ACM Standard Code Library

Huang Wei

Software Engineering

Computer and Software College

Hangzhou Dianzi University

October, 2008

ACM 算法模板集

Contents

4. 单源最短路径(Dijkstra 算法)
5. 全源最短路径(Folyd 算法)
6. 拓扑排序
7. 网络预流和最大流
8. 网络最小费用最大流
9. 网络最大流(高度标号预流推进)
10. 最大团
11. 最大匹配(Hungary，HopcroftKarp 算法)
12. 带权二分图最优匹配(KM 算法)
13. 强连通分量(Kosaraju 算法)
14. 强连通分量(Gabow 算法)
15. 无向图割边割点和双连通分量
16. 最小树形图 O(N^3)
17. 最小树形图 O(VE)

六.几何算法

1. 几何模板
2. 球面上两点最短距离
3. 三点求圆心坐标
4. 三角形几个重要的点

七.专题讨论

1. 树状数组
2. 字典树
3. 后缀树
4. 线段树
5. 并查集
6. 二叉堆
7. 逆序数(归并排序)
8. 树状 DP
9. 欧拉路
10. 八数码
11. 高斯消元法
12. 字符串匹配(KMP 算法)
13. 全排列,全组合
14. 二维线段树
15. 稳定婚姻匹配
16. 后缀数组
17. 左偏树
18. 标准 RMQ-ST
19. 度限制最小生成树
20. 最优比率生成树(0/1 分数规划)
21. 最小花费置换
22. 区间 K 大数

第一章 常用函数和 STL

一. 常用函数

```
#include <stdio.h>
int getchar( void );              //读取一个字符, 一般用来去掉无用字符
char *gets( char *str );          //读取一行字符串

#include <stdlib.h>
void * malloc( size_t size );     //动态内存分配, 开辟大小为 size 的空间
void qsort( void *buf, size_t num, size_t size, int (*compare)(const void *, const
void *) );                        //快速排序
Sample:
int compare_ints( const void* a, const void* b )
{
int* arg1 = (int*) a;       int* arg2 = (int*) b;
if( *arg1 < *arg2 ) return -1;
else if( *arg1 == *arg2 ) return 0;
else return 1;
}
int array[] = { -2, 99, 0, -743, 2, 3, 4 };     int array_size = 7;
qsort( array, array_size, sizeof(int), compare_ints );

#include <math.h>
//求反正弦, arg∈[-1, 1], 返回值∈[-pi/2, +pi/2]
double asin( double arg );
//求正弦, arg 为弧度, 弧度=角度*Pi/180.0, 返回值∈[-1, 1]
double sin( double arg );
//求 e 的 arg 次方
double exp( double arg );
//求 num 的对数, 基数为 e
double log( double num );
//求 num 的根
double sqrt( double num );
//求 base 的 exp 次方
double pow( double base, double exp );

#include <string.h>
//初始化内存, 常用来初始化数组
void* memset( void* buffer, int ch, size_t count );
memset( the_array, 0, sizeof(the_array) );
//printf 是它的变形, 常用来将数据格式化为字符串
int sprintf( char *buffer, const char *format, ... );
sprintf(s, "%d%d", 123, 4567); //s="1234567"
```

```
//scanf 是它的变形, 常用来从字符串中提取数据
int sscanf( const char *buffer, const char *format, ... );
Sample:
char result[100]="24 hello", str[100];          int num;
sprintf( result, "%d %s", num,str );//num=24;str="hello" ;
//字符串比较, 返回值<0 代表 str1<str2, =0 代表 str1=str2, >0 代表 str1>str2
int strcmp( const char *str1, const char *str2 );
```

二. 常用 STL

[标准 container 概要]

vector<T>	大小可变的向量, 类似数组的用法, 容易实现删除
list<T>	双向链表
queue<T>	队列, empty(), front(), pop(), push()
stack<T>	栈, empty(), top(), pop(), push()
priority_queue<T>	优先队列, empty(), top(), pop(), push()
set<T>	集合
map<key,val>	关联数组, 常用来作 hash 映射

[标准 algorithm 摘录]

for_each()	对每一个元素都唤起（调用）一个函数
find()	查找第一个能与引数匹配的元素
replace()	用新的值替换元素, O(N)
copy()	复制（拷贝）元素, O(N)
remove()	移除元素
reverse()	倒置元素
sort()	排序, O(N log(N))
partial_sort()	部分排序
binary_search()	二分查找
merge()	合并有序的序列, O(N)

[C++ String 摘录]

copy()	从别的字符串拷贝
empty()	判断字符串是否为空
erase()	从字符串移除元素
find()	查找元素
insert()	插入元素
length()	字符串长度
replace()	替换元素
substr()	取子字符串
swap()	交换字符串

第二章 重要公式与定理

1. Fibonacci Number

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610 …

Formula:

$$F_0 = 1$$

$$F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2}$$

$$F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}} = \left[\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n\right]$$

2. Lucas Number

1, 3, 4, 7, 11, 18, 29, 47, 76, 123...

Formula:

$$L_n = \left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n$$

3. Catalan Number

1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012…

Formula:

$$Cat_n = \frac{C(2n,n)}{n+1}$$

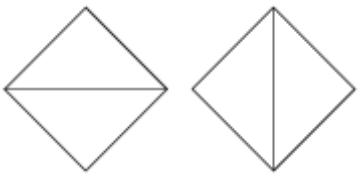
$$Cat_n = \sum_{i=0}^{n-1} Cat_i * Cat_{n-1-i}$$

Application:

1) 将 n + 2 边形沿弦切割成 n 个三角形的不同切割数

Sample:

n = 2;



n = 3;

2) n + 1 个数相乘，给每两个元素加上括号的不同方法数

Sample:

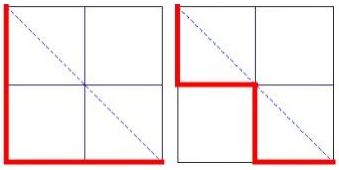
n = 2; (1 (2 3)), ((1 2) 3)
n = 3; (1 (2 (3 4))), (1 ((2 3) 4)) , ((1 2) (3 4)), ((1 (2 3)) 4), (((1 2) 3) 4)

3) n 个节点的不同形状的二叉树数(严《数据结构》P.155)

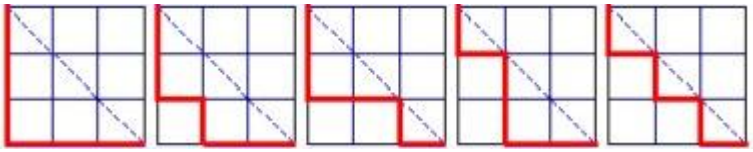
4) 从 n * n 方格的左上角移动到右下角不升路径数

Sample:

n = 2;



n = 3;



4. Stirling Number(Second Kind)

S(n, m)表示含 n 个元素的集合划分为 m 个集合的情况数
或者是 n 个有标号的球放到 m 个无标号的盒子中，要求无一为空，其不同的方案数

Formula:

$$S_{n,m}\begin{cases}0 & (m=0 \ \| \ n<m)\\ S_{n-1,m-1}+m\times S_{n-1,m} & (n>m\geq 1)\end{cases}$$

$$S_{n,m}=\frac{1}{m!}\sum_{i=0}^{m}(-1)^i\times C(m,i)\times(m-i)^n$$

Special Cases:

$$S_{n,0}=0$$

$$S_{n,1}=1$$

$$S_{n,2}=2^{n-1}-1$$

$$S_{n,3}=\frac{1}{6}(3^n-3\times 2^n+3)$$

$$S_{n,n-1}=C(n,2)$$

$$S_{n,n}=1$$

5. Bell Number

n 个元素集合所有的划分数

Formula:

$$B_n=\sum_{i=0}^{n}S_{n,i}$$

6. Stirling's Approximation

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

7. Sum of Reciprocal Approximation

EulerGamma = 0.57721566490153286060651209;

$$\sum_{i=1}^{n}\frac{1}{i} = \ln(n) + EulerGamma \quad (n \to \infty)$$

8. Young Tableau

Young Tableau(杨式图表)是一个矩阵，它满足条件:
如果格子[i, j]没有元素，则[i+1, j]也一定没有元素
如果格子[i, j]有元素 a[i, j],则[i+1, j]要么没有元素，要么 a[i+1, j] > a[i, j]
Y[n]代表 n 个数所组成的杨式图表的个数

Formula:

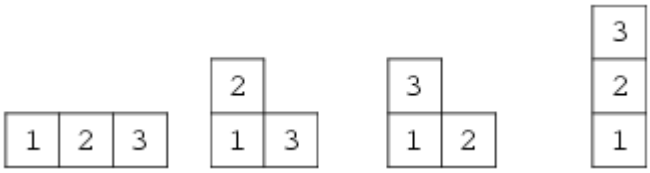
$$Y_1 = 1$$

$$Y_2 = 2$$

$$Y_n = Y_{n-1} + (n-1) \times Y_{n-2} \quad (n > 2)$$

Sample:

n = 3;



9. 整数划分

将整数 n 分成 k 份，且每份不能为空，任意两种分法不能相同

1) 不考虑顺序

```
for(int p=1; p<=n ;p++)
    for(int i=p; i<=n ;i++)
        for(int j=k; j>=1 ;j--)
            dp[i][j] += dp[i-p][j-1];
cout<< dp[n][k] <<endl;
```

2) 考虑顺序

dp[i][j] = dp[i-k][j-1]; (k=1..i)

3) 若分解出来的每个数均有一个上限 m

dp[i][j] = dp[i-k][j-1]; (k=1..m)

10. 错排公式

$D_1 = 0$

$D_2 = 1$

$D_n = (n-1)\times(D_{n-1} + D_{n-2})$

11. 三角形内切圆半径公式

$$p = \frac{a+b+c}{2}$$

$$s = \sqrt{p(p-a)(p-b)(p-c)}$$

$$r = \frac{2s}{a+b+c}$$

12. 三角形外接圆半径公式

$$R = \frac{abc}{4s}$$

13. 圆内接四边形面积公式

$$p = \frac{a+b+c+d}{2}$$

$$s = \sqrt{(p-a)(p-b)(p-c)(p-d)}$$

14. 基础数论公式

1) 模取幂

$a^n \% b = ((((a\%b)*a)\%b)\ldots)\%b$

2) n 的约数的个数

若 n 满足 $n = p_1^{n_1} + p_2^{n_2} + \ldots + p_m^{n_m}$，则 n 的约数的个数为

$(n_1+1)(n_2+1)\ldots(n_m+1)$

第三章 大数模板

```
typedef int hugeint;
//应不大于,以防乘法时溢出
const int Base = 1000;
const int Capacity = 1000;

struct xnum
{
    int Len;
    int Data[Capacity];
    xnum() : Len(0) {}
    xnum(const xnum& V) : Len(V.Len) {
        memcpy(Data, V.Data, Len * sizeof *Data);
    }
    xnum(int V) : Len(0) {
        for (; V > 0; V /= Base) Data[Len++] = V % Base;
    }
    xnum(char S[]);
    xnum& operator=(const xnum& V) {
        Len = V.Len;
        memcpy(Data, V.Data, Len * sizeof *Data);
        return *this;
    }
    int& operator[](int Index) { return Data[Index]; }
    int operator[](int Index) const { return Data[Index]; }

    void print(){
        printf("%d",Len==0?0:Data[Len-1]);
        for(int i=Len-2;i>=0;i--)
            for(int j=Base/10;j>0;j/=10)
                printf("%d",Data[i]/j%10);
    }
};

xnum::xnum(char S[])
{
    int I, J;
    Data[Len = 0] = 0;
    J = 1;
    for (I = strlen(S)-1; I>=0; I--) {
        Data[Len] += (S[I] - '0') * J;
        J *= 10;
        if (J >= Base) J = 1, Data[++Len] = 0;
```

```
    }
    if (Data[Len] > 0) Len++;
}

int compare(const xnum& A, const xnum& B)
{
    int I;
    if (A.Len != B.Len) return A.Len > B.Len ? 1 : -1;
    for (I = A.Len - 1; I >= 0 && A[I] == B[I]; I--);
    if (I < 0) return 0;
    return A[I] > B[I] ? 1 : -1;
}

xnum operator+(const xnum& A, const xnum& B)
{
    xnum R;
    int I;
    int Carry = 0;
    for (I = 0; I < A.Len || I < B.Len || Carry > 0; I++)
    {
        if (I < A.Len) Carry += A[I];
        if (I < B.Len) Carry += B[I];
        R[I] = Carry % Base;
        Carry /= Base;
    }
    R.Len = I;
    return R;
}

xnum operator-(const xnum& A, const xnum& B)
{
    xnum R;
    int Carry = 0;
    R.Len = A.Len;
    int I;
    for (I = 0; I < R.Len; I++)
    {
        R[I] = A[I] - Carry;
        if (I < B.Len) R[I] -= B[I];
        if (R[I] < 0) Carry = 1, R[I] += Base;
        else Carry = 0;
    }
    while (R.Len > 0 && R[R.Len - 1] == 0) R.Len--;
    return R;
```

```
}

xnum operator*(const xnum& A, const int B)
{
    int I;
    if (B == 0) return 0;
    xnum R;
    hugeint Carry = 0;
    for (I = 0; I < A.Len || Carry > 0; I++)
    {
        if (I < A.Len) Carry += hugeint(A[I]) * B;
        R[I] = Carry % Base;
        Carry /= Base;
    }
    R.Len = I;
    return R;
}

xnum operator*(const xnum& A, const xnum& B)
{
    int I;
    if (B.Len == 0) return 0;
    xnum R;
    for (I = 0; I < A.Len; I++)
    {
        hugeint Carry = 0;
        for (int J = 0; J < B.Len || Carry > 0; J++)
        {
            if (J < B.Len) Carry += hugeint(A[I]) * B[J];
            if (I + J < R.Len) Carry += R[I + J];
            if (I + J >= R.Len) R[R.Len++] = Carry % Base;
            else R[I + J] = Carry % Base;
            Carry /= Base;
        }
    }
    return R;
}

xnum operator/(const xnum& A, const int B)
{
    xnum R;
    int I;
    hugeint C = 0;
    for (I = A.Len - 1; I >= 0; I--)
```

```
    {
        C = C * Base + A[I];
        R[I] = C / B;
        C %= B;
    }
    R.Len = A.Len;
    while (R.Len > 0 && R[R.Len - 1] == 0) R.Len--;
    return R;
}

//div
xnum operator/(const xnum& A, const xnum& B)
{
    int I;
    xnum R, Carry = 0;
    int Left, Right, Mid;
    for (I = A.Len - 1; I >= 0; I--)
    {
        Carry = Carry * Base + A[I];
        Left = 0;
        Right = Base - 1;
        while (Left < Right)
        {
            Mid = (Left + Right + 1) / 2;
            if (compare(B * Mid, Carry) <= 0) Left = Mid;
            else Right = Mid - 1;
        }
        R[I] = Left;
        Carry = Carry - B * Left;
    }
    R.Len = A.Len;
    while (R.Len > 0 && R[R.Len - 1] == 0) R.Len--;
    return R;
}

//mod
xnum operator%(const xnum& A, const xnum& B)
{
    int I;
    xnum R, Carry = 0;
    int Left, Right, Mid;
    for (I = A.Len - 1; I >= 0; I--)
    {
        Carry = Carry * Base + A[I];
```

```
        Left = 0;
        Right = Base - 1;
        while (Left < Right)
        {
            Mid = (Left + Right + 1) / 2;
            if (compare(B * Mid, Carry) <= 0) Left = Mid;
            else Right = Mid - 1;
        }
        R[I] = Left;
        Carry = Carry - B * Left;
    }
    R.Len = A.Len;
    while (R.Len > 0 && R[R.Len - 1] == 0) R.Len--;
    return Carry;
}

istream& operator>>(istream& In, xnum& V)
{
    char Ch;
    for (V = 0; In >> Ch;)
    {
        V = V * 10 + (Ch - '0');
        if (cin.peek() <= ' ') break;
    }
    return In;
}

ostream& operator<<(ostream& Out, const xnum& V)
{
    int I;
    Out << (V.Len == 0 ? 0 : V[V.Len - 1]);
    for (I = V.Len - 2; I >= 0; I--)
        for (int J = Base / 10; J > 0; J /= 10)
            Out << V[I] / J % 10;
        return Out;
}

xnum gcd(xnum a,xnum b)
{
    if(compare(b,0)==0) return a;
    else return gcd(b,a%b);
}

int div(char *A,int B)
```

```
{
    int I;
    int C = 0;
    int Alen=strlen(A);
    for (I = 0; I <Alen; I++)
    {
        C = C * Base + A[I]-'0';
        C %= B;
    }
    return C;
}

xnum C(int n,int m)
{
    int i;
    xnum sum = 1;

    for(i = n; i >= n-m+1; i --)
        sum = sum*i;
    for(i = 1; i <= m; i ++)
        sum = sum/i;

    return sum;
}

#define MAXN 9999
#define DLEN 4
class BigNum {
private:
    int a[1000];//可以控制大数的位数
    int len; //大数长度
public:
    BigNum() {len = 1;memset(a,0,sizeof(a));}
    BigNum(const int);
    BigNum(const char*);
    BigNum(const BigNum &);
    BigNum &operator=(const BigNum &);
    BigNum operator+(const BigNum &) const;
    BigNum operator-(const BigNum &) const;
    BigNum operator*(const BigNum &) const;
    BigNum operator/(const int    &) const;
    BigNum operator^(const int    &) const;
    int    operator%(const int    &) const;
    bool    operator>(const BigNum & T)const;
```

```
    void print();
};

BigNum::BigNum(const int b) {
    int c,d = b;
    len = 0;
    memset(a,0,sizeof(a));
    while(d > MAXN) {
        c = d - (d / (MAXN + 1)) * (MAXN + 1);
        d = d / (MAXN + 1);    a[len++] = c;
    }
    a[len++] = d;
}
BigNum::BigNum(const char*s) {
    int t,k,index,l,i;
    memset(a,0,sizeof(a));
    l=strlen(s);
    len=l/DLEN;
    if(l%DLEN)len++;
    index=0;
    for(i=l-1;i>=0;i-=DLEN) {
        t=0;k=i-DLEN+1;
        if(k<0)k=0;
        for(int j=k;j<=i;j++)
            t=t*10+s[j]-'0';
        a[index++]=t;
    }
}
BigNum::BigNum(const BigNum & T) : len(T.len) {
    int i;
    memset(a,0,sizeof(a));
    for(i = 0 ; i < len ; i++)a[i] = T.a[i];
}
BigNum & BigNum::operator=(const BigNum & n) {
    len = n.len;
    memset(a,0,sizeof(a));
    int i;
    for(i = 0 ; i < len ; i++)
        a[i] = n.a[i];
    return *this;
}
BigNum BigNum::operator+(const BigNum & T) const {
    BigNum t(*this);
    int i,big;//位数
```

```
    big = T.len > len ? T.len : len;
    for(i = 0 ; i < big ; i++) {
        t.a[i] +=T.a[i];
        if(t.a[i] > MAXN) {
            t.a[i + 1]++;
            t.a[i] -=MAXN+1;
        }
    }
    if(t.a[big] != 0) t.len = big + 1;
    else t.len = big;
    return t;
}
BigNum BigNum::operator-(const BigNum & T) const {
    int i,j,big;
    bool flag;
    BigNum t1,t2;
    if(*this>T) {
        t1=*this;
        t2=T;
        flag=0;
    } else {
        t1=T;
        t2=*this;
        flag=1;
    }
    big=t1.len;
    for(i = 0 ; i < big ; i++) {
        if(t1.a[i] < t2.a[i]) {
            j = i + 1;
            while(t1.a[j] == 0) j++;
            t1.a[j--]--;
            while(j > i) t1.a[j--] += MAXN;
            t1.a[i] += MAXN + 1 - t2.a[i];
        } else t1.a[i] -= t2.a[i];
    }
    t1.len = big;
    while(t1.a[len - 1] == 0 && t1.len > 1) {
        t1.len--;
        big--;
    }
    if(flag)t1.a[big-1]=0-t1.a[big-1];
    return t1;
}
BigNum BigNum::operator*(const BigNum & T) const {
```

```
    BigNum ret;
    int i,j,up;
    int temp,temp1;
    for(i = 0 ; i < len ; i++) {
        up = 0;
        for(j = 0 ; j < T.len ; j++) {
            temp = a[i] * T.a[j] + ret.a[i + j] + up;
            if(temp > MAXN) {
                temp1 = temp - temp / (MAXN + 1) * (MAXN + 1);
                up = temp / (MAXN + 1);
                ret.a[i + j] = temp1;
            } else {
                up = 0;
                ret.a[i + j] = temp;
            }
        }
        if(up != 0)
            ret.a[i + j] = up;
    }
    ret.len = i + j;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1) ret.len--;
    return ret;
}
BigNum BigNum::operator/(const int & b) const {
    BigNum ret;
    int i,down = 0;
    for(i = len - 1 ; i >= 0 ; i--) {
        ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
        down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
    }
    ret.len = len;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1) ret.len--;
    return ret;
}
int BigNum::operator %(const int & b) const {
    int i,d=0;
    for (i = len-1; i>=0; i--) {
        d = ((d * (MAXN+1))% b + a[i])% b;
    }
    return d;
}
BigNum BigNum::operator^(const int & n) const {
    BigNum t,ret(1);
    if(n<0)exit(-1);
```

```
    if(n==0)return 1;
    if(n==1)return *this;
    int m=n;
    while(m>1) {
        t=*this;
        int i;
        for(i=1;i<<1<=m;i<<=1) {
            t=t*t;
        }
        m-=i;
        ret=ret*t;
        if(m==1)ret=ret*(*this);
    }
    return ret;
}
bool BigNum::operator>(const BigNum & T) const {
    int ln;
    if(len > T.len) return true;
    else if(len == T.len) {
        ln = len - 1;
        while(a[ln] == T.a[ln] && ln >= 0) ln--;
        if(ln >= 0 && a[ln] > T.a[ln]) return true;
        else return false;
    } else return false;
}
void BigNum::print() {
    int i;
    cout << a[len - 1];
    for(i = len - 2 ; i >= 0 ; i--) {
        cout.width(DLEN);
        cout.fill('0');
        cout << a[i];
    }
}
//读取整数
const int ok = 1;
int get_val(int & ret) {
    ret = 0;
    char ch;
    while ((ch=getchar()) > '9' || ch < '0') ;
    do {
        ret = ret*10 + ch - '0';
    } while ((ch=getchar()) <= '9' && ch >= '0') ;
    return ok;
```

```
}

//带负数
int get_val(int & ret) {
    ret = 0;
    char ch;
    bool neg = false;
    while (((ch=getchar()) > '9' || ch < '0') && ch!='-') ;
    if (ch == '-') {
        neg = true;
        while ((ch=getchar()) > '9' || ch < '0') ;
    }
    do {
        ret = ret*10 + ch - '0';
    } while ((ch=getchar()) <= '9' && ch >= '0') ;
    ret = (neg? -ret : ret);
    return ok;
}

//读取整数,可判EOF和EOL
const int eof = -1;
const int eol = -2;
int get_val(int & ret) {
    ret = 0;
    char ch;
    while (((ch=getchar()) > '9' || ch < '0') && ch!=EOF) ;
    if (ch == EOF) return eof;
    do {
        ret = ret*10 + ch - '0';
    } while ((ch=getchar()) <= '9' && ch >= '0') ;
    if (ch == '\n') return eol;
    return ok;
}

//读取浮点数
int get_val(double & ret) {
    ret = 0;
    double base = 0.1;
    char ch;
    bool dot = false, neg = false;
    while (((ch=getchar()) > '9' || ch < '0') && ch != '.' && ch != '-') ;
    if (ch == '-') {
        neg = true;
        while (((ch=getchar()) > '9' || ch < '0') && ch != '.' && ch != '-') ;
```

```
    }
    do {
        if (ch == '.') {
            dot = true;
            continue;
        }
        if (dot) {
            ret += (ch-'0') * base;
            base *= 0.1;
        } else ret = ret*10 + (ch-'0');
    } while (((ch=getchar()) <= '9' && ch >= '0') || ch == '.') ;
    ret = (neg? -ret : ret);
    return ok;
}

typedef long long LL;
//LL MultiMod(LL a, LL b, LL c) {
// if (b)
//   return (a * (b & 1) % c + (MultiMod(a, b >> 1, c) << 1)) % c;
// return 0;
//}

LL MultiMod(LL a, LL b, LL c) {
    LL ret = 0, d = a;
    for (; b; b >>= 1, d <<= 1, d %= c)
        if ((b & 1))
            ret = (ret + d) % c;
    return ret;
}

// 128-bits integer's power with mod in O(64*LogN)
LL ModPower(LL base, LL exp, LL mod) {
    LL ret = 1;
    for (; exp; exp >>= 1, base = MultiMod(base, base, mod))
        if ((exp & 1))
            ret = MultiMod(ret, base, mod);
    return ret;
}
```

