在 3 个 for 循环里添加标记即可   */

int n;
int head,tail,Start,Finish;
int link[N];     //表示哪个点匹配了哪个点
int Father[N];   //这个就是增广路的 Father……但是用起来太精髓了
int Base[N];     //该点属于哪朵花
int Q[N];
bool mark[N];
bool map[N][N];
bool InBlossom[N];
bool in_Queue[N];

void BlossomContract(int x,int y)
{
    fill(mark,mark+n+1,false);
    fill(InBlossom,InBlossom+n+1,false);
    #define pre Father[link[i]]
    int lca,i;
    for (i=x;i;i=pre) {i=Base[i]; mark[i]=true; }
    for (i=y;i;i=pre) {i=Base[i]; if (mark[i]) {lca=i; break;} }  //寻找 lca 之旅……一定要注
意 i=Base[i]
    for (i=x;Base[i]!=lca;i=pre){
        if (Base[pre]!=lca) Father[pre]=link[i]; //对于 BFS 树中的父边是匹配边的点，Father 向
后跳
        InBlossom[Base[i]]=true;
        InBlossom[Base[link[i]]]=true;
    }
    for (i=y;Base[i]!=lca;i=pre){
        if (Base[pre]!=lca) Father[pre]=link[i]; //同理
        InBlossom[Base[i]]=true;
        InBlossom[Base[link[i]]]=true;
    }
    #undef pre
    if (Base[x]!=lca) Father[x]=y;     //注意不能从 lca 这个奇环的关键点跳回来
    if (Base[y]!=lca) Father[y]=x;
    for (i=1;i<=n;i++){
        if (InBlossom[Base[i]]){
            Base[i]=lca;
            if (!in_Queue[i]){
                Q[++tail]=i;
                in_Queue[i]=true;     //要注意如果本来连向 BFS 树中父结点的边是非匹配边的点，可
能是没有入队的
            }
        }
    }
}

void Change()
{
    int x,y,z;
    z=Finish;
    while (z){
        y=Father[z];
        x=link[y];
        link[y]=z;
        link[z]=y;
        z=x;
    }
}

void FindAugmentPath()
```

```
{
    fill(Father,Father+n+1,0);
    fill(in_Queue,in_Queue+n+1,false);
    for (int i=1;i<=n;i++) Base[i]=i;  //Init 属于同一花朵
    head=0; tail=1;
    Q[1]=Start; //当前节点进入队列
    in_Queue[Start]=1;
    while (head!=tail){
        int x=Q[++head];
        for (int y=1;y<=n;y++){
            if (map[x][y] && Base[x]!=Base[y] && link[x]!=y){   //无意义的边
               if ( Start==y || link[y] && Father[link[y]] )    //精髓地用 Father 表示该点是否
                    BlossomContract(x,y);
                else if (!Father[y]){
                    Father[y]=x;
                    if (link[y]){
                        Q[++tail]=link[y];
                        in_Queue[link[y]]=true;
                    }
                    else{
                        Finish=y;
                        Change();
                        return;
                    }
                }
            }
        }
    }
}

int Edmonds()
{
    int i,cnt=0;
    memset(link,0,sizeof(link));
    memset(Father,0,sizeof(Father));
    for (Start=1;Start<=n;Start++){
        if (link[Start]==0)
            FindAugmentPath();   //如果点没有匹配，那么找 BFS 增广路
    }

    for(i=1;i<=n;i++)
        if(link[i])cnt++;
    return cnt;
}
```

5.13 Dinic

```
/*   Dinic   O(n^2m)
   如果所有容量均为 1，复杂度 O(min(n^(2/3) * m, m^(1/2)) * m)
   对于二分图，复杂度 O(n^(1/2) * m)
   构造层次图，然后在层次图上 DFS 找增光路，路径 d[u]=d[v]+1
   加当前弧优化，效率提高                              */

struct Edge{
    int u,v,cap;
}e[N*N];

int first[N],next[N*N],d[N],cur[N];
int n,m,S,T,mt;

void adde(int a,int b,int val)
```

```
{
    e[mt].u=a;e[mt].v=b;
    e[mt].cap=val;
    next[mt]=first[a];first[a]=mt++;
    e[mt].u=b;e[mt].v=a;
    e[mt].cap=0;
    next[mt]=first[b];first[b]=mt++;
}

