ACM 模板

ECUST ACMer　Ch_g

2010 - …

目录

1. 数学

```
/*==================================================================================*\
  1)   线性筛素数
\*==================================================================================*/
const int N = 25600000;
bool a[N]; //判断是否为合数
int p[N/2],num;
void prime(){
   memset(a, 0, sizeof(a));
   int i, j;
    num = 0;
   for(i = 2; i < N; ++i){
      if(a[i]==0) p[num++] = i;
      for(j = 0; j<num && i*p[j]<N ; ++j){
          a[i*p[j]] = 1;
          if(i%p[j]==0) break;
      }
   }
}
/*==================================================================================*\
  2)   GCD 最大公约数
\*==================================================================================*/
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}
/*==================================================================================*\
  3)   com(n,r)%mod  和  n!快速质因数分解
\*==================================================================================*/
//这里直接用a*b%c,而没有用模板中的函数product_mod 和 product_mod是为了提高效率
typedef long long lld;
const int N = 2000000;
bool a[N];
int p[N/2],num,n1[N/2],n2[N/2],n3[N/2];
void prime();
//------------------------以上为引用模板内其他函数------------------------//
template<class Type>
Type pow(Type a,Type b,Type c){
    Type r=1;
    while(b){
       if(b & 1)r=r*a%c;
       a=a*a%c;
       b>>=1;
    }
    return r;
}
void div(int *list,int n){
    int i,tmp;
    for (i=0;i<num;i++) list[i]=0;
    for (i=0;i<num && p[i]<=n;i++){
        tmp=n;
        while (tmp/p[i]){list[i]+=tmp/p[i];tmp/=p[i];}
    }
}
lld com(lld n,lld r,int mod){
    lld res=1,temp;
    div(n1,n);  div(n2,r);  div(n3,n-r);
    for(int i=0;i<num;i++){
        n1[i]-=n2[i]+n3[i];
        if(n1[i]==0)continue;
        temp=pow(p[i],n1[i],mod);
        res=(res*temp)%mod;
    }
    return res;
}
int main(){
    prime();
    ……
}
/*================================================================================*\
  4)   组合数(n 最大取 61)(若用 int 型 n 最大取 33)
\*================================================================================*/
long long com(int n,int r){
    if(r > n)return 0;
    if(n-r < r)r=n-r;
    int i,j;
    long long s=1;
    for(i=0,j=1;i<r;i++){
        s*=n-i;
        for(;j<=r && s%j==0;j++)s/=j;
    } return s;
}
/*================================================================================*\
  5)   double 型组合数(n 最大可取 1029,一般 6 位精确值)
\*================================================================================*/
double lnc(int n,int r){ //求ln(c(n,r))
    if(r > n-r)r = n-r;
    double s1=0,s2=0;
    for(int i=0;i<r;i++)  s1 += log((double)n-i);
    for(int i=2;i<=r;i++)  s2 += log((double)i);
    return s1-s2;
}
double com(int n,int r){
    if(r>n)return 0;
    return exp(lnc(n,r));
}
```

```
/*==========================================================================*\
  6)   a*b % c
\*==========================================================================*/
template<class Type>
Type product_mod(Type a,Type b,Type c){
   Type r=0;
   while(b>0){
      if(b & 1)r=(r+a)%c;
      a=(a+a)%c;
      b>>=1;
   } return r;
}
/*==========================================================================*\
  7)   a^b % c
\*==========================================================================*/
template<class Type>
Type product_mod(Type a,Type b,Type c);
//-----------------------以上为引用模板内其他函数-----------------------//
template<class Type>
Type power_mod(Type a,Type b,Type c){
   Type r=1;
   while(b){
      if(b & 1)r=product_mod(r,a,c);
      a=product_mod(a,a,c);
      b>>=1;
   } return r;
}
/*==========================================================================*\
  8)   素数测试
\*==========================================================================*/
//3e+14以内100%正确，(long long型/2)之内
//也应该是正确的，不过没有找到理论依据
template<class Type>
Type product_mod(Type a,Type b,Type c);
template<class Type>
Type power_mod(Type a,Type b,Type c);
//-----------------------以上为引用模板内其他函数-----------------------//
bool isprime(long long n){
   if(n<2)return false;
   if(n==2)return true;
   if(n%2==0)return false;
   int i,j,k=0;
   int p[]={2,3,5,7,11,13,17,23,37,51,61};
   long long a,m=n-1;
   while(m % 2 == 0)m>>=1,k++;
   for(i=0;i<11;i++){
       if(p[i]>=n)return true;
       a = power_mod((long long)p[i],m,n);
       if(a==1)continue;
       for(j = 0; j < k; j ++){
          if(a==n-1)break;
          a = product_mod(a,a,n);
       }
       if(j==k)return false ;
   }  return true;
}
/*=========================================================================================*\
  9)    整数的因子分解
\*=========================================================================================*/
#intlude<cstdlib>
using namespace std;
Type gcd(Type a,Type b);
Type product_mod(Type a,Type b,Type c);
bool isprime(long long n);
//--------------------------以上为引用模板内其他函数--------------------------//
typedef long long lld;
lld pollard_rho(lld c,lld n){
   int i=1;
   lld x=rand()%n, y=x;
   int k=2;
   do{
      i++;
      lld d=gcd(n+y-x,n);
      if(d>1 && d<n)return d;
      if(i==k)y=x,k*=2;
      x=(product_mod(x,x,n)+n-c)%n;
   }while(y!=x);
   return n;
}
lld rho(lld n){
   if(isprime(n))return n;
   while(1){
      lld t = pollard_rho(rand()%(n-1)+1,n);
      if(t<n){
         lld a=rho(t),b=rho(n/t);
         return a<b ? a:b;
      }
   }
}
/*=========================================================================================*\
  10)   所有数位相加
\*=========================================================================================*/
int dig(int x){
   if(x==0)return 0;
   return (x+8)%9+1;
}
```

```
/*==================================================================================*\
  11)  统计 x 二进制表示中 1 的个数
\*==================================================================================*/
template<class Type>
int count(Type x){
   Type n=x>>1;
   while(n){x-=n;n>>=1;}
   return x;
}
```

```
/*==================================================================================*\
  12)  数值转换为 char* （十进制转换）
\*==================================================================================*/
//将十进制数v转换成2至36进制的数，结果以
//字符串形式存放入dest中，r为进制数，函数//返回字符串长度
//注意v为0时，返回字符串长度为0
template<class Type>
int my_itoa(Type v,char *dest,int r){
   if(v==0){
      dest[0]='0';dest[1]='\0';
      return 0;
   }
   int len=my_itoa(v/r,dest,r);
   int t=v%r;
   if(t<10)dest[len++]='0'+t;
   else dest[len++]='a'+t-10;
   dest[len]='\0';
   return len;
}
```

```
/*==================================================================================*\
  13)  约瑟夫环问题
   n 个人(编号 1…n)围成一圈，从第 k 个人开始，从 1 报数，报到 m 出列，下一个人继续从 1 报数
\*==================================================================================*/
//-----------------------数学方法-----------------------//
```

方法一：（可以求得每次出列者编号）

```
int i=0,n,m,k,p;
scanf("%d%d%d",&k,&n,&m);
while(++i<=n){//每次计算第i个出列者的编号
   p=i*m;
   while(p>n){ p=p-n+(p-n-1)/(m-1);}
   printf("%d\n",(p+n+k-2)%n+1);
}
```

方法二：(只能求得最后剩下那人的编号，但效率较高)

```
int n,m,k,p;
scanf("%d%d%d",&k,&n,&m);
p=0;//运算时是从0开始数，数到m-1
for(int i=2;i<=n;i++)p=(p+m)%i;
printf("%d\n",(p+n+k)%n+1);
```

```
/*==========================================================================*\
  14)  生成下一个二进制中有 k 个 1 的数
\*==========================================================================*/
int nxt(int x) {
   int b = x & -x;
   int t = x + b;
   int c = t ^ x;
   int m = (c >> 2) / b;
   int r = t | m;
   return r;
}
/*==========================================================================*\
  15)  Nim 积
\*==========================================================================*/
int nimpow(int x, int y) { // x=2^a;
   if (y < 2) return y ? x : 0;
   int m, b;
   for (b = 0, m = 2; x >= m; )
      m = 1 << (1 << ++b);
   m = 1 << (1 << --b);
   int p = x>>(1<<b);
   int s = y>>(1<<b), t = y&(m-1);
   int d1 = nimpow(p, s);
   int d2 = nimpow(p, t);
   return ((d1 ^ d2) << (1 << b)) ^ nimpow(m >> 1, d1);
}
int nimx(int x, int y) {
   if (x < y) return nimx(y, x);
   if (y < 2) return y ? x : 0;
   int m, b;
   for (b = 0, m = 2; x >= m; )
      m = 1 << (1 << ++b);
   m = 1 << (1 << --b);
   int p = x>>(1<<b), q = x&(m-1);
   int s = y>>(1<<b), t = y&(m-1);
   int c1 = nimx(p, s);
   int c2 = nimx(p, t) ^ nimx(q, s);
   int c3 = nimx(q, t) ^ ((c1 ^ c2) << (1 << b));
   return c3 ^ nimpow(m >> 1, c1);
}
*==========================================================================*\
  16)  稳定婚姻问题
\*==========================================================================*/
const int N = 1010;
int n, resm[N], resw[N];
int man[N][N], woman[N][N];
int chose[N];
```

```
void stableMatch() {
   queue<int> q;
   memset(chose, 0, sizeof(chose));
   memset(resw, -1, sizeof(resw));
   for (int i = 0; i < n; ++i) q.push(i);
   while (!q.empty()) {
      int u = q.front(); q.pop();
      int v = man[u][chose[u]++];
      if (resw[v] == -1 || woman[v][resw[v]] > woman[v][u]) {
         if (resw[v] != -1) q.push(resw[v]);
         resm[u] = v;
         resw[v] = u;
      } else {
         q.push(u);
      }
   }
}

void solve() {
   int u;
   scanf("%d", &n);
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j) {
         scanf("%d", &u); man[i][j] = u - 1; //读入男士对女士的偏好次序
      }
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j) {
         scanf("%d", &u); woman[i][u - 1] = j; //读入女士对男士的偏好次序
      }
   stableMatch();
   for (int i = 0; i < n; ++i)
      printf("%d\n", resm[i] + 1);  // 输出男士有利的稳定婚姻方案
}
/*============================================================================*\
  17)  三分法(凸性函数求极值)
\*============================================================================*/
const double EPS=1e-10;
double MIN, MAX;
double cal(double x){
   /* 根据题目的意思计算 */
}

double solve(){
   double left=MIN,right=MAX;
   double mid1,mid2;
   while(left+EPS<right){
      mid1=left+(right-left)*0.381966;
      mid2=left+(right-left)*0.618034;
```

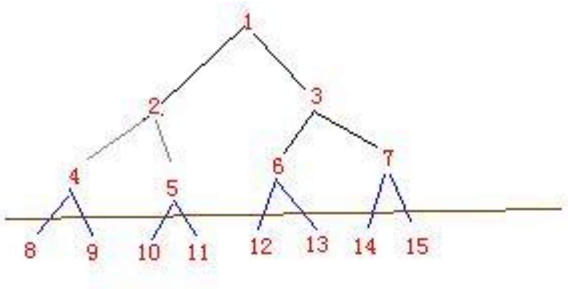
```
        if(cal(mid1)<cal(mid2)) right=mid2; //求极小值
    //  if(cal(mid1)>cal(mid2)) right=mid2; //求极大值
        else left=mid1;
    }
    return cal(left);
}

*==============================================================================*\
 18) 线性规划(watashi 的模板)
\*==============================================================================*/
const double EPS = 1e-9;
const int MAX_N = 128;
const int MAX_M = 128;
typedef double LPMAT[MAX_M + 1][MAX_M + MAX_N + 1];
typedef double LPRET[MAX_N + 1];

inline bool d_zero(double x) {
    return fabs(x) < EPS;
};
inline bool d_less(double x, double y) {
    return x + EPS < y;
};
/** 线性规划.
 * 求 b1 x1 + b2 x2 + ... + bn xn 在 x1, x2, ..., xn >= 0 时的最大值
 * m 为限制条件方程数, n 为变量数目
 * 限制条件为
 *  a_i1 x1 + a_i2 x2 + ... + a_in xn <= ci (1 <= i <= m, ci >= 0 (!!!))
 * mat 传入系数矩阵
 * mat[0] 表示目标方程, 即 mat[0][1..n] 为 b1 .. bn
 * mat[1..m]表示限制方程, mat[1..m][0] 表示 c1, c2, ... cm
 * 其余 mat[i][j] 表示 a[i][j]
 * 注意函数会改变 mat 的值
 *   i \ j 0    1    2   .... n
 *   0     0   b1   b2   .... bn
 *   1     c1  a11  a12  .... a1n
 *   ............................
 *   m     cm  am1  am2  .... amn
 * 找到解返回 true, ans 返回最大值, ret[1..n] 分别返回 x1..xn 的取值
 * 如果不存在最大值返回 false
 * 不可能无解, 因为 x1 = x2 = ... = xn = 0 必为一组解
 */
bool lp(int m, int n, LPMAT mat, double& ans, LPRET ret) {
    static int p[MAX_M + 1], q[MAX_M + MAX_N + 1];
    static double trial[MAX_M + MAX_N + 1];
    int i, j, k, l, s, h;
    double z, zbuf;

    mat[0][0] = 0;
    for (i = 0; i <= m; i++) {
        for (j = n; j > 0; j--)
            mat[i][j + m] = (i == 0) ? -mat[i][j] : mat[i][j];

        for (j = m; j > 0; j--)
            mat[i][j] = (i == j) ? 1 : 0;

        p[i] = q[i] = i;
    }

    bool flag = true;
    while (flag) {
        flag = false;
        for (j = m + n; j > 0; j--) {
            if (!d_less(mat[0][j], 0))
                continue;
            for (i = 1, l = 0; i <= m; i++) {
                if (!d_less(0, mat[i][j]))
                    continue;
                if (l == 0) {
                    l = i, s = 0;
                } else {
                    for (h = 0; ; h++) {
                        if (h == s)
                            trial[s++] = mat[l][h] / mat[l][j];
                        z = mat[i][h] / mat[i][j];
                        if (trial[h] != z) break;
                    }
                    if (d_less(z, trial[h]))
                        l = i, trial[h] = z, s = h + 1;
                }
            }
            if (l == 0) return false;  // The maximum is infinite

            for (k = 0, z = mat[l][j]; k <= m + n; k++)
                if (!d_zero(mat[l][k]))
                    mat[l][k] = mat[l][k] / z;

            for (i = 0; i <= m; i++) {
                if (i == l)
                    continue;
                for (k = 0, z = mat[i][j]; k <= m + n; k++)
                    mat[i][k] = (k == j || d_zero(zbuf = mat[i][k] - z *
mat[l][k])) ?  0 : zbuf;
            }
            q[p[l]] = 0, p[l] = j, q[j] = l;
            flag = true;
            break;
        };
    };
    ans = mat[0][0];

    for (i = 1, j = m + 1; j <= m + n; i++, j++)
        ret[i] = (q[j]) ? mat[q[j]][0] : 0;

    /* 此处可用来计算 (u1, u2, ..., um) 其中 ui >= 0, 使得在
     * a1j u1 + a2j u2 + ... + amj um >= bj (1 <= j <= n)
     * 的限制条件下 c1 u1 + c2 u2 + ... + cm um 最大.
     * 当函数返回 false 的时候此处无解. */
    //for (j = 1; j <= m; j++)
    //    u[i] = mat[0][j];

    return true;
}
```

2． 数据结构

```
/*==========================================================================*\
  1)   二分查找
\*==========================================================================*/
//a[]已经有序(数值可以不唯一)
//在[low,high)范围内查找值v
//返回第1个匹配的下标,失败返回-1
template<class Type>
int bs(Type a[],int low,int high,Type v){
    if(low==high)return -1;
    int mid;
    while (low<high){
        mid=(low+high)>>1;
        if(a[mid]<v)low=mid+1;
    //  if(a[mid]>v)low=mid+1;  //a[]递减
        else high=mid;
    }
    if(a[low]==v)return low;
    return -1;
}
/*==========================================================================*\
  2)   二分查找(大于等于 v(/大于 v)的第一个值)
\*==========================================================================*/
//范围为[low,high)
//a[]递增。若 v 最大则返回 high 的值
template<class Type>
int bsh(Type a[],int low,int high,Type v){
    int mid;
    while (low<high){
        mid=(low+high)>>1;
        if(a[mid]<v)low=mid+1;  //大于等于v
    //  if(a[mid]<=v)low=mid+1; //大于v
        else high=mid;
    }
    return low;
}
/*==========================================================================*\
  3)   最长有序子序列(递增/非递减)
\*==========================================================================*/
const int N=30010;
Type a[N],f[N]; //题目中的序列存入a[]中
int d[N];
template<class Type>
int bsh(Type a[],int low,int high,Type v)
    //递增时bsh找大于等于v的第一个值
    //非递减时bsh找大于v的第一个值
//-----------------------以上为引用模板内其他函数-----------------------//
```

```
template<class Type>
int lis(Type a[],int n){//n为a序列的总数
   int i,j,size=1;
   f[0]=a[0];d[0]=1;
   for(i=1;i<n;i++){
      j=bsh(f,0,size,a[i]);
      f[j]=a[i];d[i]=j+1;
      if(j==size)size++;
   }
   return size;
}
/*=============================================================================*\
  4)   Matrix
\*=============================================================================*/
const int N = 300; // 300左右是极限了，否则ans什么都就要开成全局变量
const int mod = 1000000007;
typedef int type;
struct matrix {
   int n;
   type a[N][N];
   void clear() {forn (i, n) forn (j, n) a[i][j] = 0; }
   matrix(){}
   matrix(int z) { n = z; clear(); }
   matrix operator + (const matrix& u) {
      matrix ans; ans.n = n;
      forn (i, n) forn (j, n) {
         ans.a[i][j] = a[i][j] + u.a[i][j];
         if (ans.a[i][j] >= mod) ans.a[i][j] %= mod;
      } return ans;
   }
   matrix operator * (const matrix& u) {
      matrix ans(n);
      forn (i, n) forn (k, n) if (a[i][k])
         forn (j, n) if (u.a[k][j]) {
            ans.a[i][j] += a[i][k] * u.a[k][j];
            if (ans.a[i][j] >= mod) ans.a[i][j] %= mod;
         }
      return ans;
   }
   matrix pow(int k) {
      matrix r(n), t = *this;
      forn (i, n) r.a[i][i] = 1;
      while (k) {
         if (k & 1) r = t * r;
         t = t * t;
         k >>= 1;
      } return r;
   }
   matrix calc(int); // A + A^2 + A^3 + ... + A^k
}mtx;

matrix matrix::calc(int k) {
   matrix r(2 * n), t, ret(n);
   forn(i, n) forn(j, n)
      r.a[i][j] = r.a[i][j + n] = a[i][j];
   forn(i, n) r.a[i + n][i + n] = 1;
   t = r.pow(k);
   forn(i, n) forn(j, n)
      ret.a[i][j] = t.a[i][j + n];
   return ret;
}
/*==================================================================================*\
  5)   树状数组 解决 RMQ 问题
\*==================================================================================*/
const int N=50010;
int n,v[N],c[N];     //v[]需初始化,下标为1…n
int lowb(const int &a){ return a & (-a);}
void preprocess(){   //v[]中已有值，树状数组c[]存放区间最大值
   int i,y,k;
   for(i=1;i<=n;i++){
      y=lowb(i); c[i]=v[i];
      for(k=1;k<y;k<<=1)
         if(c[i]<c[i-k])c[i]=c[i-k];
   }
}
int search(int a,int b){//查找[a,b]中最大值
   int res=-0x7fffffff;
   a--;
   while(1){
      for(;b-lowb(b)>=a && b;b-=lowb(b))
         if(res<c[b])res=c[b];
      if(b==a)break;
      if(res<v[b])res=v[b];
      b--;
   }
   return res;
}
/*==================================================================================*\
  6)   ST 算法 解决 RMQ 问题:
\*==================================================================================*/
//这里查询的是最大值
//下标范围0…n-1
int mx[N][20], ln[N], val[N];//val[]置为待查询数组
void init(int n){
   int i,j,k,sk;
   ln[0] = ln[1] = 0;
```

```
    for (i = 2; i < n; ++i)
        ln[i] = ln[i >> 1] + 1;
    for(i=0;i<n;i++) mx[i][0]=val[i];
    for(i=1,k=2;k<n;i++,k<<=1)
        for(j=0,sk=k>>1;j<n;j++,sk++){
            mx[j][i]=mx[j][i-1];
            if(sk<n && mx[j][i]<mx[sk][i-1])
                mx[j][i]=mx[sk][i-1];
        }
}
int query(int a,int b){
    int bl = ln[b - a + 1];
    return max(mx[a][bl],mx[b-(1<<bl)+1][bl]);
}
/*==================================================================================*\
  7)    点树
\*==================================================================================*/
/*
  点树 - (线段树的一种拓展，专门针对点操作)
  实现思路：
    先画一棵完全二叉树， 为节省空间，采用数组来实现。对这棵二叉树，叶子用于存放数据，节点用于统计叶子信息。
通过下面的三种方法，进一步节省空间：
1    节点只记录左子树叶子信息，右子树叶子信息通过当前节点和父节点等节点的值计算得出。
      因而需要指定一个点，当作根节点的"父节点"，以便计算根节点右子树信息。
      可以将根节点从1开始编号，对节点i，左孩子编号为2*i，右孩子编号为2*i+1。
2    对某些应用，叶子信息可以通过节点信息计算得出，因而不保存叶子信息，
3    完全二叉树，边界要求为2^k，为了表示[0, n)这n个点，需要将n增加到2^k，实际上，
     只要第n个叶子的父节点r存在就可以了，编号大于r的节点根本不会被访问到，因而没必要分配空间
 */
```