第四章 数论算法

1. Greatest Common Divisor 最大公约数

```
int GCD(int x, int y)
{
    int t;
    while(y > 0) {
        t = x % y;
        x = y;
        y = t;
    }
    return x;
}
```

2. Prime 素数判断

```
bool is_prime(int u)
{
    if(u == 0 || u == 1) return false;
    if(u == 2)              return true;
    if(u%2 == 0)                return false;
    for(int i=3; i <= sqrt(u) ;i+=2)
        if(u%i==0)     return false;
    return true;
}
```

3. Sieve Prime 素数筛法

```
const int M = 1000; // M : size
bool mark[M]; // true : prime number
void sieve_prime()
{
    memset(mark, true, sizeof(mark));
    mark[0] = mark[1] = false;
    for(int i=2; i <= sqrt(M) ;i++) {
        if(mark[i]) {
            for(int j=i*i; j < M ;j+=i)
                mark[j] = false;
        }
    }
}
```

4. Module Inverse 模逆元

```
// ax ≡ 1 (mod n)
int Inv(int a, int n)
{
    int d, x, y;
    d = extended_euclid(a, n, x, y);
    if(d == 1)    return (x%n + n) % n;
    else          return -1; // no solution
}
```

5. Extended Euclid 扩展欧几里德算法

```
//如果 GCD(a,b) = d, 则存在 x, y, 使 d = ax + by
// extended_euclid(a, b) = ax + by
int extended_euclid(int a, int b, int &x, int &y)
{
    int d;
    if(b == 0) {x = 1;    y = 0;    return a;}
    d = extended_euclid(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

6. Modular Linear Equation 模线性方程(同余方程)

```
//如果 GCD(a, b)不能整除 c, 则 ax + by = c 没有整数解
// ax ≡ b (mod n) n > 0
//上式等价于二元一次方程 ax – ny = b
void modular_linear_equation(int a, int b, int n)
{
    int d, x, y, x0, gcd;
    // 可以减少扩展欧几里德溢出的可能
    gcd = GCD(a, n);
    if (b%gcd != 0) {
        cout << "no solution" << endl;
        return ;
    }
    a /= gcd; b /= gcd; n /= gcd;
    d = extended_euclid(a, n, x, y);
    if( b%d == 0) {
        x0 = ( x*(b/d) ) % n; // x0 : basic solution
        int ans = n; // min x = (x0%(n/d) + (n/d)) % (n/d)
        for(int i=0; i < d ;i++) {
            ans = ( x0 + i*(n/d) ) % n;
            cout << ans << endl;
```

```
            }
        }
        else    cout << "no solution" << endl;
    }
```

7. Chinese Remainder Theorem 中国余数定理

```
    // x ≡ b[i] (mod w[i]), i∈[1, len-1]
    // 前提条件 w[i] > 0, 且 w[]中任意两个数互质
    int chinese_remainder(int b[], int w[], int len)
    {
        int i, d, x, y, m, n, ret;
        ret = 0;    n = 1;
        for(i=0; i < len ;i++)    n *= w[i];
        for(i=0; i < len ;i++) {
            m = n / w[i] ;
            d = extended_euclid(w[i], m, x, y);
            ret = (ret + y*m*b[i]) % n;
        }
        return (n + ret%n) % n;
    }
// m ≡r[i] (mod a[i])
// a[i] 可以不互素
// -1表示无解
/*
Pku 2891 Strange Way to Express Integers
假设C ≡ A1 (mod B1)，C ≡ A2 (mod B2)。
令C = A1 + X1B，那么X1B1 ≡ A2 - A1 (mod B2)。
用扩展欧几里德算法求出X1，也就求出C。
令B = lcm(B1, B2)，那么上面两条方程就可以被C’ ≡ C (mod B)代替。
迭代直到只剩下一条方程。
*/
LL chinese_remainder2()
{
    int i, j;

    if (n == 1)
        return r[0];

    LL m, x, apre;
    x = modular_linear_equation(a[0], r[1]-r[0], a[1]);
    if (x == -1)
        return -1;
    m = x*a[0] + r[0];
    apre = LCM(a[0], a[1]);
```

```
    for (i=2; i<n; i++)
    {
        x = modular_linear_equation(apre, r[i]-m, a[i]);
        if (x == -1)
            return -1;
        m = x*apre + m;
        apre = LCM(apre, a[i]);
    }
    return m;
}
```

8. Euler Function 欧拉函数

```
//求1..n-1中与n互质数的个数
int euler(int n)
{
    int ans = 1;
    int i;
    for(i=2; i*i<=n ;i++) {
        if(n%i == 0) {
            n /= i;
            ans *= i-1;
            while(n%i == 0) {
                n /= i;
                ans *= i;
            }
        }
    }
    if(n > 1) {
        ans *= n-1;
    }
    return ans;
}
```

9. Farey 总数

```
//求MAX以内所有Farey的总数
const int MAX = 1000100;
int n;
bool num[1100];//sqrt(MAX)
int prime[1100], total;
__int64 f[MAX], inc[MAX];

void cal_prime() {
    int i,j;
```

```
    memset(num, false, sizeof(num));
    total = 0;
    for(i=2;i<1100;i++) {
        if(!num[i]) {
            prime[total ++] = i;
            j = i+i;
            while(j < 1100) {
                num[j] = true;
                j += i;
            }
        }
    }
}

void cal_farey() {
    int i,j,k;
    inc[1] = 1;
    for(i=2;i<MAX;i++) {
        for(j=0;j<total;j++) {
            if(i%prime[j] == 0) {
                k = i / prime[j];
                if(k%prime[j] == 0) inc[i] = inc[k] * prime[j];
                else inc[i] = inc[k] * (prime[j] -1);
                break;
            }
        }
        if(j == total) inc[i] = i -1;
    }
    f[1] = 0;
    for(i=2;i<MAX;i++) f[i] = f[i-1] + inc[i];
}

int main() {
    cal_prime();
    cal_farey();
    while(scanf("%d", &n), n) {
        printf("%I64d\n", f[n]);
    }
}
```

10. Farey 序列构造

```
//构造5000以内的Farey序列
const int MAX = 8000000;
int total;
```

```
int n,k;
int farey[2][MAX];
void make_farey_seq(int x1,int y1,int x2, int y2)
{
    if(x1+x2 > n || y1+y2 > n) return;
    make_farey_seq(x1, y1,x1+x2, y1+y2);
    total ++;
    farey[0][total] = x1+x2;
    farey[1][total] = y1+y2;
    make_farey_seq(x1+x2, y1+y2,x2,y2);
}
int main() {
    int t;
    scanf("%d %d", &n, &t);
    total = 1;
    farey[0][1] = 0;
    farey[1][1] = 1;
    make_farey_seq(0,1,1,1);
    farey[0][total+1] = 1;
    farey[1][total+1] = 1;
    total ++;
    while(t --) {
        scanf("%d", &k);
        if(k > total) puts("No Solution");
        else printf("%d/%d\n", farey[0][k], farey[1][k]);
    }
}
```

11. Miller_Rabbin 素数测试，Pollard_rho 因式分解

```
typedef __int64 I64;
const char * pformat = "%I64d";
I64 big_rand(I64 m) {
    I64 x = rand();
    x *= rand();
    if(x < 0) x = -x;
    return x %= m;
}
// x*y % n
I64 mod_mul(I64 x, I64 y, I64 n) {
    if(x == 0 || y == 0) return 0;
    return ( ((x&1)*y)%n + (mod_mul(x>>1,y,n)<<1)%n ) % n;
}
// x^y % n
I64 mod_exp(I64 x, I64 y, I64 n) {
```

```
    I64 ret = 1;
    while(y) {
        if(y&1) ret = mod_mul(ret,x,n);
        x = mod_mul(x,x,n);
        y >>= 1;
    }
    return ret;
}
bool Miller_Rabbin(I64 n) {// O(times * (log N)^3)
    I64 i,j,x,m,k;
    if(n == 2) return true;
    if(n < 2 || !(n&1)) return false;
    m = n - 1; k = 0;
    while(!(m&1)) m >>= 1, k ++;// binary scan
    for(i=0;i<4;i++) {// test times
        x = big_rand(n-2) +2;
        x = mod_exp(x,m,n);
        if(x == 1) continue;
        for(j=0;j<k;j++) {
            if(x == n-1) break;
            x = mod_mul(x,x,n);
        }
        if(j >= k) return false;
    }
    return true;
    /*lrj P.218
    for(i=0;i<20;i++) {
    x = big_rand(n-2) +2;
    if(mod_exp(x,n-1,n) != 1) return false;
    }
    return true;
    */
}
I64 gcd(I64 x, I64 y) {
    if(x > y) std::swap(x,y);
    while(x) {
        I64 t = y % x;
        y = x;
        x = t;
    }
    return y;
}
I64 func(I64 x, I64 m) {
    return (mod_mul(x,x,m)+1) % m;
```

```
}
I64 Pollard(I64 n) {
    if(Miller_Rabbin(n)) return n;
    if(!(n&1)) return 2;
    I64 i,x,y,ret;
    i = 1;
    while(true) {
        x = i ++;
        y = func(x,n);
        ret = gcd(y-x,n);
        while(ret == 1) {
            x = func(x,n);
            y = func(func(y,n),n);
            ret = gcd((y-x+n)%n,n) % n;
        }
        if(0 < ret && ret < n) return ret;
    }
}
I64 factor[100], nfac, minfac;
void cal_factor(I64 n) {
    I64 x = Pollard(n);
    if(x == n) {
        //factor[nfac ++] = x;
        minfac = min(minfac,x);
        return;
    }
    cal_factor(x);
    cal_factor(n/x);
}
void print_factor(I64 n) {
    I64 i;
    nfac = 0;
    cal_factor(n);
    std::sort(factor,factor + nfac);
    for(i=0;i<nfac;i++) {
        if(i > 0) putchar(' ');
        printf(pformat,factor[i]);
    }
    puts("");
}
const I64 lim = 100000;
int main() {
    I64 n,t,i;
    srand((unsigned)time(NULL));
```

```
    scanf(pformat,&t);
    while(t --) {
        scanf(pformat, &n);
        if(Miller_Rabbin(n)) puts("Prime");
        else {
            if(!(n&1)) puts("2");
            else {
                for(minfac=3; minfac < lim && n % minfac ;minfac+=2) ;
                if(minfac >= lim) {
                    I64 rn = sqrt(1.0*n);
                    if(rn * rn == n) {
                        minfac = rn;
                        cal_factor(rn);
                    }
                    else {
                        minfac = n;
                        cal_factor(n);
                    }
                }
                printf(pformat,minfac);
                puts("");
            }
        }
    }
}
```

12.

第五章 图论算法

1. 最小生成树(Kruscal 算法)

```
/**** **** **** **** **** ****
*   Function Name :      最小生成树(Kruscal 算法)
*   Description :        ZJU 1203 Swordfish O(E*LogE)
**** **** **** **** **** ****/
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cmath>
using namespace std;
struct struct_edges
{
    int bv,tv; //bv 起点  tv 终点
    double w; //权值
};
struct_edges edges[10100]; //边集
struct struct_a
{
    double x;
    double y;
};
struct_a arr_xy[101];
int point[101],n,e;  //n 顶点数, e 边数(注意是无向网络)
double sum;

int kruscal_f1(int point[], int v)
{
    int i = v;
    while(point[i] > 0)     i = point[i];
    return i;
}

bool UDlesser(struct_edges a, struct_edges b)
```

```
{return a.w < b.w;}

void kruscal() //只需要准备好 n，e，递增的边集 edges[]即可使用
{
    int v1,v2,i,j;
    for(i=0; i<n ;i++)     point[i]=0;
    i = j = 0;
    while(j<n-1 && i<e) {
        v1 = kruscal_f1(point, edges[i].bv);
        v2 = kruscal_f1(point, edges[i].tv);
        if(v1 != v2) {
            sum += edges[i].w; //注意 sum 初始为 0
            point[v1]=v2;
            j++;
        }
        i++;
    }
}

int main()
{
    int k,i,j;
    cin>>n;
    k=0;
    while(n != 0) {
        sum=0;
        k++;
        for(i=0; i<n ;i++)
            cin>>arr_xy[i].x>>arr_xy[i].y;
        e=0;
        for(i=0; i<n ;i++) //从 0 开始计数
            for(j=i+1; j<n ;j++) //注意是无向网络
            {
                if(i == j) continue;
                edges[e].bv=i;
                edges[e].tv=j;
                edges[e].w=sqrt((arr_xy[i].x-arr_xy[j].x)*(arr_xy[i].x-arr_xy[j
].x)+(arr_xy[i].y-arr_xy[j].y)*(arr_xy[i].y-arr_xy[j].y));
                e++;
            }
            sort(edges,edges+e,UDlesser);  //得到一个递增的边集，注意是从 0 开
始计数
            kruscal();
```

```
            printf("Case #%d:\n",k);  //cout<<"Case #"<<k<<":"<<endl;
            printf("The minimal distance is: %.2f\n",sum);  //输出 sum
            cin>>n;
            if(n != 0) printf("\n");
    }
}
```

2. 最小生成树(Prim 算法)

```
/**** **** **** **** **** ****
*   Function Name :      最小生成树(Prim 算法)
*   Description :        ZJU 1203 Swordfish O(N^2)
**** **** **** **** **** ****/
#include <iostream>
#include <cmath>
#include <cstdio>
using namespace std;
double sum, arr_list[101][101], min;
int i, j, k=0, n;

struct struct_a
{
    float x;
    float y;
};
struct_a arr_xy[101];
struct struct_b
{
    int point;
    float lowcost;
};
struct_b closedge[101];

void prim(int n)   //prim  需要准备：n 顶点数 arr_list[][]顶点的邻接矩阵也是从 0 开始计数
{
    int i,j,k;
    k=0;
    for(j=0; j<n ;j++) {
        if(j != k) {
            closedge[j].point = k;
            closedge[j].lowcost = arr_list[k][j];
        }
```

```
    }
    closedge[k].lowcost=0;
    for(i=0; i<n ;i++) {
        min=10000;
        for(j=0; j<n ;j++) {
            if (closedge[j].lowcost != 0 && closedge[j].lowcost < min) {
                k = j;
                min = closedge[j].lowcost;
            }
        }
        sum += closedge[k].lowcost;  //不要改成 sum+=min;  sum 即为所求值
        closedge[k].lowcost = 0;
        for(j=0; j<n ;j++) {
            if(arr_list[k][j] < closedge[j].lowcost) {
                closedge[j].point = k;
                closedge[j].lowcost = arr_list[k][j];
            }
        }
    }
}
/*
  arr_list[][]= Wij 如果 Vi, Vj 有边
            0   如果 i=j
            无限大 如果没有边
*/
int main()
{
    cin>>n;
    while(n != 0) {
        sum=0;
        k++;
        for(i=0; i<n ;i++)
            cin>>arr_xy[i].x>>arr_xy[i].y;
        for(i=0; i<n ;i++)
            for(j=0; j<n ;j++) //得到邻接矩阵 arr_list[][]
                arr_list[i][j]=arr_list[j][i]=sqrt((arr_xy[i].x-arr_xy[j].x)*(arr_x
y[i].x-arr_xy[j].x)+(arr_xy[i].y-arr_xy[j].y)*(arr_xy[i].y-arr_xy[j].y));
        prim(n);

        cout<<"Case #"<<k<<":"<<endl;
        printf("The minimal distance is: %.2f\n",sum);
        cin>>n;
        if(n!=0)    printf("\n");
    }
```

```
    }
```

3. 单源最短路径(Bellman-ford 算法)

```
struct node {
    int e,v;
    node(int a = 0,int b = 0)
        : e(a), v(b) {}
};
vector< vector<node> > path;
int n,p,q;
int dist[1000100];
/*
*   SPFA (Shortest Path Faster Algorithm)
*   Bellman-Ford算法的一种队列实现，减少了不必要的冗余计算
*   返回值为false，说明队列不为空，存在负权环
*/
bool SPFA()
{
    int i,j,k,now,l;
    node next;
    bitset <1000100> vis;
    queue< int > SQ;
    memset(dist,-1,sizeof(dist));
    SQ.push(1);
    vis[1] = true;
    dist[1] = 0;

    for (i=0;i<=n;i++) {
        l = SQ.size();
        if (l == 0) break;
        while (l--) {
            now = SQ.front();
            SQ.pop();
            vis[now] = false;
            for (j=path[now].size()-1;j>=0;j--) {
                next = path[now][j];
                if (dist[next.e]==-1 || dist[next.e] > dist[now]+next.v) {
                    dist[next.e] = dist[now]+next.v;
                    if(!vis[next.e]) {
                        SQ.push(next.e);
                        vis[next.e] = true;
                    }
                }
            }
```

```
        }
    }
    return SQ.empty();
}
```

4. 单源最短路径(Dijkstra 算法)

```
/**** **** **** **** **** ****
*   Function Name :      单源最短路径 (Dijkstra 算法)
*   Description :        贪心, O(N^2), 不能有负权
**** **** **** **** **** ****/
int matrix[200][200],n;          //matrix[][], 30000 表示无限大,即无边.否则为有边,
其值为边的权值
void Dijkstra(int x,int y)     //起点 Vx 终点 Vy
{
    int i,j,k,path[40000],mark[40000];
    int min,dist[40000];
    for(i=1;i<=n;i++) {
        mark[i] = 0;
        dist[i] = matrix[x][i];
        path[i] = x;
    }
    mark[x] = 1;
    do {
        min=30000;
        k=0;
        for(i=1;i<=n;i++)
            if(mark[i]==0 && dist[i]<min) {
                min = dist[i];
                k = i;
            }
        if(k) {
            mark[k] = 1;
            for(i=1;i<=n;i++)
                if(matrix[k][i]<30000 && min+matrix[k][i]<dist[i]) {
                    dist[i] = min + matrix[k][i];
                    path[i] = k;
                }
        }
    }while(k);
    cout<<dist[y]<<endl;     //dist[y] 的值就是从 Vx 到 Vy 的最短路径值
    //如果希望得到路径，加入如下代码：
    do {
        cout<<k<<"<--";
        k = path[k];
```

```
    }while(k!=x);
    cout<<x<<endl;
}
```

5. 全源最短路径(Folyd 算法)

```
/**** **** **** **** **** ****
*   Function Name :      全源最短路径(Folyd 算法)
*   Description :          DP, O(N^3)
**** **** **** **** **** ****/
//初始化
//path[i][j]=j;
void Floyd()
{
   int i,j,k;
   for(k=0;k<vertex_number;k++) {
      for(i=0;i<vertex_number;i++) {
         for(j=0;j<vertex_number;j++) {
            if((graph[i][k]==-1) || (graph[k][j]==-1))    continue;
            if((graph[i][j]==-1) || (graph[i][j] > graph[i][k]+graph[k][j]))
            {
               graph[i][j] = graph[i][k]+graph[k][j];    /*最短路径值*/
               path[i][j] = k;    /*最短路径*/
            }
         }
      }
   }
}
```

6. 拓扑排序

```
/**** **** **** **** **** ****
*   Function Name :      拓扑排序
**** **** **** **** **** ****/
//degree[]   每个结点的入度
//f[]        每个结点所在的层
void Toplogical_sort()
{
    int i,j;
    bool p=true;
    top=0;
    while(p) {
        p=false;
        top++;
        for(i=1;i<=n;i++)
```

```
            if(degree[i]==0) {
                p=true;
                f[i]=top;
              }
           for(i=1;i<=n;i++)
              if(f[i]==top) {
                for(j=1;j<=n;j++)
                   if(map[i][j])    degree[j]--;
                degree[i]=-1;
              }
        }
       top--;
    }
```

7. 网络预流和最大流

```
/*
网络中求最大流Edmonds_Karp最短增广路算法O(VE^2)
参数含义:    n代表网络中节点数,第1节点为源点, 第n节点为汇点
             net[][]代表剩余网络,0表示无通路
             path[]保存增广路径
             neck[]代表瓶颈，保存增广路径最小容量
返回值:      最大流量
*/
const int NMAX = 210;
int net[NMAX][NMAX];
int path[NMAX], n;

int bfs()
{
    queue<int> SQ;
    int neck[NMAX], i;
    memset(path,-1,sizeof(path));
    neck[1] = INT_MAX;
    SQ.push(1);

    while(!SQ.empty()) {
        int now = SQ.front();
        SQ.pop();
        if(now == n) break ;
        for(i=1;i<=n;i++) {
            if(net[now][i] > 0 && path[i] == -1) {
                path[i] = now;
                neck[i] = min(neck[now], net[now][i]);
                SQ.push(i);
```

```
            }
        }
    }
    if(path[n] == -1) return -1;
    return neck[n];
}

int Edmonds_Karp()
{
    int now, step;
    int max_flow = 0;

    while( (step=bfs()) != -1 ) {
        max_flow += step;
        now = n;
        while(now != 1) {
            int pre = path[now];
            net[pre][now] -= step;
            net[now][pre] += step;
            now = pre;
        }
    }
    return max_flow;
}
/*
网络中求最大流HLPP高度标号预流推进算法O(V^2*E^0.5)
参数含义:    n代表网络中节点数,第0节点为源点, 第n节点为汇点
             net[][]代表剩余网络,0表示无通路
             earn[]代表各点的盈余
             high[]代表各点的高度
返回值:      最大流量
*/
const int NMAX = 110;
int earn[NMAX], net[NMAX][NMAX], high[NMAX];
int n, m;
queue<int> SQ;
void push(int u, int v) {
    int ex = min(earn[u], net[u][v]);
    earn[u] -= ex;
    net[u][v] -= ex;
    earn[v] += ex;
    net[v][u] += ex;
}
void relable(int u) {
```