int bfs()
{
    int u,i,j;
    queue<int> q;
    mem(d,0);
    q.push(S);
    d[S]=1;
    while(!q.empty()){
        u=q.front();q.pop();
        for(i=first[u];i!=-1;i=next[i]){
            if(e[i].cap && !d[e[i].v]){
                d[e[i].v]=d[u]+1;
                q.push(e[i].v);
            }
        }
    }
    return d[T];
}

int dfs(int u,int a)
{
    if(u==T || a==0)return a;
    int f,flow=0;
    for(int& i=cur[u];i!=-1;i=next[i]){      //当前弧优化，从上次的弧考虑
        if( d[u]+1==d[e[i].v] && (f=dfs(e[i].v,Min(a,e[i].cap)) )){
            e[i].cap-=f;
            e[i^1].cap+=f;
            flow+=f;
            a-=f;
            if(!a)break;
        }
    }
    return flow;
}

int dinic()
{
    int i,flow=0;
    while(bfs()){
        for(i=0;i<=n;i++)cur[i]=first[i];  //注意这里是所有点都要赋值!!!
        flow+=dfs(S,INF);
    }
    return flow;
}
```

5.14 ISAP-当前弧优化 + 距离标号

```
/*  ISAP-当前弧优化 + 距离标号    O(V2E ?)
   加了当前弧优化以及 gap 优化
   n 为图点的个数
   mt 为边的条数，初始化 0   */
```

```
struct Edge{
    int u,v,cap;
}e[N*N];

int first[N],next[N*N],d[N],cur[N],fa[N],num[N];
int n,m,S,T,mt;   //s,t 分别为源点和汇点

void adde(int a,int b,int val)  //对于一条边，需建立双向边，一个容量为 cap，反向边容量为 0!
{
    e[mt].u=a;e[mt].v=b;
    e[mt].cap=val;
    next[mt]=first[a];first[a]=mt++;
    e[mt].u=b;e[mt].v=a;
    e[mt].cap=0;
    next[mt]=first[b];first[b]=mt++;
}
/*
void bfs()    // 初始化 d[]，多次求解规模较小的网络流是，效率会有明显提升，注意注销掉 isap() 中
的 mem(d,0);!
{
    int x,i,j;
    queue<int> q;
    mem(d,-1);
    q.push(T);
    d[T]=0;
    while(!q.empty()){
        x=q.front();q.pop();
        for(i=first[x];i!=-1;i=next[i]){
            if(d[e[i].v]<0){
                d[e[i].v]=d[x]+1;
                q.push(e[i].v);
            }
        }
    }
}
*/

int augment()
{
    int x=T,a=INF;
    while(x!=S){
        a=Min(a,e[fa[x]].cap);
        x=e[fa[x]].u;
    }
    x=T;
    while(x!=S){
        e[fa[x]].cap-=a;
        e[fa[x]^1].cap+=a;
        x=e[fa[x]].u;
    }
    return a;
}

int isap()
{
    int i,x,ok,min,flow=0;
    mem(d,0);
    mem(num,0);
    num[0]=n;
    for(i=0;i<=n;i++)cur[i]=first[i];   //注意这里的边界 i<=n
    x=S;
```

```
    while(d[S]<n){
        if(x==T){
            flow+=augment();
            x=S;
        }
        ok=0;
        for(i=cur[x];i!=-1;i=next[i]){
            if(e[i].cap && d[x]==d[e[i].v]+1){
                ok=1;
                fa[e[i].v]=i;
                cur[x]=i;
                x=e[i].v;
                break;
            }
        }
        if(!ok){
            min=n-1;
            for(i=first[x];i!=-1;i=next[i])
                if(e[i].cap && d[e[i].v]<min)min=d[e[i].v];
            if(--num[d[x]]==0)break;
            num[d[x]=min+1]++;
            cur[x]=first[x];
            if(x!=S)x=e[fa[x]].u;
        }
    }
    return flow;
}
```

5.15 最高标号预留推进 -HLPP

```
/*   HLPP - from watashi    O(n^3)
   ZOJ 2364:http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=2364
   所有点的标号范围为 [0,n-1]!        */

const int MAXN = 1515;
const int MAXM = 300300;

inline int RE(int i){
    return i ^ 1;
}

struct Edge{
    int v;
    int c;
};

struct FlowNetwork
{
    int n, m, source, sink;
    vector<int> e[MAXN];
    Edge edge[MAXM * 2];

    void init(int n, int source, int sink){
        this->n = n;
        this->m = 0;
        this->source = source;
        this->sink = sink;
        for (int i = 0; i < n; ++i){
            e[i].clear();
        }
    }
```