额外增加节点0，记录所有叶子。

节点	所统计的(左子树下的)叶子
0:	8 9 10 11 12 13 14 15
1:	8 9 10 11
2:	8 9
3:	12 13
4:	8
5:	10
6:	12
7:	14

叶子8-15对应输入的数据，节点1-7记录左子树下的叶子数目
为节省空间，各节点不记录右子树的信息，且不记录叶子对应的数据

点树存储结构示意图

```
template<int N> // 表示可用区间为[0,N)，其中N必须是2的幂数;
class pointtree {
public:
    int a[2 * N];
    int size;
```

```
   void clear() {
      memset(this, 0, sizeof(*this));
   }
   void ins(int n) {
      ++size;
      for (++a[n += N]; n > 1; n >>= 1)
         if (~n & 1)
            a[n >> 1]++;
   }
   void del(int n) {
      if (!a[n += N]) return; // 没有n
      --size;
      for (--a[n]; n > 1; n >>= 1)
         if (~n & 1)
            a[n >> 1]--;
   }
   int cntls(int n) {  // 统计小于n的个数
      int c = 0; // 若统计小于等于则c=a[i];
      for (n += N; n > 1; n >>= 1)
         if (n & 1)
            c += a[n >> 1];
      return c;
   }
   int cntgt(int n) {  // 统计大于n的个数
      return size - a[N + n] - cntls(n);
   }

   /*
    * 解决：求点集中第i小的数(由0数起)
    * 注意：如果i>=size 返回N-1
    */
   int operator[](int n) {
      int i = 1;
      while (i < N) {
         if (n < a[i]) i <<= 1;
         else n -= a[i], i = i << 1 | 1;
      }
      return i - N;
   }
};
pointtree<1 << 18> t;
/*==============================================================================*\
  8)   SBT
\*==============================================================================*/
// 注意没有内存回收
template<typename Type>
class sbtree { // SizeBalanceTree
public:
```

```
void clear() {
   tot = 0;
   lc[0] = rc[0] = 0;
   sz[0] = 0;
   root = 0;
}
int size() {return sz[root];}
bool empty() {return root == 0;}
void Build(int s,int e) {Build(root,s,e);}
bool Find(Type val)  {return Find(root, val);}
void Insert(Type val)   {Insert(root, val);}
void Delete(Type val)   {Delete(root, val);}
Type DeleteSelect(int k) {return DeleteSelect(root, k);}
void DeleteSmall(Type val) {DeleteSmall(root, val);}
int Rank(Type val)   {return Rank(root, val);}
Type Select(int k)   {return Select(root, k);}
Type pred(Type val)  {return pred(root, val);}
Type succ(Type val) {return succ(root, val);}
Type getMin()  {
   int temp = root;
   while (lc[temp]) temp = lc[temp];
   return key[temp];
}
Type getMax()  {
   int temp = root;
   while (rc[temp]) temp = rc[temp];
   return key[temp];
}
Type DeleteMax() {
   int temp = root;
   if(rc[root] == 0) {
      root = lc[root];
      return key[temp];
   }
   while (rc[rc[temp]]) {
      sz[temp] --;
      temp = rc[temp];
   }
   sz[temp] --;
   Type ret = key[rc[temp]];
   rc[temp] = lc[rc[temp]];
   return ret;
}
Type DeleteMin() {
   int temp = root;
   if(lc[root] == 0) {
      root = rc[root];
      return key[temp];
```

```
            }
            while (lc[lc[temp]]) {
                sz[temp] --;
                temp = lc[temp];
            }
            sz[temp] --;
            Type ret = key[lc[temp]];
            lc[temp] = rc[lc[temp]];
            return ret;
        }
    private:
        int sz[maxn];   //sz[i]表示以i为根的子树的大小
        Type key[maxn]; //存放val值 (左儿子 <= 根 <= 右儿子)
        int lc[maxn];   //左儿子
        int rc[maxn];   //右儿子
        int root , tot;
            // 建树之前先clear()
        void Build(int &root,int s,int e) {  //以[s, e]的连续整数建SBT
            if(s > e) return ;
            int mid = (s + e)/2;
            root = ++tot;
            key[root] = mid;
            lc[root] = 0;
            rc[root] = 0;
            sz[root] = e - s + 1;
            if(s == e) return ;
            Build(lc[root] , s , mid - 1);
            Build(rc[root] , mid + 1 , e);
        }
        bool Find(int &root,Type k) {
            if (root == 0) {
                return false;
            } else if (k < key[root]) {
                return Find(lc[root] , k);
            } else {
                return (key[root] == k || Find(rc[root] , k));
            }
        }
        void Insert(int &root,Type val) {
            if (root == 0) {
                root = ++tot;
                lc[root] = rc[root] = 0;
                sz[root] = 1;
                key[root] = val;
                return ;
            }
            sz[root] ++;
            if (val < key[root]) {
```

```
            Insert(lc[root] , val);
        } else {
            Insert(rc[root] , val);
        }
        maintain(root , !(val < key[root]));
    }
    Type Delete(int &root,Type val) {//删除一个val值(val值必须存在)(无内存回收)
        sz[root]--;
        if ((key[root] == val) || (val < key[root] && lc[root] == 0) ||
(key[root] < val && rc[root] == 0)) {
                Type ret = key[root];
                if ( lc[root] == 0 || rc[root] == 0 )
                    root = lc[root] + rc[root];
                else   //从左子树中取最大的节点取代当前节点
                    key[root] = Delete(lc[root] , key[root] + 1);
                return ret;
        } else {
            if ( val < key[root] )
                return Delete(lc[root] , val);
            else
                return Delete(rc[root] , val);
        }
    }
    void DeleteSmall(int &root , Type val){//删除小于val的所有数(破坏树的平衡)
        if ( root == 0 ) return ;
        if ( key[root] < val ) {
            root = rc[root];
            DeleteSmall(root , val);
        } else {
            DeleteSmall(lc[root] , val);
            sz[root] = 1 + sz[lc[root]] + sz[rc[root]];
        }
    }
    int Rank(int &root , Type val) { //查询val值rank多少(val值必须存在)
        if ( key[root] == val ) {
            return 1;
        } else if ( val < key[root] ) {
            return Rank(lc[root], val);
        } else {
            return sz[lc[root]] + 1 + Rank(rc[root] , val);
        }
    }
    Type Select(int &root , int k) { // 查询第k小元素(从1开始)
        if ( sz[lc[root]] + 1 == k ) {
            return key[root];
        } else if ( k > sz[lc[root]] ) {
            return Select(rc[root] , k - 1 - sz[lc[root]]);
        } else {
```

```
      return Select(lc[root] , k);
   }
}
Type DeleteSelect(int &root,int k) {  // 删除第k小元素
   sz[root]--;
   if ( sz[lc[root]] + 1 == k ) {
      Type ret = key[root];
      if (lc[root] == 0 || rc[root] == 0 )
         root = lc[root] + rc[root];
      else
         key[root] = Delete(lc[root] , key[root] + 1);
      return ret;
   } else if ( k > sz[lc[root]] ) {
      return DeleteSelect(rc[root] , k - 1 - sz[lc[root]]);
   } else {
      return DeleteSelect(lc[root] , k);
   }
}
Type pred(int &root , Type val) {  // 查询小于val的最大值
   if (root == 0) {
      return val;
   } else if (val > key[root]) {
      Type ret = pred(rc[root] , val);
      if(ret == val) return key[root];
      return ret;
   } else {
      return pred(lc[root] , val);
   }
}
Type succ(int &root , Type val) {  // 查询大于val的最小值
   if (root == 0) {
      return val;
   } else if (key[root] > val) {
      Type ret = succ(lc[root] , val);
      if (ret == val) return key[root];
      return ret;
   } else {
      return succ(rc[root] , val);
   }
}
void LeftRotate(int &root) {
   int temp = rc[root];
   rc[root] = lc[temp];
   lc[temp] = root;
   sz[temp] = sz[root];
   sz[root] = 1 + sz[lc[root]] + sz[rc[root]];
   root = temp;
}
```

```
   void RightRotate(int &root) {
      int temp = lc[root];
      lc[root] = rc[temp];
      rc[temp] = root;
      sz[temp] = sz[root];
      sz[root] = 1 + sz[lc[root]] + sz[rc[root]];
      root = temp;
   }
   void maintain(int &root , bool flag) {
      if (root == 0) return ;
      if ( !flag ) {  // 调整左子树
         if ( sz[lc[lc[root]]] > sz[rc[root]] ) {
            RightRotate( root );
         } else if ( sz[rc[lc[root]]] > sz[rc[root]] ) {
            LeftRotate( lc[root] );
            RightRotate( root );
         } else {
            return ;
         }
      } else {   // 调整右子树
         if ( sz[rc[rc[root]]] > sz[lc[root]] ) {
            LeftRotate( root );
         } else if ( sz[lc[rc[root]]] > sz[lc[root]] ) {
            RightRotate( rc[root] );
            LeftRotate( root );
         } else {
            return ;
         }
      }
      maintain(lc[root] , false);
      maintain(rc[root] , true);
      maintain(root , false);
      maintain(root , true);
   }
};
/*=========================================================================*\
  9)   Splay Tree
\*=========================================================================*/
/*
 * Splay Tree
 * 所处理的数组下标为1-N，为实现方便，在0和N+1的位置增加一个key为inf的结点
 * select()函数中的kth与实际下边的关系如下
 * inf - num - num - num - num - inf
 *  0     1      2     3    4     5
 * 另外用null节点替换空指针
 */
const int MAX = 200005;
#define type int
```

```
struct node {
   int size, rev;
   type key, minv, delta;
   node *ch[2], *pre;
   void add(type v) {
      if (size == 0) return;
      delta += v;
      minv += v;
      key += v;
   }
   void reverse() {
      if (size == 0) return;
      rev ^= 1;
      swap(ch[0], ch[1]);
      /* 逆序后会变化的值注意修改(比如左右连续长度) */
   }
   void update() {
      size = ch[0]->size + ch[1]->size + 1;
      minv = min(key, min(ch[0]->minv, ch[1]->minv));
   }
   void pushdown() {
      if (delta) {
         ch[0]->add(delta);
         ch[1]->add(delta);
      }
      if (rev) {
         ch[0]->reverse();
         ch[1]->reverse();
      }
      delta = rev = 0;
   }
};

type arr[MAX];
node* Hash[MAX]; // Hash[i]指向key = i的节点，方便查找其位置(key值唯一)

#define keytree root->ch[1]->ch[0]
class Splay {
   int cnt, top;
   node *stk[MAX], data[MAX];
public:
   node *root, *null;

   /*
    * 获得一个新的节点，之前删除的节点会放到stk中以便再利用
    */
   node *Newnode(type var) {
      node *p;
      if (top) p = stk[top--];
      else p = &data[cnt++];
      p->key = p->minv = var;
      p->size = 1;
      p->delta = p->rev = 0;
      p->ch[0] = p->ch[1] = p->pre = null;
      return p;
   }

   void init() {
      top = cnt = 0;
      null = Newnode(inf);
      null->size = 0;
      root = Newnode(inf);
      root->ch[1] = Newnode(inf);
      root->ch[1]->pre = root;
      root->update();
   }

   /*
    * 用arr数组中[l,r]区间内的值建树
    */
   void maketree(int l, int r) {
      init();
      keytree = build(l, r);
      keytree->pre = root->ch[1];
      splay(keytree, null);
   }
   node *build(int l, int r) {
      if (l > r) return null;
      int mid = (l + r) >> 1;
      node *p = Newnode(arr[mid]);
      Hash[arr[mid]] = p;
      p->ch[0] = build(l, mid - 1);
      p->ch[1] = build(mid + 1, r);
      if (p->ch[0] != null)
         p->ch[0]->pre = p;
      if (p->ch[1] != null)
         p->ch[1]->pre = p;
      p->update();
      return p;
   }

   /*
    * 旋转操作, c=0 表示左旋, c=1 表示右旋
    */
   void rotate(node *x, int c) {
      node *y = x->pre;
      y->pushdown();
      x->pushdown();
      y->ch[!c] = x->ch[c];
      if (x->ch[c] != null)
         x->ch[c]->pre = y;
      x->pre = y->pre;
      if (y->pre != null)
         y->pre->ch[ y == y->pre->ch[1] ] = x;
      x->ch[c] = y;
      y->pre = x;
      y->update();
      if (y == root) root = x;
   }

   /*
    * 旋转使x成为f的子节点，若f为null则x旋转为根节点
    * x会执行pushdown和update的操作
    */
   void splay(node *x, node *f) {
      x->pushdown();
      while (x->pre != f) {
         if (x->pre->pre == f) {
            rotate(x, x->pre->ch[0] == x);
            break;
         }
         node *y = x->pre;
         node *z = y->pre;
         int c = (y == z->ch[0]);
         if (x == y->ch[c]) {
            rotate(x, !c); rotate(x, c); // 之字形旋转
         } else {
            rotate(y, c); rotate(x, c);  // 一字形旋转
         }
      }
      x->update();
   }

   /*
    * 找到位置为k的节点，并将其升至x的儿子
    * k节点会执行pushdown和update的操作
    */
   void select(int kth, node *x) {
      node *cur = root;
      while (true) {
         cur->pushdown();
         int tmp = cur->ch[0]->size;
         if (tmp == kth) break;
         else if (tmp < kth) {
               kth -= tmp + 1;
               cur = cur->ch[1];
           } else {
               cur = cur->ch[0];
           }
       }
       splay(cur, x);
   }

   /*
    * 在X位置后插入值为Y的节点。
    * "insert(2,4)" on {1, 2, 3, 4, 5} results in {1, 2, 4, 3, 4, 5}
    * 做法:将X位置的节点a升至根节点，再将X+1位置的节点b升至a的右儿子
    * 此时b的左儿子一定为空， 将新插入的节点作为b的左儿子。
    */
   void insert(int x, type y) {
       select(x, null);
       select(x + 1, root);
       keytree = Newnode(y);
       keytree->pre = root->ch[1];
       root->ch[1]->update();
       splay(keytree, null);
   }
   /*
    * 在x位置后插入arr数组中[l,r]区间内的数
    */
   void insert(int x, int l, int r) {
       select(x, null);
       select(x + 1, root);
       keytree = build(l, r);
       keytree->pre = root->ch[1];
       root->ch[1]->update();
       splay(keytree, null);
   }

   /*
    * 回收x为根的子树
    */
   void erase(node *x) {
       if (x == null) return;
       erase(x->ch[0]);
       erase(x->ch[1]);
       stk[++top] = x;
   }
   /*
    * 删除区间[x, y]范围的数
    */
   void dele(int x, int y) {
       select(x - 1, null);
       select(y + 1, root);
       erase(keytree);
       keytree = null;
       root->ch[1]->update();
       root->update();
   }
   /*
    * 删除x位置的数。
    * "DELETE(2)" on {1, 2, 3, 4, 5} results in {1, 3, 4, 5}
    * 做法：找到并将其升至根节点，调用deleroot();
    */
   void dele(int x) {
       select(x, null);
       deleroot();
   }
   /*
    * 删除某节点
    * 做法：将其升至根节点，调用deleroot()
    */
   void dele(node* t) {
       splay(t, null);
       deleroot();
   }
   /*
    * 删除根节点，以其右子树的最左边节点替换之
    */
   void deleroot() {
       node *oldRoot = root;
       root = root->ch[1];
       root->pre = null;
       select(0, null);
       root->ch[0] = oldRoot->ch[0];
       root->ch[0]->pre = root;
       root->update();
       stk[++top] = oldRoot;
   }


   /*
    * 区间增加key值 "add(2, 4, 1)" on {1, 2, 3, 4, 5} results in {1, 3, 4, 5, 5}
    */
   void add(int x, int y, type d) {
       select(x - 1, null);
       select(y + 1, root);
       keytree->add(d);
       splay(keytree, null);
   }
```

```
/*
 * 区间翻转 "reverse(2,4)" on {1, 2, 3, 4, 5} results in {1, 4, 3, 2, 5}
 */
void reverse(int x, int y) {
    select(x - 1, null);
    select(y + 1, root);
    keytree->reverse();
}
/*
 * 区间[x, y]循环右移d，实质是交换区间[a, b]和[b + 1, c]，其中b = y-d%(y-x+1)
 * "revolve(2, 4, 2)" on {1, 2, 3, 4, 5} results in {1, 3, 4, 2, 5}
 * 做法：将b+1位置的节点x升至根节点，将c+1位置的节点y升至x的右儿子，将c位置的节点z升至y的左儿子
 * 将a-1位置的节点v升至x的左儿子，此时v的右儿子即是[a, b]，将其赋给z的右儿子。
 * 当d = 1时，节点x与节点z是同一节点，特殊处理。
 */
void revolve(int x, int y, int d) {
    int len = y - x + 1;
    d = (d % len + len) % len;
    if (d == 0) return;
    if (d == 1) {
        dele(y);
        insert(x - 1, stk[top]->key);
    } else {
        select(y - d + 1, null);
        select(y + 1, root);
        select(x - 1, root);
        select(y, root->ch[1]);
        node *p = root->ch[0]->ch[1];
        root->ch[0]->ch[1] = null;
        root->ch[0]->update();
        root->ch[1]->ch[0]->ch[1] = p;
        p->pre = root->ch[1]->ch[0];
        splay(p, null);
    }
}

/*
 * 求区间最小值
 * "MIN(2,4)" on {1, 2, 3, 4, 5} is 2
 * 做法：找到X-1位置上的节点a并将其升至根节点，再找到Y+1位置上的
 * 的节点b并将其作为a的右儿子。则b的左儿子即所求区间。
 */
type getMin(int x, int y) {
    select(x - 1, null);
    select(y + 1, root);
    return keytree->minv;
}
```

```
   /*
    * 查找key = i的节点的位置
    * 做法：将该节点升至根节点，统计其左儿子个数
    * 注意：由于首位插入了一个无用节点，所以原序列的第i个数有i个左儿子
    */
   int query(type i) {
      splay(Hash[i], null);
      int ans = root->ch[0]->size;
   // reverse(1, ans);
   // dele(1);
      return ans;
   }

   void debug() {vis(root);}
   void vis(node* t) {
      if (t == null) return;
      vis(t->ch[0]);
      printf("node%2d:lson %2d,rson %2d,pre %2d,sz=%2d,key=%2d\n",
            t - data, t->ch[0] - data, t->ch[1] - data,
            t->pre - data, t->size, t->key);
      vis(t->ch[1]);
   }
} spt;
/*===========================================================================*\
  10)  动态树(边权)
\*===========================================================================*/
const int N = 10010;
#define type int
struct node {
   int size;
   bool isroot;
   type val, maxv;
   node *ch[2], *pre;
   void init(type v, node *null) {
      val = maxv = v;
      ch[0] = ch[1] = pre = null;
      size = isroot = 1;
   }
   void pushdown() {}
   void update() {
      size = ch[0]->size + ch[1]->size + 1;
      maxv = max(val, max(ch[0]->maxv, ch[1]->maxv));
   }
};
int n;
type cost[N * 2];
int head[N], que[N], id[N], e;
int ev[N * 2], nxt[N * 2];
```

```
void addedge(int u, int v, type c) {
   ev[e] = v; cost[e] = c; nxt[e] = head[u]; head[u] = e++;
   ev[e] = u; cost[e] = c; nxt[e] = head[v]; head[v] = e++;
}
struct LinkCutTree {
   node data[N], *null;
   void maketree() {
      null = data;
      null->init(-inf, null);
      null->size = 0;
      int bg = 0, ed = 0;
      que[ed++] = 1;
      data[1].init(-inf, null);
      while (bg < ed) {
         int u = que[bg++];
         for (int i = head[u]; ~i; i = nxt[i]) {
            int v = ev[i];
            if (data[u].pre == data + v) continue;
            que[ed++] = v;
            data[v].init(cost[i], null);
            data[v].pre = data + u;
            id[i / 2 + 1] = v;  // 记录边的权值在哪个节点上，下标均为从1开始
         }
      }
   }
   void rotate(node *x, int c) {
      node *y = x->pre;
      y->pushdown();
      x->pushdown();
      y->ch[!c] = x->ch[c];
      if (x->ch[c] != null)
         x->ch[c]->pre = y;
      x->pre = y->pre;
      if (y->isroot) y->isroot = 0, x->isroot = 1;
      else y->pre->ch[ y == y->pre->ch[1] ] = x;
      x->ch[c] = y;
      y->pre = x;
      y->update();
   }
   void splay(node *x) {
      x->pushdown();
      while (!x->isroot) {
         if (x->pre->isroot) {
            rotate(x, x->pre->ch[0] == x);
            break;
         }
         node *y = x->pre;
         node *z = y->pre;
         int c = (y == z->ch[0]);
         if (x == y->ch[c]) {
            rotate(x, !c); rotate(x, c);
         } else {
            rotate(y, c); rotate(x, c);
         }
      }
      x->update();
   }

   node* access(node *x) {
      node *y;
      for (y = null; x != null; y = x, x = x->pre) {
         splay(x);   x->ch[1]->isroot = 1;
         x->ch[1] = y;  y->isroot = 0;