```
    int i, mmin = INT_MAX;
    for(i=0;i<=n;i++) {
        if(net[u][i] > 0 && high[i] >= high[u]) {
            mmin = min(mmin, high[i]);
        }
    }
    high[u] = mmin +1;
}
void discharge(int u) {
    int i, vn;
    while(earn[u] > 0) {
        vn = 0;
        for(i=0;i<=n && earn[u] > 0;i++) {
            if(net[u][i] > 0 && high[u] == high[i]+1) {
                push(u,i);
                vn ++;
                if(i != n) SQ.push(i);
            }
        }
        if(vn == 0) relable(u);
    }
}
void init_preflow() {
    int i;
    memset(high,0,sizeof(high));
    memset(earn,0,sizeof(earn));
    while(!SQ.empty()) SQ.pop();
    high[0] = n+1;
    for(i=1;i<=n;i++) {
        if(net[0][i] > 0) {
            earn[i] = net[0][i];
            earn[0] -= net[0][i];
            net[i][0] = net[0][i];
            net[0][i] = 0;
            if(i != n) SQ.push(i);
        }
    }
}
int high_label_preflow_push() {
    int i,j;
    init_preflow();
    while(!SQ.empty()) {
        int overp = SQ.front();
        SQ.pop();
```

```
            discharge(overp);
        }
        return earn[n];
}
//带gap优化的高标预流
const int N = 128;
const int INF = 1 << 28;

class Edge {
public:
    int u, v, cuv, cvu, flow;
    Edge() {}
    Edge(int cu, int cv, int ccu, int ccv) : u(cu), v(cv), cuv(ccu), cvu(ccv), flow(0)
{}
    int other(int p) const { return p == u ? v : u; }
    int cap(int p) const { return p == u ? cuv-flow : cvu+flow; }
    void addFlow(int p, int f) { flow += (p == u ? f : -f); }
};

class NodeList {
private:
    int level, next[N], index[2*N], v;
public:
    void clear(int cv) { v = cv; level = -1; memset(index, -1, sizeof(index)); }
    void insert(int n, int h) { next[n] = index[h]; index[h] = n; level >?= h; }
    int remove();
    bool empty() const { return level < 0; }
};
int NodeList::remove() {
    int r = index[level]; index[level] = next[index[level]];
    while(level >= 0 && index[level] == -1) level--;
    return r;
}

class Network {
private:
    vector<Edge> eg;
    vector<Edge*> net[N];
    int v, s, t;
    NodeList list;
    int h[N], hn[2*N], e[N], cur[N];
    void initNet();
    void initFlow();
    void initHeight();
```

```
    void push(int);
    void relabel(int);
    void discharge(int);
    void gapHeuristic(int);
public:
    bool build();
    int maxFlow(int, int);
};
void Network::gapHeuristic(int k) {
    if(hn[k] != 0 || k >= v+1) return;
    for(int i = 0; i < v; i++)
        if(h[i] > k && h[i] <= v && i != s)
        { hn[h[i]]--; hn[v+1]++; h[i] = v+1; }
}
void Network::initNet() {
    for(int i = 0; i < v; i++) net[i].clear();
    for(int i = eg.size()-1; i >= 0; i--) {
        net[eg[i].u].push_back(&eg[i]);
        net[eg[i].v].push_back(&eg[i]);
    }
}
void Network::initHeight() {
    memset(h, 0, sizeof(h)); memset(hn, 0, sizeof(hn));
    memset(e, 0, sizeof(e)); e[s] = INF;
    for(int i = 0; i < v; i++) h[i] = v;
    queue<int> Q; Q.push(t); h[t] = 0;
    while(!Q.empty()) {
        int p = Q.front(); Q.pop();
        for(int i = net[p].size()-1; i >= 0; i--) {
            int u = net[p][i]->other(p), ec = net[p][i]->cap(u);
            if(ec != 0 && h[u] == v && u != s) { h[u] = h[p]+1; Q.push(u); }
        }
    }
    for(int i = 0; i < v; i++) hn[h[i]]++;
}
void Network::initFlow() {
    initNet(); initHeight();
    for(int i = 0; i < v; i++) cur[i] = net[i].size()-1;
    list.clear(v);
    for(; cur[s] >= 0; cur[s]--) push(s);
}
void Network::push(int u) {
    Edge* te = net[u][cur[u]];
    int ex = min(te->cap(u), e[u]), p = te->other(u);
```

```
    if(e[p] == 0 && p != t) list.insert(p, h[p]);
    te->addFlow(u, ex); e[u] -= ex; e[p] += ex;
}
void Network::relabel(int u) {
    int mh = 2*v, oh = h[u];
    for(int i = net[u].size()-1; i >= 0; i--) {
        int p = net[u][i]->other(u);
        if(net[u][i]->cap(u) != 0) mh <?= h[p]+1;
    }
    hn[h[u]]--; hn[mh]++; h[u] = mh; cur[u] = net[u].size()-1;
    gapHeuristic(oh);
}
void Network::discharge(int u) {
    while(e[u] > 0)
        if(cur[u] < 0) relabel(u);
        else if(net[u][cur[u]]->cap(u) > 0 && h[u] ==
h[net[u][cur[u]]->other(u)]+1) push(u);
        else cur[u]--;
}
bool Network::build() {
    int m, np, nc;
    int a, b, l, i;
    if(scanf("%d %d %d %d", &v, &np, &nc, &m) != 4) return false;
    v += 2; eg.clear();
    for(i = 0; i < m; i++) {
        scanf("\n(%d,%d)%d", &a, &b, &l);
        eg.push_back(Edge(a+2, b+2, l, 0));
    }
    for(i = 0; i < np; i++) {
        scanf("\n(%d)%d", &a, &l);
        eg.push_back(Edge(0, a+2, l, 0));
    }
    for(i = 0; i < nc; i++) {
        scanf("\n(%d)%d", &a, &l);
        eg.push_back(Edge(a+2, 1, l, 0));
    }
    return true;
}
int Network::maxFlow(int ss, int tt) {
    s = ss; t = tt; initFlow();
    while(!list.empty()) {
        int u = list.remove();
        discharge(u);
    }
```

```
    return e[t];
}

int main()
{
    Network net;
    while(net.build()) printf("%d\n", net.maxFlow(0, 1));
    return 0;
}

/*
网络中求最大流Dinic算法O(V^2E)
适用于稠密图，实际复杂度低于HLPP模板
参数含义:    n代表网络中节点数,第0节点为源点，第n节点为汇点
             net代表网络，使用前向星表示法存储边
             dis[]代表从源点出发的距离标号
             path[]代表模拟栈中的路径信息
             cur[]代表模拟栈的现场保存
返回值:      最大流量
*/
const int NMAX = 21000;
const int MMAX = 250000<<1;

struct EDGE {
    int u, v, cap, flow;
    int next;
    EDGE(int _u=0, int _v=0, int _c=0, int _f=0)
        : u(_u), v(_v), cap(_c), flow(_f) {}
};
const int ENDFLAG = -1;
struct EDGELIST {
    int start[NMAX];
    int last[NMAX];
    int tot;
    EDGE arc[MMAX];
    void clear() {
        tot = ENDFLAG +1;
        memset(last, ENDFLAG, sizeof(last));
    }
    void push_back(EDGE edge) {
        edge.next = ENDFLAG;
        arc[tot] = edge;
        if (last[edge.u] != ENDFLAG) arc[ last[edge.u] ].next = tot;
        else start[edge.u] = tot;
```

```
        last[edge.u] = tot;
        tot ++;
    }
    // 创建双向弧
    void add_arc(EDGE edge) {
        push_back(edge);
        push_back(EDGE(edge.v,edge.u,edge.cap));
    }
}net;

int que[2][NMAX];
int qf[2],qe[2],qnow;

#define push_que(a) (que[qnow][ qe[qnow]++ ] = (a))
#define pop_que2    (que[qnow^1][ qf[qnow^1]++ ])
#define switch_que  qnow ^= 1; \
                    qf[qnow] = qe[qnow] = 0;
#define empty_que2      (qf[qnow^1] >= qe[qnow^1])
#define size_que2   (qe[qnow^1] - qf[qnow^1])

int n, m;
int dis[NMAX];
int path[NMAX], deep;
int cur[NMAX];
bool bfs() {
    int i, j;
    memset(dis,-1,sizeof(dis));
    dis[0] = 0;
    qnow = 0;
    switch_que;
    push_que(0);
    switch_que;
    while (!empty_que2) {
        int l = size_que2;
        while (l --) {
            int u = pop_que2;
            for (i=net.start[u];i!=ENDFLAG;i=net.arc[i].next) {
                int v = net.arc[i].v;
                if (dis[v]==-1 && net.arc[i].cap>net.arc[i].flow) {
                    push_que(v);
                    dis[v] = dis[u]+1;
                    if (v == n) return true;
                }
            }
```

```
        }
        switch_que;
    }
    return false;
}
int Dinic()
{
    int i, j;
    int u;
    int maxflow = 0;
    while (bfs()) {
        memcpy(cur,net.start,sizeof(cur));
        for (deep=u=0;true;) {
            if (u==n) {
                int neck = INT_MAX, pos;
                for (i=0;i<deep;i++) {
                    int res = net.arc[path[i]].cap - net.arc[path[i]].flow;
                    if (res<neck) {
                        neck = res;
                        pos = i;
                    }
                }
                maxflow += neck;
                for (i=0;i<deep;i++) {
                    net.arc[path[i]].flow += neck;
                    net.arc[path[i]^1].flow -= neck;
                }
                deep = pos;
                u = net.arc[path[deep]].u;
            }
            for (i=cur[u];i!=ENDFLAG;i=net.arc[i].next) {
                if (net.arc[i].cap>net.arc[i].flow
                    && dis[u]+1==dis[net.arc[i].v]) break;
            }
            cur[u] = i;
            if (i!=ENDFLAG) {
                path[deep++] = i;
                u = net.arc[i].v;
            }
            else {
                if (deep==0) break;
                dis[u] = -1;
                u = net.arc[path[--deep]].u;
            }
```

```
        }
    }
    return maxflow;
}
```

8. 网络最小费用最大流

```
/**** **** **** **** **** ****
网络中最小费用最大流O(V*E^2)
参数含义:    n代表网络中的总节点数,第0节点为源点, 第n节点为汇点
             net[][]代表剩余网络
             cost[][]代表单位费用
             path[]保存增广路径
             ecost[]源点到各点的最短路
算法:初始最小费用和最大流均为0，寻找单位费用最短路
在最短路中求出最大流，即为增广路，再修改剩余网络，直到无可增广路为止
返回值:      最小费用，最大流量
**** **** **** **** **** ****/
const int NMAX = 210;
int net[NMAX][NMAX], cost[NMAX][NMAX];
int path[NMAX], ecost[NMAX];
int n;
bool bellman_ford()
{
    int i,j;
    memset(path,-1,sizeof(path));
    fill(ecost, ecost+NMAX, INT_MAX);
    ecost[0] = 0;

    bool flag = true;
    while(flag) {
        flag = false;
        for(i=0;i<=n;i++) {
            if(ecost[i] == INT_MAX) continue ;
            for(j=0;j<=n;j++) {
                if(net[i][j] > 0 && ecost[i]+cost[i][j] < ecost[j]) {
                    flag = true;
                    ecost[j] = ecost[i]+cost[i][j];
                    path[j] = i;
                }
            }
        }
    }
    return ecost[n] != INT_MAX;
}
```

```
int min_cost_max_flow()
{
    int i,j;
    int mincost = 0, maxflow = 0;
    while( bellman_ford() ) {
        int now = n;
        int neck = INT_MAX;
        while(now != 0) {
            int pre = path[now];
            neck = min(neck, net[pre][now]);
            now = pre;
        }
        maxflow += neck;
        now = n;
        while(now != 0) {
            int pre = path[now];
            net[pre][now] -= neck;
            net[now][pre] += neck;
            cost[now][pre] = - cost[pre][now];
            mincost += cost[pre][now] * neck;
            now = pre;
        }
    }
    return mincost;
}
/**** **** **** **** **** ****
网络中最小费用最大流O(V*E^2) 邻接表SPFA实现
参数含义:    n代表网络中的总节点数,第s节点为源点, 第t节点为汇点
             net代表剩余网络
             path[]保存增广路径
             ecost[]源点到各点的最短路
返回值:      最小费用, 最大流量
**** **** **** **** **** ****/
// POJ 3422
const int NMAX = 5100; // 点数
const int MMAX = 30000; // 边数
const int INF = 0x7f7f7f7f;

int path[NMAX], ecost[NMAX];
int n;
int s, t;

struct EDGE {
```

```
    int u, v, cap, cost, flow;
    int next;
    EDGE(int _u=0, int _v=0, int _c=0, int _ct=0, int _f=0)
        : u(_u), v(_v), cap(_c), flow(_f), cost(_ct) {}
};
const int ENDFLAG = -1;
struct EDGELIST {
    int start[NMAX];
    int last[NMAX];
    int tot;
    EDGE arc[MMAX];
    void clear() {
        tot = ENDFLAG +1;
        memset(last, ENDFLAG, sizeof(last));
    }
    void push_back(EDGE edge) {
        edge.next = ENDFLAG;
        arc[tot] = edge;
        if (last[edge.u] != ENDFLAG) arc[ last[edge.u] ].next = tot;
        else start[edge.u] = tot;
        last[edge.u] = tot;
        tot ++;
    }
    // 创建双向弧
    void add_arc(EDGE edge) {
        push_back(edge);
        push_back(EDGE(edge.v,edge.u,0,INF));
    }
}net;

int que[2][NMAX];
int qf[2],qe[2],qnow;

#define push_que(a) (que[qnow][ qe[qnow]++ ] = (a))
#define pop_que2    (que[qnow^1][ qf[qnow^1]++ ])
#define switch_que  qnow ^= 1; \
                    qf[qnow] = qe[qnow] = 0;
#define empty_que2      (qf[qnow^1] >= qe[qnow^1])
#define size_que2   (qe[qnow^1] - qf[qnow^1])

bool SPFA()
{
    int i,j;
    bitset <NMAX> vis;
```

```
    memset(ecost, 0x7f, sizeof(ecost));
    memset(path, -1, sizeof(path));

    bool flag = true;
    qnow = 1;
    switch_que;
    push_que(s);
    vis[s] = 1;
    ecost[s] = 0;

    for (j=0; j<n && flag; j++)
    {
        flag = false;
        switch_que;
        int l = size_que2;
        while (l --)
        {
            int now = pop_que2;
            vis[now] = 0;
            for (i=net.start[now]; i!=ENDFLAG; i=net.arc[i].next)
            {
                EDGE ed = net.arc[i];
                if (ed.cap>ed.flow && ecost[ed.v]>ecost[now]+ed.cost)
                {
                    flag = true;
                    ecost[ed.v] = ecost[now]+ed.cost;
                    path[ed.v] = i;
                    if (! vis[ed.v])
                    {
                        vis[ed.v] = 1;
                        push_que(ed.v);
                    }
                }
            }
        }
    }
    return ecost[t] != INF;
}

int min_cost_max_flow()
{
    int i,j;
    int mincost = 0, maxflow = 0;
```

```
    while( SPFA() ) {
        int pre = path[t];
        int neck = INT_MAX;
        while(pre != -1) {
            int res = net.arc[pre].cap - net.arc[pre].flow;
            neck = min(neck, res);
            pre = path[net.arc[pre].u];
        }
        maxflow += neck;
        mincost += ecost[t] * neck;
        pre = path[t];
        while(pre != -1) {
            net.arc[pre].flow += neck;
            net.arc[pre^1].flow -= neck;
            net.arc[pre^1].cost = - net.arc[pre].cost;
            pre = path[net.arc[pre].u];
        }
    }
    return mincost;
}
```

9. 网络最大流(高度标号预流推进)

```
/*
函数接口:   int Relabel_To_Front(int s,int d) O(V^2*sqrt(E))
参数含义:   s 为源点，d 为汇点
返回值 :   网络最大流

调用函数前的初始化工作:ver 置为网络中节点的个数，c[i][j]代表节点 i 到
节点 j 的流量，vl[i]存放 i 与相邻的所有节点
其它全局变量均初始化为零
*/
const int VEX = 405;  //网络中顶点数
const int HMAX = 810; //最大高度的定义,只要大于顶点的 2 倍就可以了

int f[VEX][VEX];  //流量
int c[VEX][VEX];  //边最大容量
int h[VEX];       //节点高度
int e[VEX];       //节点容量
int ver;          //节点数目
vector<int> vl[VEX];  //邻接表，vl[i]存放与 i 相邻的节点

void Push(int u,int v)  //流推进，由节点 u 推向 v
{
    int cf = c[u][v] - f[u][v];   //u,v 边的容量
```

```
    int d = e[u] < cf ? e[u] : cf;

    f[u][v] += d;
    f[v][u] = -f[u][v];
    e[u] -= d;
    e[v] += d;
}

void Relabel(int u)  //对 u 重新标号
{
    int i,t,cf;
    int hmin = HMAX;

    for(i = 0 ; i < vl[u].size() ; i++){  //寻找相邻最低点
        t = vl[u][i];
        cf = c[u][t] - f[u][t];
        if(cf > 0 && h[u] <= h[t] && h[t] < hmin)
            hmin = h[t];
    }
    h[u] = hmin + 1;
}

void Init_Preflow(int s)  //初始化网络流，s 为源点
{
    int i;
    int u;

    h[s] = ver;  //初始化高度
    for(i = 0 ; i < vl[s].size() ; i++){
        u = vl[s][i];
        f[s][u] = c[s][u];
        f[u][s] = -c[s][u];
        e[u] = c[s][u];
        e[s] -= c[s][u];
    }
}

void Discharge(int u)
{
    int i = 0;
    int cf,v;

    if(vl[u].size() == 0) return;
    while(e[u] > 0){
```

```
        if(i < vl[u].size()) {
            v = vl[u][i];
            cf = c[u][v] - f[u][v];
        }
        if(i >= vl[u].size()){
            Relabel(u);
            i = 0;
        }
        else if(cf > 0 && h[u] == h[v] + 1)
            Push(u,v);
        else
            i++;
    }
}

int Relabel_To_Front(int s,int d) //s 为源点，d 为汇点
{
    int u,i,old_h;
    list<int> l;
    list<int>::iterator iter;

    Init_Preflow(s);

    iter = l.begin();
    for(i = 0 ; i < ver ; i++){
        if(i != s && i != d)
            l.insert(iter,i);
    }
    iter = l.begin();
    while(iter != l.end()){
        u = *iter;
        old_h = h[u];
        Discharge(u);
        if(h[u] > old_h){
            l.erase(iter);
            l.insert(l.begin(),u);
            iter = l.begin();
        }
        iter++;
    }
    return e[ver - 1];
}
```

10. 最大团

```
/**** **** **** **** **** ****
*   Function Name :          最大独立集，最大团
*   Description :          PKU 1419 Graph Coloring
*   团：指G的一个完全子图，该子图不包含在任何其他的完全子图当中
*   最大独立集：补图的最大团
*   最大团：指其中包含顶点最多的团
**** **** **** **** **** ****/
#include <cstdio>
#include <string>
#define NMAX 110
bool path[NMAX][NMAX];
int n, mmax;
int dp[NMAX];
bool v[NMAX];
int seq[NMAX], seq_pos;
//seq记录最大团集合
bool dfs(int pos, int size)
{
    int i, j, unvis;
    bool tv[NMAX];

    unvis = 0;
    for (i=pos;i<n;i++) {
        if (!v[i]) {
            unvis ++;
        }
    }
    if (unvis == 0) {//|U| = 0
        if (size > mmax) {
            mmax = size;
            seq_pos = 0;
            seq[ seq_pos ++] = pos+1;
            return true;
        }
        return false;
    }

    for (i=pos; i < n && unvis > 0 ;i++) {
        if (!v[i]) {
            if (unvis + size <= mmax || dp[i] + size <= mmax) {
                return false;
            }
            v[i] = true;//U = U\{vi}
            unvis --;
```

```
            memcpy(tv, v, sizeof(v));
            for (j=0;j<n;j++) {//U ∩N(vi);
                if (!path[i][j]) {
                    v[j] = true;
                }
            }
            if ( dfs(i, size+1) ) {
                seq[ seq_pos ++] = pos+1;
                return true;
            }
            memcpy(v, tv, sizeof(v));
        }
    }//while U is not empty
    return false;
}

int max_clique()
{
    int i,j;
    mmax = 0;

    for (i=0;i<n;i++) {
        path[i][i] = false;
    }
    for (i=n-1;i>=0;i--) {
        for (j=0;j<n;j++) {//Si ∩N(vi);
            v[j] = !path[i][j];
        }
        dfs(i, 1);
        dp[i] = mmax;
    }
    return mmax;
}

int main()
{
    int i,j,x,y,e;
    int m,tn;
    scanf("%d", &m);
    while (m --) {
        scanf("%d %d", &n, &e);
        memset(path,0,sizeof(path));
        for (i=0;i<e;i++) {
            scanf("%d %d", &x,&y);
```

```
            x--; y--;
            path[x][y] = path[y][x] = true;
        }
        //max independent set in original graph
        //max clique in inverse graph
        for (i=0;i<n;i++) {
            for (j=0;j<n;j++) {
                path[i][j] = !path[i][j];
            }
        }
        memset(dp,0,sizeof(dp));
        printf("%d\n", max_clique());

        printf("%d", seq[0]);
        for (i=1;i<seq_pos;i++) {
            printf(" %d", seq[i]);
        }
        printf("\n");
    }
}
```

11. 最大二分图匹配(匈牙利算法)

```
/**** **** **** **** **** ****
*   Function Name :    最大二分图匹配(匈牙利算法)
*   Description :    HDOJ 2063 过山车
*   二分图: 指所有顶点分成集合 M 和 N, M 或 N 中任意两个在同一集合中的点互不相连
*   匹配: 一组边顶点分别在两个集合中, 并且任意两条边都没有相同顶点
*   最大匹配: 所能得到的最大的边的个数
**** **** **** **** **** ****/
#include<cstdio>
#include<memory>
#include<vector>
using namespace std;
const int Max=1100;
vector< vector<int> > Bmap;
int n, m, k, nm;
int mark[Max];
bool flag[Max];

bool dfs(int pos)
{
   int i, pre, tp;
   for(i=0; i < Bmap[pos].size() ;i++) {
      tp = Bmap[pos][i];
```