```
void addEdge(int a, int b, int c){
    edge[m].v = b;
    edge[m].c = c;
    e[a].push_back(m++);
    edge[m].v = a;
    edge[m].c = 0;
    e[b].push_back(m++);
}

int c[MAXN * 2];
int d[MAXN];
int w[MAXN];
int done[MAXN];

void bfs(){
    queue<int> q;
    fill(c, c + n * 2, 0);
    c[n + 1] = n - 1;
    fill(d, d + n, n + 1);
    d[source] = n;
    d[sink] = 0;
    q.push(sink);
    while (!q.empty()){
        int u = q.front();
        q.pop();
        --c[n + 1];
        ++c[d[u]];
        for (size_t i = 0; i < e[u].size(); ++i){
            Edge &cra = edge[RE(e[u][i])];
            int v = edge[e[u][i]].v;
            if (d[v] == n + 1 && cra.c > 0){
                d[v] = d[u] + 1;
                q.push(v);
            }
        }
    }
}

int hlpp(){
    vector<queue<int> > q(n * 2);
    vector<bool> mark(n, false);
    int todo = -1;

    bfs();
    mark[source] = mark[sink] = true;
    fill(w, w + n, 0);
    for (size_t i = 0; i < e[source].size(); ++i){
        Edge &arc = edge[e[source][i]];
        Edge &cra = edge[RE(e[source][i])];
        int v = arc.v;
        w[v] += arc.c;
        cra.c += arc.c;
        arc.c = 0;
        if (!mark[v]){
            mark[v] = true;
            q[d[v]].push(v);
            todo = max(todo, d[v]);
        }
    }
    fill(done, done + n, 0);
    while (todo >= 0)
```

```
        {
            if (q[todo].empty()){
                --todo;
                continue;
            }
            int u = q[todo].front();
            mark[u] = false;
            q[todo].pop();
            while (done[u] < (int)e[u].size()){
                Edge &arc = edge[e[u][done[u]]];
                int v = arc.v;
                if (d[u] == d[v] + 1 && arc.c > 0){
                    Edge &cra = edge[RE(e[u][done[u]])];
                    int f = min(w[u], arc.c);
                    w[u] -= f;
                    w[v] += f;
                    arc.c -= f;
                    cra.c += f;
                    if (!mark[v]){
                        mark[v] = true;
                        q[d[v]].push(v);
                    }
                    if (w[u] == 0){
                        break;
                    }
                }
                ++done[u];
            }
            if (w[u] > 0){
                int du = d[u];
                --c[d[u]];
                d[u] = n * 2;
                for (size_t i = 0; i < e[u].size(); ++i){
                    Edge &arc = edge[e[u][i]];
                    int v = arc.v;
                    if (d[u] > d[v] + 1 && arc.c > 0){
                        d[u] = d[v] + 1;
                        done[u] = i;
                    }
                }
                ++c[d[u]];
                if (c[du] == 0){
                    for (int i = 0; i < n; ++i){
                        if (d[i] > du && d[i] < n + 1){
                            --c[d[i]];
                            ++c[n + 1];
                            d[i] = n + 1;
                        }
                    }
                }
                mark[u] = true;
                q[d[u]].push(u);
                todo = d[u];
            }
        }
        return w[sink];
    }
};
```

5.16 最小费用流

```
/*   最小费用流   O(k*E)
   每次用 SPFA 扩展     */

struct Edge{
    int u,v,cap,w;
}e[N*N];

int d[N],first[N],next[N*N],inq[N],p[N];
int n,m,s,t,mt;

void adde(int a,int b,int c,int val){
    e[mt].u=a,e[mt].v=b,e[mt].cap=c,e[mt].w=val;
    next[mt]=first[a],first[a]=mt++;
    e[mt].u=b,e[mt].v=a,e[mt].cap=0,e[mt].w=-val;
    next[mt]=first[b],first[b]=mt++;
}