         x->update();
      }
      return y;
   }
   node* findroot(node *x) {
      x = access(x);
      while (1) {
         x->pushdown();
         if (x->ch[0] == null) return x;
         else x = x->ch[0];
      }
   }

   void modify(int u, type val) {
      splay(data + u);
      data[u].val = val;
      data[u].update();
   }
   type qmax(int u, int v) {
      access(data + u);
      node *x = data + v;
      type ans;
      for (node *y = null; x != null; y = x, x = x->pre) {
         splay(x);
         if (x->pre == null)
            ans = max(x->ch[1]->maxv, y->maxv);
         x->ch[1]->isroot = 1;
         x->ch[1] = y;  y->isroot = 0;
         x->update();
      }
      return ans;
   }
```

```
   void debug() {
      for (int i = 1; i <= n; ++i) {
         printf("node%2d:ls%2d,rs%2d,pre%2d,val=%2d,max=%2d\n",
               i, data[i].ch[0] - data, data[i].ch[1] - data,
               data[i].pre - data, data[i].val, data[i].maxv);
      }
   }
}lct;
/*==========================================================================*\
  11)  动态树(点权)
\*==========================================================================*/
const int N = 30010;
#define type int

int n, q;
int head[N], que[N], e;
int ev[N * 2], nxt[N * 2];
type val[N]

void addedge(int u, int v) {
   ev[e] = v; nxt[e] = head[u]; head[u] = e++;
   ev[e] = u; nxt[e] = head[v]; head[v] = e++;
}

struct node {
   int size;
   bool rev, isroot;
   type val, maxv, sum;
   node *ch[2], *pre;
   void init(type v, node *null) {
      val = maxv = sum = v;
      ch[0] = ch[1] = pre = null;
      rev = 0;
      size = isroot = 1;
   }
   void reverse() {
      if (size == 0) return;
      swap(ch[0], ch[1]);
      rev ^= 1;
   }
   void pushdown() {
      if (rev) {
         ch[0]->reverse();
         ch[1]->reverse();
      }
      rev = 0;
   }
   void update() {
      size = ch[0]->size + ch[1]->size + 1;
      sum = ch[0]->sum + ch[1]->sum + val;
      maxv = max(val, max(ch[0]->maxv, ch[1]->maxv));
   }
};

struct LinkCutTree {
   node data[N], *null;
   void maketree() {
      null = data;
      null->init(-inf, null);
      null->size = null->sum = 0;
      int bg = 0, ed = 0;
      que[ed++] = 1;
      data[1].init(val[1], null);
      while (bg < ed) {
         int u = que[bg++];
         for (int i = head[u]; ~i; i = nxt[i]) {
            int v = ev[i];
            if (data[u].pre == data + v) continue;
            que[ed++] = v;
            data[v].init(val[v], null);
            data[v].pre = data + u;
         }
      }
   }
   void rotate(node *x, int c) {
      node *y = x->pre;
      y->pushdown();
      x->pushdown();
      y->ch[!c] = x->ch[c];
      if (x->ch[c] != null)
         x->ch[c]->pre = y;
      x->pre = y->pre;
      if (y->isroot) y->isroot = 0, x->isroot = 1;
      else y->pre->ch[ y == y->pre->ch[1] ] = x;
      x->ch[c] = y;
      y->pre = x;
      y->update();
   }
   void splay(node *x) {
      x->pushdown();
      while (!x->isroot) {
         if (x->pre->isroot) {
            rotate(x, x->pre->ch[0] == x);
            break;
         }
         node *y = x->pre;
         node *z = y->pre;
         int c = (y == z->ch[0]);
         if (x == y->ch[c]) {
            rotate(x, !c); rotate(x, c);
         } else {
            rotate(y, c); rotate(x, c);
         }
      }
      x->update();
   }

   node* access(node *x) {
      node *y;
      for (y = null; x != null; y = x, x = x->pre) {
         splay(x);  x->ch[1]->isroot = 1;
         x->ch[1] = y;  y->isroot = 0;
         x->update();
      }
      return y;
   }
   node* findroot(node *x) {
      x = access(x);
      while (1) {
         x->pushdown();
         if (x->ch[0] == null) return x;
         else x = x->ch[0];
      }
   }

   /*
    * 把x为根的子树与原来节点断开，移动到y节点下(必须合法)
    */
   void move(int x, int y) {
      access(data + x);
      splay(data + x);
      data[x].ch[0]->pre = null;
      data[x].ch[0]->isroot = 1;
      data[x].ch[0] = null;
      data[x].pre = data + y;
      access(data + x);
   }

   /*
    * 判断x是否在y到根节点的路径上，包括x==y的情况
    */
   bool isfather(int x, int y) {
      if (findroot(data + x) !=
         findroot(data + y)) return false;
      access(data + y);
      splay(data + x);
      return data[x].pre == null;
   }

   void evert(node *x) { // 将x节点置为根,权值在边上时不能进行
      x = access(x);
      x->reverse();
   }
   void modify(int u, type val) {
      splay(data + u);
      data[u].val = val;
      data[u].update();
   }
   type qmax(int u, int v) {
      evert(data + u);
      return access(data + v)->maxv;
   }
   type qsum(int u, int v) {
      evert(data + u);
      return access(data + v)->sum;
   }

   void debug() {
      bool flag = true;
      while (flag) {
         flag = false;
         for (int i = 1; i <= n; ++i)
            if (data[i].rev) {
               data[i].pushdown();
               flag = true;
            }
      }
      for (int i = 1; i <= n; ++i) {
         printf("node%2d:ls%2d,rs%2d,pre%2d,val=%2d,max=%2d\n",
               i, data[i].ch[0] - data, data[i].ch[1] - data,
               data[i].pre - data, data[i].val, data[i].maxv);
      }
   }
}lct;
/*==================================================================================*\
  12) 块状链表
\*==================================================================================*/
const int TOT = 2 * 1024 * 1024 + 10;
const int SIZE = 3000; // size of each block
const int MAX = TOT / SIZE * 2 + 100;

#define type int
```

```
struct BlockList {
   type data[MAX][SIZE];
   int cnt[MAX];
   int next[MAX];
   int free[MAX];
   int top;

   void init() {
      for (int i = 1; i < MAX; ++i)
         free[i] = i;
      top = 1;
      next[0] = -1;
      cnt[0] = 0;
   }
   int newnode(int n, type from[], int nxt) {
      int b = free[top++];
      next[b] = nxt;
      cnt[b] = n;
      memcpy(data[b], from, sizeof(type) * n);
      return b;
   }
   void delnode(int t) {
      free[--top] = t;
   }
   /*
    * 找到p所在的块b，p置为在b中的相对位置，下标均从0开始
    * p == cnt[b]时不会跳到下一块，而是位于此块的最后一个的后面
    */
   void find(int &p, int &b) {
      for (b = 0; b != -1 && p > cnt[b]; b = next[b])
         p -= cnt[b];
   }

   /*
    * b block : [0,end] -> [0,p-1] & [p,end]
    */
   void splite(int b, int p) {
      if (b == -1 || p == cnt[b]) return;
      int t = newnode(cnt[b] - p, data[b] + p, next[b]);
      next[b] = t;
      cnt[b] = p;
   }
   void maintain(int b) {
      for (; b != -1; b = next[b])
         for (int t = next[b]; t != -1 && cnt[b] + cnt[t] <= SIZE;
               t = next[b]) {
            memcpy(data[b] + cnt[b], data[t], sizeof(type) * cnt[t]);
            cnt[b] += cnt[t];
            next[b] = next[t];
            delnode(t);
         }
   }
   /*
    * 在p位置上插入n个数，该位置上的数向后挪，下标从0开始
    */
   void insert(int p, int n, type from[]) {
      int b, t, i;
      find(p, b);
      splite(b, p);
      for (i = 0; i + SIZE <= n; i += SIZE) {
         t = newnode(SIZE, from + i, next[b]);
         next[b] = t;
         b = t;
      }
      if (n - i) {
         t = newnode(n - i, from + i, next[b]);
         next[b] = t;
      }
      maintain(b);
   }
   void erase(int p, int n) {
      int b, e;
      find(p, b);
      splite(b, p);
      for (e = next[b]; e != -1 && n > cnt[e]; e = next[e])
         n -= cnt[e];
      splite(e, n);
      e = next[e];
      for (int t = next[b]; t != e; t = next[b]) {
         next[b] = next[t];
         delnode(t);
      }
      maintain(b);
   }
   void get(int p, int n, type to[]) {
      int b, t, i;
      find(p, b);
      i = min(n, cnt[b] - p);
      memcpy(to, data[b] + p, sizeof(type) * i);
      for (t = next[b]; t != -1 && i + cnt[t] <= n;
            i += cnt[t], t = next[t])
         memcpy(to + i, data[t], sizeof(type) * cnt[t]);
      if (n - i && t != -1)
         memcpy(to + i, data[t], sizeof(type) * (n - i));
   }
}lst;
```

```
/*==============================================================================*\
 13)  左偏树
\*==============================================================================*/
const int N = 100010;
const int na = 0;
#define typec int // type of key val
struct node {
   typec key;
   int l, r, f, dist;
} tr[N];
int iroot(int i) { // find i's root
   if (i == na) return i;
   while (tr[i].f != na) i = tr[i].f;
   return i;
}
int merge(int rx, int ry) { // two root: rx, ry
   if (rx == na) return ry;
   if (ry == na) return rx;
   if (tr[rx].key > tr[ry].key) swap(rx, ry);  //最小堆变最大堆只要修改这里
   int r = merge(tr[rx].r, ry);
   tr[rx].r = r; tr[r].f = rx;
   if (tr[r].dist > tr[tr[rx].l].dist)
      swap(tr[rx].l, tr[rx].r);
   if (tr[rx].r == na) tr[rx].dist = 0;
   else tr[rx].dist = tr[tr[rx].r].dist + 1;
   return rx; // return new root
}
int ins(int i, typec key, int root) { // add a new node(i, key)
   tr[i].key = key;
   tr[i].l = tr[i].r = tr[i].f = na;
   tr[i].dist = 0;
   return root = merge(root, i); // return new root
}
int del(int i) { // delete node i
   if (i == na) return i;
   int x, y, l, r;
   l = tr[i].l;
   r = tr[i].r;
   y = tr[i].f;
   tr[i].l = tr[i].r = tr[i].f = na;
   tr[x = merge(l, r)].f = y;
   if (y != na && tr[y].l == i) tr[y].l = x;
   if (y != na && tr[y].r == i) tr[y].r = x;
   for (; y != na; x = y, y = tr[y].f) {
      if (tr[tr[y].l].dist < tr[tr[y].r].dist)
         swap(tr[y].l, tr[y].r);
      if (tr[tr[y].r].dist + 1 == tr[y].dist) break;
      tr[y].dist = tr[tr[y].r].dist + 1;
   }
   if (x != na) return iroot(x); // return new root
   else return iroot(y);
}
node top(int root) {
   return tr[root];
}
node pop(int &root) {
   node out = tr[root];
   int l = tr[root].l, r = tr[root].r;
   tr[root].l = tr[root].r = tr[root].f = na;
   tr[l].f = tr[r].f = na;
   root = merge(l, r);
   return out;
}
int change(int i, typec val) { // tr[i].key = val
   if (i == na) return i;
   if (tr[i].l == na && tr[i].r == na && tr[i].f == na) {
      tr[i].key = val;
      return i;
   }
   int rt = del(i);
   return ins(i, val, rt);
}
void init(int n) {
   tr[na].l = tr[na].r = tr[na].f = na;
   tr[na].dist = -1;
   for (int i = 1; i <= n; i++) {
      scanf("%d", &tr[i].key); //%d: type of key
      tr[i].l = tr[i].r = tr[i].f = na;
      tr[i].dist = 0;
   }
}
/*==================================================================================*\
  14)  划分树
\*==================================================================================*/
#define L(x) (x << 1)
#define R(x) (x << 1 | 1)
const int N = 100010;
int n;

struct node {
   int lft, rht;
   int getmid() { return (lft + rht) >> 1; }
}tree[N * 4]; // 注意"*4"
int val[20][N], sorted[N];
int toleft[20][N];
```

```
void build(int root, int lft, int rht, int d) {
   tree[root].lft = lft;
   tree[root].rht = rht;
   if (lft == rht) return;
   int mid = (lft + rht) >> 1;
   int same = mid - lft + 1; //same表示和val_mid相等且分到左边的数目
   for (int i = lft; i <= rht; ++i)
      if (val[d][i] < sorted[mid])
         same--;
   int lpos = lft;
   int rpos = mid + 1;
   for (int i = lft; i <= rht; ++i) {
      if (i == lft) toleft[d][i] = 0;
      else toleft[d][i] = toleft[d][i - 1];

      if (val[d][i] < sorted[mid]) {
         toleft[d][i]++;
         val[d + 1][lpos++] = val[d][i];
      } else if (val[d][i] > sorted[mid]) {
         val[d + 1][rpos++] = val[d][i];
      } else if (same) {
         same--;
         toleft[d][i]++;
         val[d + 1][lpos++] = val[d][i];
      } else {
         val[d + 1][rpos++] = val[d][i];
      }
   }
   build(L(root), lft, mid, d + 1);
   build(R(root), mid + 1, rht, d + 1);
}

int query(int root, int lft, int rht, int d, int k) {
   if (lft == rht) return val[d][lft];
   int s;  //s表示[lft , rht]有多少个分到左边
   int ss; //ss表示[tree[root].lft , lft - 1]有多少个分到左边
   if (lft == tree[root].lft) {
      s = toleft[d][rht];
      ss = 0;
   } else {
      s = toleft[d][rht] - toleft[d][lft - 1];
      ss = toleft[d][lft - 1];
   }
   if (k <= s) {  //有多于k个分到左边,显然去左儿子区间找第k个
      int left = tree[root].lft + ss;
      int right = left + s - 1;
      return query(L(root), left, right, d + 1, k);
   } else {
        int b = lft - tree[root].lft - ss;
                // b表示[tree[root].lft , lft - 1]有多少个分到右边
        int left = tree[root].getmid() + b + 1;
        int right = left + rht - lft - s;
        return query(R(root), left, right, d + 1, k - s);
    }
}
/*==================================================================*\
 15)  矩形面积并
\*==================================================================*/
#define L(x) (x << 1)
#define R(x) (x << 1 | 1)
const int N = 100010 * 2;
const double eps = 1e-8;
//typedef double type;
typedef ll type;
type bin[N];

struct line {
    type x, y0, y1;
    int d;
    line(){}
    line(type x, type y0, type y1, int d) :
        x(x), y0(y0), y1(y1), d(d) {}
    bool operator < (const line& u) const {
//      if (fabs(x - u.x) > eps) return x < u.x;
        if (x != u.x) return x < u.x;
        return d > u.d;
    }
}lin[N];

struct node {
    int lft, rht, c; // c 为区间被覆盖的层数
    type m;    // 区间被覆盖长度
    int getmid() { return (lft + rht) >> 1; }
    void update(int);
}tree[N * 4];

void node::update(int root) {
    if (c) m = bin[rht] - bin[lft];
    else if (lft + 1 == rht) m = 0;
    else m = tree[L(root)].m + tree[R(root)].m;
}


void build(int root, int lft, int rht) {
    tree[root].lft = lft;
    tree[root].rht = rht;
```

```
   tree[root].c = 0;
   tree[root].m = 0;
   if (lft + 1 == rht) return;
   int mid = (lft + rht) >> 1;
   build(L(root), lft, mid);
   build(R(root), mid, rht);
}

void insert(int root, int lft, int rht, int d) {
   if (lft <= tree[root].lft && tree[root].rht <= rht) {
      tree[root].c += d;
      tree[root].update(root);
      return;
   }
   int mid = tree[root].getmid();
   if (lft < mid) insert(L(root), lft, rht, d);
   if (mid < rht) insert(R(root), lft, rht, d);
   tree[root].update(root);
}

bool Equal(double a, double b) {return fabs(a - b) < eps;}

void solve() {
   type x0, x1, y0, y1;
   int n, tot = 0;
   scanf("%d", &n);
   forn (i, n) {
//    scanf("%lf%lf%lf%lf", &x0, &y0, &x1, &y1);
      scanf("%lld%lld%lld%lld", &x0, &y0, &x1, &y1);
      bin[tot] = y0;  lin[tot++] = line(x0, y0, y1, 1);
      bin[tot] = y1;  lin[tot++] = line(x1, y0, y1, -1);
   }
   sort(lin, lin + tot);
   sort(bin, bin + tot);
// tot = unique(bin, bin + tot, Equal) - bin;
   tot = unique(bin, bin + tot) - bin;
   build(1, 0, tot - 1); // 注意tot不能等于0
   type ans = 0;
   forn (i, n + n) {
//    int lft = lower_bound(bin, bin + tot, lin[i].y0 - eps) - bin;
//    int rht = lower_bound(bin, bin + tot, lin[i].y1 - eps) - bin;
      int lft = lower_bound(bin, bin + tot, lin[i].y0) - bin;
      int rht = lower_bound(bin, bin + tot, lin[i].y1) - bin;
      if (lft != rht) insert(1, lft, rht, lin[i].d);
      ans += tree[1].m * (lin[i + 1].x - lin[i].x);
   }
   printf("%lld\n", ans);
}
```

```
/*==========================================================================*\
  16)  矩形周长并
\*==========================================================================*/
#define L(x) (x << 1)
#define R(x) (x << 1 | 1)
const int N = 100010 * 2;
const double eps = 1e-8;
//typedef double type;
typedef int type;
type bin[N];

struct line {
   type x, y0, y1;
   int d;
   line(){}
   line(type x, type y0, type y1, int d) :
      x(x), y0(y0), y1(y1), d(d) {}
   bool operator < (const line& u) const {
//    if (fabs(x - u.x) > eps) return x < u.x;
      if (x != u.x) return x < u.x;
      return d > u.d;
   }
}lin[N];

struct node {
   int lft, rht, c; // c 为区间被覆盖的层数
   int cnt, lbd, rbd; // lbd和rbd表示边界，cnt表示需要统计几根线段
   type m;    // 区间被覆盖长度
   void init() {
      cnt = lbd = rbd = c = 0;
      m = 0;
   }
   int getmid() { return (lft + rht) >> 1; }
   void update(int);
}tree[N * 4];

void node::update(int root) {
   if (c) {
      m = bin[rht] - bin[lft];
      lbd = rbd = cnt = 1;
   } else if (lft + 1 == rht) {
      init();
   } else {
      m = tree[L(root)].m + tree[R(root)].m;
      cnt = tree[L(root)].cnt + tree[R(root)].cnt
            - (tree[L(root)].rbd & tree[R(root)].lbd);
      lbd = tree[L(root)].lbd;
      rbd = tree[R(root)].rbd;
   }
}

void build(int root, int lft, int rht) {
   tree[root].lft = lft;
   tree[root].rht = rht;
   tree[root].init();
   if (lft + 1 == rht) return;
   int mid = (lft + rht) >> 1;
   build(L(root), lft, mid);
   build(R(root), mid, rht);
}

void insert(int root, int lft, int rht, int d) {
   if (lft <= tree[root].lft && tree[root].rht <= rht) {
      tree[root].c += d;
      tree[root].update(root);
      return;
   }
   int mid = tree[root].getmid();
   if (lft < mid) insert(L(root), lft, rht, d);
   if (mid < rht) insert(R(root), lft, rht, d);
   tree[root].update(root);
}

bool Equal(double a, double b) {return fabs(a - b) < eps;}

void solve() {
   type x0, x1, y0, y1;
   int n, tot = 0;
   scanf("%d", &n);
   forn (i, n) {
//    scanf("%lf%lf%lf%lf", &x0, &y0, &x1, &y1);
      scanf("%d%d%d%d", &x0, &y0, &x1, &y1);
      bin[tot] = y0;  lin[tot++] = line(x0, y0, y1, 1);
      bin[tot] = y1;  lin[tot++] = line(x1, y0, y1, -1);
   }
   sort(lin, lin + tot);
   sort(bin, bin + tot);
// tot = unique(bin, bin + tot, Equal) - bin;
   tot = unique(bin, bin + tot) - bin;
   build(1, 0, tot - 1); // 注意tot不能等于0
   type ans = 0, len = 0;
   forn (i, n + n) {
//    int lft = lower_bound(bin, bin + tot, lin[i].y0 - eps) - bin;
//    int rht = lower_bound(bin, bin + tot, lin[i].y1 - eps) - bin;
      int lft = lower_bound(bin, bin + tot, lin[i].y0) - bin;
      int rht = lower_bound(bin, bin + tot, lin[i].y1) - bin;
      if (lft != rht) insert(1, lft, rht, lin[i].d);
      ans += tree[1].cnt * (lin[i + 1].x - lin[i].x) * 2;
//    ans += fabs(tree[1].m - len);
      ans += abs(tree[1].m - len);
      len = tree[1].m;
   }
   printf("%d\n", ans);
}
```