```
            if( !flag[tp] ) {
                flag[tp] = true;
                pre = mark[tp];
                mark[tp] = pos;
                if(pre==-1 || dfs(pre))    return true;
                mark[tp] = pre;
            }
        }
        return false;
    }

    inline int Max_Match()
    {
        int mmax = 0, i;
        for(i=1; i <= m ;i++) {
            memset(flag,0,sizeof(flag));
            if( dfs(i) )    mmax++;
        }
        return mmax;
    }

    int main()
    {
        int i, j, id, id2;
        while(scanf("%d", &k)==1 && k) {
            scanf("%d%d",&m, &n);
            nm = n + m;
            Bmap.clear();    Bmap.resize(nm+10);
            memset(mark,-1,sizeof(mark));
            for(j=0; j < k ;j++) {
                scanf("%d %d", &id, &id2);
                id2 += m;
                Bmap[id].push_back(id2);
            }
            printf("%d\n", Max_Match());
        }
    }
// 二分匹配HopcroftKarp算法O(sqrt(V)*E)
// 贪心一个初始匹配可以加速
#include <iostream>
#include <queue>
using namespace std;

const int MAXN = 3002;
```

```
const int INF = 1<<30;

struct node{
    int x, y;
}G[MAXN], U[MAXN];

int n, m, t, nx, ny, dis;
int x[MAXN], y[MAXN], vs[MAXN];
int Isf[MAXN];
bool adj[MAXN][MAXN];
int ds[MAXN], dt[MAXN];

void input(){
    scanf("%d %d", &t, &m);
    for(int i = 0; i < m; i++){
        scanf("%d %d %d", &G[i].x, &G[i].y, &vs[i]);
        vs[i] *= vs[i];
    }
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        scanf("%d %d", &U[i].x, &U[i].y);
    memset(adj, 0, sizeof(adj));
    if(m < n){
        nx = m, ny = n;
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++){
                int a = G[i].x - U[j].x, b = G[i].y - U[j].y;
                if( (a*a + b*b) < vs[i]*t*t)
                    adj[i][j] = 1;
            }
    }else{
        nx = n, ny = m;
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++){
                int a = G[j].x - U[i].x, b = G[j].y - U[i].y;
                if( (a*a + b*b) <= vs[j]*t*t)
                    adj[i][j] = 1;
            }
    }
}

bool Search() {
    memset(ds, -1, sizeof(ds));
    memset(dt, -1, sizeof(dt));
```

```
    queue<int> Q; dis = INF;
    for(int i = 0; i < nx; i++)
        if(x[i] == -1){
            Q.push(i);
            ds[i] = 0;
        }
        while(!Q.empty()) {
            int u = Q.front(); Q.pop();
            if(ds[u] > dis) break;
            for(int v = 0; v < ny; v++){
                if(adj[u][v]){
                    if(dt[v] != -1) continue;
                    dt[v] = ds[u]+1;
                    if(y[v] == -1) dis = dt[v];
                    else{
                        ds[y[v]] = dt[v]+1;
                        Q.push(y[v]);
                    }
                }
            }
        }
        return (dis != INF);
}

bool DFS(int u) {
    for(int v = 0; v < ny; v++){
        if(!Isf[v] && adj[u][v] && dt[v] == ds[u] + 1){
            Isf[v] = true;
            if(y[v] != -1 && dt[v] == dis) continue;
            if(y[v] == -1 || DFS(y[v])){
                y[v] = u; x[u] = v;
                return 1;
            }
        }
    }
    return 0;
}

int HopcroftKarp() {
    int cnt = 0;
    for(int i = 0; i < nx; i++) x[i] = -1;
    for(int i = 0; i < ny; i++) y[i] = -1;
    while(Search()){
        memset(Isf, 0, sizeof(Isf));
```

```
        for(int i = 0; i < nx; i++)
            if(ds[i] == 0 && DFS(i))
                cnt++;
    }
    return cnt;
}

int main(){
    int test;
    scanf("%d", &test);
    for(int k = 1; k <= test; k++){
        input();
        printf("Scenario #%d:\n%d\n\n", k, HopcroftKarp());
    }
    return 0;
}
```

12. 带权二分图最优匹配(KM 算法)

```
//二分图带权匹配O(N^3)
const int MAXN = 509;
const int INF = 0x1fffffff;

int bpCostMatch(int c[][MAXN], int nx, int ny) {
    static int lx[MAXN], ly[MAXN], slack[MAXN];
    static int open[MAXN], prev[MAXN], pnt[MAXN], x[MAXN], y[MAXN];
    int i, j, k, s, head, tail;
    int d, ans = 0;

    if (nx > ny) ny = nx;
    for (i = 0; i < nx; i++) lx[i] = -INF;
    for (i = 0; i < ny; i++) ly[i] = 0;
    for (i = 0; i < nx; i++)
        for (j = 0; j < ny; j++)
            if ((lx[i] - c[i][j]) < 0)
                lx[i] = c[i][j];
    memset(x, -1, sizeof(x)); memset(y, -1, sizeof(y));
    for (i = 0; i < nx; i++) {
        memset(prev, -1, sizeof(prev));
        for (j = 0; j < ny; j++) slack[j] = INF;
        open[0] = i; head = 0; tail = 1;
        while (x[i] < 0) {
            for (; head < tail && x[i] < 0; head++)
                for (s = open[head], j = 0; j < ny && x[i] < 0; j++)
                    if (prev[j] < 0) {
```

```
                    if ((d = lx[s] + ly[j] - c[s][j]) > 0) {
                        if ((slack[j] - d) > 0) {
                            slack[j] = d; pnt[j] = s;
                        }
                        continue;
                    }
                    open[tail++] = y[j]; prev[j] = s;
                    if (y[j] >= 0) continue;
                    while (j >= 0) {
                        s = prev[j]; y[j] = s; k = x[s]; x[s] = j; j = k;
                    }
                }
                if (x[i] >= 0) break;
                for (d = INF, j = 0; j < ny; j++)
                    if (prev[j] < 0 && (d - slack[j]) > 0)
                        d = slack[j];
                for (j = 0; j < tail; j++) lx[open[j]] -= d;
                for (j = 0; j < ny; j++)
                    if (prev[j] >= 0)
                        ly[j] += d;
                    else if (slack[j] < INF)
                        slack[j] -= d;
                for (j = 0; j < ny; j++)
                    if (prev[j] < 0 && slack[j] == 0) {
                        open[tail++] = y[j]; prev[j] = pnt[j];
                        if (y[j] >= 0) continue;
                        while (j >= 0) {
                            s = prev[j]; y[j] = s; k = x[s]; x[s] = j; j = k;
                        }
                        break;
                    }
            }
        }
        for (i = 0; i < nx; i++)
            if (c[i][x[i]] > -INF) {
                if (c[i][x[i]] < 0)
                    return -1;
                ans += c[i][x[i]];
            } else return -1;
        return ans;
    }

    int N, M, E;
    int c[MAXN][MAXN];
```

```
int cas;

int main() {
    int i, j, a, b, w, ans;
    while (scanf("%d%d%d", &N, &M, &E) != EOF) {
        for (i = 0; i < N; i++)
            for (j = 0; j < M; j++)
                c[i][j] = -INF;
        for (i = 0; i < E; i++) {
            scanf("%d%d%d", &a, &b, &w);
            if (w < 0) continue;
            if (c[a][b] < w)
                c[a][b] = w;
        }
        if (N > M) ans = -1;
        else ans = bpCostMatch(c, N, M);
        printf("Case %d: %d\n", ++cas, ans);
    }
    return 0;
}

/*
wywcgs 的KM O(n^3)
只需要把Graph里的n(顶点数)和edge[x][y](边权)赋值，第一维为x点，第二维为y点。
然后调用KMMatch()函数即可，返回值为最大权完美匹配。
最小权匹配可将每条边权取反，然后类似求最大权匹配即可。
匹配信息保存在xmate[]和ymate[]中。其中xmate[i]为x[i]的匹配点，ymate[i]为y[i]的匹配点。
*/
#include <cstdio>
#include <queue>
#include <algorithm>
using namespace std;

const int N = 310;
const int INF = 1 << 28;

class Graph {
private:
    bool xckd[N], yckd[N];
    int n, edge[N][N], xmate[N], ymate[N];
    int lx[N], ly[N], slack[N], prev[N];
    queue<int> Q;
    bool bfs();
```

```
    void agument(int);
public:
    bool make();
    int KMMatch();
};
bool Graph::bfs() {
    while(!Q.empty()) {
        int p = Q.front(), u = p>>1; Q.pop();
        if(p&1) {
            if(ymate[u] == -1) { agument(u); return true; }
            else { xckd[ymate[u]] = true; Q.push(ymate[u]<<1); }
        } else {
            for(int i = 0; i < n; i++)
                if(yckd[i]) continue;
                else if(lx[u]+ly[i] != edge[u][i]) {
                    int ex = lx[u]+ly[i]-edge[u][i];
                    if(slack[i] > ex) { slack[i] = ex; prev[i] = u; }
                } else {
                    yckd[i] = true; prev[i] = u;
                    Q.push((i<<1)|1);
                }
        }
    }
    return false;
}
void Graph::agument(int u) {
    while(u != -1) {
        int pv = xmate[prev[u]];
        ymate[u] = prev[u]; xmate[prev[u]] = u;
        u = pv;
    }
}
int Graph::KMMatch() {
    memset(ly, 0, sizeof(ly));
    for(int i = 0; i < n; i++) {
        lx[i] = -INF;
        for(int j = 0; j < n; j++) lx[i] >?= edge[i][j];
    }
    memset(xmate, -1, sizeof(xmate)); memset(ymate, -1, sizeof(ymate));
    bool agu = true;
    for(int mn = 0; mn < n; mn++) {
        if(agu) {
            memset(xckd, false, sizeof(xckd));
            memset(yckd, false, sizeof(yckd));
```

```
            for(int i = 0; i < n; i++) slack[i] = INF;
            while(!Q.empty()) Q.pop();
            xckd[mn] = true; Q.push(mn<<1);
        }
        if(bfs()) { agu = true; continue; }
        int ex = INF; mn--; agu = false;
        for(int i = 0; i < n; i++)
            if(!yckd[i]) ex <?= slack[i];
        for(int i = 0; i < n; i++) {
            if(xckd[i]) lx[i] -= ex;
            if(yckd[i]) ly[i] += ex;
            slack[i] -= ex;
        }
        for(int i = 0; i < n; i++)
            if(!yckd[i] && slack[i] == 0) { yckd[i] = true; Q.push((i<<1)|1); }
    }
    int cost = 0;
    for(int i = 0; i < n; i++) cost += edge[i][xmate[i]];
    return cost;
}

bool Graph::make()
{
    int i, j;
    while (scanf("%d", &n) == 1)
    {
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                scanf("%d", &edge[i][j]);
        return true;
    }
    return false;
}

int main()
{
    Graph g;

    while(g.make()) printf("%d\n", g.KMMatch());

    return 0;
}
```

13. 强连通分量(Kosaraju 算法)

```
/*
有向图的强连通分量Kosaraju算法O（E）
参数含义:    使用邻接表来保存图
             path原图，npath逆图
             scc强连通个数
             id[x]=y表示第x个顶点属于y强连通
*/
#define NMAX 11000
vector< vector< int > > path;
vector< vector< int > > npath;
int n,m, scc;
int order[NMAX], order_pos, id[NMAX];
bool vis[NMAX];

void dfs(int pos)
{
    int i,j,l;
    vis[pos] = true;
    l = path[pos].size();
    for (i=0;i<l;i++) {
        j = path[pos][i];
        if (!vis[j]) {
            dfs(j);
        }
    }
    order[ order_pos ++ ] = pos;//make order
}

void ndfs(int pos)
{
    int i,j,l;
    vis[pos] = true;
    id[pos] = scc;
    l = npath[pos].size();
    for (i=0;i<l;i++) {
        j = npath[pos][i];
        if (!vis[j]) {
            ndfs(j);
        }
    }
}

void Kosaraju()
{
```

```
    int i,j;
    //dfs in original graph
    memset(vis, 0, sizeof(vis));
    order_pos = 0;
    for (i=1; i<=n ;i++) {
        if (!vis[i]) {
            dfs(i);
        }
    }
    //dfs in inverse graph
    memset(vis, 0, sizeof(vis));
    memset(id, 0, sizeof(id));
    scc = 1;
    for (i=order_pos-1; i>=0 ;i--) {
        if (!vis[ order[i] ]) {
            ndfs(order[i]);
            scc ++;
        }
    }
    scc --;
}
```

14. 强连通分量(Gabow 算法)

```
/*
有向图的强连通分量Gabow算法O（E）
参数含义:    使用邻接表来保存图
             path原图
             scc强连通个数
             id[x]=y表示第x个顶点属于y强连通
*/
#define NMAX 11000
vector< vector< int > > path;
int n,m, scc, step;
int order[NMAX], order_pos, id[NMAX];
int order2[NMAX], order2_pos;
int vis[NMAX];

void dfs(int pos)
{
    int i,j,next,l,pre;
    vis[pos] = step ++;
    order[ order_pos ++ ] = pos;
    order2[ order2_pos ++ ] = pos;
    l = path[pos].size();
```

```
    for (i=0;i<l;i++) {
        next = path[pos][i];
        if (vis[next] == 0) {
            dfs(next);
        }
        else if (id[next] == 0) {//have a circle and belong to nothing
            while (vis[ order2[order2_pos -1] ] > vis[next]) {
                order2_pos --;
            }
        }
    }//for i
    if (order2[order2_pos -1] == pos) {//if pos back to begin of scc
        order2_pos --;
    }
    else {
        return ;
    }
    do {//record scc
        pre = order[order_pos -1];
        id[pre] = scc;
        order_pos --;
    } while(pre != pos);
    scc ++;
}

void Gabow()
{
    int i,j,l;
    //dfs in original graph
    memset(id, 0, sizeof(id));
    memset(vis, 0, sizeof(vis));
    scc = step = 1;
    order_pos = order2_pos = 0;
    for (i=1; i<=n ;i++) {
        if (vis[i] == 0) {
            dfs(i);
        }
    }
    scc --;
}
```

15. 无向图割边割点和双连通分量

```
#define mclear(x) memset((x), 0, sizeof((x)))
const int MAX = 5100;
```

```
int n,m,deep;
vector<int> path[MAX];
int vis[MAX], low[MAX];
vector<int> cutpoint;//割点
vector< pair<int,int> > bridge;//割边,桥
int nbcc;//双连通分量数
stack< pair<int,int> > order;
vector<int> bcc[MAX];//双连通分量

void dfs(int pos, int father) {
    int i,j, total = 0;
    bool cut = false;
    int reback = 0;//处理平行边
    vis[pos] = low[pos] = deep ++;
    int ls = path[pos].size();
    for(j=0;j<ls;j++) {
        i = path[pos][j];
        if(i == father) reback ++;
        if(vis[i] == 0) {
            pair<int,int> e(pos, i);
            order.push(e);
            dfs(i, pos);
            if(low[i] >= vis[pos]) {
                nbcc ++;
                bcc[nbcc].clear();
                pair<int,int> r;
                do {
                    r = order.top();
                    order.pop();
                    bcc[nbcc].push_back(r.second);
                }while(e != r);
                bcc[nbcc].push_back(r.first);
            }
            total ++;
            low[pos] = min(low[i], low[pos]);
            if((vis[pos] == 1 && total > 1) ||
                (vis[pos] != 1 && low[i] >= vis[pos])) cut = true;
            if(low[i] > vis[pos]) bridge.push_back(e);
        }
        else if(i != father) {
            low[pos] = min(vis[i], low[pos]);
        }
    }
    if(reback > 1) low[pos] = min(low[pos], vis[father]);
```

```
    if(cut) cutpoint.push_back(pos);
}

void find_cut() {
    int i;
    mclear(vis); mclear(low);
    cutpoint.clear(); bridge.clear();
    nbcc = 0;
    while(!order.empty()) order.pop();
    for(i=1;i<=n;i++) {
        if(vis[i] == 0) {
            deep = 1;
            dfs(i, i);
        }
    }
}
/*******************************
图的DFS信息构建by oyjpArt
g矩阵: g[i][j] -> 0 : 无边
              1 : 可重复访问边
              -1: 非可重复访问边
说明:以为在无向图中u->v访问之后就不能再从v->u访问了
故{u, v}访问了之后{v, u}要置-1
如果是有向图则没有这个规则
gc矩阵:gc[i][j]-> 0 : 无边
1 : 树枝边
2 : 反向边
3 : 正向边
4 : 交叉边
d数组: 顶点的开始访问时间表
f数组: 顶点的结束访问时间表
c数组: 顶点颜色表0白色-1灰色1黑色
p数组: 顶点的前驱表
l数组: 顶点的L值(最顶层的祖先层数)
b数组: 顶点的度数表

关于标号函数LOW()
LOW(U)代表的是与U以及U的子孙直接相连的结点的最高辈分（深度）
d[U]                    U首次被访问时
LOW[U] =     min(LOW[U], d[W])     访问边{U,W}
min(LOW[U], LOW[S])  U的儿子S的关联边全部被访问时
/*******************************/
const int maxn = 100;
int n, g[maxn][maxn], gc[maxn][maxn];
```

```
int d[maxn], f[maxn], l[maxn], p[maxn], c[maxn], b[maxn];
int time;

void dfs_visit(int u) {//递归搜索以U为根的深度优先树
    int v;
    c[u] = -1;          //置顶点为灰色//去掉这句之后适用于有向图(后面设置不可访问亦同)
    time++; d[u] = time, l[u] = time;
    for(v = 1; v<=n; v++)
        if(g[u][v] > 0)
            if(c[v] == 0)  {          //如果v是白色节点
                g[v][u] = -1;         //不可再访问
                gc[u][v] = 1;         //树枝边
                b[u]++;                //度数
                p[v] = u;             //记录父亲节点
                dist_visit(v);        //递归搜索以v为根的深度优先树
                if(l[v] < l[u])       //v是u的后代
                    l[u] = l[v];      //u的儿子v的关联边搜索完后计算父亲的low值
                g[v][u] = 1;          //恢复可访问标志
            } else {
                if(c[v] < 0) {        //若顶点为灰色
                    if(l[v] < l[u])   //u与v相连
                        l[u] = l[v];
                    gc[u][v] = 2;     //反向边
                } else  {               //黑色
                    if(d[v] > d[u])
                        gc[u][v] = 3;          //正向边
                    else
                        gc[u][v] = 4;          //交叉边
                }
            }
    c[u] = 1;               //DFS完毕置黑色吧
    time++; f[u] = time;
}

void dfs() {
    int u;
    memset(gc, 0, sizeof(gc));
    memset(c, 0, sizeof(c));
    memset(b, 0, sizeof(b));
    time = 0;
    for(u = 1; u <= n; u++)
        if(c[u] == 0) {
            p[u] = 0;
```

```
            dfs_visit(u);
        }
}
```

16. 最小树形图 O(N^3)

```
/*
最小树形图O（N^3）
参数含义:    使用邻接矩阵来保存图，邻接表O（VE）
             path原图
             pre保存最小入弧的权
             del表示被缩去的点
             fpre保存最小树形图的逆路径
例题:        TJU 2248 Channel Design
*/
const int NMAX = 110;
const int INF = 0x7f7f7f7f;
int n;
int path[NMAX][NMAX];
int pre[NMAX];
bool vis[NMAX], del[NMAX];
int min_cost;
int fold[NMAX], fpre[NMAX];
void dfs(int pos) {
    int i;
    vis[pos] = true;
    for(i=1;i<=n;i++) {
        if(path[pos][i] != INF && !vis[i]) dfs(i);
    }
}
bool is_connect(int root) {
    int i;
    memset(vis, 0, sizeof(vis));
    dfs(root);
    for(i=1;i<=n;i++) {
        if(!vis[i]) return false;
    }
    return true;
}
//O(N^3)
bool min_tree_graph(int root) {
    int i,j,k;
    //make sure every node(except root) have in-arc
    if(!is_connect(root)) return false;
```

```
    memset(del, 0, sizeof(del));
    min_cost = 0;
    for(i=0;i<=n;i++) fold[i] = fpre[i] = i;
    while(true) {
        for(i=1;i<=n;i++) {
            if(del[i] || i == root) continue;
            pre[i] = i;
            path[i][i] = INF;//delete self-cycle
            for(j=1;j<=n;j++) {
                if(del[j]) continue;
                if(path[j][i] < path[ pre[i] ][i]) pre[i] = fpre[fold[i]] = j;
            }
        }//find min in-arc
        for(i=1;i<=n;i++) {
            if(del[i] || i == root) continue;
            j = i;
            memset(vis, 0, sizeof(vis));
            while(!vis[j] && j != root) {
                vis[j] = true;
                j = pre[j];
            }
            if(j == root) continue;//no cycle
            i = j;//cycle begin node
            min_cost += path[ pre[i] ][i];
            for(j=pre[i]; j != i ;j=pre[j]) {
                del[j] = true;//fold cycle
                min_cost += path[ pre[j] ][j];//add cycle cost
            }
            for(j=1;j<=n;j++) {
                if(del[j]) continue;
                if(path[j][i] != INF) path[j][i] -= path[ pre[i] ][i];
            }//i is new fold node
            for(j=pre[i]; j != i ;j=pre[j]) {
                for(k=1;k<=n;k++) {
                    if(del[k]) continue;
                    path[i][k] = min(path[i][k], path[j][k]);
                    if(path[k][j] != INF && path[k][i] > path[k][j] -
path[ pre[j] ][j]) {
                        path[k][i] = path[k][j] - path[ pre[j] ][j];
                        fold[i] = j;//record fold node
                        fpre[i] = j;
                    }
                }
            }//make new graph
```

```
            break;
        }
        if(i > n) {
            for(i=1;i<=n;i++) {
                if(del[i] || i == root) continue;
                min_cost += path[ pre[i] ][i];
            }
            break;
        }//graph no cycle
    }//while have cycle
    return true;
}
//print path in min tree graph
void print_mtg(int root) {
    int i, total = n;
    memset(vis, 0, sizeof(vis));
    for(i=1;i<=n;i++) vis[fpre[i]] = true;
    for(i=1;i<=n;i++) {
        if(!vis[i]) {
            int pos = i;
            while(pos != root) {
                printf("%d <- ", pos);
                pos = fpre[pos];
            }
            printf("%d\n", root);
        }
    }
}
int main() {
    int i,m;
    while(scanf("%d %d", &n,&m), !(n==0 && m==0)) {
        memset(path, 0x7f, sizeof(path));
        while(m --) {
            int x,y,z;
            scanf("%d %d %d", &x,&y,&z);
            path[x][y] = min(path[x][y], z);
        }
        if( !min_tree_graph(1) ) puts("impossible");
        else printf("%d\n", min_cost);
    }
}
```

17. 最小树形图 O(VE)

```
const int NMAX = 1500;
```

```
const int INF = 0x7f7f7f7f;
struct LINKT {
    int ls;
    int adj[NMAX];
    void clear() {ls = 0;}
    int operator [] (const int pos) {return adj[pos];}
    int size() {return ls;}
    void push_back(const int pos) {adj[ls ++] = pos;}
};
int n;
int path[NMAX][NMAX];
LINKT epath[NMAX], nepath[NMAX];
int pre[NMAX];
bool vis[NMAX], del[NMAX];
int min_cost;
int fold[NMAX], fpre[NMAX];
void dfs(int pos) {
    int i;
    vis[pos] = true;
    for(i=0;i<epath[pos].ls;i++) {
        if(!vis[ epath[pos].adj[i] ]) dfs(epath[pos].adj[i]);
    }
}
bool is_connect(int root) {
    int i;
    memset(vis, 0, sizeof(vis));
    dfs(root);
    for(i=1;i<=n;i++) {
        if(!vis[i]) return false;
    }
    return true;
}
//O(VE)
bool min_tree_graph(int root) {
    int i,j,k;
    //make sure every node(except root) have in-arc
    if(!is_connect(root)) return false;
    memset(del, 0, sizeof(del));
    min_cost = 0;
    for(i=0;i<=n;i++) fold[i] = fpre[i] = i;
    while(true) {
        for(i=1;i<=n;i++) {
            if(del[i] || i == root) continue;
            pre[i] = i;
```