int Mincost()
{
    int i,j,x,a,cost=0;
    queue<int> q;
    p[s]=-1;
    while(1){
        a=INF;
        mem(d,INF);
        mem(inq,0);
        d[s]=0;
        q.push(s);
        while(!q.empty()){
            x=q.front();q.pop();
            inq[x]=0;
            for(i=first[x];i!=-1;i=next[i]){
                if(e[i].cap && d[e[i].u]+e[i].w<d[e[i].v]){
                    d[e[i].v]=d[e[i].u]+e[i].w;
                    p[e[i].v]=i;
                    if(!inq[e[i].v]){
                        q.push(e[i].v);
                        inq[e[i].v]=1;
                    }
                }
            }
        }
        if(d[t]==INF)break;
        for(i=p[t];i!=-1;i=p[e[i].u])
            if(e[i].cap<a)a=e[i].cap;
        for(i=p[t];i!=-1;i=p[e[i].u]){
            e[i].cap-=a;
            e[i^1].cap+=a;
        }
        cost+=d[t]*a;
    }
    return cost;
}
```

5.17 无向图的最小割

```
/*    Stoer_Wagner  O(n^3)  (POJ 2914)
   求无向图的最小割，把图分为两个子图的最小花费
   邻接矩阵建图,w[][] 点从 0 开始
   算法流程:
```

```
    1.min=MAXINT，固定一个顶点 P
    2. 从点 P 用“类似”prim 的 s 算法扩展出“最大生成树”，记录最后扩展的顶点和最后扩展的边
    3. 计算最后扩展到的顶点的切割值（即与此顶点相连的所有边权和），若比 min 小更新 min
    4. 合并最后扩展的那条边的两个端点为一个顶点（当然他们的边也要合并，这个好理解吧？）
    5. 转到 2，合并 N-1 次后结束
    6.min 即为所求，输出 min
  prim 本身复杂度是 O(n^2)，合并 n-1 次，算法复杂度即为 O(n^3)
  如果在 prim 中加堆优化，复杂度会降为 O((n^2)logn)            */

int w[N][N];
int v[N], dis[N];
bool vis[N];
int n,m;

int Stoer_Wagner(int n){
    int i, j, res = INF;
    for(i = 0; i < n; i ++)
        v[i] = i;
    while(n > 1){
        int k = 1, pre = 0;
        for(i = 1; i < n; i ++){
            dis[v[i]] = w[v[0]][v[i]];
            if(dis[v[i]] > dis[v[k]])
                k = i;
        }
        memset(vis, 0, sizeof(vis));
        vis[v[0]] = true;
        for(i = 1; i < n; i ++){
            if(i == n-1){
                res = min(res, dis[v[k]]);
                for(j = 0; j < n; j ++){
                    w[v[pre]][v[j]] += w[v[j]][v[k]];
                    w[v[j]][v[pre]] += w[v[j]][v[k]];
                }
                v[k] = v[-- n];
            }
            vis[v[k]] = true;
            pre = k;
            k = -1;
            for(j = 1; j < n; j ++)
                if(!vis[v[j]]){
                    dis[v[j]] += w[v[pre]][v[j]];
                    if(k == -1 || dis[v[k]] < dis[v[j]])
                        k = j;
                }
        }
    }
    return res;
}
```

6 计算几何

6.1 凸包

```
/*   凸包  O(n*lgn)
 Graham 算法
 p[] 为初始点集
 res[] 为凸包点集，逆时针排序  */

struct Point{
    double x, y;
}p[N],res[N];

bool mult(Point sp, Point ep, Point op)
{
    return (sp.x - op.x) * (ep.y - op.y)>= (ep.x - op.x) * (sp.y - op.y);
}

bool operator < (const Point &l, const Point &r)
{
    return l.y < r.y || (l.y == r.y && l.x < r.x);
}

int graham(Point pnt[], int n, Point res[])
{
    int i, len, k = 0, top = 1;
    sort(pnt, pnt + n);
    if (n == 0) return 0;
    res[0] = pnt[0];
    if (n == 1) return 1;
    res[1] = pnt[1];
    if (n == 2) return 2;
    res[2] = pnt[2];
    for (i = 2; i < n; i++){
        while (top && mult(pnt[i], res[top], res[top-1]))top--;
        res[++top] = pnt[i];
    }
    len = top;
    res[++top] = pnt[n - 2];
    for (i = n - 3; i >= 0; i--){
        while (top!=len && mult(pnt[i], res[top], res[top-1])) top--;
        res[++top] = pnt[i];
    }
    return top; // 返回凸包中点的个数
}
```

6.2 3 维凸包