3. 搜索 & 动态规划

```
/*==============================================================================*\
 1)  插头 DP(括号匹配)
\*==============================================================================*/
//这份是简单路径(4进制) 的插头DP
const int N = 10;
int n, m, maze[N][N];

const int H = 40007;
struct Hash {
   int head[H], nxt[H], size;
   int dp[H], msk[H];
   void clear() {
      size = 0; clr(head, -1);
   }
   void push(int m, int val) {
      int x = m % H;
      for (int i = head[x]; ~i; i = nxt[i]) {
         if (msk[i] == m) {
            dp[i] = max(dp[i], val);
            return;
         }
      }
      dp[size] = val;
      msk[size] = m;
      nxt[size] = head[x];
      head[x] = size++;
   }
}hp[2], *scr, *des;

int getlft(int msk, int pos) {
   int cnt = 1;
   for (int i = pos - 1; i >= 0; --i) {
      int t = (msk >> (i * 2)) & 3;
      if (t == 1) cnt--;
      if (t == 2) cnt++;
      if (cnt == 0) return 1 << (i * 2);
   } return 0;
}
int getrht(int msk, int pos) {
   int cnt = 1;
   for (int i = pos + 1; i < 32; ++i) {
      int t = (msk >> (i * 2)) & 3;
      if (t == 1) cnt++;
      if (t == 2) cnt--;
      if (cnt == 0) return 1 << (i * 2);
   } return 0;
```

```
}
int cntthree(int msk) {
   int cnt = 0;
   while (msk) {
      if ((msk & 3) == 3) cnt++;
      msk >>= 2;
   }
   return cnt;
}

int plugDP() {
   scr = hp; des = hp + 1;
   int ans = 0;
   scr->clear();
   scr->push(0, 0);
   forn (i, n) forn (j, m) {
      des->clear();
      forn (k, scr->size) {
         int msk = scr->msk[k];
         int val = scr->dp[k] + maze[i][j];

//       printf("dp[%d][%d][", i, j);
//       forn (t, m + 1) printf("%d", (msk >> (t * 2)) & 3);
//       printf("] = %d\n", val - maze[i][j]);

         int lft = (msk >> (j * 2)) & 3;
         int up = (msk >> (j * 2 + 2)) & 3;
         int now = msk & ~(15 << (j * 2));
         if (maze[i][j] == 0) {
            if (lft == 0 && up == 0)
               des->push(now, val);
         } else if (lft == 0 && up == 0) {
            des->push(now, val - maze[i][j]);
            des->push(now | 9 << (j * 2), val);
            if (cntthree(msk) > 1) continue;
            des->push(now | 3 << (j * 2), val);
            des->push(now | 3 << (j * 2 + 2), val);
         } else if (lft == 0 || up == 0) {
            des->push(now | (lft + up) << (j * 2), val);
            des->push(now | (lft + up) << (j * 2 + 2), val);
            if (cntthree(msk) > 1) continue;
            if (lft + up == 3) {
               if (now == 0) ans = max(ans, val);
            } else {
               int pos = lft == 0 ? j + 1 : j;
               if (lft + up == 1) now += getrht(msk, pos);
               if (lft + up == 2) now += getlft(msk, pos) * 2;
               des->push(now, val);
            }
         } else if (lft == 3 && up == 3) {
            if (now == 0) ans = max(ans, val);
         } else if (lft == 3 || up == 3) {
            int pos = lft == 3 ? j + 1 : j;
            if (lft + up == 4) now += getrht(msk, pos);
            if (lft + up == 5) now += getlft(msk, pos) * 2;
            des->push(now, val);
         } else if (lft == up) {
            if (lft == 1) now -= getrht(msk, j + 1);
            if (lft == 2) now += getlft(msk, j);
            des->push(now, val);
         } else if (lft == 2 && up == 1) {
            des->push(now, val);
         }
      }
      swap(scr, des);
      if (j == m - 1) {
         des->clear();
         forn (k, scr->size) if (scr->msk[k] < 1 << (m * 2))
            des->push(scr->msk[k] << 2, scr->dp[k]);
         swap(scr, des);
      }
   }
   return ans;
}

void solve() {
   scanf("%d%d", &n, &m);
   int ans = 0;
   forn (i, n) forn (j, m) {
      scanf("%d", &maze[i][j]);
      ans = max(ans, maze[i][j]);
   }
   printf("%d\n", max(ans, plugDP()));
}

int main() {
   int cas;
   scanf("%d", &cas);
   forn (i, cas) {
      solve();
   }
   return 0;
 }
```

```
/*============================================================================*\
  2)    插头 DP(最小表示法)
\*============================================================================*/
/*
题目: Black and White (UVA 10572)
题目内容或思路:   插头dp(广义路径) (陈丹琪论文题)
    转移的时候,每个格子有一到两个填色方案
    以黑色为例转移如下:
不合法情况: (情况1)If 最后一格 && 左白 && 上白 && 左上黑   Then 照成不连通,非法
      (情况2)If 左黑 && 上黑 && 左上黑        Then 形成2x2的格子,非法
      (情况3)If 上白 && 轮廓线上没有和上边相连的格子
            If 轮廓线上有白色的格子     Then 照成不连通, 非法 (情况3.1)
            If 当前格不是最后两格       Then 非法           (情况3.2)
           (因为如果剩下格子有白色,照成不连通 剩下格子全黑色,必然有2x2的黑色格子)
    If 左黑 && 上黑        Then 合并两个连通块
    Else If 左白 && 上白      Then 形成新的连通块
    Else If 左黑 && 上白      Then 和左边的合并
    Else If 左白 && 上黑      Then 和上边的合并
    最后对所有状态判断一下最多只能存在两个连通块
*/
const int N = 10;
int n, m, code[N], bin[N];
char maze[N][N];
typedef int type;

const int H = 17009;
struct Hash {
   int head[H], nxt[H], size;
   int msk[H], col[H];
   type dp[H];
   void clear() { size = 0; clr(head, -1); }
   int push(int xmsk, int xcol, type val) {
      int x = xmsk % H;
      for (int i = head[x]; ~i; i = nxt[i]) {
         if (msk[i] == xmsk && col[i] == xcol) {
            dp[i] += val;
            return i;
         }
      }
      msk[size] = xmsk;
      col[size] = xcol;
      dp[size] = val;
      nxt[size] = head[x];
      return head[x] = size++;
   }
}hp[2], *scr, *des;
int pre[66][H];
```

```
void decode(int msk, int code[]) {
   forn (i, m) code[i] = (msk >> (i * 3)) & 7;
}

int encode(int code[]) {
   int msk = 0, cnt = 0; clr(bin, -1);
   forn (i, m) {
      if (bin[code[i]] == -1)
         bin[code[i]] = cnt++;
      msk |= bin[code[i]] << (i * 3);
   } return msk;
}

void trans(int i, int j, int k, int cur) {
   decode(scr->msk[k], code);
   int col = scr->col[k];
   int lft = (j == 0) ? -1 : (col >> (j - 1)) & 1;
   int up = (i == 0) ? -1 : (col >> (j + 1)) & 1;
   int p = (j == 0 || i == 0) ? -1 : (col >> j) & 1;

// printf("dp[%d][%d][", i, j);
// forn (t, m) printf("%d", code[t]);
// printf("][");
// forn (t, m + 1) printf("%d", (col >> t) & 1);
// printf("] = %lld (%d)\n", scr->dp[k], cur);

   if (i == n - 1 && j == m - 1 && lft == 1 - cur
      && up == 1 - cur && p == cur) return;  // 情况1
   if (lft == cur && up == cur && p == cur) return; // 情况2
   if (up != -1 && up != cur) {
      int cnt = 0;
      forn (u, m) if (code[u] == code[j]) cnt++;
      if (cnt == 1) {     // 情况3
         forn (u, m + 1) if (u != j && u != j + 1)
            if (((col >> u) & 1) == up) return; // 情况3.1
         if (i * m + j < n * m - 2) return; // 情况3.2
      }
   }
   if (lft == cur && up == cur) {
      int t = code[j];
      forn (u, m) if (code[u] == t)
         code[u] = code[j - 1]; // 合并
   } else if (lft != cur && up != cur) {
      code[j] = 8;    // 新连通块
   } else if (lft == cur) {
      code[j] = code[j - 1];
   }
   if (p == -1) col |= cur << j;
   else col ^= ((cur ^ p) << j);
   if (j == m - 1) col = (col << 1) & ~(1 << (m + 2));
   int id = des->push(encode(code), col, scr->dp[k]);
   pre[i * m + j][id] = k << 1 | cur;
}
ll plugDP(int &id) {
   scr = hp; des = scr + 1;
   scr->clear(); scr->push(0, 0, 1);
   forn (i, n) forn (j, m) {
      des->clear();
//    puts("");
      forn (k, scr->size) {
         if (maze[i][j] != 'o') trans(i, j, k, 1);
         if (maze[i][j] != '#') trans(i, j, k, 0);
      }
      swap(scr, des);
   }
   type ans = 0;
   forn (k, scr->size) {
      decode(scr->msk[k], code);
      int block = 0;
      forn (i, m) block = max(block, code[i]);
      if (block <= 1) {
         ans += scr->dp[k];
         id = k << 1;
      }
   } return ans;
}
void solve() {
   int id;
   scanf("%d%d", &n, &m);
   forn (i, n) scanf("%s", maze[i]);
   type ans = plugDP(id);
   printf("%d\n", ans);
   if (ans == 0) { puts(""); return;}
   ford (i, n) ford (j, m) {
      id = pre[i * m + j][id >> 1];
      maze[i][j] = id & 1 ? '#' : 'o';
   }
   forn (i, n) puts(maze[i]);
   puts("");
}
int main() {
   int cas; scanf("%d", &cas);
   forn (i, cas) solve();
   return 0;
 }
```

```
/*==========================================================================*\
  3)   重复覆盖问题(Dancing Links + IDA*)
\*==========================================================================*/
int u[M], d[M], l[M], r[M];
int col[M], s[N];
// 上面变量都需初始化
void remove(int &c){
   for(int i = d[c]; i != c ; i = d[i])
      l[r[i]] = l[i], r[l[i]] = r[i];
}
void resume(int &c){
   for(int i = u[c]; i != c ; i = u[i])
      l[r[i]] = i, r[l[i]] = i;
}
int h() {
   bool hash[51];
   memset(hash,false,sizeof(hash));
   int ret = 0;
   for(int c = r[0]; c; c = r[c]) if(!hash[c]){
      ret++;
      hash[c] = true;
      for(int i = d[c] ; i != c ; i = d[i])
         for(int j = r[i] ; j != i ; j = r[j])
            hash[col[j]] = true;
   }
   return ret;
}

bool dfs(int deep,int lim) {
   if(deep + h() > lim) return false;
   if(r[0] == 0) return true;
   int idx , i , j , minnum = 99999;
   for(i = r[0] ; i; i = r[i])
      if(s[i] < minnum)
         minnum = s[i], idx = i;
   for(i = d[idx]; i != idx; i = d[i]) {
      remove(i);
      for(j = r[i]; j != i ; j = r[j]) remove(j);
      if(dfs(deep+1,lim)) return true;
      for(j = l[i]; j != i ; j = l[j]) resume(j);
      resume(i);
   }
   return false;
}
```