```
        path[i][i] = INF;//delete self-cycle
        for(j=0;j<nepath[i].ls;j++) {
            int t = nepath[i].adj[j];
            if(del[t]) continue;
            if(path[t][i] < path[ pre[i] ][i]) pre[i] = fpre[fold[i]] = t;
        }
    }//find min in-arc
    for(i=1;i<=n;i++) {
        if(del[i] || i == root) continue;
        j = i;
        memset(vis, 0, sizeof(vis));
        while(!vis[j] && j != root) {
            vis[j] = true;
            j = pre[j];
        }
        if(j == root) continue;//no cycle
        i = j;//cycle begin node
        min_cost += path[ pre[i] ][i];
        for(j=pre[i]; j != i ;j=pre[j]) {
            del[j] = true;//fold cycle
            min_cost += path[ pre[j] ][j];//add cycle cost
        }
        for(j=0;j<nepath[i].ls;j++) {
            int t = nepath[i].adj[j];
            if(del[t]) continue;
            path[t][i] -= path[ pre[i] ][i];
        }//i is new fold node
        for(j=pre[i]; j != i ;j=pre[j]) {
            for(k=0;k<epath[j].ls;k++) {
                int t = epath[j].adj[k];
                if(del[t]) continue;
                if(path[i][t] == INF) {
                    epath[i].push_back(t);
                    nepath[t].push_back(i);
                }
                path[i][t] = min(path[i][t], path[j][t]);
            }
            for(k=0;k<nepath[j].ls;k++) {
                int t = nepath[j].adj[k];
                if(del[t]) continue;
                if(path[t][i] == INF) {
                    epath[t].push_back(i);
                    nepath[i].push_back(t);
                }
```

```
                    if(path[t][i] > path[t][j] - path[ pre[j] ][j]) {
                        path[t][i] = path[t][j] - path[ pre[j] ][j];
                        fold[i] = j;//record fold node
                        fpre[i] = j;
                    }
                }
            }//make new graph
            break;
        }
        if(i > n) {
            for(i=1;i<=n;i++) {
                if(del[i] || i == root) continue;
                min_cost += path[ pre[i] ][i];
            }
            break;
        }//graph no cycle
    }//while have cycle
    return true;
}
```

18.

第六章 几何算法

```
/*****************************************
* COMPUTATIONAL GEOMETRY ROUTINES
* WRITTEN BY : LIU Yu (C) 2003
*****************************************/
//   叉乘
//   两个点的距离
//   点到直线距离
//   返回直线 Ax + By + C =0  的系数
//   线段
//   圆
//   两个圆的公共面积
//   矩形
//   根据下标返回多边形的边
//   两个矩形的公共面积
//   多边形  ,逆时针或顺时针给出 x,y
//   多边形顶点
//   多边形的边
//   多边形的周长
```

```
//    判断点是否在线段上
//    判断两条线断是否相交，端点重合算相交
//    判断两条线断是否平行
//    判断两条直线断是否相交
//    直线相交的交点
//    判断是否简单多边形
//    求多边形面积
//    判断是否在多边形上
//    判断是否在多边形内部
//    点阵的凸包，返回一个多边形
//   最近点对的距离
#include <cmath>
#include <cstdio>
#include <memory>
#include <algorithm>
#include <iostream>
using namespace std;

typedef double TYPE;

#define Abs(x) (((x)>0)?(x):(-(x)))
#define Sgn(x) (((x)<0)?(-1):(1))
#define Max(a,b) (((a)>(b))?(a):(b))
#define Min(a,b) (((a)<(b))?(a):(b))

#define Epsilon 1e-10
#define Infinity 1e+10
#define Pi 3.14159265358979323846

TYPE Deg2Rad(TYPE deg)
{return (deg * Pi / 180.0);}

TYPE Rad2Deg(TYPE rad)
{return (rad * 180.0 / Pi);}

TYPE Sin(TYPE deg)
{return sin(Deg2Rad(deg));}

TYPE Cos(TYPE deg)
{return cos(Deg2Rad(deg));}

TYPE ArcSin(TYPE val)
{return Rad2Deg(asin(val));}
```

```
TYPE ArcCos(TYPE val)
{ return Rad2Deg(acos(val));}

TYPE Sqrt(TYPE val)
{ return sqrt(val);}

struct POINT
{
   TYPE x;
   TYPE y;
   TYPE z;
   POINT() : x(0), y(0), z(0) {};
   POINT(TYPE _x_, TYPE _y_, TYPE _z_ = 0) : x(_x_), y(_y_), z(_z_) {};
};

// cross product of (o->a) and (o->b)
// 叉乘
TYPE Cross(const POINT & a, const POINT & b, const POINT & o)
{return (a.x - o.x) * (b.y - o.y) - (b.x - o.x) * (a.y - o.y);}

// planar points' distance
//  两个点的距离
TYPE Distance(const POINT & a, const POINT & b)
{return Sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + (a.z - b.z) * (a.z
 - b.z));}

struct LINE
{
   POINT a;
   POINT b;
   LINE() {};
   LINE(POINT _a_, POINT _b_) : a(_a_), b(_b_) {};
};

//点到直线距离
double PointToLine(POINT p0 ,POINT p1 ,POINT p2 ,POINT &cp)
{
    double d = Distance(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 Abs(s);
}
```

```
// 返回直线 Ax + By + C =0  的系数
void Coefficient(const LINE & L, TYPE & A, TYPE & B, TYPE & C)
{
   A = L.b.y - L.a.y;
   B = L.a.x - L.b.x;
   C = L.b.x * L.a.y - L.a.x * L.b.y;
}

void Coefficient(const POINT & p,const TYPE a,TYPE & A,TYPE & B,TYPE & C)
{
   A = Cos(a);
   B = Sin(a);
   C = - (p.y * B + p.x * A);
}

// 线段
struct SEG
{
   POINT a;
   POINT b;
   SEG() {};
   SEG(POINT _a_, POINT _b_):a(_a_),b(_b_) {};
};

// 圆
struct CIRCLE
{
   TYPE x;
   TYPE y;
   TYPE r;
   CIRCLE() {}
   CIRCLE(TYPE _x_, TYPE _y_, TYPE _r_) : x(_x_), y(_y_), r(_r_) {}
};

POINT Center(const CIRCLE & circle)
{ return POINT(circle.x, circle.y);}

TYPE Area(const CIRCLE & circle)
{ return Pi * circle.r * circle.r;}

//两个圆的公共面积
TYPE CommonArea(const CIRCLE & A, const CIRCLE & B)
{
   TYPE area = 0.0;
```

```
    const CIRCLE & M = (A.r > B.r) ? A : B;
    const CIRCLE & N = (A.r > B.r) ? B : A;

    TYPE D = Distance(Center(M), Center(N));

    if ((D < M.r + N.r) && (D > M.r - N.r))
    {
        TYPE cosM = (M.r * M.r + D * D - N.r * N.r) / (2.0 * M.r * D);
        TYPE cosN = (N.r * N.r + D * D - M.r * M.r) / (2.0 * N.r * D);

        TYPE alpha = 2.0 * ArcCos(cosM);
        TYPE beta  = 2.0 * ArcCos(cosN);

        TYPE TM = 0.5 * M.r * M.r * Sin(alpha);
        TYPE TN = 0.5 * N.r * N.r * Sin(beta);

        TYPE FM = (alpha / 360.0) * Area(M);
        TYPE FN = (beta / 360.0) * Area(N);

        area = FM + FN - TM - TN;
    }
    else if (D <= M.r - N.r)
    {
        area = Area(N);
    }
    return area;
}

//    矩形
//    矩形的线段
//        2
//     --------------- b
//     |           |
//    3 |           | 1
//    a ---------------
//        0

struct RECT
{
    POINT a;                            // 左下点
    POINT b;                            // 右上点
    RECT() {};
    RECT(const POINT & _a_, const POINT & _b_)
```

```
    {a = _a_; b = _b_;}
};

//根据下标返回多边形的边
SEG Edge(const RECT & rect, int idx)
{
    SEG edge;
    while (idx < 0) idx += 4;
    switch (idx % 4)
    {
    case 0:
        edge.a = rect.a;
        edge.b = POINT(rect.b.x, rect.a.y);
        break;
    case 1:
        edge.a = POINT(rect.b.x, rect.a.y);
        edge.b = rect.b;
        break;
    case 2:
        edge.a = rect.b;
        edge.b = POINT(rect.a.x, rect.b.y);
        break;
    case 3:
        edge.a = POINT(rect.a.x, rect.b.y);
        edge.b = rect.a;
        break;
    default:
        break;
    }
    return edge;
}

TYPE Area(const RECT & rect)
{return (rect.b.x - rect.a.x) * (rect.b.y - rect.a.y);}

//  两个矩形的公共面积
TYPE CommonArea(const RECT & A, const RECT & B)
{
    TYPE area = 0.0;

    POINT LL(Max(A.a.x, B.a.x), Max(A.a.y, B.a.y));
    POINT UR(Min(A.b.x, B.b.x), Min(A.b.y, B.b.y));

    if ((LL.x <= UR.x) && (LL.y <= UR.y))
```

```
    {
        area = Area(RECT(LL, UR));
    }
    return area;
}

// 多边形 ,逆时针或顺时针给出 x,y
struct POLY
{
    int n;        //n 个点
    TYPE * x;     //x,y 为点的指针，首尾必须重合
    TYPE * y;
    POLY() : n(0), x(NULL), y(NULL) {};
    POLY(int _n_, const TYPE * _x_, const TYPE * _y_)
    {
        n = _n_;
        x = new TYPE[n + 1];
        memcpy(x, _x_, n*sizeof(TYPE));
        x[n] = _x_[0];

        y = new TYPE[n + 1];
        memcpy(y, _y_, n*sizeof(TYPE));
        y[n] = _y_[0];
    }
};

//多边形顶点
POINT Vertex(const POLY & poly, int idx)
{
    idx %= poly.n;
    return POINT(poly.x[idx], poly.y[idx]);
}

//多边形的边
SEG Edge(const POLY & poly, int idx)
{
    idx %= poly.n;
    return SEG(POINT(poly.x[idx], poly.y[idx]),
        POINT(poly.x[idx + 1], poly.y[idx + 1]));
}

//多边形的周长
TYPE Perimeter(const POLY & poly)
{
```

```
    TYPE p = 0.0;
    for (int i = 0; i < poly.n; i++)
        p = p + Distance(Vertex(poly, i), Vertex(poly, i + 1));
    return p;
}

bool IsEqual(TYPE a, TYPE b)
{return (Abs(a - b) < Epsilon);}

bool IsEqual(const POINT & a, const POINT & b)
{return (IsEqual(a.x, b.x) && IsEqual(a.y, b.y));}

bool IsEqual(const LINE & A, const LINE & B)
{
    TYPE A1, B1, C1;
    TYPE A2, B2, C2;

    Coefficient(A, A1, B1, C1);
    Coefficient(B, A2, B2, C2);

    return IsEqual(A1 * B2, A2 * B1) &&
        IsEqual(A1 * C2, A2 * C1) &&
        IsEqual(B1 * C2, B2 * C1);
}

// 判断点是否在线段上
bool IsOnSeg(const SEG & seg, const POINT & p)
{
    return (IsEqual(p, seg.a) || IsEqual(p, seg.b)) ||
        (((p.x - seg.a.x) * (p.x - seg.b.x) < 0 ||
        (p.y - seg.a.y) * (p.y - seg.b.y) < 0) &&
        (IsEqual(Cross(seg.b, p, seg.a), 0)));
}

//判断两条线断是否相交，端点重合算相交
bool IsIntersect(const SEG & u, const SEG & v)
{
    return (Cross(v.a, u.b, u.a) * Cross(u.b, v.b, u.a) >= 0) &&
        (Cross(u.a, v.b, v.a) * Cross(v.b, u.b, v.a) >= 0) &&
        (Max(u.a.x, u.b.x) >= Min(v.a.x, v.b.x)) &&
        (Max(v.a.x, v.b.x) >= Min(u.a.x, u.b.x)) &&
        (Max(u.a.y, u.b.y) >= Min(v.a.y, v.b.y)) &&
        (Max(v.a.y, v.b.y) >= Min(u.a.y, u.b.y));
}
```

```
//判断两条线断是否平行
bool IsParallel(const LINE & A, const LINE & B)
{
   TYPE A1, B1, C1;
   TYPE A2, B2, C2;

   Coefficient(A, A1, B1, C1);
   Coefficient(B, A2, B2, C2);

   return (A1 * B2 == A2 * B1) &&
      ((A1 * C2 != A2 * C1) || (B1 * C2 != B2 * C1));
}

//判断两条直线断是否相交
bool IsIntersect(const LINE & A, const LINE & B)
{return !IsParallel(A, B);}

//直线相交的交点
POINT Intersection(const LINE & A, const LINE & B)
{
   TYPE A1, B1, C1;
   TYPE A2, B2, C2;

   Coefficient(A, A1, B1, C1);
   Coefficient(B, A2, B2, C2);

   POINT I(0, 0);
   I.x = - (B2 * C1 - B1 * C2) / (A1 * B2 - A2 * B1);
   I.y =   (A2 * C1 - A1 * C2) / (A1 * B2 - A2 * B1);
   return I;
}

bool IsInCircle(const CIRCLE & circle, const RECT & rect)
{
   return (circle.x - circle.r >= rect.a.x) &&
      (circle.x + circle.r <= rect.b.x) &&
      (circle.y - circle.r >= rect.a.y) &&
      (circle.y + circle.r <= rect.b.y);
}

//判断是否简单多边形
bool IsSimple(const POLY & poly)
{
```

```
    if (poly.n < 3)
        return false;
    SEG L1, L2;
    for (int i = 0; i < poly.n - 1; i++)
    {
        L1 = Edge(poly, i);
        for (int j = i + 1; j < poly.n; j++)
        {
            L2 = Edge(poly, j);
            if (j == i + 1)
            {
                if (IsOnSeg(L1, L2.b) || IsOnSeg(L2, L1.a))
return false;
            }
            else if (j == poly.n - i - 1)
            {
                if (IsOnSeg(L1, L2.a) || IsOnSeg(L2, L1.b))
return false;
            }
            else
            {
                if (IsIntersect(L1, L2))   return false;
            }
        } // for j
    } // for i
    return true;
}

//求多边形面积
TYPE Area(const POLY & poly)
{
    if (poly.n < 3) return TYPE(0);
    double s = poly.y[0] * (poly.x[poly.n - 1] - poly.x[1]);
    for (int i = 1; i < poly.n; i++)
    {
        s += poly.y[i] * (poly.x[i - 1] - poly.x[(i + 1) % poly.n]);
    }
    return s/2;
}

//判断是否在多边形上
bool IsOnPoly(const POLY & poly, const POINT & p)
{
    for (int i = 0; i < poly.n; i++)
```

```
    {
        if (IsOnSeg(Edge(poly, i), p))   return true;
    }
    return false;
}

//判断是否在多边形内部
bool IsInPoly(const POLY & poly, const POINT & p)
{
    SEG L(p, POINT(Infinity, p.y));
    int count = 0;
    for (int i = 0; i < poly.n; i++)
    {
        SEG S = Edge(poly, i);
        if (IsOnSeg(S, p))
        {
            return false;                    //如果想让 在 poly 上则返回 true,则改为 true
        }
        if (!IsEqual(S.a.y, S.b.y))
        {
            POINT & q = (S.a.y > S.b.y)?(S.a):(S.b);
            if (IsOnSeg(L, q))
            {
                ++count;
            }
            else if (!IsOnSeg(L, S.a) && !IsOnSeg(L, S.b) && IsIntersect(S, L))
            {
                ++count;
            }
        }
    }
    return (count % 2 != 0);
}

// 点阵的凸包，返回一个多边形
POLY ConvexHull(const POINT * set, int n)          // 不适用于点少于三个的情况
{
    POINT * points = new POINT[n];
    memcpy(points, set, n * sizeof(POINT));

    TYPE * X = new TYPE[n];
    TYPE * Y = new TYPE[n];

    int i, j, k = 0, top = 2;
```

```
for(i = 1; i < n; i++)
{
   if ((points[i].y < points[k].y) ||
      ((points[i].y == points[k].y) &&
      (points[i].x < points[k].x)))
   {
      k = i;
   }
}

std::swap(points[0], points[k]);

for (i = 1; i < n - 1; i++)
{
   k = i;
   for (j = i + 1; j < n; j++)
   {
      if ((Cross(points[j], points[k], points[0]) > 0) ||
         ((Cross(points[j], points[k], points[0]) == 0) &&
         (Distance(points[0], points[j]) < Distance(points[0], points[k]))))
      {
         k = j;
      }
   }
   std::swap(points[i], points[k]);
}

X[0] = points[0].x; Y[0] = points[0].y;
X[1] = points[1].x; Y[1] = points[1].y;
X[2] = points[2].x; Y[2] = points[2].y;

for (i = 3; i < n; i++)
{
   while (Cross(points[i], POINT(X[top], Y[top]),
      POINT(X[top - 1], Y[top - 1])) >= 0 && top>0)
   {
      top--;
   }
   ++top;
   X[top] = points[i].x;
   Y[top] = points[i].y;
}

delete [] points;
```

```
    POLY poly(++top, X, Y);
    delete [] X;
    delete [] Y;
    return poly;
}

//最近点对的距离, Written By PrincessSnow
#define MAXN 100000
POINT pt[MAXN];

bool cmp(POINT n1, POINT n2)
{return (n1.x<n2.x || n1.x==n2.x && n1.y<n2.y);}

double Get(double dis, int mid, int start, int end)
{
    int s=mid, e=mid, i, j;
    double t;
    while(s > start && pt[mid].x - pt[s].x <= dis)     s--;
    while(e < end && pt[e].x - pt[mid].x <= dis)     e++;
    for(i=s; i <= e; i++)
        for(j=i+1; j <= e && j <= i+7; j++) {
            t = Distance(pt[i], pt[j]);
            if(t < dis)     dis=t;
        }
     return dis;
}

double ClosestPairDistance(int start, int end)
{
    int m = end-start+1, mid, i;
    double t1, t2, dis=-1, t;
    if(m <= 3) {
        for(i=start; i < end; i++) {
            t = Distance(pt[i] , pt[i+1]);
            if(t < dis || dis == -1)     dis = t;
        }
        t = Distance(pt[start] , pt[end]);
        if(t < dis) dis=t;
        return dis;
    }

    if(m%2 == 0)     mid = start + m/2 - 1;
    else             mid = start + m/2;
    if(m%2 == 0) {
```

```
        t1 = ClosestPairDistance(start, mid);
        t2 = ClosestPairDistance(mid+1, end);
    }
    else {
        t1 = ClosestPairDistance(start, mid);
        t2 = ClosestPairDistance(mid+1, end);
    }
    if(t1 < t2)     dis = t1;
    else          dis = t2;
    dis = Get(dis, mid, start, end);
    return dis;
}
```

1. 球面上两点最短距离

```
// 计算圆心角 lat 表示纬度, -90 <= w <= 90, lng 表示经度
// 返回两点所在大圆劣弧对应圆心角, 0 <= angle <= pi
double angle(double lng1, double lat1, double lng2, double lat2)
{
    double dlng = fabs(lng1 - lng2) * pi / 180;
    while(dlng >= pi+pi)     dlng -= pi+pi;
    if(dlng > pi)     dlng = pi + pi - dlng;
    lat1 *= pi / 180,     lat2 *= pi / 180;
    return acos( cos(lat1)*cos(lat2)*cos(dlng) + sin(lat1)*sin(lat2) );
}

// 计算距离, r 为球半径
double line_dist(double r, double lng1, double lat1, double lng2, double lat2)
{
    double dlng = fabs(lng1 - lng2) * pi / 180;
     while(dlng >= pi+pi)     dlng -= pi+pi;
    if(dlng > pi)     dlng = pi + pi - dlng;
    lat1 *= pi / 180,     lat2 *= pi / 180;
    return r * sqrt( 2 - 2*( cos(lat1)*cos(lat2)*cos(dlng) + sin(lat1)*sin(lat2)
) );
}

// 计算球面距离, r 为球半径
double sphere_dist(double r, double lng1, double lat1, double lng2, double lat
2)
{
    return r * angle(lng1, lat1, lng2, lat2);
}
```

2. 三点求圆心坐标

```
double GetRadiusBy3Points(double x1, double y1,
                          double x2, double y2,
                          double x3, double y3,
                          double &x, double &y)
{
    // 由 ( x - x1 )^2 + ( y - y1 )^2 = ( x - x2 )^2 + ( y - y2 )^2 得
    // 2*( x2 - x1 )*x + 2*( y2 - y1 )*y = x2^2 - x1^2 + y2^2 - y1^2
    // 同理得
    // 2*( x3 - x2 )*x + 2*( y3 - y2 )*y = x3^2 - x2^2 + y3^2 - y2^2
    // 由行列式解方程得 x , y
    double a11, a12, a21, a22, b1, b2;
    double d, d1, d2 ;
    a11 = 2 * ( x3 - x2 );
    a12 = 2 * ( y3 - y2 );
    a21 = 2 * ( x2 - x1 );
    a22 = 2 * ( y2 - y1 );

    b1 = x3*x3 - x2*x2 + y3*y3 - y2*y2;
    b2 = x2*x2 - x1*x1 + y2*y2 - y1*y1;

    d = a11*a22 - a12*a21;
    d1 = b1*a22 - a12*b2;
    d2 = a11*b2 - b1*a21;
    // x , y 是圆心坐标
    x = d1 / d;
    y = d2 / d;
    return  (x1 - x)*(x1 - x) + (y1 - y)*(y1 - y);
}
```

3. 三角形几个重要的点

设三角形的三条边为 a, b, c，且不妨假设 a <= b <= c
三角形的面积可以根据海伦公式算得，如下：
s = sqrt(p * (p - a) * (p - b) * (p - c)), p = (a + b + c) / 2

1. 费马点(该点到三角形三个顶点的距离之和最小)
有个有趣的结论：若三角形的三个内角均小于 120 度，
那么该点连接三个顶点形成的三个角均为 120 度；若三角形存在一个内角
大于 120 度，则该顶点就是费马点)
计算公式如下:
若有一个内角大于 120 度（这里假设为角 C)，则距离为 a + b
若三个内角均小于 120 度，则距离为
sqrt((a * a + b * b + c * c + 4 * sqrt(3.0) * s) / 2),其中

2.内心----角平分线的交点

令 x = (a + b - c) / 2, y = (a - b + c) / 2, z = (-a + b + c) / 2, h = s / p
计算公式为 sqrt(x * x + h * h) + sqrt(y * y + h * h) + sqrt(z * z + h * h)

3.重心----中线的交点
计算公式如下：

```
2.0 / 3 * (sqrt((2 * (a * a + b * b) - c * c) / 4)
         + sqrt((2 * (a * a + c * c) - b * b) / 4)
         + sqrt((2 * (b * b + c * c) - a * a) / 4))
```

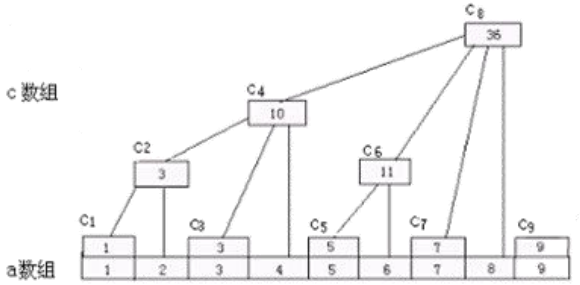
4.垂心----垂线的交点
计算公式如下：