```
//1. 输入点集  2. 构建凸包（hull.construct()） 3. 输出相应操作：面积、体积等。。。
const int MAX=110;
cosnt double PR=1e-8;
struct TPoint
{
    double x,y,z;
    TPoint(){}
    TPoint(double _x,double _y,double _z):x(_x),y(_y),z(_z){}
    TPoint operator-(const TPoint p) {return TPoint(x-p.x,y-p.y,z-p.z);}
   TPoint operator*(const TPoint p) {return TPoint(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);}//叉
积
    double operator^(const TPoint p) {return x*p.x+y*p.y+z*p.z;}//点积
};
```

```
struct fac//
{
    int a,b,c;//凸包一个面上的三个点的编号
    bool ok;//该面是否是最终凸包中的面
};
struct T3dhull
{
    int n;//初始点数
    TPoint ply[N];//初始点
    int trianglecnt;//凸包上三角形数
    fac tri[N];//凸包三角形
    int vis[N][N];//点 i 到点 j 是属于哪个面
    double dist(TPoint a){return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}//两点长度
    double area(TPoint a,TPoint b,TPoint c){return dist((b-a)*(c-a));}//三角形面积 *2
    double volume(TPoint a,TPoint b,TPoint c,TPoint d){return (b-a)*(c-a)^(d-a);}//四面体有向
体积 *6
    double ptoplane(TPoint &p,fac &f)//正：点在面同向
    {
        TPoint m=ply[f.b]-ply[f.a],n=ply[f.c]-ply[f.a],t=p-ply[f.a];
        return (m*n)^t;
    }
    void deal(int p,int a,int b)
    {
        int f=vis[a][b];
        fac add;
        if(tri[f].ok)
        {
            if((ptoplane(ply[p],tri[f]))>PR) dfs(p,f);
            else
            {
                add.a=b,add.b=a,add.c=p,add.ok=1;
                vis[p][b]=vis[a][p]=vis[b][a]=trianglecnt;
                tri[trianglecnt++]=add;
            }
        }
    }
    void dfs(int p,int cnt)//维护凸包，如果点 p 在凸包外更新凸包
    {
        tri[cnt].ok=0;
        deal(p,tri[cnt].b,tri[cnt].a);
        deal(p,tri[cnt].c,tri[cnt].b);
        deal(p,tri[cnt].a,tri[cnt].c);
    }
    bool same(int s,int e)//判断两个面是否为同一面
    {
        TPoint a=ply[tri[s].a],b=ply[tri[s].b],c=ply[tri[s].c];
        return fabs(volume(a,b,c,ply[tri[e].a]))<PR
            &&fabs(volume(a,b,c,ply[tri[e].b]))<PR
            &&fabs(volume(a,b,c,ply[tri[e].c]))<PR;
    }
    void construct()//构建凸包
    {
        int i,j;
        trianglecnt=0;
        if(n<4) return ;
        bool tmp=true;
        for(i=1;i<n;i++)//前两点不共点
        {
            if((dist(ply[0]-ply[i]))>PR)
            {
                swap(ply[1],ply[i]); tmp=false; break;
            }
```

```
    }
    if(tmp) return;
    tmp=true;
    for(i=2;i<n;i++)//前三点不共线
    {
        if((dist((ply[0]-ply[1])*(ply[1]-ply[i])))>PR)
        {
            swap(ply[2],ply[i]); tmp=false; break;
        }
    }
    if(tmp) return ;
    tmp=true;
    for(i=3;i<n;i++)//前四点不共面
    {
        if(fabs((ply[0]-ply[1])*(ply[1]-ply[2])^(ply[0]-ply[i]))>PR)
        {
            swap(ply[3],ply[i]); tmp=false; break;
        }
    }
    if(tmp) return ;
    fac add;
    for(i=0;i<4;i++)//构建初始四面体
    {
        add.a=(i+1)%4,add.b=(i+2)%4,add.c=(i+3)%4,add.ok=1;
        if((ptoplane(ply[i],add))>0) swap(add.b,add.c);
        vis[add.a][add.b]=vis[add.b][add.c]=vis[add.c][add.a]=trianglecnt;
        tri[trianglecnt++]=add;
    }
    for(i=4;i<n;i++)//构建更新凸包
    {
        for(j=0;j<trianglecnt;j++)
        {
            if(tri[j].ok&&(ptoplane(ply[i],tri[j]))>PR)
            {
                dfs(i,j); break;
            }
        }
    }
    int cnt=trianglecnt;
    trianglecnt=0;
    for(i=0;i<cnt;i++)
    {
        if(tri[i].ok)
            tri[trianglecnt++]=tri[i];
    }
}
double area()//表面积
{
    double ret=0;
    for(int i=0;i<trianglecnt;i++)
        ret+=area(ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
    return ret/2.0;
}
double volume()//体积
{
    TPoint p(0,0,0);
    double ret=0;
    for(int i=0;i<trianglecnt;i++)
        ret+=volume(p,ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
    return fabs(ret/6);
}
int facetri() {return trianglecnt;}//表面三角形数
```