4. 图论

```
/*==================================================================*\
  1)   二分图最佳匹配(kuhn munkras 算法)
   最大权匹配/最小权匹配，复杂度 O(n^3)
\*==================================================================*/
#define N 200
#define INF 0x7fffffff
int g[N][N],nx,ny;   //需要初始化
int mx[N],my[N],lx[N],ly[N]; //lx[],ly[]为KM算法中Xi与Yi的顶点标号
bool sx[N],sy[N];    //标记是否在交错树上
int prev[N],slack[N];   //prev[i]为Y中i点在交错树上的前点；slack为松弛量
int q[N*2],head,tail;
void augment(int v){ //增广
   while(v!=-1){
      int pv=mx[prev[v]];
      mx[prev[v]]=v; my[v]=prev[v];v=pv;
   }
}
bool bfs(){
   while(head!=tail){
      int p=q[head++],u=p>>1;
      if(p & 1){
         if(my[u]==-1){ augment(u);return true; }
         else { q[tail++]=my[u]<<1; sx[my[u]]=true; }
      }
      else for(int i=0;i<ny;i++)
         if(sy[i])continue;
         else if(lx[u]+ly[i]!=g[u][i]){
            int ex=lx[u]+ly[i]-g[u][i];
            if(slack[i]>ex){ slack[i]=ex; prev[i]=u; }
         }
         else { prev[i]=u; sy[i]=true; q[tail++]=i*2+1; }
   }return false;
}
int KMmatch(bool maxsum = true){    //默认为最大权匹配
   int i,j,ex,cost=0;
   if(!maxsum) for(i=0;i<nx;i++) for(j=0;j<ny;j++) g[i][j]*=-1;
   memset(mx,-1,sizeof(mx));
   memset(my,-1,sizeof(my));
   memset(ly,0,sizeof(ly));
   for(i=0;i<nx;i++)
      for(lx[i]=-INF,j=0;j<ny;j++)
         lx[i]=max(lx[i],g[i][j]);
   for(int live=0;live<nx;live++){
      memset(sx,0,sizeof(sx)); memset(sy,0,sizeof(sy));
      for(i=0;i<ny;i++)slack[i]=INF;
      head=tail=0; q[tail++]=live*2; sx[live]=true;
      while(!bfs()){
         for(ex=INF,i=0;i<ny;i++)if(!sy[i]) ex=min(ex,slack[i]);
         for(i=0;i<nx;i++) if(sx[i])lx[i]-=ex;
         for(j=0;j<ny;j++){ if(sy[j])ly[j]+=ex;slack[j]-=ex;}
         for(i=0;i<ny;i++)
            if(!sy[i] && slack[i]==0){q[tail++]=i*2+1;sy[i]=true;}
      }
   }
   if(!maxsum) for(i=0;i<nx;i++) for(j=0;j<ny;j++) g[i][j]*=-1;
   for(i=0;i<nx;i++)cost+=g[i][mx[i]];
   return cost;
}
/*==================================================================================*\
  2)   二分图最佳匹配(kuhn munkras 算法)按交大模板改的
   复杂度 O(n^4)(上界较宽)。比上一个模板的优点是代码短。
\*==================================================================================*/
#define N 200
int mx[N], my[N], lx[N], ly[N];
bool sx[N], sy[N];
int nx, ny, g[N][N]; //需要初始化
bool path(int u) {
   sx[u] = 1;
   forn (v, ny) if (g[u][v] == lx[u] + ly[v] && !sy[v]) {
      sy[v] = 1;
      if (my[v] == -1 || path(my[v])) {
         mx[u] = v; my[v] = u; return 1;
      }
   } return 0;
}
int KMmatch(bool maxsum = true){  //默认为最大权匹配
   int j, ret = 0;
   if (!maxsum) forn (i, nx) forn (j, ny) g[i][j] *= -1;
   clr(ly, 0);  clr(mx, -1);  clr(my, -1);
   forn (i, nx) for(lx[i] = -inf, j = 0; j < ny; j++)
      lx[i] = max(lx[i], g[i][j]);
   forn (u, nx) if (mx[u] == -1) {
      clr(sx, 0);  clr(sy, 0);
      while(!path(u)){
         int ex=inf;
         forn (i, nx) if (sx[i]) forn (j, ny) if (!sy[j])
            ex = min(ex, lx[i] + ly[j] - g[i][j]);
         forn (i, nx) if (sx[i]) { lx[i] -= ex; sx[i] = 0; }
         forn (j, ny) if (sy[j]) { ly[j] += ex; sy[j] = 0; }
      }
   }
   if (!maxsum) forn (i, nx) forn (j, ny) g[i][j] *= -1;
   forn (i, nx) ret += g[i][mx[i]];
   return ret;
```

```
}
/*============================================================================*\
  3)   次小生成树
\*============================================================================*/
#define N 1010
#define inf 0x3fffffff
typedef int type;
int n, pre[N];
type cost[N][N], lowc[N], maxc[N][N];
bool vis[N];
type prim(){
   type res = 0, minc;
   clr(vis, 0);  clr(pre, 0);
   forn (i, n) forn (j, n) maxc[i][j] = -inf;
   vis[0] = 1;  pre[0] = -1;
   forn (i, n) lowc[i] = cost[0][i];
   forn (i, n - 1) {
      minc = inf; int p = -1;
      forn (j, n) if (!vis[j] && minc > lowc[j]) {
         minc = lowc[j];  p = j;
      }
      if(p == -1) return -1;
      res += minc;
      forn (j, n)
         if (vis[j]) maxc[j][p] = maxc[p][j] =
            max(maxc[j][ pre[p] ], cost[ pre[p] ][p]);
         else if(lowc[j]>cost[p][j]){lowc[j]=cost[p][j];pre[j]=p;}
      vis[p] = 1;
   } return res;
}
type secondmst(){
   type res = prim(), minc = inf;
   forn (i, n) for (int j = i + 1; j < n; ++j)
      if(pre[i] != j && pre[j] != i)
         minc = min(minc, cost[i][j] - maxc[i][j]);
   return res + minc;
}
/*============================================================================*\
  4)   BellmanFord 单源最短路（吉大模板）
\*============================================================================*/
#define typec int // type of cost
const typec inf=0x3f3f3f3f; // max of cost
int n, m, pre[V], edge[E][3]; typec dist[V];
bool relax (int u, int v, typec c){
   if (dist[v] > dist[u] + c) {
      dist[v] = dist[u] + c;
      pre[v] = u; return 1;
   }return 0;
```

```
}
int bellman (int src){
   int i, j;
   for(i=0;i<n;++i){ dist[i]=inf; pre[i]=-1;}
   dist[src] = 0; bool flag;
   for (i=1; i<n; ++i){
      flag = false; // 优化
      for (j=0; j<m; ++j)
         if(relax(edge[j][0], edge[j][1],edge[j][2]))
            flag = true;
      if ( !flag ) break;
   }
   for (j=0; j<m; ++j)
      if( relax(edge[j][0], edge[j][1], edge[j][2]) )
         return 0; // 有负圈
   return 1;
}
/*==================================================================*\
 5)   Floyd 求最小环（吉大模板）
\*==================================================================*/
const int N=110;
int n,m,g[N][N],dist[N][N];
int r[N][N],out[N],ct;
int solve(int i,int j,int k){
   ct=0;
   while(j!=i){ out[ct++]=j;j=r[i][j];}
   out[ct++]=i;out[ct++]=k;
   return 0;
}
int main(){
   int i,j,k;
   scanf("%d%d",&n,&m);
   for(i=0;i<n;i++)for(j=0;j<n;j++){
      g[i][j]=INF;r[i][j]=i;
   }
   for(i=0;i<m;i++){
      int x,y,l;
      scanf("%d%d%d",&x,&y,&l);
      if(l<g[--x][--y])g[x][y]=g[y][x]=l;
   }
   memmove(dist,g,sizeof(dist));
   int _min=INF;
   for(k=0;k<n;k++){
      for(i=0;i<k;i++)if(g[k][i]<INF)
         for(j=i+1;j<k;j++)
            if(dist[i][j]<INF && g[k][j]<INF &&
               _min>dist[i][j]+g[k][i]+g[k][j])
                  _min=dist[i][j]+g[k][i]+g[k][j],solve(i,j,k);
      for(i=0;i<n;i++)if(dist[i][k]<INF)
         for(j=0;j<n;j++)
            if(dist[k][j]<INF && dist[i][j]>dist[i][k]+dist[k][j])
               dist[i][j]=dist[i][k]+dist[k][j],r[i][j]=r[k][j];
   }
   if(_min<INF){
      for(ct--;ct>=0;ct--){
         printf("%d",out[ct]+1);
         if(ct)printf(" ");
      }
   }
   else printf("No solution.");
   printf("\n");
   return 0;
}
/*==================================================================*\
 6)   Tarjan 强连通分量 O(N + M)
\*==================================================================*/
const int N = 10010;
const int M = 50010;
int e, ev[M], nxt[M], head[N];
bool instack[N];
int dfn[N], low[N], dindex, q[N], ed;
int belong[N], bcnt;  //记录连通分量
void addedge(int u, int v){
   ev[e] = v; nxt[e] = head[u]; head[u] = e++;
}
void tarjan(int u){
   int i, v;
   dfn[u] = low[u] = ++dindex;//时间戳
   instack[u] = 1;
   q[ed++] = u;
   for(i = head[u]; i != -1; i = nxt[i]){
      v = ev[i];
      if (!dfn[v]){
         tarjan(v);
         low[u] = min(low[u], low[v]);
      }else if(instack[v])
         low[u] = min(low[u], dfn[v]);
   }
   if (dfn[u] == low[u]){
      do{
         belong[v = q[--ed]] = bcnt;
         instack[v] = 0;
      }while(v != u);
      bcnt++;
   }
}
```

```
void solve(){
   int n, m, u, v;
   scanf("%d%d", &n, &m);
   e = 0;
   memset(head, -1, sizeof(head));
   while(m--){
      scanf("%d%d", &u, &v);
      addedge(--u, --v);
   }
   ed = bcnt = dindex = 0;
   memset(dfn,0,sizeof(dfn));
   memset(instack, 0, sizeof(instack));
   for(int i = 0; i < n; ++i)
      if(!dfn[i])tarjan(i);
}
/*============================================================================*\
 7)   2-SAT + 缩点 O(N + M)
\*============================================================================*/
const int N = 8010 * 2;
const int M = 20010 * 2;
int dfn[N], low[N], q[N], ed, dindex;
int belong[N], bcnt, order[N], f[N];
bool instack[N], vis[N];
set<pii> edge;

struct graph {
   int e, head[N];
   int ev[M], nxt[M];
   void init() { e = 0; clr(head, -1); }
   void addedge(int u, int v) {
      ev[e] = v; nxt[e] = head[u]; head[u] = e++;
   }
   void input(int m) {
      int u, v;
      init();
      forn (i, m) {
         scanf("%d%d", &u, &v); // u和v不能共存
         u--; v--;
         addedge(u, v ^ 1);
         addedge(v, u ^ 1);
      }
   }
   void tarjan(int);
   void toposort(int u) {
      vis[u] = 1;
      for (int i = head[u]; ~i; i = nxt[i]) {
         int v = ev[i];
         if (!vis[v]) toposort(v);
      } order[--dindex] = u;
   }
   void dfs(int u) {
      vis[u] = 1;
      for (int i = head[u]; ~i; i = nxt[i]) {
         int v = ev[i]; if (!vis[v]) dfs(v);
      }
   }
   void build(int);
}g, s;

void graph::build(int n) { // 缩点建新图
   edge.clear();  s.init();
   for (int u = 0; u < n; ++u)
      for (int i = head[u]; ~i; i = nxt[i]) {
         int a = belong[u];
         int b = belong[ev[i]];
         if (a != b && edge.find(MP(b, a)) == edge.end()) {
            edge.insert(MP(b, a));
            s.addedge(b, a);   // 建反向边
         }
      }
}

void graph::tarjan(int u) {...} // 调用前面tarjan的模板

void solve(int n) {
   bcnt = dindex = ed = 0;
   clr(dfn, 0);  clr(instack, 0);
   forn (i, n) if (!dfn[i]) g.tarjan(i);
   forn (i, n) {
      if (belong[i] == belong[i ^ 1]) {
         puts("NIE"); return;   // 无可行解
      }
      f[belong[i]] = belong[i ^ 1];
   }
   g.build(n);
   dindex = bcnt; clr(vis, 0);
   forn (i, bcnt) if (!vis[i]) s.toposort(i);
   set<int> res;
   clr(vis, 0);
   forn (i, bcnt) {
      int u = order[i];
      if (!vis[u]) {
         res.insert(u);
         s.dfs(f[u]);
      }
   }
   forn (i, n)
      if (res.find(belong[i]) != res.end())
         printf("%d\n", i + 1);
}

int main() {
   int n, m;
   while (~scanf("%d%d", &n, &m)) {
      g.input(m);
      solve(n * 2);
   } return 0;
}
/*==============================================================================*\
  8)   LCA
\*==============================================================================*/
const int N = 10010;
const int M = N * 2;
const int H = 20;

struct graph {
   int e, head[N];
   int ev[M], nxt[M];
   void init() { e = 0; clr(head, -1); }
   void addedge(int u, int v) {
      ev[e] = v; nxt[e] = head[u]; head[u] = e++;
      ev[e] = u; nxt[e] = head[v]; head[v] = e++;
   }
};

int ln[N];
struct LCA {
   int pnt[N][H], depth[N], stk[N];
   void init() { // 求1-N所有log2(x)的值，只需初始化一次
      ln[0] = ln[1] = 0;
      for (int i = 2; i < N; ++i)
         ln[i] = ln[i >> 1] + 1;
   }
   int getfather(int x, int len) {
      while (len > 0) {
         x = pnt[x][ ln[len] ];
         len -= 1 << ln[len];
      } return x;
   }
   int lca(int x, int y) {
      int low = 0, high = min(depth[x], depth[y]);
      x = getfather(x, depth[x] - high);
      y = getfather(y, depth[y] - high);
      if (x == y) return x;
      while (high - low > 1) {
         int mid = ln[high - low - 1];
         int nx = pnt[x][mid];
         int ny = pnt[y][mid];
         mid = high - (1 << mid);
         if (nx == ny) low = mid;
         else { high = mid; x = nx; y = ny; }
      }
      return pnt[x][ln[high - low]];
   }

   /********下面求得depth[]和pnt[][]值，也可以通过其他方式求得********/
   void build(const graph& g, int root, int n) {
      forn (i, n) {
         depth[i] = -1;
         clr(pnt[i], -1);
      }
      int top = 1;
      depth[ stk[0] = root ] = 0;
      while (top) { // 这里默认g为一颗树，若为森林需要修改此处
         int u = stk[--top];
         for (int i = g.head[u]; ~i; i = g.nxt[i]) {
            int v = g.ev[i];
            if (depth[v] != -1) continue;
            stk[top++] = v;
            pnt[v][0] = u;
            depth[v] = depth[u] + 1;
         }
      }
      for (int i = 1; i < H; ++i)
         forn (u, n) if (pnt[u][i - 1] != -1)
            pnt[u][i] = pnt[ pnt[u][i - 1] ][i - 1];
   }
};
/*==============================================================================*\
  9)    Tarjan(边双连通)
\*==============================================================================*/
const int N = 10010, M = 20010 * 2;
struct graph {
   int e, head[N], ev[M], nxt[M];
   void init() { e = 0; clr(head, -1); }
   void addedge(int u, int v) {
      ev[e] = v; nxt[e] = head[u]; head[u] = e++;
      ev[e] = u; nxt[e] = head[v]; head[v] = e++;
   }
};

struct Biconnected {
```

```
int dfn[N], low[N], tim;
int bridge[M]; // 边e为桥则 bridge[e >> 1] == 1
int cut[N];  //点u为割点则 cut[u] == 1
int belong[N], bnt;  // 按桥分块，每个点属于哪个块

void tarjan(const graph& g, int u, bool isroot) {
   dfn[u] = low[u] = ++tim;
   int cnt = 0;
   for (int i = g.head[u]; ~i; i = g.nxt[i]) {
      if (bridge[i >> 1]) continue;
      bridge[i >> 1] = -1;
      int v = g.ev[i];
      if (!dfn[v]) {
         cnt++;
         tarjan(g, v, false);
         if (dfn[u] <= low[v]) cut[u] = 1;
         if (dfn[u] < low[v]) bridge[i >> 1] = 1;
         low[u] = min(low[u], low[v]);
      } else
         low[u] = min(low[u], dfn[v]);
   }
   if (isroot && cnt < 2) cut[u] = 0;
}
void dfs(const graph& g, int u, int mark) {
   belong[u] = mark;
   for (int i = g.head[u]; ~i; i = g.nxt[i]) {
      if (bridge[i >> 1] == 1) continue;
      int v = g.ev[i];
      if (belong[v] == -1) dfs(g, v, mark);
   }
}
void work(const graph& g, int n, graph& tr) {
   tim = 0; clr(cut, 0); clr(dfn, 0); clr(bridge, 0);
   forn (i, n) if (!dfn[i]) tarjan(g, i, true);

   bnt = 0; clr(belong, -1);
   forn (i, n) if (belong[i] == -1) dfs(g, i, bnt++);

   set<pii> edge;  tr.init();
   forn (i, g.e / 2) if (bridge[i] == 1) {  // 缩点建树
      int u = belong[g.ev[i * 2]];
      int v = belong[g.ev[i * 2 + 1]];
      if (edge.find(MP(u, v)) != edge.end()) continue;
      edge.insert(MP(u, v));
      edge.insert(MP(v, u));
      tr.addedge(u, v);
   }
}
```

```
};
/*==============================================================*\
  10)  Tarjan(点双连通)
\*==============================================================*/
const int N = 100010;
const int M = 100010 * 2;

struct graph {
   int e, head[N];
   int ev[M], nxt[M];
   void init() { e = 0; clr(head, -1); }
   void addedge(int u, int v) {
      ev[e] = v; nxt[e] = head[u]; head[u] = e++;
      ev[e] = u; nxt[e] = head[v]; head[v] = e++;
   }
};

struct Biconnected {
   int dfn[N], low[N], tim;
   int stk[M], top;
   int cut[N];  //点u为割点则 cut[u] == 1
   int belong[M], bcnt;  // 每条边属于哪个连通块

   void tarjan(const graph& g, int u, int e) {
      int v, edge, son = 0;
      dfn[u] = low[u] = ++tim;
      for (int i = g.head[u]; ~i; i = g.nxt[i]) {
         v = g.ev[i];
         if (i == (e ^ 1)) continue;
         if (dfn[v] >= dfn[u]) continue;
         stk[top++] = i >> 1;
         if (!dfn[v]) {
            son++;
            tarjan(g, v, i);
            low[u] = min(low[u], low[v]);
            if (dfn[u] <= low[v]) {
               cut[u] = true;
               do {
                  edge = stk[--top];
                  belong[edge] = bcnt;
               } while (edge != i >> 1);
               bcnt++;
            }
         } else
            low[u] = min(low[u], dfn[v]);
      }
      if (e == -1 && son < 2) cut[u] = 0;
   }
```

```
    void work(const graph& g, int &n, graph& tr) {
        bcnt = tim = top = 0;
        clr(dfn, 0); clr(cut, 0);
        forn (i, n) if (!dfn[i]) tarjan(g, i, -1);

        /*********下面缩点(割点和每个块分别缩为一点)*********/
        set<pii> edge;
        int tot = n;
        n = bcnt;  tr.init();
        forn (u, tot) if (cut[u] == 1) {
            for (int i = g.head[u]; ~i; i = g.nxt[i]) {
                int z = belong[i >> 1];
                if (edge.find(MP(n, z)) != edge.end()) continue;
                tr.addedge(n, z);
                edge.insert(MP(n, z));
                edge.insert(MP(z, n));
            }
            n++;
        }
    }
};
/*==============================================================================*\
  11)  最大团(输出方案)
\*==============================================================================*/
const int N = 110;
int n, m, g[N][N], id[N];
int list[N][N], s[N], degree[N], behide[N];
int found, curmax, curobj;


void sortdegree() {
    for (int i = 1; i <= n; ++i) id[i] = i;
    for (int j, k, l, i = 1; i <= n; ++i) {
        for (k = i, j = i + 1; j <= n; ++j)
            if (degree[j] < degree[k]) k = j;
        if (k != i) {
            swap(id[i], id[k]);
            swap(degree[i], degree[k]);
            for (l = 1; l <= n; ++l) swap(g[i][l], g[k][l]);
            for (l = 1; l <= n; ++l) swap(g[l][i], g[l][k]);
        }
    }
}

void dfs(int d, vi &t) {
    if (d - 1 > curmax) {found = 1; return ;}
    int i, j;
```

```
   for (i = 1; i < list[d - 1][0] - curmax + d; ++i)
      if (!found && d + behide[list[d - 1][i] + 1] > curmax &&
            (list[d-1][0]==i || d+behide[list[d-1][i+1]]>curmax)) {
         for (j = i + 1, list[d][0] = 0; j <= list[d - 1][0]; ++j)
            if (g[list[d - 1][j]][list[d - 1][i]])
               list[d][++list[d][0]] = list[d - 1][j];
         t[d - 1] = list[d - 1][i];
         if (list[d][0] == 0 || d + behide[list[d][1]] > curmax)
            dfs(d + 1, t);
      }
}

void gao() {
   vi ans(1, 1);
   sortdegree(); behide[n + 1] = 0; behide[n] = 1;
   for (int j, i = n - 1; i > 0; --i) {
      curmax = behide[i + 1]; found = list[1][0] = 0;
      for (j = i + 1; j <= n; ++j)
         if (g[j][i]) list[1][++list[1][0]] = j;
      vi tmp(curmax + 1);
      tmp[0] = i;
      dfs(2, tmp); behide[i] = curmax + found;
      if (found) ans = tmp;
   }
   printf("%d\n", sz(ans));
   for (int i = 0; i < sz(ans); ++i)
      ans[i] = id[ans[i]];
   sort(ans.begin(), ans.end());
   for (int i = 0; i < sz(ans); ++i) {
      if (i) printf(" ");
      printf("%d", ans[i]);
   }
   puts("");
}

void solve() {
   int u, v;
   clr(g, 0);
   scanf("%d%d", &n, &m);
   for (int i = 0; i < m; ++i) {
      scanf("%d%d", &u, &v);
      g[u][v] = g[v][u] = 1;
   }
   for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j)
         g[i][j] ^= 1;
   gao();
}
```

```
/*==========================================================================*\
  12)  Minimal Steiner Tree O(4^k*V+2^k*E*logE)
\*==========================================================================*/
typedef int typec;
const int V = 1000, E = 4010, K = 8;
const typec inf = 0x3f3f3f3f;
int ev[E], nxt[E], head[V], vis[V], ch[V], e, n, k;
typec cost[E], dp[V][1 << K];
struct node {
   int v; typec c;
   node(int v = 0, typec c = 0) : v(v), c(c) {}
   bool operator < (const node &u) const {
      return c > u.c;
   }
};
void addedge(int u, int v, typec c) {
   ev[e] = v; cost[e] = c; nxt[e] = head[u]; head[u] = e++;
   ev[e] = u; cost[e] = c; nxt[e] = head[v]; head[v] = e++;
}
int steiner(int ch[], int k) {
   memset(dp, 0x3f, sizeof(dp));
   for (int i = 0; i < k; ++i) dp[ch[i]][1 << i] = 0;
   for (int i = 0; i < n; ++i) dp[i][0] = 0;
   int tot = 1 << k;
   priority_queue<node> que;
   for (int i = 0; i < tot; ++i) {
      for (int j = 0; j < n; ++j)
         for (int u = i; u > 0; u = (u - 1) & i)
            dp[j][i] = min(dp[j][i], dp[j][u] + dp[j][i ^ u]);
//权值在点上时 dp[j][i] = min(dp[j][i], dp[j][u] + dp[j][i ^ u] - cost[j]);
      memset(vis, 0, sizeof(vis));
      for (int j = 0; j < n; ++j)
         if (dp[j][i] != inf)
            que.push(node(j, dp[j][i]));
      while (!que.empty()) {
         int u = que.top().v; que.pop();
         if (vis[u]) continue;
         vis[u] = 1;
         for (int j = head[u]; ~j; j = nxt[j]) {
            int v = ev[j];
            if (dp[v][i] > dp[u][i] + cost[j]) { // 权值在点上用cost[v]
               dp[v][i] = dp[u][i] + cost[j]; // 权值在点上用cost[v]
               que.push(node(v, dp[v][i]));
            }
         }
      }
   } return dp[ch[0]][(1 << k) - 1];
}
```

```
/*==========================================================================*\
  13)  最大流 SAP(V^2 * E)(递归版)
\*==========================================================================*/
```

```
//typedef double typec;
//const typec inf = 1e100;
//const double eps = 1e-8
typedef int typec;
const typec inf = 0x3f3f3f3f;
const int N = 210, M = 410 * 2;

struct FlowNetwork {
   int n, e, head[N], d[N], vd[N], pre[N];
   int nxt[M], ev[M];
   typec c[M];
   void init() { e = 0; clr(head, -1); }
   void addedge(int u, int v, typec w) {
      ev[e]=v; c[e]=w; nxt[e]=head[u]; head[u]=e++;
      ev[e]=u; c[e]=0; nxt[e]=head[v]; head[v]=e++;//无向图c[e]=w;
   }
   typec maxflow(int u, int s, int t, typec flow){
      if(u == t)return flow;
      typec temp, ans = 0;
      for(int i = head[u]; i != -1; i = nxt[i]){
         //if(c[i] < eps || d[u] != d[ev[i]] + 1)continue;
         if(c[i] <= 0 || d[u] != d[ev[i]] + 1)continue;
         temp = maxflow(ev[i], s, t, min(c[i], flow - ans));
         c[i] -= temp; c[i ^ 1] += temp;
         ans += temp;
         if(ans == flow)return ans;
      }
      if(d[s] >= n)return ans;
      if(--vd[d[u]] == 0)d[s] = n;
      vd[++d[u]]++;
      return ans;
   }
   typec sap(int s, int t, int n) {
      clr(d, 0); clr(vd, 0);
      this->n = vd[0] = n;
      typec ans = 0;
      while(d[s] < n) ans += maxflow(s, s, t, inf);
      return ans;
   }
};
```

```
/*==========================================================================*\
  14)  最大流 SAP(V^2 * E)(非递归)
\*==========================================================================*/
```

```
//typedef double typec;
//const typec inf = 1e100;
```

```
//const double eps = 1e-8
typedef int typec;
const typec inf = 0x3f3f3f3f;
const int N = 210, M = 410 * 2;

struct FlowNetwork {
   int e, head[N], d[N], vd[N], pre[N], cur[N];
   int nxt[M], eu[M], ev[M];
   typec c[M];
   void init() { e = 0; clr(head, -1); }
   void addedge(int u, int v, typec w) {
      eu[e]=u; ev[e]=v; c[e]=w; nxt[e]=head[u]; head[u]=e++;
      eu[e]=v; ev[e]=u; c[e]=0; nxt[e]=head[v]; head[v]=e++;
            //无向图中第二个c[e]=w;
   }
   typec sap(int s, int t, int n){
      int i, u;
      clr(d, 0); clr(vd, 0);
      vd[0] = n;
      cur[u = s] = head[s];
      pre[s] = -1;
      typec temp, ans = 0;
      while (d[s] < n) {
         if(u == t){
            for(temp = inf, i = pre[u]; ~i; i = pre[eu[i]])
               temp = min(temp, c[i]);
            for(i = pre[u]; ~i; i = pre[eu[i]]){
               c[i] -= temp; c[i ^ 1] += temp;
            }
            ans += temp;  u = s;
         }
         for (i = cur[u]; ~i; i = nxt[i])
            //if (c[i] > eps && d[u] == d[ev[i]] + 1){
            if (c[i] > 0 && d[u] == d[ev[i]] + 1){
               cur[u] = i;    //当前弧优化
               pre[u = ev[i]] = i;
               break;
            }
         if(i == -1){
            cur[u] = head[u];
            if (--vd[d[u]] == 0)break;
            vd[++d[u]]++;
            if(u != s)u = eu[pre[u]];
         }
      }
      return ans;
   }
};
```

```
/*==================================================================*\
  15)  最大流(预流推进)(watashi 代码)
\*==================================================================*/
/* watashi的预流推进代码(ZOJ2364/SGU212) */
const int MAXN = 1515;
const int MAXM = 300300;

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

struct Edge { int v, 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];
   }
```

```
};

int main() {
   int re;
   int n, m, l, r, s, t;
   FlowNetwork fn;

   scanf("%d", &re);
   for (int ri = 1; ri <= re; ++ri) {
      if (ri > 1) {
         puts("");
      }
      scanf("%d%d%d", &n, &m, &l);
      for (int i = 0; i < n; ++i) {
         scanf("%d", &r);
         if (r == 1) {
            s = i;
         } else if (r == l) {
            t = i;
         }
      }
      fn.init(n, s, t);
      for (int i = 0; i < m; ++i) {
         scanf("%d%d%d", &s, &t, &r);
         fn.addEdge(s - 1, t - 1, r);
      }
      fn.hlpp();
      for (int i = 0; i < m; ++i) {
         printf("%d\n", fn.edge[RE(i << 1)].c);
      }
   }

   return 0;
}
/*==================================================================================*\
  16)  最小费用流(SPFA)
\*==================================================================================*/
// 注：费用不能有负环
typedef int typef;
typedef int typec;
const int N = 5100, M = 40010;
const typef inff = 0x3f3f3f3f;
const typec infc = 0x3f3f3f3f;

struct MinCostMaxFlow {
   int e, ev[M], nxt[M], head[N];
   typec cost[M], dist[N];
   typef cap[M];
   int pnt[N], road[N], q[N], bg, ed;
   bool vis[N];

   void init() { e = 0; clr(head, -1); }

   void addedge(int u, int v, typef f, typec c) { //u->v flow=f, cost=c
      ev[e]=v; cap[e]=f; cost[e]=c; nxt[e]=head[u]; head[u]=e++;
      ev[e]=u; cap[e]=0; cost[e]=-c; nxt[e]=head[v]; head[v]=e++;
   }

   bool spfa(int s, int t, int n) {
      forn (i, n) dist[i] = infc, vis[i] = 0;
      bg = ed = dist[s] = 0;
      pnt[s] = s; q[ed++] = s;
      while (bg != ed) {
         int u = q[bg++]; vis[u] = 0;
         if (bg == N) bg = 0;
         for (int i = head[u]; ~i; i = nxt[i]) {
            if (cap[i] <= 0)continue;
            int v = ev[i];
            if (dist[v] > dist[u] + cost[i]) {
               dist[v] = dist[u] + cost[i];
               pnt[v] = u; road[v] = i;
               if (!vis[v]) {
                  q[ed++] = v; vis[v] = 1;
                  if(ed == N)ed = 0;
               }
            }
         }
      }
      return dist[t] != infc;
   }

   void mincost(int s, int t, int n, typef &f, typec &c) {
      c = f = 0;
      while(spfa(s, t, n)){
         typef minf = inff;
         for(int u = t; u != s; u = pnt[u])
            minf = min(minf, cap[road[u]]);
         for(int u = t; u != s; u = pnt[u]){
            cap[road[u]] -= minf;
            cap[road[u] ^ 1] += minf;
         }
         f += minf;
         c += minf * dist[t];
      }
   }
};
```

```
/*================================================================*\
  17)  最小费用流(ZKW)
\*================================================================*/
const int V=440, E= 4010*2;

struct MinCostMaxFlow {
   struct etype {
      int t, c, f;
      etype *next, *pair;
      etype() {}
      etype(int T,int C,int F,etype* N): t(T), c(C), f(F), next(N) {}
      void* operator new(unsigned, void* p){return p;}
   } *e[V], Te[E+E], *Pe;

   int S, T, n, pis, cost;
   bool v[V];

   void init() { clr(e, 0); Pe = Te; }

   void addedge(int s, int t, int f, int c) {
      e[s] = new(Pe++) etype(t, +c, f, e[s]);
      e[t] = new(Pe++) etype(s, -c, 0, e[t]);
      e[s]->pair = e[t];
      e[t]->pair = e[s];
   }

   int aug(int u, int m) {
      if (u == T) return cost += pis * m, m;
      v[u] = true;
      int f = m;
      for (etype *i = e[u]; i; i = i->next)
         if (i->f && i->c == 0 && !v[i->t]) {
            int d = aug(i->t, min(f, i->f));
            i->f -= d, i->pair->f += d, f -= d;
            if (f == 0) return m;
         }
      return m - f;
   }

   bool modlabel() {
      static int d[V]; memset(d, 0x3f, sizeof(d)); d[T] = 0;
      static deque<int> Q; Q.push_back(T);
      while(Q.size()) {
         int dt, u = Q.front(); Q.pop_front();
         for(etype *i = e[u]; i; i = i->next)
            if(i->pair->f && (dt = d[u] - i->c) < d[i->t])
               (d[i->t] = dt) <= d[Q.size() ? Q.front() : 0]
                     ? Q.push_front(i->t) : Q.push_back(i->t);
```

```
        }
        forn (i, n) for(etype *j = e[i]; j; j = j->next)
            j->c += d[j->t] - d[i];
        pis += d[S];
        return d[S] < inf;
    }

    int mincost(int s, int t, int tot) {
        S = s; T = t; n = tot;
        pis = cost = 0;
        while(modlabel())
            do memset(v, 0, sizeof(v));
            while(aug(S, inf));
        return cost;
    }
};
/*==================================================================================*\
  18)  一般图匹配（带花树）
\*==================================================================================*/
/*  从网上摘抄的代码，不明其中细节，不过验证过其正确性
    注意：下标从1开始   */
#define MAXE 250*250*2
#define MAXN 250
deque<int> Q;
//g[i][j]存放关系图：i,j是否有边,match[i]存放i所匹配的点
bool g[MAXN][MAXN], inque[MAXN], inblossom[MAXN];
int match[MAXN], pre[MAXN], base[MAXN];

//找公共祖先
int findancestor(int u, int v) {
    bool inpath[MAXN] = { false };
    while (1) {
        u = base[u];
        inpath[u] = true;
        if (match[u] == -1) break;
        u = pre[match[u]];
    }
    while (1) {
        v = base[v];
        if (inpath[v]) return v;
        v = pre[match[v]];
    }
}

//压缩花
void reset(int u, int anc) {
    while (u != anc) {
        int v = match[u];
```

```
      inblossom[base[u]] = 1;
      inblossom[base[v]] = 1;
      v = pre[v];
      if (base[v] != anc)
         pre[v] = match[u];
      u = v;
   }
}

void contract(int u, int v, int n) {
   int anc = findancestor(u, v);
   clr(inblossom, 0);
   reset(u, anc);
   reset(v, anc);
   if (base[u] != anc) pre[u] = v;
   if (base[v] != anc) pre[v] = u;
   for (int i = 1; i <= n; i++)
      if (inblossom[base[i]]) {
         base[i] = anc;
         if (!inque[i]) {
            Q.push_back(i);
            inque[i] = 1;
         }
      }
}

bool dfs(int S, int n) {
   for (int i = 0; i <= n; i++)
      pre[i] = -1, inque[i] = 0, base[i] = i;
   Q.clear();
   Q.push_back(S);
   inque[S] = 1;
   while (!Q.empty()) {
      int u = Q.front();
      Q.pop_front();
      for (int v = 1; v <= n; v++) {
         if (g[u][v] && base[v] != base[u] && match[u] != v) {
            if (v == S || (match[v] != -1 && pre[match[v]] != -1))
               contract(u, v, n);
            else if (pre[v] == -1) {
               pre[v] = u;
               if (match[v] != -1)
                  Q.push_back(match[v]), inque[match[v]] = 1;
               else {
                  u = v;
                  while (u != -1) {
                     v = pre[u];
                     int w = match[v];
                     match[u] = v;
                     match[v] = u;
                     u = w;
                  }
                  return true;
               }
            }
         }
      }
   }
   return false;
}

int work(int n) {
   clr(match,-1);
   int ans = 0;
   for (int i = 1; i <= n; i++)
      if (match[i] == -1 && dfs(i, n))
         ans++;
   return ans;
}
/*==================================================================*\
  19)  无根树的最小树形图
\*==================================================================*/
/*
有固定根的最小树形图求法O(VE)：交大和吉大模板上都有这里就不熬述了
对于不固定根的最小树形图，wy教主有一巧妙方法。摘录如下：
新加一个点，和每个点连权相同的边，这个权大于原图所有边权的和，这样这个图固定跟的最小树形图和原图不固定跟的最小树形图就是对应的了。
*/
```