```
3 * (c / 2 / sqrt(1 - cosC * cosC))
```

4.

第七章 专题讨论

1. 树状数组




```
/**** **** **** **** **** ****
*   Function Name :          树状数组
*   Description :            HDOJ 1166 敌兵布阵
*                        减少冗余统计，是线段树的一种变化
**** **** **** **** **** ****/
#include<cstdio>
int data[50001], s[50001], T[50001];

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

inline int sum(int end)
{
    int sum = 0;
    while(end > 0) {
        sum += T[end];
        end -= lowbit(end);
```

```
    }
    return sum;
}

inline void plus(int pos, int num, int count)
{
    while(pos <= count) {
        T[pos] += num;
        pos += lowbit(pos);
    }
}

int main()
{
    char buffer[10];
    int i, j, t, n, a, b;
    scanf("%d", &t);
    for(i=1; i <= t ;i++) {
        scanf("%d", &n);
        T[0] = s[0] = data[0] = 0;
        for(j=1; j <= n ;j++) {
            scanf("%d", &data[j]);
            s[j] = s[j - 1] + data[j];
            T[j] = s[j] - s[j - lowbit(j)];
        }
        printf("Case %d:\n", i);
        while(scanf("%s", buffer) == 1 && buffer[0] != 'E') {
            scanf("%d%d", &a, &b);
            switch(buffer[0]) {
            case 'Q':
                printf("%d\n", sum(b) - sum(a) + data[a]); break;
            case 'A':
                plus(a, b, n); data[a] += b;    break;
            case 'S':
                plus(a, -b, n); data[a] -= b; break;
            }
        }
    }
}
```

2. 字典树

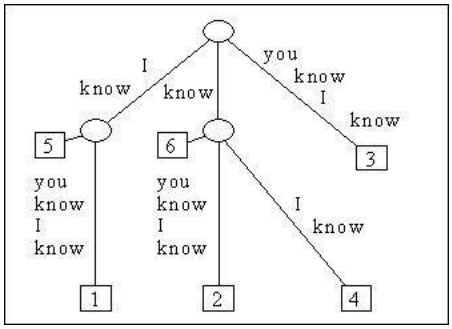
```
/**** **** **** **** **** ****
*   Function Name :      字典树(多路查找树)
*   Description :           HDOJ 1075 What Are You Talking About
```

```
*                          易于字符保存，插入和查找，时间复杂度都是线性
**** **** **** **** **** ****/
#include <cstdio>
#include <string>
using namespace std;

struct trie
{
   trie * next[26];
   int index;
};
trie *thead;
char dic[1000000][20];

inline trie * newnode()
{
   int i;
   trie *t;
   t=(trie*)malloc(sizeof(trie));
   memset(t,0,sizeof(trie));
   return t;
}

void insert(trie * s,char x[],int pos)
{
   int i;
   trie *t;
   for(i=0; x[i] ; i++) {
      if( s->next[ x[i]-'a' ] )    s=s->next[ x[i]-'a' ];
      else {
         t=newnode();
         s->next[ x[i]-'a' ]=t;
         s=t;
      }
   }//for
   s->index=pos;
}

void deltrie(trie * s)
{
   int i;
   for(i=0; i < 26 ;i++) {
      if( s->next[i] )
         deltrie(s->next[i]);
```

```
    }
    free(s);
    s=NULL;
}

int find(trie *s, char x[])
{
    int i;
    if(x[0] == 0)    return -1;
    for(i=0; x[i] ; i++) {
        if( s->next[ x[i]-'a' ] )    s=s->next[ x[i]-'a' ];
        else                         break;
    }
    if(x[i]==0)    return s->index;
    else        return -1;
}

int main()
{
    int t,n,i,j,all;
    char word[20],mars[20],ch;

    thead=newnode();
    while(scanf("%s",word)==1)
        if(word[0]=='S')    break;
     i=1;
     while(scanf("%s",dic[i])==1 && dic[i][0]!='E') {
         scanf("%s",mars);
         insert(thead,mars,i);
         i++;
     }
     all=i;
     while(scanf("%s",word)==1)
         if(word[0]=='S')    break;
     getchar();    j=0;
     while(scanf("%c",&ch)==1 && ch!='E') {
         if(ch>='a' && ch<='z') {
             mars[j]=ch;    j++;
         }
         else {
             mars[j]=0;
             t=find( thead , mars );
             j=0;
             if(t>0)    printf("%s",dic[t]);
```

```
            else if(mars[0]!=0)    printf("%s",mars);
            printf("%c",ch);
            }
    }//while
    deltrie(thead);
}
```

3. 后缀树




```
/**** **** **** **** **** ****
*   Function Name :          后缀树
*   Description :        PKU 2774 Long Long Message
*                        有效的支持字符串匹配和查询
**** **** **** **** **** ****/
#include<cstdio>
#include<string>

#define NUM                27
#define STARTCHAR          'a'
#define SPECIALCHAR        '{'
#define ERROR              -1
#define TYPE1              1
#define TYPE2              2
#define LEAF               1
#define NOTLEAF            2

struct SuffixTrie {
    int Start, End;
    SuffixTrie * Next[NUM];
    SuffixTrie * Link;
    SuffixTrie * Father;
    int Flag;
    int Length;
};

char str[100010], buf[100010];
SuffixTrie head;
```

```
SuffixTrie*P, *G, *U, *V, *q;
int W[3], len, len2;

void CreateNode(SuffixTrie * & Node) {
    int i;
    Node = (SuffixTrie * ) malloc(sizeof(SuffixTrie));
    Node -> Start = Node -> End = Node -> Length = ERROR;
    for (i = 0; i < NUM; i++) Node -> Next[i] = NULL;
    Node -> Link = Node -> Father = NULL;
    Node -> Flag = LEAF;
}

void Init(SuffixTrie & h, char s[]) {
    int i;
    h.Start = h.End = ERROR;
    for (i = 0; i < NUM; i++) h.Next[i] = NULL;
    h.Link = & h;
    h.Father = NULL;
    h.Flag = LEAF;
    h.Length = 0;

    len = strlen(s);
    s[len] = SPECIALCHAR;
    s[len + 1] = '\0';
    len++;
}

int FindV(char s[]) {
    int old;
    SuffixTrie * t, * newt;
    t = U -> Next[s[W[0]] - STARTCHAR];

    old = 0;
    while (W[2] > (t -> End) - (t -> Start) + 1 + old) {
        old += (t -> End - t -> Start + 1);
        t = t -> Next[s[W[0] + old] - STARTCHAR];
    }
    if (W[2] == (t -> End) - (t -> Start) + 1 + old) {
        V = t;
        P -> Link = V;
        return TYPE1;
    } else {
        CreateNode(newt);
        newt -> Start = t -> Start;
```

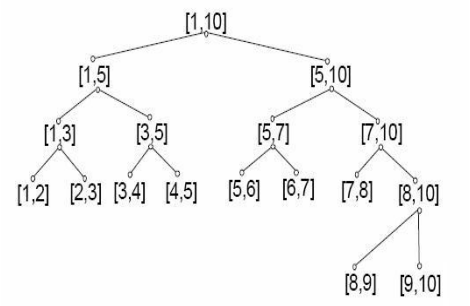
```
        newt -> End = t -> Start + W[2] - old - 1;
        newt -> Father = t -> Father;
        newt ->
            Length = newt -> Father -> Length + newt -> End - newt ->
            Start + 1;
        t -> Father -> Next[s[t -> Start] - STARTCHAR] = newt;
        t -> Start = newt -> End + 1;
        newt -> Next[s[t -> Start] - STARTCHAR] = t;
        t -> Father = newt;
        V = newt;
        P -> Link = V;
        return TYPE2;
    }
}

int Insert(SuffixTrie * Node, int start, char s[]) {
    int i, posbegin, posend;
    SuffixTrie * t;
    if (Node -> Next[s[start] - STARTCHAR] == NULL) {
        CreateNode(Node -> Next[s[start] - STARTCHAR]);
        Node -> Next[s[start] - STARTCHAR] -> Start = start;
        Node -> Next[s[start] - STARTCHAR] -> End = len - 1;
        Node -> Next[s[start] - STARTCHAR] -> Father = Node;
        Node -> Next[s[start] - STARTCHAR] ->
            Length = Node -> Length + len - start;
        Node -> Flag = NOTLEAF;
        P = Node;
        return TYPE1;
    } else {
        posbegin = Node -> Next[s[start] - STARTCHAR] -> Start;
        posend = Node -> Next[s[start] - STARTCHAR] -> End;
        for (i = posbegin; i <= posend; i++) {
            if (s[i] != s[start + i - posbegin]) break;
        }
        if (i == posend + 1) {
            return Insert(Node -> Next[s[start] - STARTCHAR], start + i - posbe
gin, s);
        } else {
            CreateNode(t);
            t -> Start = posbegin;
            t -> End = i - 1;
            t -> Flag = NOTLEAF;
            Node -> Next[s[start] - STARTCHAR] -> Start = i;
            t -> Next[s[i] - STARTCHAR] = Node -> Next[s[start] - STARTCHAR];
```

```
            t -> Next[s[i] - STARTCHAR] -> Father = t;
            Node -> Next[s[start] - STARTCHAR] = t;
            t -> Father = Node;
            t -> Length = Node -> Length + t -> End - t -> Start + 1;
            Insert(t, start + i - posbegin, s);
            G = Node;
            P = t;
            return TYPE2;
        }
    }
}

int Select(int start, char s[], int type) {
    int result1, result2, result;
    if (type == TYPE1) {
        U = P -> Link;
        result = Insert(U, start + U -> Length, s);
    } else {
        U = G -> Link;
        if (G -> Link == G) {
            W[0] = P -> Start + 1;
            W[1] = P -> End;
            W[2] = P -> End - P -> Start;
        } else {
            W[0] = P -> Start;
            W[1] = P -> End;
            W[2] = P -> End - P -> Start + 1;
        }
        if (W[2] == 0) {
            V = G;
            P -> Link = V;
            result = Insert(V, start, s);
        } else {
            result1 = FindV(s);
            result2 = Insert(V, start + V -> Length, s);
            if (result1 == TYPE2) {
                G = P -> Father;
                result = result1;
            } else result = result2;
        }
    }
    return result;
}
```

```
void BuildSuffixTrie(SuffixTrie & h, char s[]) {
   int i;
   int type;

   len = strlen(s);
   CreateNode(h.Next[s[0] - STARTCHAR]);
   h.Next[s[0] - STARTCHAR] -> Start = 0;
   h.Next[s[0] - STARTCHAR] -> End = len - 1;
   h.Next[s[0] - STARTCHAR] -> Father = & h;
   h.Next[s[0] - STARTCHAR] -> Length = h.Length + h.Next[s[0] - STARTC
HAR] -> End - h.Next[s[0] - STARTCHAR] -> Start + 1;
   h.Flag = NOTLEAF;
   type = TYPE1;
   P = & h;

   for (i = 1; i < len; i++) type = Select(i, s, type);
}

void DeleteSuffixTrie(SuffixTrie * & Node) {
   int i;
   for (i = 0; i < NUM; i++) {
      if (Node -> Next[i] != NULL) {
         DeleteSuffixTrie(Node -> Next[i]);
         Node -> Next[i] = NULL;
      }
   }
   free(Node);
}

int FindString(int start, char s[]) {
   int result;
   int i;
   int temp;
   SuffixTrie * x;
   x = P -> Next[s[start] - STARTCHAR];
   result = P -> Length;
   if (x == NULL) {
      P = P -> Link;
      return result;
   }
   temp = 0;
   for (i = start; i < len2; i++) {
      if (x -> Start + i - start - temp > x -> End) {
         temp = i - start;
```

```
            P = x;
            x = x -> Next[s[start + temp] - STARTCHAR];
            if (x == NULL) break;
        }
        if (s[i] != str[x -> Start + i - start - temp]) break;
        result++;
    }
    P = P -> Link;
    return result;
}

int Search(SuffixTrie & h, char s[]) {
    int result;
    int i;
    int temp;
    len2 = strlen(s);
    result = 0;
    P = & head;
    for (i = 0; i < len2; i++) {
        temp = FindString(i + P -> Length, s);
        if (result < temp) result = temp;
        if (result >= len2 - i) break;
    }
    return result;
}

int main() {
    int result;
    while (scanf("%s", str) != EOF) {
        Init(head, str);
        BuildSuffixTrie(head, str);
        scanf("%s", buf);
        result = Search(head, buf);
        printf("%d\n", result);
    }
}
```

4. 线段树




```
/**** **** **** **** **** ****
*    Function Name :          线段树
*    Description :        HDOJ 1542 Atlantis
*                         用于表示区间线段
**** **** **** **** **** ****/
#include<cstdio>
#include<algorithm>
using namespace std;

typedef struct ITREE_NODE {
    ITREE_NODE * pLChild, * pRChild;
    double left, right;          // 左端点，右端点
    double measure;              // 测度
    int count;                 // 覆盖计数器
    int lines;                   // 独立线段数
    int lbound, rbound;  // 覆盖左、右顶点的线段数目
}*PITREE_NODE;

inline void safe_add(int & v, int value) {
   v += value;
   if (v < 0) v = 0;
}

void itree_splite(const double * pList, PITREE_NODE pParent, const int iLeft,
const int iRight) {
   if (iRight - iLeft <= 1) return;
   int iMid = (iLeft + iRight) >> 1;
   pParent -> pLChild = new ITREE_NODE;
   pParent -> pRChild = new ITREE_NODE;
   memset(pParent -> pLChild, 0, sizeof(ITREE_NODE));
   memset(pParent -> pRChild, 0, sizeof(ITREE_NODE));
   pParent -> pLChild -> left = pList[iLeft];
   pParent -> pLChild -> right = pList[iMid];
   pParent -> pRChild -> left = pList[iMid];
   pParent -> pRChild -> right = pList[iRight];
   itree_splite(pList, pParent -> pLChild, iLeft, iMid);
```

```
    itree_splite(pList, pParent -> pRChild, iMid, iRight);
}

PITREE_NODE itree_generate(const double * pList, const int iListCount) {
    PITREE_NODE pRoot = new ITREE_NODE;
    memset(pRoot, 0, sizeof(ITREE_NODE));
    pRoot -> left = pList[0];
    pRoot -> right = pList[iListCount - 1];
    itree_splite(pList, pRoot, 0, iListCount - 1);
    return pRoot;
}

void itree_destroy(PITREE_NODE pParent) {
    if (pParent == NULL) return;
    if (pParent -> pLChild) itree_destroy(pParent -> pLChild);
    if (pParent -> pRChild) itree_destroy(pParent -> pRChild);
    delete pParent;
}

inline void itree_measure(PITREE_NODE pNode) {
    if (pNode -> count > 0)
        pNode -> measure = pNode -> right - pNode -> left;
    else if (pNode -> pLChild && pNode -> pRChild)
        pNode -> measure = pNode -> pLChild -> measure + pNode -> pRChil
d -> measure;
    else
         pNode -> measure = 0;
}

inline void itree_lines(PITREE_NODE pNode) {
    if (pNode -> count > 0) {
        pNode -> lines = 1;
    } else if (pNode -> pLChild && pNode -> pRChild) {
        if (pNode -> pLChild -> rbound && pNode -> pRChild -> lbound) {
            pNode -> lines = pNode -> pLChild -> lines + pNode -> lines - 1;
        } else {
            pNode -> lines = pNode -> pLChild -> lines + pNode -> lines;
        }
    } else {
        pNode -> lines = 0;
    }
}

// 插入的时候 value = 1, 删除的时候 value = -1
```

```
void itree_update(PITREE_NODE pParent, const double left, const double righ
t,
                  int value) {
   if (pParent -> left == left && pParent -> right == right) {
      safe_add(pParent -> count, value);
      safe_add(pParent -> lbound, value);
      safe_add(pParent -> rbound, value);
      itree_measure(pParent);
      itree_lines(pParent);
   } else {
      if (pParent -> pLChild -> right > left) {
         if (pParent -> pLChild -> right >= right) {
            itree_update(pParent -> pLChild, left, right, value);
         } else {
            itree_update(pParent -> pLChild, left,
                  pParent -> pLChild -> right, value);
            itree_update(pParent -> pRChild, pParent -> pRChild -> left,
                  right, value);
         }
      } else {
         itree_update(pParent -> pRChild, left, right, value);
      }
      itree_measure(pParent);
      itree_lines(pParent);
      if (left == pParent -> left) safe_add(pParent -> lbound, value);
        if (right == pParent -> right) {
         safe_add(pParent -> rbound, value);
      }
   }
}

void itree_insert(PITREE_NODE pParent, const double left, const double right
) {itree_update(pParent, left, right, 1); }

void itree_delete(PITREE_NODE pParent, const double left, const double right
) {itree_update(pParent, left, right, -1); }

struct EVENT {
    double x, y1, y2;
    int type;
};

bool cmp(const EVENT & a, const EVENT & b)
{ return a.x < b.x; }
```

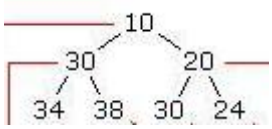
```
PITREE_NODE pRoot;
EVENT env[200];
double Y[200];
double tsize = 0.0;

int main() {
    double x1, x2, y1, y2;
    int i, n, n2, cas = 0;
    while (scanf("%d", & n) == 1 && n) {
        cas++;
        n2 = n << 1;
        for (i = 0; i < n2; i += 2) {
            scanf("%lf%lf%lf%lf", & x1, & y1, & x2, & y2);
            env[i].x = x1;
            env[i].y1 = y1;
            env[i].y2 = y2;
            env[i].type = 1;
            env[i + 1].x = x2;
            env[i + 1].y1 = y1;
            env[i + 1].y2 = y2;
            env[i + 1].type = -1;
            Y[i] = y1;
            Y[i + 1] = y2;
        }
        sort(env, env + n2, cmp);
        sort(Y, Y + n2);
        pRoot = itree_generate(Y, n2);
        for (i = 0; i < n2; ++i) {
            if (i > 0) tsize += pRoot -> measure * (env[i].x - env[i - 1].x);
            else tsize = 0.0;
            itree_update(pRoot, env[i].y1, env[i].y2, env[i].type);
        }
        itree_destroy(pRoot);
        printf("Test case #%d\nTotal explored area: %.2lf\n\n", cas, tsize);
    }
    return 0;
}
```

5. 并查集

```
/**** **** **** **** **** ****
*   Function Name :         并查集
*   Description :           集合操作, 并, 除, 判断
**** **** **** **** **** ****/
```

```
const int Max=1000;
typedef int ElemType;

int Parent[Max],Rank[Max];

int Find(int x)
{
   int temp = x, root, w;
   //搜寻根节点
   while(Parent[x]!=0)     x=Parent[x];
   root=x;
   x=temp;
    //压缩路径
   while(Parent[x]!=0) {
      w=Parent[x];
      Parent[x]=root;
      x=w;
   }
   return root;
}

int Union(int x,int y)
{
   int u, v, root;
   u=Find(x);
   v=Find(y);
   if(Rank[u] <= Rank[v]) {
      root = Parent[u] = v;
      if(Rank[u] == Rank[v])     Rank[v]++;
   }
   else     root=Parent[v]=u;
   return root;
}
```

6. 二叉堆




```
/**** **** **** **** **** ****
*   Function Name :     二叉堆
*   Description :       父结点的键值总是大於或等於任何一个子节点的键值
*                       便於寻找父节点和子节点
**** **** **** **** **** ****/
```

```
const int Max=1000;
typedef int ElemType;

ElemType Heap[Max];

int Sift_Up(int i)      //上移
{
   ElemType temp;
   bool flag;
   flag = true;
   if(i == 1)     return 0;
   do {
      if(Heap[i] > Heap[i/2])
      {temp=Heap[i];     Heap[i]=Heap[i/2];     Heap[i/2]=temp;}
      else     flag = false;
       i /= 2;
   }while(i>1 || flag);
   return 1;
}

int Sift_Down(int i,int n)   //下移
{
   bool flag;
   ElemType temp;
   flag = false;
   if(2*i > n)     return 0;
   do {
      i*=2;
      if(i+1 <= n && Heap[i+1] > Heap[i])     i++;
      if(Heap[i/2] < Heap[i])
      {temp=Heap[i];     Heap[i]=Heap[i/2];     Heap[i/2]=temp;}
      else     flag = false;
   }while(2*i<=n || flag);
   return 1;
}

int Insert(int &n,ElemType x)    //插入元素
{
   Heap[++n] = x;
   if( Sift_Up(n) )     return n;
}

int Delete(int &n,int i)   //输出元素
{
```

```
    ElemType x,y;
    x = Heap[i];    y = Heap[n];
    n--;
    if(i == n+1)    return x;
    Heap[i] = y;
    if(y >= x)    Sift_Up(i);
    else        Sift_Down(i,n);
    return x;
}

int Delete_Max(int &n)    //输出最大值
{
    ElemType x;
    x = Heap[1];
    Delete(n,1);
    return x;
}

int Make_Heap(int n)    //转换为大顶堆
{
    int i;
    for(i=n/2; i >= 1 ;i--)    Sift_Down(i,n);
    return n;
}

int HeapSort(int n)  //非降序排序
{
    int i;
    ElemType temp;
    Make_Heap(n);
    for(i=n; i >= 2 ;i--) {
        temp=Heap[i];    Heap[i]=Heap[1];    Heap[1]=temp;
        Sift_Down(1,i-1);
    }
    return 1;
}
```

7. 逆序数(归并排序)

```
/**** **** **** **** **** ****
*   Function Name :      逆序数(归并排序)
*   Description :          N*Log(N)
**** **** **** **** **** ****/
//逆序数值存放在 anti 中
int p[MAX], t[MAX], anti = 0;
```

```
void merge(int first, int last)
{
    int mid = (first+last)/2;
    int i1 = 0, i2 = first, i3 = mid+1;
    while(i2 <= mid && i3 <= last) {
        if(p[i2] > p[i3]) {
            t[i1++] = p[i3++];
            anti += mid-i2+1;
        }
        else t[i1++]=p[i2++];
    }
    while(i2 <= mid)    t[i1++] = p[i2++];
    while(i3 <= last)    t[i1++] = p[i3++];
    i1 = first;    i2 = 0;
    while(i2 < last-first+1)    p[i1++] = t[i2++];
}

void merge_sort(int first, int last)
{
    int mid;
    if(first<last) {
        mid = (first+last)/2;
        merge_sort(first, mid);
        merge_sort(mid+1, last);
        merge(first, last);
    }
}
```

8. 树状 DP

```
/**** **** **** **** **** ****
*   Function Name :       树状 DP
*   Description :         HDOJ 1561 The more, The Better
**** **** **** **** **** ****/
#include<cstdio>
#include<memory>
#include<queue>
using namespace std;
#define Max 210
int n,m,a[Max][Max];
struct inf
{
   int l,r,p;
   int v;
}tree[Max];
```