```
    int facepolygon()//表面多边形数
    {
        int ans=0,i,j,k;
        for(i=0;i<trianglecnt;i++)
        {
            for(j=0,k=1;j<i;j++)
            {
                if(same(i,j)) {k=0;break;}
            }
            ans+=k;
        }
        return ans;
    }

}hull;
```

6.3 单位圆覆盖

```
/*   单位圆覆盖  O(n^2*lg n)
  将每个点扩展为单位圆，依次枚举每个单位圆，枚举剩下的单位圆
  如果有交点，每个圆产生两个交点
  然后对产生的 2n 个交点极角排序，判断被覆盖最多的弧
  被覆盖相当于这个弧上的点为圆心的圆可以覆盖到覆盖它的那些点，所以被覆盖最多的弧就是答案了。  */

struct Node{  //点
    double x,y;
}nod[N];
struct Point{  //极角
    double angle;
    int id;   //交点方向
    bool operator < (const Point& a)const{
        return angle!=a.angle?angle<a.angle:id>a.id;
    }
}p[N*2];
int n;

double dist(Node &a,Node &b)
{
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}

int slove()
{
    int i,j,ans,tot,k,cnt;
    ans=1;
    for(i=0;i<n;i++){
        for(j=k=0;j<n;j++){
            if(j==i || dist(nod[i],nod[j])>2.0)continue;
            double angle=atan2(nod[i].y-nod[j].y,nod[i].x-nod[j].x);  //注意为 i-j 的向量方向
            double phi=acos(dist(nod[i],nod[j])/2);
            p[k].angle=angle-phi;p[k++].id=1;
            p[k].angle=angle+phi;p[k++].id=-1;
        }
        sort(p,p+k);
        for(tot=1,j=0;j<k;j++){  //累计答案
            tot+=p[j].id;
            ans=Max(ans,tot);
        }
    }
    return ans;
}
```

6.4 平面最近点对

```
/*   平面最近点对   O( n*logn*logn )
  应用 G++ 提交，G++ 提交速度一倍！
  分治法求平面最近点对      */


struct Node{  //id 为 nod 排序后的编号值，index 为排序前的标号值（随便自己定义）
    double x,y;
    int id,index;
    Node(){}
    Node(double _x,double _y,int _index):x(_x),y(_y),index(_index){}
}nod[N],temp[N];

int n;

double dist(Node &a,Node &b)
{
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}

bool cmpxy(Node a,Node b)  //先按 x 排序，然后按 y 排序
{
    return a.x!=b.x?a.x<b.x:a.y<b.y;
}

bool cmpy(Node a,Node b)  //按 y 排序
{
    return a.y<b.y;
}

pii Closest_Pair(int l,int r)  //返回排序后点的编号
{
    if(l==r || l+1==r)return pii(l,r);   //只有一个点或者两个点
    double d,d1,d2;
    int i,j,k,mid=(l+r)/2;
    //左右两边最小距离点的编号
    pii pn1=Closest_Pair(l,mid);
    pii pn2=Closest_Pair(mid+1,r);
    //左右两遍的最小距离
    d1=(pn1.first==pn1.second?OO:dist(nod[pn1.first],nod[pn1.second]));
    d2=(pn2.first==pn2.second?OO:dist(nod[pn2.first],nod[pn2.second]));
    pii ret;
    d=Min(d1,d2);
    ret=d1<d2?pn1:pn2;
    //找出在 mid-d,mid+d 之间的点
    for(i=l,k=0;i<=r;i++){
        if(fabs(nod[mid].x-nod[i].x)<=d){
            temp[k++]=nod[i];
        }
    }
    //合并两边，求最小距离
    sort(temp,temp+k,cmpy);
    for(i=0;i<k;i++){
        for(j=i+1;j<k && fabs(temp[j].y-temp[i].y)<d;j++){
            if(dist(temp[i],temp[j])<d){
                d=dist(temp[i],temp[j]);
                ret=make_pair(temp[i].id,temp[j].id);
            }
        }
    }
```