5． 字符串

```
/*==================================================================================*\
  1)   AC 自动机
\*==================================================================================*/
const int N = 1000010;
const int M = 10001 * 50;
const int ch = 26;
int sw[128];//string swap每个字符对应的Index，方便模板化
int trie[M][ch + 1], top, n, q[M], bg, ed, fail[M];
bool vis[M];
char str[N];
void init(){
   top = 1;
   memset(trie[0], 0, sizeof(trie[0]));
   for(char i = 'a', j = 0; i <= 'z'; ++i, ++j)
      sw[i] = j;
}
void ins(char *s, int rank = 1){//rank的值随题目要求而变
   int rt, nxt;
   for(rt = 0; *s; rt = nxt, ++s){
      nxt = trie[rt][sw[*s]];
      if(nxt == 0){
         memset(trie[top], 0, sizeof(trie[top]));
         trie[rt][sw[*s]] = nxt = top++;
      }
   }
// trie[rt][ch] = rank;
   trie[rt][ch]++;
}
void makefail(){
   int u, v;
   fail[0] = bg = ed = 0;
   for(int i = 0; i < ch; ++i)
      if(q[ed] = trie[0][i])
         fail[q[ed++]] = 0;
   while(bg < ed){
      u = q[bg++];
      for(int i = 0; i < ch; ++i){
         if(v = trie[u][i]){
            q[ed++] = v;
            fail[v] = trie[fail[u]][i];
            //对trie[v][ch]按trie[fail[v]][ch]里的内容进行处理
         }else
            trie[u][i] = trie[fail[u]][i];
      }
   }
}
```

```
int ac(char *s){
   int res = 0; memset(vis, 0, sizeof(vis));
   for(int i = 0; *s; ++s){
      i = trie[i][sw[*s]];
      for(int j = i; j && !vis[j]; j = fail[j]){
         vis[j] = 1;
         if(trie[j][ch])
            res += trie[j][ch];
      }
   } return res;
}
void input(){
   init(); char tmp[55];
   scanf("%d", &n);
   for(int i = 1; i <= n; ++i){
      scanf("%s", tmp); ins(tmp);
   } scanf("%s", str);
}
void solve(){
   makefail();
   printf("%d\n", ac(str));
}
/*============================================================================*\
  2)   后缀数组O(N*log(N)) + RMQ O(N*log(N))
\*============================================================================*/
/*
rank[0...7]: 4 6 8 1 2 3 5 7
string:      a a b a a a a b
-------------------------------------------
 sa[1] = 3 : a a a a b        height[1] = 0
 sa[2] = 4 : a a a b          height[2] = 3
 sa[3] = 5 : a a b            height[3] = 2
 sa[4] = 0 : a a b a a a a b  height[4] = 3
 sa[5] = 6 : a b              height[5] = 1
 sa[6] = 1 : a b a a a a b    height[6] = 2
 sa[7] = 7 : b                height[7] = 0
 sa[8] = 2 : b a a a a b      height[8] = 1
*/
const int N = 1000010;
int ua[N], ub[N], us[N];
int cmp(int *r,int a,int b,int l){
   return r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(int *r,int *sa,int n,int m){ //da(r, sa, n + 1, 256);(r[n] = 0)
   int i,j,p,*x=ua,*y=ub,*t;       //r[]存放原字符串，且从char变为int
   for(i=0;i<m;i++) us[i]=0; //sa[i]表示排名为i的后缀起始下标(i>=1,sa[i]>=0)
   for(i=0;i<n;i++) us[x[i]=r[i]]++;
   for(i=1;i<m;i++) us[i]+=us[i-1];
   for(i=n-1;i>=0;i--) sa[--us[x[i]]]=i;
   for(j=1,p=1;p<n;j*=2,m=p){
      for(p=0,i=n-j;i<n;i++) y[p++]=i;
      for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
      for(i=0;i<m;i++) us[i]=0;
      for(i=0;i<n;i++) us[x[i]]++;
      for(i=1;i<m;i++) us[i]+=us[i-1];
      for(i=n-1;i>=0;i--) sa[--us[x[y[i]]]]=y[i];
      for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
         x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
   }
}
int rank[N],height[N]; //height[i]为排第i-1和第i的后缀的公共前缀长度
void calheight(int *r,int *sa,int n){
    int i,j,k=0;
    for(i=1;i<=n;i++) rank[sa[i]]=i;
    for(i=0;i<n;height[rank[i++]]=k)
      for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);
}
int *RMQ = height; // RMQ为查询的数组,这里RMQ=height
//int RMQ[N];
int mm[N];
int best[20][N]; //best[i][j]表示[j, j + 2^i)区间中的最小值
void initRMQ(int n){
    int i,j,a,b;
    for(mm[0]=-1,i=1;i<=n;i++)
      mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
    for(i=1;i<=n;i++) best[0][i]=i;
    for(i=1;i<=mm[n];i++)
    for(j=1;j<=n+1-(1<<i);j++){
      a=best[i-1][j];
      b=best[i-1][j+(1<<(i-1))];
      if(RMQ[a]<RMQ[b]) best[i][j]=a;
      else best[i][j]=b;
    }
}
int askRMQ(int a,int b){
   int t;
   t=mm[b-a+1];b-=(1<<t)-1;
   a=best[t][a];b=best[t][b];
   return RMQ[a]<RMQ[b]?a:b;
}
int lcp(int a,int b){ //后缀r[a]和r[b]的公共前缀长度
   int t;
   a=rank[a];b=rank[b];
   if(a>b) {t=a;a=b;b=t;}
   return(height[askRMQ(a+1,b)]);
}
```

```
/*==========================================================================*\
  3)   KMP
\*==========================================================================*/
int fail[M];

void makefail(char *t, int lt){
   t--;
   for(int i = 1, j = 0; i <= lt + 1; ++i, ++j){
      fail[i] = j;
      while(j > 0 && t[i] != t[j]) j = fail[j];
   }
}
int kmp(char *s, int ls, char *t, int lt){
   --s; --t; int cnt = 0;
   for(int i = 1, j = 1; i <= ls; ++i, ++j){
      while(j > 0 && s[i] != t[j])j = fail[j];
      if(j == lt){
         cnt++;
         j = fail[lt + 1] - 1;
      }

   } return cnt; // 返回出现了几次

}
/*==========================================================================*\
  4)   字符串最小表示
\*==========================================================================*/
// flag = 1 为最小表示; flag = 0 为最大表示
int mpres(char *s, int len, bool flag) {
   int i = 0, j = 1, k = 0, t;
   while (i < len && j < len && k < len) {
      t = s[(i + k) % len] - s[(j + k) % len];
      if (t == 0) { ++k; continue; }
      if ((t > 0) == flag)
         i = max(i + k + 1, j);
      else
         j = max(j + k + 1, i);
      if (i == j) ++i;
      k = 0;
   } return min(i, j);
}
/*==========================================================================*\
  5)   扩展 KMP
\*==========================================================================*/
const int N = 100010;
char a[N], b[N];
int lcp1[N], lcp2[N];

void SelfLcp(char *t, int lt, int *lcp) {
    int j = 0, k;
    while (j + 1 < lt && t[j] == t[j + 1]) ++j;
    lcp[1] = j; k = 1;
    for (int i = 2; i < lt; ++i) {
        int len = k + lcp[k] - 1, l = lcp[i - k];
        if (l < len - i + 1) lcp[i] = l;
        else {
            j = max(0, len - i + 1);
            while (i + j < lt && t[j] == t[i + j]) ++j;
            lcp[i] = j; k = i;
        }
    }
}

//lcp1[i]为s[i,|s|]和t的公共前缀长，lcp2[i]为t[i,|t|]和t的公共前缀长
void ExtKmp(char *s, int ls, int *lcp1, char *t, int lt, int *lcp2){
    SelfLcp(t, lt, lcp2);
    int j = 0, k;
    while (j < ls && j < lt && s[j] == t[j]) ++j;
    lcp1[0] = j; k = 0;
    for (int i = 1; i < ls; ++i) {
        int len = k + lcp1[k] - 1, l = lcp2[i - k];
        if (l < len - i + 1) lcp1[i] = l;
        else {
            j = max(0, len - i + 1);
            while (i + j < ls && j < lt && s[i + j] == t[j]) ++j;
            lcp1[i] = j; k = i;
        }
    }
}
/*==================================================================================*\
  6)   O(n)回文子串算法
\*==================================================================================*/
/*
   原串： w a a b w s w f d
新串r[]： $ # w # a # a # b # w # s # w # f # d #
辅助数组P：  1 2 1 2 3 2 1 2 1 2 1 4 1 2 1 2 1 2 1
p[id]- 1 就是该回文子串在原串中的长度
*/
const int N = 110010 * 2;
char str[N];
int r[N], p[N];

void pk(int *r, int n, int *p) {
    int i, id, mx = 0;
    for (i = 1; i < n; ++i) {
        if (mx > i) p[i] = min(p[2 * id - i], mx - i);
        else p[i] = 1;
```

```
      for (; r[i + p[i]] == r[i - p[i]]; p[i]++);
      if (p[i] + i > mx) {
         mx = p[i] + i;
         id = i;
      }
   }
}

void solve() {
   scanf("%s", str);
   int len = strlen(str);
   int n = 0;
   r[n++] = '$'; r[n++] = '#';
   forn (i, len) {
      r[n++] = str[i];
      r[n++] = '#';
   }
   r[n] = 0;
   pk(r, n, p);
   int ans = 0;
   for (int i = 1; i < n; ++i)
      ans = max(ans, p[i] - 1);
   printf("%d\n", ans);
}
```