```
int tp,now;
queue<int> SQ;
char v[Max];

int main()
{
    int i,j;
    int root,pt,tv;
    while(scanf("%d%d",&n,&m)) {
        if(n==0 && m==0 )    break;
        memset(tree,0,sizeof(tree));
        memset(a,0,sizeof(a));
        memset(v,3,sizeof(v));
        while(!SQ.empty())    SQ.pop();
        for(i=1; i <= n ;i++) {
            scanf("%d%d", &root, &tree[i].v);
            if(tree[root].l == 0) {
                tree[root].l = i;
                v[root]--;
                tree[i].p = root;
            }
            else {
                pt = tree[root].l;
                while(tree[pt].r != 0)    pt = tree[pt].r;
                tree[pt].r = i;
                v[pt] -= 2;
                tree[i].p = pt;
            }
        }
        for(i=1;i<=n;i++)
            if(v[i]==3)    SQ.push(i);
          while(!SQ.empty()) {
              now = SQ.front();
              SQ.pop();

              a[now][1] = tree[now].v;
              for(i=1; i <= m ;i++)
                  a[now][i] = a[now][i] < a[ tree[now].r ][i] ? a[ tree[now].r ][i]
 : a[now][i];
              for(i=2; i <= m ;i++)
                  for(j=1; j <= i ;j++) {
                      tv = a[ tree[now].l ][j-1] + tree[now].v + a[ tree[now].r ][i
-j];
                      a[now][i] = a[now][i] < tv ? tv : a[now][i];
```

```
            }
            if(tree[ tree[now].p ].l == now)    v[ tree[now].p ]++;
            else                                v[ tree[now].p ] += 2;
            if(v[ tree[now].p ] == 3)  SQ.push(tree[now].p);
        }
        printf("%d\n",a[ tree[0].l ][m]);
    }
}
```

9. 欧拉路

```
/**** **** **** **** **** ****
*   Function Name :      欧拉路
*   Description :        ZJU 2730 Necklace
*   欧拉路的构造方法:
*   若图连同且度为奇数的节点不超过 2 个,则该图可以构造出欧拉路
*   先选一个度为奇数的节点(若没有就任选一个度为偶数的节点)
*   再以该节点为起点,用 dfs 遍历所有的弧(每条弧只遍历一次),遇到死胡同就回溯
*   在每次回溯时将所在弧按顺序记录下来,这组弧的排列就组成了一条欧拉路
**** **** **** **** **** ****/
#include<stdio.h>
#define MAXN 50
void find_path_euler(int n, int mat[][MAXN], int now, int& step, int* path)
{
    int i;
    for(i=n-1; i >= 0 ;i--)
        while(mat[now][i]) {
            mat[now][i]--, mat[i][now]--;
            find_path_u(n, mat, i, step, path);
        }
    path[ step++ ]=now;
}

int main()
{
    int n;
    int a[MAXN][MAXN];
    int i, j, cnt, mmin;
    int b[10000],c[10000];
    while(scanf("%d",&n)!=EOF) {
        for(i=0; i<n ;i++)
            for(j=0; j<n ;j++)
                if(j == i)    a[i][j] = 0;
                else          a[i][j] = a[j][i] = 1;
```

```
        cnt = 0;
        mmin = 2000000000;
        for(i=0; i<n ;i++) {
            find_path_u(n, a, i, cnt, b);
            if(cnt < mmin) {
                mmin = cnt;
                for(j=0; j<mmin ;j++)
                    c[j] = b[j];
                break;
            }
        }
        printf("%d\n", mmin-1);
        for(i=0; i<mmin-2 ;i++)
            printf("%d ",c[i]);
        printf("%d",c[i]);
        printf("\n");
    }
}
```

10. 八数码

```
/**** **** **** **** **** ****
*   Function Name :      八数码 Eight(Special Judge)
*   Description :       搜索 + 状态 hash
*   PKU(1077)       HDOJ(1043)       ZOJ(1217)
*   BFS          广搜       PKU(312ms)    HDOJ(TLE)       ZOJ(TLE)
*   BFS2       双向广搜    PKU(31ms)    HDOJ(1325ms)    ZOJ(TLE)
*   以上均是每次计算的运行耗时，ZOJ 的可以全部计算后保存状态
**** **** **** **** **** ****/
#include<cstdio>
#include<string>
#include<memory>
#include<queue>
using namespace std;
char input[100];
int state[10], s_num, e10[10], fac_n[10];
char hash_T[400000], step[10000], hash_T2[400000];
struct inf
{
   int pos;
   char mode;
};
queue<int> SQ;
queue<inf> SQ2;
```

```
int num_pos(int num,int x,int y)
{
    int temp=(x-1)*3+y;
    if(temp == num%10)    return 9;
    if(temp > num%10)    return (num / e10[9-temp+1]) %10;
    else    return (num / e10[9-temp] )%10;
}

int state_pos(int num,int x,int y)
{
    int temp=(x-1)*3+y;
    if(temp == state[9])    return 9;
    if(temp > state[9])    return state[temp-1];
    else    return state[temp];
}

inline int move(int num,char op)
{
    int t0,t1,t2;
    switch(op)
    {
    case 'r':
        if(num%10%3 == 0)    return 0;
        return num+1;
    case 'l':
        if((num-1)%10%3 ==0)    return 0;
        return num-1;
    case 'u':
        if(num%10 -3 <= 0)    return 0;
        t0 = 9-num%10 + 1;
        t1 = num / e10[t0];
        t2 = t1%1000;
        t1= t1- t2 + (t2 % 100) * 10 + t2 / 100;
        t1*= e10[t0];
        return (t1 + ( (num % e10[t0]) - 3));
    case 'd':
        if(num%10 +3 > 9)    return 0;
        t0 = 9-num%10 + 1 -3;
        t1 = num / e10[t0];
        t2 = t1%1000;
        t1= t1- t2 + (t2 % 10) * 100 + t2 / 10;
        t1*= e10[t0];
        return (t1 + ( (num % e10[t0]) + 3));
    }
```

```
}

bool be_solved()
{
    int i,j,anti=0;
    for(i=1;i<=8;i++)
        for(j=1;j<i;j++)
            if( state[i] < state[j] )
                anti++;
     if(anti%2)     return false;
     else          return true;
}

inline int hash(int num)
{
    int dig[10],i=9,j,sum,anti;
    if(num==0)    return -1;
    while(num)    dig[i]=num%10 , num/=10 , i-- ;
    sum=(9-dig[9])*fac_n[8];
    for(i=1;i<9;i++) {
        for(anti=0,j=1;j<i;j++)
            if(dig[i] < dig[j])
                anti++;
          sum += anti*fac_n[i-1];
    }
    return sum;
}

void BFS()
{
    int k,to_num,to_hash,i;
    memset(hash_T,0,sizeof(hash_T));
    while(!SQ.empty())    SQ.pop();
    SQ.push(123456789);
    hash_T[ hash(123456789) ]='e';
    while(!SQ.empty())
    {
        k=SQ.front();
        SQ.pop();

        to_num=move(k,'r');    to_hash=hash(to_num);
        if(to_hash>=0 && hash_T[ to_hash ]==0)
            hash_T[ to_hash ]='r' , SQ.push(to_num);
        to_num=move(k,'l');    to_hash=hash(to_num);
```

```
        if(to_hash>=0 && hash_T[ to_hash ]==0)
            hash_T[ to_hash ]='l' , SQ.push(to_num);
        to_num=move(k,'u');    to_hash=hash(to_num);
        if(to_hash>=0 && hash_T[ to_hash ]==0)
            hash_T[ to_hash ]='u' , SQ.push(to_num);
        to_num=move(k,'d');    to_hash=hash(to_num);
        if(to_hash>=0 && hash_T[ to_hash ]==0)
            hash_T[ to_hash ]='d' , SQ.push(to_num);
    }
}

void BFS2()
{
    int to_num,to_hash,i;
    char *phash,*phash2;
    char op;
    inf k,t;
    memset(hash_T,0,sizeof(hash_T));
    memset(hash_T2,0,sizeof(hash_T2));
    while(!SQ2.empty())     SQ2.pop();
    k.pos=s_num;     k.mode=1;
    SQ2.push(k);
    k.pos=123456789;     k.mode=2;
    SQ2.push(k);
    hash_T[ hash(s_num) ]='s';
    hash_T2[ hash(123456789) ]='e';
    while(!SQ2.empty()) {
        k=SQ2.front();
        SQ2.pop();
        to_hash=hash(k.pos);
        if(k.mode==1)
            if(hash_T2[ to_hash ]!=0)    break;
            else    phash=hash_T,phash2=hash_T2;
              if(k.mode==2)
                  if(hash_T[ to_hash ]!=0)    break;
                  else    phash=hash_T2,phash2=hash_T;
              t=k;
              t.pos=move(k.pos,'r');     to_hash=hash(t.pos);
              if(to_hash>=0 && phash[ to_hash ]==0)
                  phash[ to_hash ]='r' , SQ2.push(t);
              t.pos=move(k.pos,'l');     to_hash=hash(t.pos);
              if(to_hash>=0 && phash[ to_hash ]==0)
                  phash[ to_hash ]='l' , SQ2.push(t);
              t.pos=move(k.pos,'u');     to_hash=hash(t.pos);
```

```
            if(to_hash>=0 && phash[ to_hash ]==0)
                phash[ to_hash ]='u' , SQ2.push(t);
            t.pos=move(k.pos,'d');    to_hash=hash(t.pos);
            if(to_hash>=0 && phash[ to_hash ]==0)
                phash[ to_hash ]='d' , SQ2.push(t);
    }
    i=0;
    to_hash = hash(k.pos);
    to_num = k.pos;
    while( hash_T[ to_hash ] != 's' ) {
       switch( step[i++]=hash_T[ to_hash ] ) {
       case 'r':    op='l';break;
       case 'l':    op='r';break;
       case 'u':    op='d';break;
       case 'd':    op='u';break;
       }
       to_num=move(to_num,op);
       to_hash=hash(to_num);
    }
    while(i>0)    printf("%c",step[--i]);
    to_hash=hash(k.pos);
    to_num=k.pos;
    while( hash_T2[ to_hash ]!='e' ) {
       switch( hash_T2[ to_hash ] ) {
       case 'r':    op='l';break;
       case 'l':    op='r';break;
       case 'u':    op='d';break;
       case 'd':    op='u';break;
       }
       printf("%c",op);
       to_num=move(to_num, op );
       to_hash=hash(to_num);
    }
}

int main()
{
    int i,j;
    for(e10[0]=1,i=1;i<=9;i++)
       e10[i] =e10[i-1]*10;
    for(fac_n[0]=0,fac_n[1]=1,i=2;i<=9;i++)
       fac_n[i] =fac_n[i-1]*i;
    while( gets(input) ) {
       for(i=strlen(input)-1,j=8;i>=0;i--) {
```

```
            if(input[i]!=' ') {
                if(input[i]=='x')
                    state[9]=j+1;
                else state[j--]=input[i]-'0';
            }
        }
        for(s_num=0,i=9,j=1;i>0;i--,j*=10)
            s_num += state[i]*j;
        if( !be_solved() )
            printf("unsolvable\n");
        else {
            BFS2();
            printf("\n");
        }
    }
}
```

11. 高斯消元法

```
/**** **** **** **** **** ****
*    Function Name :        高斯消元法
*    Description :          求解线性方程组
*
*    void exchange_col(int p1,int p2,int n)
*    交换 p1 行和 p2 行的所有数据
*
*    bool gauss(int n)
*    求解系数矩阵为 n 的线性方程组，方程组无解返回 false，否则 true
*
*   x1 = x0 - f(x0)/f'(x0)   牛顿迭代法
**** **** **** **** **** ****/
const int num = 100;
double matrix[num][num + 1];   //系数矩阵，从 0 开始
double ans[num];               //结果数组

void exchange_col(int p1,int p2,int n)  //交换 p1 行和 p2 行的所有数据
{
    double t;
    int i;

    for(i = 0 ; i <= n ; i++)
        t = matrix[p1][i],matrix[p1][i] = matrix[p2][i],matrix[p2][i] = t;
}

bool gauss(int n)   //求解系数矩阵为 n 的线性方程组
```

```
{
    int i,j,k;
    int p;
    double r;

    for(i = 0 ; i < n - 1 ; i++) {
        p = i;
        for(j = i + 1 ; j < n ; j++) {   //寻找 i 列绝对值最大值位置
            if(abs(matrix[j][i]) > abs(matrix[p][i]))
                p = j;
        }
       if(p != i)     exchange_col(i,p,n);
     if(matrix[i][i] == 0) return false;
        for(j = i + 1 ; j < n ; j++) {       //剩余列进行消元
            r = matrix[j][i] / matrix[i][i];
            for(k = i ; k <= n ; k++)
                matrix[j][k] -= r * matrix[i][k];
        }
    }
    for(i = n - 1 ; i >= 0 ; i--) {   //获得结果
        ans[i] = matrix[i][n];
        for(j = n - 1 ; j > i ; j--)
            ans[i] -= matrix[i][j] * ans[j];
     if(matrix[i][i] == 0) return false;
        ans[i] /= matrix[i][i];
    }
    return true;
}
```

12. 字符串匹配(KMP 算法)

```
/**** **** **** **** **** ****
*    Function Name :        字符串匹配(KMP 算法)
*    Description :       O(N+M)
**** **** **** **** **** ****/
void get_nextval(const string & s, int * p)
{
    int i = 0,j = -1;
    p[0] = -1;
    while(i < s.size()) {
        if(j == -1 || s[i] == s[j]) {
            ++i,++j;
            if(s[i] != s[j]) p[i] = j;
            else p[i] = p[j];
        }
```

```
        else j = p[j];
    }
}
int Index_KMP(const string & s, const string & s1, int pos)
{
    int i = pos - 1,j = 0;
    int * next = new int[s1.size()];

    get_nextval(s1,next);
    while(i <= s.size() && j <= s1.size()) {
        if(j == -1 || s[i] == s1[j])    ++i,++j;
        else    j = next[j];
    }
    if(j > s1.size())    return i - s1.size();
    else    return -1;
}
```

13. 全排列,全组合

```
/**** **** **** **** **** ****
*   Function Name :      全排列,全组合
**** **** **** **** **** ****/
void createper(int n) //全排列
{
    int total,i,j,k,t,*a=new int[n],top;
    total=1;
    for(i=1;i<=n;i++) {
        a[i]=i;
        total*=i;
    }
    for(i=1;i<n;i++)    printf("%d ",a[i]);
    printf("%d\n",a[n]);
    for(i=1;i<total;i++) {
        j=n;
        while(a[j]<a[j-1]) j--;
        k=n;
        while(a[j-1]>a[k]) k--;
        t=a[j-1];
        a[j-1]=a[k];
        a[k]=t;
        top=(j+n-1)/2;
        for(k=j;k<=top;k++) {
            t=a[k];
            a[k]=a[n-k+j];
            a[n-k+j]=t;
```

```
            }
            for(j=1;j<n;j++) printf("%d ",a[j]);
            printf("%d\n",a[n]);
        }
    }

    void createfab(int m,int n) //全组合
    {
        int i,j,lcount,*a=new int[n+2];
        for(i=1;i<=n;i++)     a[i]=i;
        a[n+1]=m+1;
        for(j=1;j<n;j++)     printf("%d ",a[j]);
        printf("%d\n",a[n]);
        lcount=1;
        while(a[1]<m-n+1) {
            for(i=n;i>0;i--) {
                if(a[i]<a[i+1]-1) {
                    a[i]++;
                    for(j=i;j<n;j++)     a[j+1]=a[j]+1;
                    for(j=1;j<n;j++)     printf("%d ",a[j]);
                    printf("%d\n",a[n]);
                    lcount++;
                    break;
                }
            }
        }
    }
```

14. 二维线段树

```
/**** **** **** **** **** ****
*     Function Name :     二维线段树RMQ
*     Description :       HDOJ 1823 Luck and Love
**** **** **** **** **** ****/
#include <cstdio>
#include <string>
#include <algorithm>
using namespace std;
#define NMAX 500000
#define MQ(x,y) ((x)>(y)?(x):(y))
struct node {
    node * pleft, * pright;
    node * ytree;
    int left, right;
    int M;
```

```
}mem[NMAX];
int mem_pos;

node * new_node()
{
    node * pt = &mem[mem_pos ++];
    memset(pt,0,sizeof(node));
    pt ->M = -1;//maximum or minimum
    return pt;
}

node * create_tree(int x1, int y1, int x2, int y2, bool flag)
{
    node * root = new_node();
    if(flag) {// first dimension
        root ->left = x1;
        root ->right = y1;
        root ->ytree = create_tree(x1, y1, x2, y2, false);
        if(x1 != y1) {
            int mid = (x1+y1)/2;
            root ->pleft = create_tree(x1, mid, x2, y2, true);
            root ->pright = create_tree(mid+1, y1, x2, y2, true);
        }
    }
    else {// second dimension
        root ->left = x2;
        root ->right = y2;
        if(x2 != y2) {
            int mid = (x2+y2)/2;
            root ->pleft = create_tree(x1, y1, x2, mid, false);
            root ->pright = create_tree(x1, y1, mid+1, y2, false);
        }
    }
    return root;
}

void update(node * root, int d1, int d2, int v, bool flag)
{
    int mid = (root ->left + root ->right)/2;
    if(flag) {// first dimension
        update(root ->ytree, d1, d2, v, false);
        if(root ->left < root ->right) {
            if(d1 <= mid) {
                update(root ->pleft, d1, d2, v, true);
```

```
            }
            else {
                update(root ->pright, d1, d2, v, true);
            }
        }
    }
    else {// second dimension
        if(root ->left == root ->right) {
            root ->M = MQ(root ->M, v);
        }
        else {
            if(d2 <= mid) {
                update(root ->pleft, d1, d2, v, false);
            }
            else {
                update(root ->pright, d1, d2, v, false);
            }
            root ->M = MQ(root ->pleft ->M, root ->pright ->M);
        }
    }
}

int query(node * root, int x1, int y1, int x2, int y2, bool flag)
{
    int lmq, rmq;
    int mid = (root ->left + root ->right)/2;
    if(flag) {// first dimension
        if(root ->left == x1 && root ->right == y1) {
            return query(root ->ytree, x1, y1, x2, y2, false);
        }
        else {
            if(y1 <= mid) {
                return query(root ->pleft, x1, y1, x2, y2, true);
            }
            if(x1 > mid) {
                return query(root ->pright, x1, y1, x2, y2, true);
            }
            lmq = query(root ->pleft, x1, mid, x2, y2, true);
            rmq = query(root ->pright, mid+1, y1, x2, y2, true);
        }
    }
    else {// second dimension
        if(root ->left == x2 && root ->right == y2) {
            return root ->M;
```

```
        }
        else {
            if(y2 <= mid) {
                return query(root ->pleft, x1, y1, x2, y2, false);
            }
            if(x2 > mid) {
                return query(root ->pright, x1, y1, x2, y2, false);
            }
            lmq = query(root ->pleft, x1, y1, x2, mid, false);
            rmq = query(root ->pright, x1, y1, mid+1, y2, false);
        }
    }
    return MQ(lmq, rmq);
}

int main()
{
    int m;
    char cmd;
    while(scanf("%d", &m), m) {
        mem_pos = 0;
        node * root = create_tree(100,200,0,1000,true);
        while(m --) {
            getchar();
            cmd = getchar();
            if(cmd == 'I') {
                int h, ia, il;
                double a,l;
                scanf("%d %lf %lf", &h, &a, &l);
                ia = 10*(a+0.05);
                il = 10*(l+0.05);
                update(root, h, ia, il, true);
            }
            else {
                int h1, h2, ia1, ia2;
                double a1, a2;
                scanf("%d %d %lf %lf", &h1, &h2, &a1, &a2);
                ia1 = 10*(a1+0.05);
                ia2 = 10*(a2+0.05);
                if(h1 > h2) {
                    swap(h1, h2);
                }
                if(ia1 > ia2) {
                    swap(ia1, ia2);
```

```
                }
                int t = query(root, h1, h2, ia1, ia2, true);
                if(t == -1) {
                    puts("-1");
                }
                else {
                    printf("%.1lf\n", t / 10.0);
                }
            }
        }
    }
```

15. 稳定婚姻匹配

```
/**** **** **** **** **** ****
*    Function Name :    稳定婚姻匹配gale_shapley算法
*    Description :      HDOJ 1522 Marriage is Stable
**** **** **** **** **** ****/
//rmw[i][j]代表i男对女生的喜欢排名
//lwm[i][j]代表i女对j男的喜欢程度
const int MAX = 510;
int w,m,n;
int rmw[MAX][MAX];
int lmw[MAX][MAX], lwm[MAX][MAX];
int couple[MAX];
char sman[MAX][110], swoman[MAX][110];

queue<int> SQ;
void gale_shapley()
{
    int i,man,woman;
    while(!SQ.empty()) {
        SQ.pop();
    }
    memset(couple,-1,sizeof(couple));
    for(i=1;i<=n;i++) {
        SQ.push(i);
    }
    while(!SQ.empty()) {
        man = SQ.front();
        for(i=1;i<=n;i++) {
            if(rmw[man][i] != -1) {
                //选择为被拒绝且最喜欢的女生
                woman = rmw[man][i];
                rmw[man][i] = -1;
```

```
                int pre = couple[woman];
                if(pre == -1) {
                    couple[woman] = man;
                    SQ.pop();
                    break;
                }
                else {
                    if(lwm[woman][man] > lwm[woman][pre]) {
                        SQ.pop();
                        SQ.push(pre);
                        couple[woman] = man;
                        break;
                    }
                }
            }
        }
    }//while
}
```

16. 后缀数组

```
/**** **** **** **** **** ****
*    Function Name :    后缀数组O(NLogN)
*    Description :      PKU 2774 Long Long Message
**** **** **** **** **** ****/
#include <cstdio>
#include <string>
using namespace std;
const int MAX = 250000;
char txt[MAX];
int mem[3][MAX], c[MAX], height[MAX];
int * SA, * nSA, * Rank, * nRank;
int len, l1, l2;
//O(NlogN)
//SA[ rank ] = who;
//Suffix(SA[i]) < Suffix(SA[i+1]) , 1≤i<n
//Rank[ who ] = rank;
//k-Rank[i]代表加上满足Suffix(j) <k Suffix(i)的j的个数
void init()
{
    l1 = strlen(txt);
    txt[l1] = 1;//特殊结尾
    gets(txt + l1+1);
    l2 = strlen(txt + l1+1);
    len = l1 + l2+1;
```

```
    txt[len ++] = 1; //特殊结尾
}
//性质.1 对k≥n ， Suffix(i) <k Suffix(j) 等价于Suffix(i) < Suffix(j)
//性质.2 Suffix(i) =2k Suffix(j) 等价于
//Suffix(i) =k Suffix(j) 且Suffix(i+k) =k Suffix(j+k)
//性质.3 Suffix(i) <2k Suffix(j) 等价于
//Suffix(i) <k Suffix(j) 或(Suffix(i) =k Suffix(j) 且Suffix(i+k) <k Suffix(j+k))
void suffix_array()
{
    int i, j, k;
    SA = mem[0]; nSA = mem[1]; Rank = mem[2];
    memset(c, 0, sizeof(c));
    for(i=0;i<len;i++) {
        c[ txt[i] ] ++;
    }
    for(i=0;i<128;i++) {
        c[i+1] += c[i];
    }
    for(i=0;i<len;i++) {
        SA[ -- c[txt[i]] ] = i;
    }
    Rank[ SA[0] ] = 0;
    for(i=1;i<len;i++) {
        Rank[ SA[i] ] = Rank[ SA[i-1] ];
        if(txt[ SA[i] ] != txt[ SA[i-1] ]) {
            Rank[ SA[i] ] ++;
        }
    }
    for(k=1;k<len && Rank[SA[len-1]]<len-1;k*=2) {
        memset(c, 0, sizeof(c));
        for(i=0;i<len;i++) {
            c[ Rank[SA[i]] ] ++;
        }
        for(i=1;i<len;i++) {
            c[i] += c[i-1];
        }
        for(i=len-1;i>=0;i--) {
            if(SA[i] >= k) {
                nSA[ -- c[ Rank[SA[i]-k] ] ] = SA[i] - k;
            }
        }
        for(i=len-k;i<len;i++) {
            nSA[ -- c[ Rank[i] ] ] = i;
        }
```