```
    return ret;
}

void Init()   //初始化点
{
    int i;
    double x,y;
    scanf("%d",&n);
    for(i=0;i<t;i++){
        scanf("%lf%lf",&x,&y);
        nod[i]=Node(x,y,i);
    }
    sort(nod,nod+n,cmpxy);
    for(i=0;i<n;i++)nod[i].id=i;  //排序后节点编号
}
```

6.5 判定点在多边形内

```
/*   点在多边形内
   从点 p 做一条射线，看射线和多边形的交点有几个，奇数个为相交，偶数个不相交。。。
   num_node 为多边形点的个数
   nod[] 为多边形点集
   p 为多判断的点            */

struct Node{
    double x,y;
}nod[N];
int n;

int chaji(Node &a,Node &b){
    return a.x*b.y-b.x*a.y;
}

int ponls(Node &a,Node &b,Node &p)
{
    if( (p.x==a.x && p.y==a.y) || (p.x==b.x && p.y==b.y) )return 2;
    Node r1,r2;
    r1.x=a.x-b.x,r1.y=a.y-b.y;
    r2.x=p.x-b.x,r2.y=p.y-b.y;
    if(!chaji(r1,r2) && p.x>=min(a.x,b.x) && p.x<=max(a.x,b.x)
        && p.y>=min(a.y,b.y) && p.y<=max(a.y,b.y))
        return 1;
    return 0;
}

int quick(Node &l1,Node &l2,Node &r1,Node &r2)
{

    if(min(l1.x,l2.x)>max(r1.x,r2.x)
        || min(l1.y,l2.y)>max(r1.y,r2.y)
        || max(l1.x,l2.x)<min(r1.x,r2.x)
        || max(l1.y,l2.y)<min(r1.y,r2.y))
        return 0;
    return 1;
}

int las(Node &l1,Node &l2,Node &r1,Node &r2)
{
    Node a,b,c;
    a.x=l1.x-r1.x;
    a.y=l1.y-r1.y;
```

```
    b.x=r2.x-r1.x;
    b.y=r2.y-r1.y;
    c.x=l2.x-r1.x;
    c.y=l2.y-r1.y;
    if( ((a.x*b.y)-(b.x*a.y))*((c.x*b.y)-(b.x*c.y))<0)return 1;
    else return 0;
}

int pinply(int num_node,Node nod[],Node &p)
{
    int i,j,cou=0;
    Node ray;
    ray.x=-1,ray.y=p.y;
    for(i=0;i<num_node;i++){
        j=(i+1)%num_node;
        if(ponls(nod[i],nod[j],p))return 0;
        if(nod[i].y!=nod[j].y){
            if(ponls(p,ray,nod[i]) && nod[i].y==max(nod[i].y,nod[j].y))
                cou++;
            else if(ponls(p,ray,nod[j]) && nod[j].y==max(nod[i].y,nod[j].y))
                cou++;
            else if(quick(nod[i],nod[j],p,ray) && las(nod[i],nod[j],p,ray)
                && las(p,ray,nod[i],nod[j]))
                cou++;
        }
    }
    return cou&1;
}
```

7 动态规划

7.1 简单状态压缩

```
/*  POJ-1160   */

int ma[N][N];
LL f[2][1<<N];
int T,n,m;

int main()
{
 //   freopen("in.txt","r",stdin);
    int i,j,k,x,y,up,icase=0,p;
    scanf("%d",&T);
    while(T--)
    {
        mem(f,0);
        scanf("%d%d",&n,&m);
        for(i=0;i<n;i++)
            for(j=0;j<m;j++)
                scanf("%d",&ma[i][j]);
        f[0][0]=p=1;
        up=1<<(m+1);
        for(i=0;i<n;i++,mem(f[p=!p],0)){
            for(j=0;j<m;j++,mem(f[p=!p],0)){
                x=1<<j;
                y=1<<(j+1);
                for(k=0;k<up;k++){
                    if(ma[i][j]){
                        f[p][k^x^y]+=f[!p][k];
                        if((k&x) && (k&y))continue;
                        if((k&x)^(k&y))f[p][k]+=f[!p][k];
                    }
                    else if(!(k&x) && !(k&y)){
                        f[p][k]+=f[!p][k];
                    }
                }
            }
            for(j=0;j<(1<<m);j++)f[p][j<<1]=f[!p][j];
        }

        printf("Case %d: There are %I64d ways to eat the trees.\n",++icase,f[!p][0]);
    }
    return 0;
}
```

7.2 插头 DP(哈密顿回路数)