6． 计算几何

```
/*==================================================================*\
  1)   二维几何基本操作
\*==================================================================*/
#define sqr(x)  ((x)*(x))
const double eps = 1e-8;
const double pi = acos(-1.0);

int dcmp(double x) {
   if (x < -eps) return -1; else return x > eps;
}
struct cpoint {
   double x, y;
   cpoint(){}
   cpoint(double x, double y) : x(x), y(y) {}
   cpoint operator + (const cpoint &u) const {
      return cpoint(x + u.x, y + u.y);
   }
   cpoint operator - (const cpoint &u) const {
      return cpoint(x - u.x, y - u.y);
   }
   cpoint operator * (const double &s) const {
      return cpoint(x * s, y * s);
   }
   cpoint operator / (const double &s) const {
      return cpoint(x / s, y / s);
   }
   double operator * (const cpoint &u) const { // 叉积
      return x * u.y - y * u.x;
   }
   double operator ^ (const cpoint &u) const { // 点积
      return x * u.x + y * u.y;
   }

   cpoint turnleft() const { // 左转90度
      return cpoint(-y, x);
   }
   cpoint turnleft(double ang) const {
      double c = cos(ang), s = sin(ang);
      return cpoint(x * c - y * s, y * c + x * s);
   }
   cpoint turnright() const { // 右转90度
      return cpoint(y, -x);
   }
   cpoint turnright(double ang) const {
      double c = cos(ang), s = sin(ang);
      return cpoint(x * c + y * s, y * c - x * s);
   }
   cpoint trunc(double s) { // 向量长度变为s
      double r = s / len();
      return cpoint(x * r, y * r);
   }
   double len() { return sqrt(x * x + y * y); }
   void get() { scanf("%lf%lf", &x, &y); }

   bool operator == (const cpoint& u) const {
      return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0;
   }
   bool operator < (const cpoint& u) const {
      if (dcmp(x - u.x)) return x < u.x;
      else return dcmp(y - u.y) < 0;
   }
};

double cross(cpoint o, cpoint p, cpoint q) { // 叉积
   return (p - o) * (q - o);
}
double dot(cpoint o, cpoint p, cpoint q) { // 点积
   return (p - o) ^ (q - o);
}
double dis(cpoint p, cpoint q) { // 两点距离
   return sqrt(sqr(p.x - q.x) + sqr(p.y - q.y));
}
double dissqr(cpoint p, cpoint q) {    // 距离平方
   return sqr(p.x - q.x) + sqr(p.y - q.y);
}
double angle(cpoint p0,cpoint p1,cpoint p2) {//计算角p1p0p2,范围在(-pi, pi]
   double cr = cross(p0, p1, p2);
   double dt = dot(p0, p1, p2);
   if (dcmp(cr) == 0) cr = 0.0;
   if (dcmp(dt) == 0) dt = 0.0;
   return atan2(cr, dt); //p0p1到p0p2逆时针为正
}
bool PointOnLine(cpoint p0, cpoint p1, cpoint p2) { // 判点p0在直线上p1p2
   return dcmp(cross(p0, p1, p2)) == 0;
}
bool PointOnSegment(cpoint p0, cpoint p1, cpoint p2){//判点p0在线段上p1p2
   return dcmp(cross(p0, p1, p2)) == 0 && dcmp(dot(p0, p1, p2)) <= 0;
}
//  判断直线相对位置 1=相交， 0=平行， -1=重合，  cp返回交点
int LineInter(cpoint p1, cpoint p2, cpoint p3, cpoint p4, cpoint &cp){
   double u = cross(p1, p2, p3), v = cross(p2, p1, p4);
   if (dcmp(u + v)) {
      cp = (p3 * v + p4 * u) / (u + v);
      return 1;
   }
   if (dcmp(u)) return 0;
   if (dcmp(cross(p3, p4, p1))) return 0;
   return -1;
}
// 判断线段相交，cp返回交点
int SegmentInter(cpoint p1,cpoint p2,cpoint p3,cpoint p4,cpoint &cp){
   int ret = LineInter(p1, p2, p3, p4, cp);
   if (ret == 1)
      return PointOnSegment(cp, p1, p2) && PointOnSegment(cp, p3, p4);
   if (ret==-1&&(PointOnSegment(p1,p3,p4)||PointOnSegment(p2,p3,p4)
         || PointOnSegment(p3, p1, p2) || PointOnSegment(p4, p1, p2) ))
      return -1;
   return 0;
}

// 判线段和直线相交：线段的两个端点和直线上任意两点的叉积异号即相交(叉积为0则点在直线上)

// 判断线段相交
int SegmentInterTest(cpoint p1, cpoint p2, cpoint p3, cpoint p4) {
   if (max(p1.x, p2.x) + eps < min(p3.x, p4.x) ||
      max(p3.x, p4.x) + eps < min(p1.x, p2.x) ||
      max(p1.y, p2.y) + eps < min(p3.y, p4.y) ||
      max(p3.y, p4.y) + eps < min(p1.y, p2.y)) return 0 ;
   int d1 = dcmp(cross(p3, p4, p2));
   int d2 = dcmp(cross(p3, p4, p1));
   int d3 = dcmp(cross(p1, p2, p4));
   int d4 = dcmp(cross(p1, p2, p3));
   if (d1 * d2 == 1 || d3 * d4 == 1) return 0 ;
   if (d1 == 0 && d2 == 0 && d3 == 0 && d4 == 0) return -1;
   return 1 ;
}
// 判点在多边形内 0 = outside; 1 = inside; 2 = boundary
int PointInPolygon(cpoint cp, cpoint p[], int n) {
   int i, k, d1, d2, wn = 0;
   p[n] = p[0];
   for (i = 0; i < n; i++) {
      if (PointOnSegment(cp, p[i], p[i + 1])) return 2 ;
      k = dcmp(cross(p[i], p[i + 1], cp));
      d1 = dcmp(p[i + 0].y - cp.y);
      d2 = dcmp(p[i + 1].y - cp.y);
      if (k > 0 && d1 <= 0 && d2 > 0) wn++;
      if (k < 0 && d2 <= 0 && d1 > 0) wn--;
   }
   return wn != 0;
}
// 点到直线的距离，cp为点p0在直线上的射影
double PointToLine(cpoint p0, cpoint p1, cpoint p2, cpoint &cp) {
   double d = dis(p1, p2);
   double s = cross(p1, p2, p0) / d ;
   cp.x = p0.x + s * (p2.y - p1.y) / d;
   cp.y = p0.y - s * (p2.x - p1.x) / d;
   return fabs(s);  // s为有向距离
}
// 点在直线上的射影(可以拓展到三维)
cpoint PointProjLine(cpoint p0, cpoint p1, cpoint p2) {
   double t = dot(p1, p2, p0) / dot(p1, p2, p2);
   return p1 + (p2 - p1) * t;
}
// 求多边形面积(凸凹都可)
double PolygonArea(cpoint p[], int n) {
   p[n] = p[0]; double s = 0;
   for (int i = 0; i < n; ++i) s += p[i] * p[i + 1];
   return s; // 顺时针方向s为负
}
/*============================================================================*\
 2)   二维凸包
\*============================================================================*/
cpoint bp;
int PolarCmp(const cpoint &p1, const cpoint &p2) {
   int u = dcmp(cross(bp, p1, p2));
   return u > 0 || (u == 0 && dcmp(dissqr(bp, p1)-dissqr(bp, p2)) < 0);
}
// ch中的点为逆时针顺序，边界无三点共线
void graham(cpoint pin[], int n, cpoint ch[], int &m) {
   int i, j, k, u, v;
   memcpy(ch, pin, n * sizeof(cpoint));
   for (i = k = 0; i < n; ++i) {
      u = dcmp(ch[i].x - ch[k].x);
      v = dcmp(ch[i].y - ch[k].y);
      if (v < 0 || (v == 0 && u < 0)) k = i;
   }
   bp = ch[k];
   sort(ch, ch + n, PolarCmp);
   n =  unique(ch, ch + n) - ch; // 注意重载”==”
   if (n <= 1) { m = n; return ;}
   if (dcmp(cross(ch[0], ch[1], ch[n - 1])) == 0) {
      m = 2; ch[1] = ch[n - 1]; return;
   }
   ch[n++] = ch[0];
   for (i = 1, j = 2; j < n; ++j) {
      while (i > 0 && dcmp(cross(ch[i - 1], ch[i], ch[j])) <= 0) i--;
      ch[++i] = ch[j];
   }
   m = i;
}
```

```
/*==============================================================*\
  3)   Pick 公式（网格）
\*==============================================================*/
/*给定顶点坐标均是整点（或正方形格点）的简单多边形，其面积S和内部格点数目a、边上格点数目b的
关系： S = a + b/2 - 1 */
/*==============================================================*\
  4)   三角形的费马点
\*==============================================================*/
//到三顶点的距离和最短，费马点
/*当三角形最大的顶角小于120度的时候，三角形内一点到三顶点之间的距离最小是与三顶点夹角都成
120度的点P；当最到顶点大于等于120度，该顶点取最小值
补充一下，当三角形的最大角小于120度时，费尔码点在三角形内，作法有多种，可以从任二边向外作等
边三角形，联接正三角形的顶点和原三角形的对角，两者联线交点即所求。*/
/*==============================================================*\
  5)   旋转卡壳求凸包直径 O(N)
\*==============================================================*/
//返回值凸包直径的平方（最远两点距离的平方）
double rotating(cpoint cp[], int n){
    int i = 1; double res = 0.0;
    cp[n] = cp[0];
    for(int j = 0; j < n; j ++) {
        while(dcmp( fabs(cross(cp[i + 1], cp[j], cp[j + 1])) -
            fabs(cross(cp[i], cp[j], cp[j + 1])) ) > 0)
            i = (i + 1) % n;
        //cp[i]和cp[j],cp[i + 1]和cp[j + 1]可能是对踵点
        res = max(res, max(dissqr(cp[i], cp[j]),
            dissqr(cp[i + 1], cp[j + 1])));
    }
    return res;
}
/*==============================================================*\
  6)   旋转卡壳求两凸包最短距离 O(N)
\*==============================================================*/
double PointToSeg(cpoint p0, cpoint p1, cpoint p2) {//点到线段距离
    cpoint cp;
    double d = PointToLine(p0, p1, p2, cp);
    if (PointOnSegment(cp, p1, p2)) return d;
    else return min(dis(p0, p1), dis(p0, p2));
}
//两平行线段距离
double DisPallSeg(cpoint p0,cpoint p1,cpoint p2, cpoint p3) {
    return min( min(PointToSeg(p0, p2, p3),PointToSeg(p1, p2, p3)),
                min(PointToSeg (p2, p0, p1), PointToSeg(p3, p0, p1)) );
}
void anticlockwise(cpoint cp[], int n) {
    for (int i = 0; i < n - 2; ++i) {
        double t = cross(cp[i], cp[i + 1], cp[i + 2]);
        if (dcmp(t) > 0) return ;
        if (dcmp(t) < 0) {
            reverse(cp, cp + n);
            return;
        }
    }
}
// 旋转卡壳，两凸包必须逆时针，并且需要把两凸包交换再做一遍
double rotating(cpoint ch1[], int n, cpoint ch2[], int m) {
    int p = 0, q = 0;
    for (int i = 0; i < n; ++i)
        if (dcmp(ch1[i].y - ch1[p].y) < 0)
            p = i;
    for (int i = 0; i < m; ++i)
        if (dcmp(ch2[i].y - ch2[q].y) > 0)
            q = i;
    ch1[n] = ch1[0];
    ch2[m] = ch2[0];
    double tmp, res = 1e99;
    for (int i = 0; i < n; ++i) {
        while ((tmp = cross(ch1[p], ch1[p + 1], ch2[q + 1]) -
            cross(ch1[p], ch1[p + 1], ch2[q])) > eps)
            q = (q + 1) % m;
        if (dcmp(tmp) < 0)
            res = min(res, PointToSeg(ch2[q], ch1[p], ch1[p + 1]));
        else
            res = min(res,DisPallSeg(ch1[p],ch1[p+1],ch2[q],ch2[q+1]));
        p = (p + 1) % n;
    }
    return res;
}
double solve() {
    anticlockwise(ch1, n);
    anticlockwise(ch2, m);
    return min(rotating(ch1, n, ch2, m), rotating(ch2, m, ch1, n));
}
/*==================================================================*\
  7)   半平面交 O(N * log(N))
\*==================================================================*/
struct cvector {
    cpoint s, e;
    double ang, d;
};
void setline(double x1,double y1,double x2,double y2,cvector &v) {
    v.s.x = x1; v.s.y = y1;
    v.e.x = x2; v.e.y = y2;
    v.ang = atan2(y2 - y1, x2 - x1);
    if (dcmp(x1 - x2))v.d = (x1 * y2 - x2 * y1) / fabs(x1 - x2);
    else               v.d = (x1 * y2 - x2 * y1) / fabs(y1 - y2);
```

```
}
bool parallel(const cvector &a, const cvector &b) { //判向量平行
   double u = (a.e.x - a.s.x) * (b.e.y - b.s.y)
         - (a.e.y - a.s.y) * (b.e.x - b.s.x);
   return dcmp(u) == 0;
}

//求两向量(直线)交点(两向量不能平行或重合)
cpoint CrossPoint(const cvector &a, const cvector &b) {
   cpoint res;
   double u = cross(a.s, a.e, b.s), v = cross(a.e, a.s, b.e);
   res.x = (b.s.x * v + b.e.x * u) / (u + v);
   res.y = (b.s.y * v + b.e.y * u) / (u + v);
   return res;
}
//半平面交排序函数[优先顺序: 1.极角2.前面的直线在后面的左边]
bool VecCmp(const cvector &l, const cvector &r) {
   if (dcmp(l.ang - r.ang)) return l.ang < r.ang;
   return l.d < r.d;
}

cvector deq[N]; //用于计算的双端队列

//      获取半平面交的多边形（多边形的核）
//      注意:1.半平面在向量左边, 2.函数会改变vec[]中的值
//函数运行后如果n[即返回多边形的点数量]为0则
//不存在半平面交的多边形（不存在区域或区域面积无穷大）
void HalfPanelCross(cvector vec[], int n, cpoint cp[], int &m) {
   int i, tn; m = 0;
   sort(vec, vec + n, VecCmp);
   for (i = tn = 1; i < n; ++i) //平面在向量左边的筛选
      if(dcmp(vec[i].ang - vec[i - 1].ang))
         vec[tn++] = vec[i];
   n = tn;
   int bot = 0, top = 1;
   deq[0] = vec[0];
   deq[1] = vec[1]; // vec[]大小不可小于2
   for (i = 2; i < n; ++i) {
      if (parallel(deq[top], deq[top - 1]) ||
         parallel(deq[bot], deq[bot + 1]) ) return ;
      while ( bot < top && dcmp( cross(vec[i].s, vec[i].e,
         CrossPoint(deq[top], deq[top - 1])) ) < 0 )
         top--;
      while ( bot < top && dcmp( cross(vec[i].s, vec[i].e,
         CrossPoint(deq[bot], deq[bot + 1])) ) < 0 )
         bot++;
      deq[++top] = vec[i];
   }
   while ( bot < top && dcmp( cross(deq[bot].s, deq[bot].e,
      CrossPoint(deq[top], deq[top - 1])) ) < 0 )
      top--;
   while ( bot < top && dcmp( cross(deq[top].s, deq[top].e,
      CrossPoint(deq[bot], deq[bot + 1])) ) < 0 )
      bot++;
   if (top <= bot + 1) return ; // 两条或两条以下的直线，面积无穷大
   for (i = bot; i < top; ++i)
      cp[m++] = CrossPoint(deq[i], deq[i + 1]);
   if (bot < top + 1)
      cp[m++] = CrossPoint(deq[bot], deq[top]);
   for (i = 0; i < m; ++i) {
      if (dcmp(cp[i].x) == 0) cp[i].x = 0;
      if (dcmp(cp[i].y) == 0) cp[i].y = 0;
   }
}

/**********************传入的半平面为方程时需要*************************/
// 1 = 构造向量成功， 0 = 无解，2 = 有无穷解
int makevec(double a,double b,double c,cvector &v) {//ax + by + c>=0
   if (dcmp(b) > 0)
      setline(0, -c / b, 1, -(a + c) / b, v);
   else if (dcmp(b) < 0)
      setline(1, -(a + c) / b, 0, -c / b, v);
   else if (dcmp(a) > 0)
      setline(-c / a, 0, -c / a, -1, v);
   else if (dcmp(a) < 0)
      setline(-c / a, 0, -c / a, 1, v);
   else if (dcmp(c) >= 0)
      return 2;
   else
      return 0;
   return 1;
}
/**********************传入的半平面为方程时需要*************************/

int n, m;
cvector v[N];
cpoint cp[N];
void solve() {
   scanf("%d", &n);
   double x1, x2, y1, y2;
   for (int i = 0; i < n; ++i) {
      scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
      setline(x1, y1, x2, y2, v[i]);
   }
   double high = 10000.0;
   double low = 0.0;
```

```
    setline(low, low, high, low, v[n++]);
    setline(high, low, high, high, v[n++]);
    setline(high, high, low, high, v[n++]);
    setline(low, high, low, low, v[n++]);

    HalfPanelCross(v, n, cp, m);
    if (m < 3)
       printf("0.0\n");
    else {
       cp[m] = cp[0];
       double area = 0;
       for (int i = 0; i < m; ++i)
          area += cp[i].x * cp[i + 1].y - cp[i].y * cp[i + 1].x;
       printf("%.1lf\n", area / 2);
    }
}
/*==========================================================================*\
  8)   最小圆覆盖 (随机增量法 O(N))
\*==========================================================================*/
void center(cpoint p0, cpoint p1, cpoint p2, cpoint &cp) {//三角形外心
    double a1=p1.x-p0.x, b1=p1.y-p0.y, c1=(sqr(a1)+sqr(b1))/2;
    double a2=p2.x-p0.x, b2=p2.y-p0.y, c2=(sqr(a2)+sqr(b2))/2;
    double d = a1 * b2 - a2 * b1;
    cp.x = p0.x + (c1 * b2 - c2 * b1) / d;
    cp.y = p0.y + (a1 * c2 - a2 * c1) / d;
}
void MinCir(cpoint cp[], int n, cpoint &c, double &r) {
    random_shuffle(cp, cp + n);
    c = cp[0]; r = 0;
    for (int i = 1; i < n; ++i) {
       if (dcmp(dis(cp[i], c) - r) <= 0) continue;
       c = cp[i]; r = 0;
       for (int j = 0; j < i; ++j) {
          if (dcmp(dis(cp[j], c) - r) <= 0) continue;
          c.x = (cp[i].x + cp[j].x) / 2;
          c.y = (cp[i].y + cp[j].y) / 2;
          r = dis(c, cp[j]);
          for (int k = 0; k < j; ++k) {
             if (dcmp(dis(cp[k], c) - r) <= 0) continue;
             center(cp[i], cp[j], cp[k], c);
             r = dis(c, cp[k]);
          }
       }
    }
}
/*==========================================================================*\
  9)   最近圆对(二分 + 扫描线)
\*==========================================================================*/
```

```
const int N = 50010;
const double eps = 1e-8;
int n, lft[N], rht[N];  set<int> s;
struct circle {  double x, y, r; }cir[N];

int dcmp(double x) {if (x < -eps) return -1; else return x > eps;}
bool cmp(circle a, circle b) { return a.y < b.y; }
bool cmp2(int i, int j){return cir[i].x-cir[i].r < cir[j].x-cir[j].r;}
bool cmp3(int i, int j) {return cir[i].x+cir[i].r < cir[j].x+cir[j].r;}

double dis(circle a, circle b) {
   return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y)) - a.r - b.r;
}
bool inter(circle a, circle b, double delta) {
   return dcmp(dis(a, b) - delta * 2) < 0;
}
bool insert(int i, double mid) {
   set<int>::iterator it = s.insert(lft[i]).first;
   if (it != s.begin()) {
      if (inter(cir[*--it], cir[lft[i]], mid))
         return false;
      ++it;
   }
   if (++it != s.end() && inter(cir[*it], cir[lft[i]], mid))
      return false;
   return true;
}
bool remove(int j, double mid) {
   set<int>::iterator it = s.find(rht[j]);
   if (it != s.begin() && it != --s.end()) {
      int a = *--it;  ++it;
      int b = *++it;
      if (inter(cir[a], cir[b], mid))
         return false;
   }
   s.erase(rht[j]);
   return true;
}
bool check(double mid) {
   s.clear();
   for (int i = 0, j = 0; i < n || j < n;) {
      if (j == n) {
         if (!insert(i++, mid)) return false;
      } else if (i == n) {
         if (!remove(j++, mid)) return false;
      } else if (dcmp(cir[lft[i]].x - cir[lft[i]].r - mid
         - cir[rht[j]].x - cir[rht[j]].r - mid) <= 0) {
            if (!insert(i++, mid)) return false;
```

```
        } else {
            if (!remove(j++, mid)) return false;
        }
    } return true;
}

void solve() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%lf%lf%lf", &cir[i].x, &cir[i].y, &cir[i].r);
        lft[i] = rht[i] = i;
    }
    sort(cir, cir + n, cmp);
    sort(lft, lft + n, cmp2);
    sort(rht, rht + n, cmp3);
    double mid, low = 0, high = dis(cir[0], cir[1]) / 2;
    while (dcmp(high - low)) {
        mid = (high + low) / 2;
        if (check(mid)) low = mid;
        else high = mid;
    }
    printf("%lf\n", low + high);
}
/*==================================================================================*\
  10)  圆和多边形面积的交
\*==================================================================================*/
double r;   int n; //多边形点数

struct cpoint {
   double x, y;
   cpoint(){}
   cpoint(double x, double y) : x(x), y(y) {}
}pin[N]; //需要初始化多边形

int dcmp(double x) {
   if (x < -eps) return -1; else return x > eps;
}

void gao(cpoint u, cpoint v, double r, vector<cpoint>& ret) {
   ret.push_back(u);
   double a = sqr(v.x - u.x) + sqr(v.y - u.y);
   double b = 2 * ((v.x - u.x) * u.x + (v.y - u.y) * u.y);
   double c = sqr(u.x) + sqr(u.y) - r * r;
   double d = b * b - 4 * a * c;
   if (d < 0) return;   d = sqrt(d);
   double t1 = (-b + d) / (2 * a);
   double t2 = (-b - d) / (2 * a);
   if (t1 > t2) swap(t1, t2);
```

```cpp
   if (dcmp(t1) > 0 && dcmp(1 - t1) > 0)
      ret.push_back( cpoint(u.x+(v.x-u.x)*t1, u.y+(v.y-u.y)*t1));
   if (dcmp(t2) > 0 && dcmp(1 - t2) > 0 && dcmp(t2 - t1) > 0)
      ret.push_back( cpoint(u.x+(v.x-u.x)*t2, u.y+(v.y-u.y)*t2));
}

double tri(const cpoint& u, const cpoint& v) {
   return u.x * v.y - u.y * v.x;
}

double arc(const cpoint& u, const cpoint& v, double r) {
   double t = atan2(v.y, v.x) - atan2(u.y, u.x);
   while (t > pi) t -= 2 * pi;
   while (t < -pi) t += 2 * pi;
   return r * r * t;
}

// 圆和多边形的公共面积(圆心在原点)
double area(double r, cpoint pin[], int n) {
   double ans = 0; pin[n] = pin[0];
   vector<cpoint> v;
   for (int i = 0; i < n; ++i)
      gao(pin[i], pin[i + 1], r, v);
   v.push_back(v.front());
   for (int i = 1; i < (int)v.size(); ++i) {
      if (sqrt( sqr((v[i - 1].x + v[i].x) / 2)
            + sqr((v[i - 1].y + v[i].y) / 2) ) < r)
         ans += tri(v[i - 1], v[i]);
      else
         ans += arc(v[i - 1], v[i], r);
   } return fabs(ans / 2);
}
/*==========================================================================*\
  11) 圆的面积并和交 O(N^2*log(N))
\*==========================================================================*/
const int N = 1010;
const double eps = 1e-8;
const double pi = acos(-1.0);
double area[N]; // area[i]记录了覆盖i层的面积
int n;
int dcmp(double x) {
   if (x < -eps) return -1; else return x > eps;
}
struct cp {
   double x, y, r, angle; int d; // d表示层数
   cp(){}
   cp(double xx, double yy, double ang = 0, int t = 0) {
      x = xx;  y = yy;  angle = ang;  d = t;
   }
   void get() {
      scanf("%lf%lf%lf", &x, &y, &r);
      d = 1; // 注意每个圆的层数要初始化为1
   }
}cir[N], tp[N * 2];
double dis(cp a, cp b) {
   return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}
double cross(cp p0, cp p1, cp p2) {
   return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) * (p2.x - p0.x);
}
int CirCrossCir(cp p1,double r1, cp p2,double r2, cp &cp1, cp &cp2){
   double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx * mx;
   double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my * my;
   double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) * (sq - sqr(r1 + r2));
   if (d + eps < 0) return 0; if (d < eps) d = 0; else d = sqrt(d);
   double x = mx * ((r1 + r2) * (r1 - r2) + mx * sx) + sx * my2;
   double y = my * ((r1 + r2) * (r1 - r2) + my * sy) + sy * mx2;
   double dx = mx * d, dy = my * d; sq *= 2;
   cp1.x = (x - dy) / sq; cp1.y = (y + dx) / sq;
   cp2.x = (x + dy) / sq; cp2.y = (y - dx) / sq;
   if (d > eps) return 2; else return 1;
}

bool circmp(const cp& u, const cp& v) {
   return dcmp(u.r - v.r) < 0;
}

bool cmp(const cp& u, const cp& v) {
   if (dcmp(u.angle - v.angle)) return u.angle < v.angle;
   return u.d > v.d;
}

double calc(cp cir, cp cp1, cp cp2) {
   double ans = (cp2.angle - cp1.angle) * sqr(cir.r)
      - cross(cir, cp1, cp2) + cross(cp(0, 0), cp1, cp2);
   return ans / 2;
}

void CirUnion(cp cir[], int n, double area[]) {
   memset(area + 1, 0, sizeof(double) * n);
   cp cp1, cp2;
   sort(cir, cir + n, circmp);
   for (int i = 0; i < n; ++i)
      for (int j = i + 1; j < n; ++j)
         if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) <= 0)
            cir[i].d++;
   for (int i = 0; i < n; ++i) {
      int tn = 0, cnt = 0;
      for (int j = 0; j < n; ++j) {
         if (i == j) continue;
         if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r,
            cp2, cp1) < 2) continue;
         cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
         cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
         cp1.d = 1;    tp[tn++] = cp1;
         cp2.d = -1;   tp[tn++] = cp2;
         if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
      }
      tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
      tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, -pi, cnt);
      sort(tp, tp + tn, cmp);
      int s = cir[i].d + tp[0].d;
      for (int j = 1; j < tn; ++j) {
         area[s] += calc(cir[i], tp[j - 1], tp[j]);
         s += tp[j].d;
      }
   }
}

void solve() {
   for (int i = 0; i < n; ++i)
      cir[i].get();
   CirUnion(cir, n, area);
   for (int i = 1; i <= n; ++i) {
      area[i] -= area[i + 1];
      printf("[%d] = %.3lf\n", i, area[i]);
   }
}

/*==========================================================================*\
 12)  凸多边形面积并O(N^2*log(N))
\*==========================================================================*/
// 注意 cpoint 重载 "==" 和 "<"
#define forn(i, n) for(int i = 0; i < (int)(n); ++i)
#define MP make_pair
#define SZ(a) (int)(a.size())

const int N = 510;
const int M = 6;

struct poly {
   int n;
   cpoint cp[M];
   void get() {
//    scanf("%d", &n);
      n = 4;
      forn (i, n) cp[i].get();
   }
   bool check() {
      cp[n] = cp[0];
      double area = 0;
      forn (i, n) area += cp[i] * cp[i + 1];
      if (dcmp(area) == 0) return false;
      if (area < 0) reverse(cp, cp + n);
      cp[n] = cp[0];
      return true;
   }
}ply[N];

int n;
typedef pair<cpoint, cpoint> segment;
#define line cpoint
segment seg1[N * M], seg2[N * M];
line lin[N * M]; //line借用cpoint结构分别用x,y表示直线y = kx + b中的k,b

line getline(cpoint u, cpoint v) {
   double k;
   if (dcmp(u.x - v.x) == 0) k = 1e200;
   else k = (u.y - v.y) / (u.x - v.x);
   return line(k, u.y - k * u.x);
}

bool getcut(line lin, cpoint a, cpoint b, cpoint& cp) {
   double t = lin.x * (a.x - b.x) - (a.y - b.y);
   if (dcmp(t) == 0) return false;
   double x = ((a * b) - (a.x - b.x) * lin.y) / t;
   cp = cpoint(x, lin.x * x + lin.y);
   return true;
}

double calc(segment seg[], int m) {
   int ln = 0; double ans = 0;
   cpoint A, B, cp;
   forn (i, m) lin[ln++] = getline(seg[i].first, seg[i].second);
   sort(lin, lin + ln);
   ln = unique(lin, lin + ln) - lin;

   forn (i, ln) {
      vector<pair<double, int> > mark;
      forn (j, n) {
         bool touch = 0;
         forn (k, ply[j].n)
            if (lin[i] == getline(ply[j].cp[k], ply[j].cp[k + 1])) {
               touch = 1; break;
            }
         if (touch) continue; // 共线
         vector<cpoint> cut;
         forn (k, ply[j].n) {
            A = ply[j].cp[k];
            B = ply[j].cp[k + 1];
            if (!getcut(lin[i], A, B, cp)) continue;
            if (dcmp((A - cp) ^ (B - cp)) <= 0)
               cut.push_back(cp);
         }
         sort(cut.begin(), cut.end());
         cut.resize(unique(cut.begin(), cut.end()) - cut.begin());
         if (SZ(cut) == 2) {
            mark.push_back(MP(cut[0].x, 0));
            mark.push_back(MP(cut[1].x, 1));
         }
      }

      forn (j, m)
         if (lin[i] == getline(seg[j].first, seg[j].second)) {
            double mn = min(seg[j].first.x, seg[j].second.x);
            double mx = max(seg[j].first.x, seg[j].second.x);
            mark.push_back(MP(mn, 2));
            mark.push_back(MP(mx, 3));
         }

      sort(mark.begin(), mark.end());

      double last = mark[0].first;
      int in = 0, ct = 0;
      forn (j, SZ(mark)) {
         double y0 = lin[i].x * last + lin[i].y;
         double y1 = lin[i].x * mark[j].first + lin[i].y;
         if (!in && ct)
            ans += (y0 + y1) * (mark[j].first - last) / 2;
         last = mark[j].first;
         if (mark[j].second == 0) in++;
         if (mark[j].second == 1) in--;
         if (mark[j].second == 2) ct++;
         if (mark[j].second == 3) ct--;
      }
   }
   return ans;
}
```

```
double PolyUnion(poly ply[], int n) {
   int n1, n2, tot = n;
   n1 = n2 = n = 0;
   forn (i, tot) if (ply[i].check())
      ply[n++] = ply[i]; // 去除共线多边形
   forn (i, n) forn (j, ply[i].n) {
      cpoint A = ply[i].cp[j];
      cpoint B = ply[i].cp[j + 1];
      if (dcmp(A.x - B.x) > 0) seg1[n1++] = MP(A, B);
      if (dcmp(A.x - B.x) < 0) seg2[n2++] = MP(A, B);
   }
   return calc(seg1, n1) - calc(seg2, n2);
}
/*============================================================================*\
  13)  三维几何基本操作
\*============================================================================*/
struct vpoint {
   double x, y, z;
   vpoint(){}
   vpoint(double x, double y, double z) : x(x), y(y), z(z) {}
   vpoint operator + (const vpoint &u) const {
      return vpoint(x + u.x, y + u.y, z + u.z);
   }
   vpoint operator - (const vpoint &u) const {
      return vpoint(x - u.x, y - u.y, z - u.z);
   }
   vpoint operator * (const double &s) const {
      return vpoint(x * s, y * s, z * s);
   }
   vpoint operator / (const double &s) const {
      return vpoint(x / s, y / s, z / s);
   }
   vpoint operator * (const vpoint &u) const { // 叉积
      return vpoint(
            y * u.z - z * u.y,
            z * u.x - x * u.z,
            x * u.y - y * u.x  );
   }
   double operator ^ (const vpoint &u) const { // 点积
      return x * u.x + y * u.y + z * u.z;
   }

   vpoint trunc(double s) { // 向量长度变为s
      double r = s / len();
      return vpoint(x * r, y * r, z * r);
   }
   double len() { return sqrt(x * x + y * y + z * z); }
   void get() { scanf("%lf%lf%lf", &x, &y, &z); }
   double angle(vpoint v) { // 计算转到v向量的夹角
      return acos((*this ^ v) / (len() * v.len()));
   }

   bool operator == (const vpoint &u) const {
      return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0
            && dcmp(z - u.z) == 0;
   }
   bool operator < (const vpoint &u) const {
      if (dcmp(x - u.x)) return x < u.x;
      if (dcmp(y - u.y)) return y < u.y;
      return dcmp(z - u.z) < 0;
   }
};

// 三角形面积*2
double area(vpoint a, vpoint b, vpoint c) {
   return ((b - a) * (c - a)).len();
}
// 四面体有向体积*6 ( a在平面bcd的正向时为正(右手定则) )
double volume(vpoint a, vpoint b, vpoint c, vpoint d) {
   return (b - a) * (c - a) ^ (d - a);
}
// 判四点共面
bool onplane(vpoint a, vpoint b, vpoint c, vpoint d) {
   return dcmp((b - a) * (c - a) ^ (d - a)) == 0;
}



/*********************点线关系*********************/

// 判点p在直线AB上 (没用过但比较可靠)
bool PointOnLine(vpoint p, vpoint A, vpoint B) {
   return dcmp( ((A - p) * (B - p)).len() ) == 0;
}
// 判点p在线段AB上 (没用过但比较可靠)
bool PointOnSeg(vpoint p, vpoint A, vpoint B) {
   return dcmp( ((A - p) * (B - p)).len() ) == 0
         && dcmp( (A - p) ^ (B - p) ) <= 0;
}
// 点p在直线AB上的射影
vpoint PointProjLine(vpoint p, vpoint A, vpoint B) {
   double t = ((p - A) ^ (B - A)) / ((B - A) ^ (B - A));
   return A + (B - A) * t;
}
```

```
/*********************线线关系*********************/

// 异面直线AB和CD的距离 (没用过但比较可靠)
double dist(vpoint A, vpoint B, vpoint C, vpoint D) {
   vpoint n = (B - A) * (D - C);
   return fabs(n ^ (C - A)) / n.len();
}

// 判定直线AB和CD位置关系，并求交点  (没用过)
// -2:异面  -1:重合  0:平行  1:相交
int LineInter(vpoint A, vpoint B, vpoint C, vpoint D, vpoint& tp) {
   if (dcmp( (B - A) * (C - A) ^ (D - A) )) return -2;
   vpoint v = (B - A) * (D - C);
   vpoint u = (C - A) * (D - C);
   double t;
   if (dcmp(v.z))     t = u.z / v.z;
   else if (dcmp(v.x)) t = u.x / v.x;
   else if (dcmp(v.y)) t = u.y / v.y;
   else if (dcmp( ((C - A) * (C - B)).len() )) return 0;
   else return -1;
   tp = A + (B - A) * t;
   return 1;
}

// 判定线段AB和CD位置关系，并求交点 (没用过)
// -2:异面  -1:共线且有重叠  0:不相交  1:相交
int SegmentInter(vpoint A, vpoint B, vpoint C, vpoint D, vpoint& tp)
{
   int ret = LineInter(A, B, C, D, tp);
   if (ret == -2 || ret == 0) return ret;
   if (ret == 1)
      return PointOnSeg(tp, A, B) && PointOnSeg(tp, C, D);
   if (ret == -1 && (PointOnSeg(A, C, D) || PointOnSeg(B, C, D)
         || PointOnSeg(C, A, B) || PointOnSeg(D, A, B) ))
      return -1;
   return 0;
}




/*********************点面关系*********************/

// 点A到平面n.x * x + n.y * y + n.z * z + d = 0的距离
double PointToPlane(vpoint A, vpoint n, double d) {
   return fabs((A ^ n) + d) / n.len();
}
```

```
// 点A在平面n.x*(x-o.x)+n.y*(y-o.y)+n.z*(z-o.z) = 0上的射影(点法式)
vpoint PointProjPlane(vpoint A, vpoint n, vpoint o) {
   double t = ((A - o) ^ n) / (n ^ n);
   return A - n * t;
}
```

```
/**********************线面关系**********************/

// 直线AB和平面n.x*x+n.y*y+n.z*z+d=0的交点(n为法向量)
// -1:直线在平面上   0:没有交点(平行)    1:有交点
int LineInterPlane(vpoint A, vpoint B, vpoint n,
      double d, vpoint &vp) {
   B = B - A;
   double s = (A ^ n) + d;
   double t = n ^ B;
   if (dcmp(t) == 0)  // 法向量和直线垂直
      return dcmp(s) ? 0 : -1;
   vp = A - B * (s / t);
   return 1;
}
```

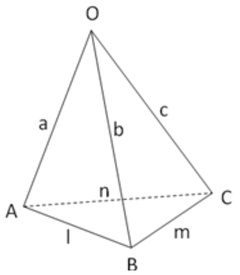
```
/**********************面面关系**********************/

// 求平面 n1.x*x+n1.y*y+n1.z*z+d1=0 和
// 平面 n2.x*x+n2.y*y+n2.z*z+d2=0 的 交线AB (没用过)
// -1:重合    0:平行    1:相交
int PlaneInter(vpoint n1, double d1, vpoint n2, double d2,
      vpoint& A, vpoint& B) {
   vpoint v = n1 * n2;
   vpoint n = n1 * d2 - n2 * d1;
   if (dcmp(v.z))     A = vpoint(n.y/v.z, -n.x/v.z, 0);
   else if (dcmp(v.x)) A = vpoint(0, n.z/v.x, -n.y/v.x);
   else if (dcmp(v.y)) A = vpoint(-n.z/v.y, 0, n.x/v.y);
   else if (dcmp( n.len() )) return 0;
   else return -1;
   B = A + v.trunc(10.0); // 改变v值以免B点坐标过大
   return 1;
}
```

```
/********************旋转********************/
// 点p绕向量AB旋转ang角度后的坐标(旋转按照右手定则)
vpoint rotate(vpoint p, vpoint A, vpoint B, double ang) {
   vpoint o = PointProjLine(p, A, B);
   vpoint r = p - o;
   vpoint e = (B - A).trunc(1.0) * r;
   return r * cos(ang) + e * sin(ang) + o;
}
/*================================================================================*\
  14)  三维凸包 O(N^2)
\*================================================================================*/
/****************************************
  Name: 3D Convex Hull
  Author: Isun
  Date: 1-10-10 17:20
  Description: 求三维空间点集凸包——增量法O(n^2)
****************************************/
#define eps 1e-7
#define MAXV 305

struct pt{ //三维点
   double x, y, z;
   pt(){}
   pt(double _x, double _y, double _z): x(_x), y(_y), z(_z){}
   pt operator - (const pt p1){return pt(x - p1.x, y - p1.y, z - p1.z);}
   pt operator * (pt p) {    //叉乘
      return pt(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-y*p.x);
   }
   double operator ^ (pt p){return x*p.x+y*p.y+z*p.z;}  //点乘
};

struct _3DCH{
   struct fac{
      int a, b, c;   //表示凸包一个面上三个点的编号
      bool ok;    //表示该面是否属于最终凸包中的面
   };
   int n;   //初始点数
   pt P[MAXV]; //初始点
   int cnt; //凸包表面的三角形数
   fac F[MAXV*8]; //凸包表面的三角形
   int to[MAXV][MAXV];

   double vlen(pt a){return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}//向量长度
   double area(pt a,pt b,pt c){return vlen((b-a)*(c-a));}//三角形面积*2
   double volume(pt a, pt b, pt c, pt d) {  //四面体有向体积*6
      return (b-a)*(c-a)^(d-a);
   }
```

```
double ptof(pt &p, fac &f){ //正：点在面同向
   pt m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];
   return (m * n) ^ t;
}

void deal(int p, int a, int b){
   int f = to[a][b];
   fac add;
   if (F[f].ok){
      if (ptof(P[p], F[f]) > eps)
         dfs(p, f);
      else{
         add.a = b, add.b = a, add.c = p, add.ok = 1;
         to[p][b] = to[a][p] = to[b][a] = cnt;
         F[cnt++] = add;
      }
   }
}

void dfs(int p, int cur){
   F[cur].ok = 0;
   deal(p, F[cur].b, F[cur].a);
   deal(p, F[cur].c, F[cur].b);
   deal(p, F[cur].a, F[cur].c);
}

bool same(int s, int t){
   pt &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];
   return fabs(volume(a, b, c, P[F[t].a])) < eps
      && fabs(volume(a, b, c, P[F[t].b])) < eps
      && fabs(volume(a, b, c, P[F[t].c])) < eps;
}

//构建三维凸包
void construct(){
   cnt = 0;
   if (n < 4) return;

/*********此段是为了保证前四个点不公面，若已保证，可去掉********/
   bool sb = 1;
   for (int i = 1; i < n; i++){ //使前两点不公点
      if (vlen(P[0] - P[i]) > eps){
         swap(P[1], P[i]);
         sb = 0;   break;
      }
   }  if (sb)return;

   sb = 1;
   for (int i = 2; i < n; i++){ //使前三点不公线
      if (vlen((P[0] - P[1]) * (P[1] - P[i])) > eps){
         swap(P[2], P[i]);
         sb = 0;  break;
      }
   }  if (sb)return;

   sb = 1;
   for (int i = 3; i < n; i++){ //使前四点不共面
      if (fabs((P[0]-P[1]) * (P[1]-P[2]) ^ (P[0]-P[i])) > eps){
         swap(P[3], P[i]);
         sb = 0;   break;
      }
   }  if (sb) return;
   /*********此段是为了保证前四个点不公面********/

   fac add;
   for (int i = 0; i < 4; i++){
      add.a=(i+1)%4, add.b=(i+2)%4, add.c=(i+3)%4, add.ok=1;
      if (ptof(P[i], add) > 0)
         swap(add.b, add.c);
      to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;
      F[cnt++] = add;
   }
   for (int i = 4; i < n; i++){
      for (int j = 0; j < cnt; j++){
         if (F[j].ok && ptof(P[i], F[j]) > eps){
            dfs(i, j);  break;
         }
      }
   }
   int tmp = cnt;   cnt = 0;
   for (int i = 0; i < tmp; i++)
      if (F[i].ok)
         F[cnt++] = F[i];
}
double area(){ //表面积
   double ret = 0.0;
   for (int i = 0; i < cnt; i++)
      ret += area(P[F[i].a], P[F[i].b], P[F[i].c]);
   return ret / 2.0;
}
double volume(){ //体积
   pt O(0, 0, 0);
   double ret = 0.0;
   for (int i = 0; i < cnt; i++)
      ret += volume(O, P[F[i].a], P[F[i].b], P[F[i].c]);
   return fabs(ret / 6.0);
   }
   //表面三角形数
   int facetCnt_tri() { return cnt; }

   int facetCnt(){ //表面多边形数
      int ans = 0;
      for (int i = 0; i < cnt; i++){
         bool nb = 1;
         for (int j = 0; j < i; j++) if (same(i, j)){
            nb = 0; break;
         } ans += nb;
      } return ans;
   }
};
_3DCH hull; //内有大数组，不易放在函数内
int main() {
   while (~scanf("%d", &hull.n)){
      for (int i = 0; i < hull.n; i++)
         scanf("%lf%lf%lf", &hull.P[i].x, &hull.P[i].y,
&hull.P[i].z);
      hull.construct();
      printf("%d\n", hull.facetCnt());
   } return 0;
}
```

```
/*==============================================================================*\
  15)  欧拉四面体公式
\*==============================================================================*/
```

已知四面体的六条棱长，求体积：



$$V^2 = \frac{1}{36}\begin{vmatrix} a^2 & \frac{a^2+b^2-n^2}{2} & \frac{a^2+c^2-m^2}{2} \\ \frac{a^2+b^2-n^2}{2} & b^2 & \frac{b^2+c^2-l^2}{2} \\ \frac{a^2+c^2-m^2}{2} & \frac{b^2+c^2-l^2}{2} & c^2 \end{vmatrix}$$

```
#define sqr(x)  ((x)*(x))
double V(double a,double b,double c,double l,double m,double n) {
   double abl = a*a + b*b - l*l;
   double bcm = b*b + c*c - m*m;
   double can = c*c + a*a - n*n;
   return sqrt(sqr(2*a*b*c) + abl*bcm*can
      - sqr(a*bcm) - sqr(b*can) - sqr(c*abl)) / 12;
}
```

7． 其他

```
/*================================================================================*\
  1)    读入输出优化
\*================================================================================*/
int readint(){  //用于整数
    char c;
    while (c = getchar(), '-' != c && !isdigit(c))
      if(c == EOF)return EOF;
    int f = 1;
    if ('-' == c) {
        f = -1;
        c = getchar();
    }
    int x = c - '0';
    while (isdigit(c = getchar())) {
        x = x * 10 + c - '0';
    }
    return x * f;
}
void write(int a){   //用于正整数
    if(a>9)write(a/10);
    putchar(a%10+'0');
}
double readdouble() {  //用于double型的读入
    char c;
    while (c = getchar(), c != '-' && c != '.' && !isdigit(c));
    int f = 1;
    if (c == '-') {
        f = -1;
        c = getchar();
    }
    double p = 1, res = 0;
    if (c != '.') {
        res = c - '0';
        while (isdigit(c = getchar())) {
            res = res * 10 + c - '0';
        }
    }
    if (c != '.') return res * f;
    while (isdigit(c = getchar())) {
        res = res * 10 + c - '0';
        p *= 10;
    }
    return res * f / p;
}
```