```
/*  插头 DP 最小表示法 求哈密顿回路数
   g[][] 为图
   code[] 为最小表示法
   如果 TLE，可调节 Hash 表大小    */

const int N=15,INF=0x3f3f3f3f,MOD=4001,STA=1000010;

int g[N][N],code[N],ma[N];
int n,m,ex,ey;

struct Hash{     //Hash 表,MOD 为表长,STA 为表大小
    int first[MOD],next[STA],size;
```

```
    LL f[STA],sta[STA];
    void init(){
        size=0;
        mem(first,-1);
    }
    void add(LL st,LL ans){
        int i,u=st%MOD;
        for(i=first[u];i!=-1;i=next[i]){
            if(sta[i]==st){
                f[i]+=ans;
                return;
            }
        }
        sta[size]=st;
        f[size]=ans;
        next[size]=first[u];
        first[u]=size++;
    }
}hs[2];

void shift(int p)    //换行移位
{
    int k;
    LL sta;
    for(k=0;k<hs[!p].size;k++){
        sta=hs[!p].sta[k]<<3;  //注意修改移位大小
        hs[p].add(sta,hs[!p].f[k]);
    }
}

LL getsta()   //最小表示法
{
    LL i,cnt=1,sta=0;
    mem(ma,-1);
    ma[0]=0;
    for(i=0;i<=m;i++){
        if(ma[code[i]]==-1)ma[code[i]]=cnt++;
        code[i]=ma[code[i]];
        sta|=(LL)code[i]<<(3*i);   //注意修改移位大小
    }
    return sta;
}

void getcode(LL sta)
{
    int i;
    for(i=0;i<=m;i++){
        code[i]=sta&7;
        sta>>=3;    //注意修改移位大小
    }
}

void unblock(int i,int j,int p)
{
    int k,t;
    LL cnt,x,y;
    for(k=0;k<hs[!p].size;k++){
        getcode(hs[!p].sta[k]);
        x=code[j],y=code[j+1];
        cnt=hs[!p].f[k];
        if(x && y){     //合并连通分量
            code[j]=code[j+1]=0;
```

```
            if(x!=y){
                for(t=0;t<=m;t++)
                    if(code[t]==y)code[t]=x;
                hs[p].add(getsta(),cnt);
            }
            else if(i==ex && j==ey){   //最后一个点特殊处理
                hs[p].add(getsta(),cnt);
            }
        }
        else if(x&&!y || !x&&y){   //延续连通分量
            t=x?x:y;
            if(g[i+1][j]){
                code[j]=t;code[j+1]=0;
                hs[p].add(getsta(),cnt);
            }
            if(g[i][j+1]){
                code[j]=0;code[j+1]=t;
                hs[p].add(getsta(),cnt);
            }
        }
        else if(g[i+1][j] && g[i][j+1]){  //创建新连通分量
            code[j]=code[j+1]=8;
            hs[p].add(getsta(),cnt);
        }
    }
}

void block(LL j,int p)
{
    int k;
    for(k=0;k<hs[!p].size;k++){
        getcode(hs[!p].sta[k]);
        code[j]=code[j+1]=0;
        hs[p].add(getsta(),hs[!p].f[k]);
    }
}

LL slove()
{
    int i,j,p;
    hs[0].init();
    hs[p=1].init();
    hs[0].add(0,1);
    for(i=0;i<n;i++){
        for(j=0;j<m;j++){
            if(g[i][j])unblock(i,j,p);
            else block(j,p);  //p=!p 优化
            hs[p=!p].init();
        }
        shift(p);   //换行移位
        hs[p=!p].init();
    }
    for(i=0;i<hs[!p].size;i++)
        if(hs[!p].sta[i]==0)return hs[!p].f[i];
    return 0;
}
```