```
        nRank = SA;
        nRank[ nSA[0] ] = 0;
        for(i=1;i<len;i++) {
            nRank[ nSA[i] ] = nRank[ nSA[i-1] ];
            if(Rank[nSA[i]] != Rank[nSA[i-1]] || Rank[nSA[i]+k] !=
Rank[nSA[i-1]+k]) {
                nRank[nSA[i]] ++;
            }
        }
        SA = nSA;
        nSA = Rank;
        Rank = nRank;
    }
}
//LCP(i,j)=lcp(Suffix(SA[i]),Suffix(SA[j])
//height[i]=LCP(i,i+1)，≤i<n
int getlcp()
{
    int i, j, k, rs;
    for (i = 0, k = 0;i < len;i++) {
        if (Rank[i] == len - 1) {
            height[Rank[i]] = k = 0;
        }
        else {
            if (k > 0) {
                k --;
            }
            j = SA[Rank[i] +1];
            while(txt[i + k] == txt[j + k]) {
                k ++;
            }
            height[Rank[i]] = k;
        }
    }
    for (i = 0, rs = 0;i < len - 1;i++) {
        if (rs < height[i] && (SA[i] < l1) != (SA[i+1] < l1)) {
            rs = height[i];
        }
    }
    int t = min(l1,l2);
    return min(t, rs);
}

int main()
```

```
{
    gets(txt);
    init();
    suffix_array();
    printf("%d\n", getlcp());
    return 0;
}
```

17. 左偏树

```
/**** **** **** **** **** ****
*    Function Name :    左偏树
*    Description :      HDOJ 1512 Monkey King
*                       二叉堆的变形，方便堆的合并
**** **** **** **** **** ****/
#include <cstdio>
#include <string>
#include <queue>
#include <algorithm>
using namespace std;
const int MAX = 101000;
struct node {
    int v, dis;//键值，距离
    node * pl, * pr;//左右子树
    node * pf;//父节点
}mem[MAX];
int mem_pos;
int value[MAX], n;

node * new_node() {
    node * pt = mem + (mem_pos ++);
    memset(pt, 0, sizeof(node));
    return pt;
}
//清除节点休息
inline void clear(node * pos) {
    if(pos == NULL) return;
    pos->pl = pos->pr = pos->pf = NULL;
    pos->dis = 0;
}
//合并堆O(log N)
node * merge(node * pa, node * pb) {
    if(pa == NULL) return pb;
    if(pb == NULL) return pa;
    //maximum vertex heap
```

```
    if(pb->v > pa->v) std::swap(pa, pb);
    pa->pr = merge(pa->pr, pb);
    if(pa->pr) {
        if(pa->pl == NULL || pa->pr->dis > pa->pl->dis) {
            std::swap(pa->pl, pa->pr);
        }
    }
    if(pa->pr == NULL) pa->dis = 0;
    else pa->dis = pa->pr->dis +1;
    if(pa->pl) pa->pl->pf = pa;
    if(pa->pr) pa->pr->pf = pa;
    return pa;
}
//插入节点
inline node * insert(node * root, node * val) {
    return merge(root, val);
}
//删除最大项
inline node * delete_max(node * root) {
    node * pt = root;
    root = merge(root ->pl, root ->pr);
    if(root) root->pf = NULL;
    clear(pt);
    return root;
}
//取得最大值
inline int get_max(node * root) {
    return root ->v;
}
//构建左偏树O(N)
inline node * make_leftist_tree() {
    queue<node *> SQ;
    node * ptemp;
    int i;
    ptemp = new_node();
    for(i=0;i<n;i++) {
        ptemp->v = value[i];
        SQ.push(ptemp);
    }
    while(!SQ.empty()) {
        int l = SQ.size();
        if(l == 1) return SQ.front();
        while(l --) {
            node * pa = SQ.front();
```

```
            SQ.pop();
            node * pb = SQ.front();
            SQ.pop();
            SQ.push(merge(pa, pb));
        }
    }
}
//删除已知任意点O(log N)
inline void delete_any(node * pos) {
    node * ppre = pos->pf;
    node * pnew = delete_max(pos);
    if(pnew) pnew->pf = ppre;
    if(ppre) {
        if(ppre->pl == pos) ppre->pl = pnew;
        else ppre->pr = pnew;
    }
    while(ppre) {
        int vl = -1, vr = -1;
        if(ppre->pl) vl = ppre->pl->dis;
        if(ppre->pr) vr = ppre->pr->dis;
        if(vl < vr) std::swap(ppre->pl, ppre->pr);
        if(vr +1 == ppre->dis) return;
        ppre->dis = vr +1;
        pnew = ppre;
        ppre = ppre->pf;
    }
}

node ltree[MAX];
int main() {
    int i,j;
    int m,t;
    while(scanf("%d", &n)==1) {
        for(i=0;i<n;i++) {
            scanf("%d", &t);
            ltree[i].v = t;
            ltree[i].dis = 0;
            ltree[i].pl = ltree[i].pr = ltree[i].pf = NULL;
        }
        scanf("%d", &m);
        int a,b;
        while(m --) {
            scanf("%d %d", &a,&b);
            a --; b --;
```

```
            node * pa, * pb;
            pa = ltree +a;
            pb = ltree +b;
            while(pa->pf) pa = pa->pf;
            while(pb->pf) pb = pb->pf;
            if(pa == pb)     puts("-1");
            else {
                node * p1 = delete_max(pa);
                node * p2 = delete_max(pb);
                pa->v /= 2;
                pb->v /= 2;
                p1 = insert(p1, pa);
                p1 = insert(p1, pb);
                p1 = merge(p1, p2);
                printf("%d\n", get_max(p1));
            }
        }
    }
}
```

18. 标准 RMQ-ST

```
/**** **** **** **** **** ****
*    Function Name :    标准RMQ-ST
*    Description :      PKU 3264 Balanced Lineup
**** **** **** **** **** ****/
#include <cstdio>
#include <string>
#include <algorithm>
using namespace std;

const int MAX = 51000;
const int LOGMAX = 16;
int n,q;
int st_max[LOGMAX][MAX], st_min[LOGMAX][MAX];

void make_st()
{
    int i,j,k;
    for(j=1; (1<<j) <= n ;j++) {
        k = 1<<(j-1);
        for(i=0; i+k < n ;i++) {
            st_max[j][i] = max(st_max[j-1][i], st_max[j-1][i+k]);
            st_min[j][i] = min(st_min[j-1][i], st_min[j-1][i+k]);
        }
```

```
    }
}

int rmq(int a,int b,int flag)
{
    int dis = abs(b-a) +1;
    int k;
    for(k=0; (1<<k) <= dis ;k++) ;
    k --;
    if(flag > 0) {
        return max(st_max[k][a], st_max[k][b-(1<<k)+1]);
    }
    else {
        return min(st_min[k][a], st_min[k][b-(1<<k)+1]);
    }
}

int main()
{
    while(scanf("%d %d", &n,&q)==2) {
        int i;
        for(i=0;i<n;i++) {
            scanf("%d", &st_max[0][i]);
            st_min[0][i] = st_max[0][i];
        }
        make_st();
        for(i=0;i<q;i++) {
            int a,b;
            scanf("%d %d", &a,&b);
            printf("%d\n", rmq(a-1,b-1,1) - rmq(a-1,b-1,-1));
        }
    }
}
```

19. 度限制最小生成树

```
/**** **** **** **** **** ****
*    Function Name :    度限制最小生成树
*    Description :      PKU 1639 Picnic Planning
*                       有一个顶点有度限制，如果所有点都有限制，当限制>4时是NP
**** **** **** **** **** ****/
#include <cstdio>
#include <string>
#include <queue>
#include <vector>
```

```
#include <map>
#include <algorithm>
using namespace std;
const int MAX = 50;
int t,n,m;
map<string , int> names;
int path[MAX][MAX];
//dmax[i]: vi->park，不与park相连的边的最大权值
int dmax[MAX];
struct node {
    int s,t;
    int dis;
    bool operator < (const node & tt) const {
        return dis > tt.dis;
    }
};
bool vis[MAX];
//block[i]: vi所属连通分量
//bs: 连通分量数目
//v0min[i][2]: park与第i个连通分量的最小权值[0]，连接顶点[1]
int block[MAX], v0min[MAX][2], bs;
//mst: 度限制生成树
vector<int> mst[MAX];
queue<int> sq;
//最小花费，park下标，限制度数
int cost, park, deg;
//O(NlogN) prime求所有连通分量mst
void prime_all_mst() {
    int i,j;
    priority_queue<node> pq;
    node now, next;
    memset(vis, 0,sizeof(vis));
    for(i=0;i<=n;i++) mst[i].clear();
    vis[park] = true;
    block[park] = 1; bs = 1;//park为第个连通分量
    cost = 0;
    for(i=1;i<=n;i++) {
        if(!vis[i]) {
            bs ++;
            while(!pq.empty()) pq.pop();
            now.s = i; now.t = i; now.dis = 0;
            pq.push(now);
            while(!pq.empty()) {
                now = pq.top();
```

```
                pq.pop();
                if(vis[now.t]) continue;
                vis[now.t] = true;
                mst[now.s].push_back(now.t);
                mst[now.t].push_back(now.s);
                block[now.t] = bs;
                cost += now.dis;
                next.s = now.t;
                for(j=1;j<=n;j++) {
                    if(!vis[j] && path[next.s][j] != -1) {
                        next.t = j;
                        next.dis = path[next.s][j];
                        pq.push(next);
                    }
                }
            }
        }
    }
}
//O(N) park连接各连通分量
bool connect_block() {
    int i,j,k;
    //选取连接相邻连通分量的最小边
    for(i=2;i<=bs;i++) v0min[i][0] = INT_MAX;
    while(!sq.empty()) sq.pop();
    for(i=1;i<=n;i++) {
        if(path[park][i] != -1 && v0min[block[i]][0] > path[park][i]) {
            v0min[block[i]][0] = path[park][i];
            v0min[block[i]][1] = i;
        }
    }
    k = 0;
    for(i=2;i<=bs;i++) {
        if(v0min[i][0] != INT_MAX) {
            cost += v0min[i][0];
            path[park][ v0min[i][1] ] = -1;
            dmax[ v0min[i][1] ] = INT_MIN;
            sq.push(v0min[i][1]);//用来初始化dmax
            k ++;//能连通的分量数
        }
    }
    //图连通，且限制度数大于等于连通分量数
    deg -= bs-1;
    return k >= bs-1 && deg >= 0;
```

```
}
//O(N) 计算dmax
void cal_dmax() {
    int i;
    memset(vis, 0, sizeof(vis));
    while(!sq.empty()) {
        int now = sq.front();
        sq.pop();
        vis[now] = true;
        for(i=0;i<mst[now].size();i++) {
            int next = mst[now][i];
            if(!vis[next]) {
                dmax[next] = max(dmax[now], path[now][next]);
                sq.push(next);
                vis[next] = true;
            }
        }
    }
}
//O(N) 差额最小删除操作
void del_path(int pos, int val) {
    int i;
    queue<int> sq2;
    memset(vis, 0, sizeof(vis));
    sq2.push(pos);
    vis[pos] = true;
    while(!sq2.empty()) {
        int now = sq2.front();
        sq2.pop();
        for(i=0;i<mst[now].size();i++) {
            int next = mst[now][i];
            if(!vis[next]) {
                if(val == path[now][next]) {
                    mst[now].erase(mst[now].begin() +i);
                    return;
                }
                sq2.push(next);
                vis[next] = true;
            }
        }
    }
}
//O(deg*N)
bool deg_limit_mst() {
```

```
    int i,j,v;
    int minv,minp;
    cal_dmax();
    for(i=0;i<deg;i++) {
        minv = INT_MAX; minp = -1;
        for(j=1;j<=n;j++) {
            if(path[park][j] != -1) {//差额最小选择操作
                if(minv > path[park][j] - dmax[j]) {
                    minv = path[park][j] - dmax[j];
                    minp = j;
                }
            }
        }
        v = cost + minv;
        if(minp == -1 || v >= cost) return false;
        cost = v;
        path[park][minp] = -1;//差额最小添加删除操作
        del_path(minp, dmax[minp]);
        mst[park].push_back(minp);
        while(!sq.empty()) sq.pop();
        sq.push(minp);
        dmax[minp] = INT_MIN;
        cal_dmax();
    }
    for(i=0;i<mst[park].size();i++) mst[ mst[park][i] ].push_back(park);
    return true;
}

int main() {
    int i,j;
    char n1[20], n2[20];
    names.clear();
    scanf("%d", &m);
    memset(path, -1, sizeof(path));
    n = 1;
    for(i=0;i<m;i++) {
        int x,y,z;
        scanf("%s %s %d", n1,n2,&z);
        x = names[string(n1)];
        y = names[string(n2)];
        if(x == 0) names[string(n1)] = x = n ++;
        if(y == 0) names[string(n2)] = y = n ++;
        if(strcmp(n1,"Park") == 0) park = x;
        else if(strcmp(n2,"Park") == 0) park = y;
```

```
            path[x][y] = path[y][x] = z;
        }
        n --;
        scanf("%d", &deg);
        prime_all_mst();
        connect_block();
        deg_limit_mst();
        printf("Total miles driven: %d\n", cost);
}
```

20. 最优比率生成树

```
/**** **** **** **** **** ****
*     Function Name :    最优比率生成树(迭代法)
*     Description :       PKU 2728 Desert King
**** **** **** **** **** ****/
#include <cstdio>
#include <string>
#include <cmath>
#include <algorithm>
using namespace std;

const int MAX = 1100;
int n;
struct point {
    int x,y,z;
}vi[MAX];
struct node {
    int s, t;
    double dis;
    bool operator < (const node & tt) const {
        return dis > tt.dis;
    }
};
double dist[MAX][MAX];
bool vis[MAX];
double rate;

double prime() {
    double cost = 0;
    double len = 0;
    double d[MAX],v;
    int pre[MAX];
    int i,j;
    memset(vis, 0, sizeof(vis));
```

```
    vis[0] = true;
    for(i=1;i<n;i++) {
        d[i] = abs(vi[0].z-vi[i].z) - rate*dist[0][i];
        pre[i] = 0;
    }
    for(i=1;i<n;i++) {
        double minv = INT_MAX;
        int minp = -1;
        for(j=1;j<n;j++) {
            if(!vis[j] && minv > d[j]) {
                minv = d[j];
                minp = j;
            }
        }
        vis[minp] = true;
        cost += abs(vi[pre[minp]].z - vi[minp].z);
        len += dist[pre[minp]][minp];
        for(j=1;j<n;j++) {
            if(!vis[j] && d[j] > (v=abs(vi[minp].z-vi[j].z) - rate*dist[minp][j])) {
                d[j] = v;
                pre[j] = minp;
            }
        }
    }
    return cost / len;
}

int main() {
    int i,j;
    while(scanf("%d", &n), n) {
        for(i=0;i<n;i++) {
            scanf("%d %d %d", &vi[i].x, &vi[i].y, &vi[i].z);
        }
        for(i=0;i<n;i++) {
            dist[i][i] = 0;
            for(j=i+1;j<n;j++) {
                dist[i][j] = dist[j][i] = sqrt(1.0*(vi[i].x-vi[j].x)*(vi[i].x-vi[j].x) +
(vi[i].y-vi[j].y)*(vi[i].y-vi[j].y));
            }
        }
        rate = 0;
        while(true) {
            double pre = rate;
            rate = prime();
```

```
            if(fabs(rate - pre) < 0.001) break;
        }
        printf("%.3lf\n", rate);
    }
}
```

21. 最小花费置换

```
//Cow Sorting
//对一个轮换进行处理的时候，应该考虑在轮换内进行交换，或与轮换外的元素交换之后，使代价值更小
#include <cstdio>
#include <string>
#include <functional>
#include <algorithm>
using namespace std;
int g[10100],n;
bool vis[10100];
int pos[101000];
struct node {
    int v,p;
    bool operator < (const node & t) const {
        return v < t.v;
    }
}g2[10100];
int main() {
    int i,j;
    int sum, mmin;
    while(scanf("%d", &n)==1) {
        sum = 0;
        mmin = INT_MAX;
        for(i=1;i<=n;i++) {
            scanf("%d", g+i);
            sum += g[i];
            g2[i].v = g[i];
            g2[i].p = i;
            mmin = min(mmin, g[i]);
            vis[i] = false;
        }
        sort(g2+1,g2+n+1);
        for(i=1;i<=n;i++) {
            pos[ g2[i].v ] = g2[i].p;
        }
        for(i=1;i<=n;i++) {
            if(!vis[i]) {
```

```
                int tpos = i;
                int len = 0;
                int tmin = INT_MAX;
                do {
                    tmin = min(tmin, g[tpos]);
                    vis[tpos] = true;
                    tpos = pos[ g2[tpos].v ];
                    len ++;
                } while(tpos != i);
                //选择两种方案中的最优方案
                sum += min( (len-2)*tmin, (len+1)*mmin +tmin);
            }
        }
        printf("%d\n", sum);
    }
}
```

22. 区间 K 大数

```
//POJ 2104
#include <cstdio>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
const int NMAX = 100000;
const int LOGNMAX = 17 +1;
int sortseq[LOGNMAX][NMAX];
int num[NMAX];
struct node {
    int l,r,d;
    node * pl,* pr;
}mem[(NMAX<<1)+100];
int mempos,n,m;
node * root;
node * make_tree(int l,int r,int d) {
    node * rt = mem+(mempos ++);
    rt->l = l; rt->r = r; rt->d = d;
    if (l == r) {
        sortseq[d][l] = num[l];
        return rt;
    }
    int mid = (l+r) >> 1;
    rt->pl = make_tree(l,mid,d+1);
    rt->pr = make_tree(mid+1,r,d+1);
```

```
    int i=l,j=mid+1,k=l;
    while (i<=mid && j<=r) {
        if (sortseq[d+1][i] < sortseq[d+1][j]) sortseq[d][k++] =
sortseq[d+1][i++];
        else sortseq[d][k++] = sortseq[d+1][j++];
    }
    while (i<=mid) sortseq[d][k++] = sortseq[d+1][i++];
    while (j<=r) sortseq[d][k++] = sortseq[d+1][j++];
    return rt;
}
int s,t,rank;
int query(node * rt,int val) {
    int i,mid,ret;
    if (s <= rt->l && rt->r <= t) {
        if (val <= sortseq[rt->d][rt->l]) return 0;
        else if (sortseq[rt->d][rt->r] < val) return rt->r - rt->l +1;
        else if (sortseq[rt->d][rt->r] == val) return rt->r - rt->l;
        int l = rt->l, r = rt->r, mid;
        while (l <= r) {
            mid = (l+r) >> 1;
            if (val <= sortseq[rt->d][mid]) r = mid-1;
            else l = mid+1;
        }
        return l - rt->l;
    }
    else {
        ret = 0;
        mid = (rt->l+rt->r) >> 1;
        if (s <= mid) ret += query(rt->pl,val);
        if (mid+1 <= t) ret += query(rt->pr,val);
        return ret;
    }
}
// 二分查找时遇到相同值的处理非常重要
int main() {
    int i,j,l,r;
    scanf("%d %d",&n,&m);
    for (i=0;i<n;i++) scanf("%d",num+i);
    mempos = 0;
    root = make_tree(0,n-1,0);
    while (m --) {
        s = get_val()-1; t = get_val()-1; rank = get_val()-1;
        l = 0, r = n-1;
        while (l <= r) {
```

```
            int mid = (l+r) >> 1;
            // 二分查找sortseq[0][mid]在区间[s,t]中的排名
            int pos = query(root,sortseq[0][mid]);
            if (rank < pos) r = mid-1;
            else l = mid+1;
        }
        printf("%d\n",sortseq[0][r]);
    }
}
```

23. LCA – RMQ-ST

```
//POJ 3417
//online O(nlogn)-O(1)
#include <cstdio>
#include <string>
#include <queue>
#include <algorithm>
using namespace std;
typedef __int64 bigint;
const int MAX = 100010;
const int STMAX = 200010;
const int LOGMAX = 18;

int n,m;
const int ENDFLAG = 0;
struct EDGELIST {
    int start[MAX];
    int last[MAX];
    int edge[MAX<<1][2];//pos,listnext
    int tot;

    void clear() {
        tot = ENDFLAG +1;
        memset(last,ENDFLAG,sizeof(last));
        memset(start,ENDFLAG,sizeof(start));
    }
    void push_back(int s,int t) {
        edge[tot][0] = t;
        edge[tot][1] = ENDFLAG;
        if (last[s] != ENDFLAG) {
            edge[ last[s] ][1] = tot;
        }
        else {
            start[s] = tot;
```

```
        }
        last[s] = tot;
        tot ++;
        //swap
        if (s == t) return;
        edge[tot][0] = s;
        edge[tot][1] = ENDFLAG;
        if (last[t] != ENDFLAG) {
            edge[ last[t] ][1] = tot;
        }
        else {
            start[t] = tot;
        }
        last[t] = tot;
        tot ++;
    }
}tree;
int cov[MAX];
bool vis[MAX];
bigint cnt[2];

int root[MAX],son[MAX];
int stn;
int st_min[LOGMAX][STMAX];
int order[STMAX],first[MAX],deep[STMAX];

void make_st() {
    int i,j,k;
    for (i=0;i<stn;i++) st_min[0][i] = i;
    for(j=1; 1<<j <= stn ;j++) {
        k = 1<<(j-1);
        for(i=0; i+k < stn ;i++) {
            if (deep[st_min[j-1][i]] < deep[st_min[j-1][i+k]])
                st_min[j][i] = st_min[j-1][i];
            else
                st_min[j][i] = st_min[j-1][i+k];
        }
    }
}

int rmq(int a,int b) {
    int dis = abs(b-a) +1;
    int k;
    for(k=0; (1<<k) <= dis ;k++) ;
```

```
    k --;
    int ret = st_min[k][a];
    if (deep[ret] > deep[st_min[k][b-(1<<k)+1]])
        ret = st_min[k][b-(1<<k)+1];
    return ret;
}

int lca(int a,int b) {
    int x = first[a],y = first[b];
    if (x > y) swap(x,y);
    return order[rmq(x,y)];
}

int ordcnt;
int sq[MAX];
int qf,qe;
void dfs(int pos,int d) {
    int i,j;
    vis[pos] = true;
    first[pos] = ordcnt;
    deep[ordcnt] = d;
    order[ordcnt ++] = pos;
    for (i=tree.start[pos]; i != ENDFLAG ;i=tree.edge[i][1]) {
        int next = tree.edge[i][0];
        if (vis[next]) continue;
        son[pos] ++;
        root[next] = pos;
        dfs(next,d+1);
        deep[ordcnt] = d;
        order[ordcnt ++] = pos;
    }
    if (son[pos] == 0) sq[qe ++] = pos;
}

void treedp() {
    son[0] = -1;
    while (qf < qe) {
        int now = sq[qf ++];
        if (cov[now] <= 1) cnt[ cov[now] ] ++;
        son[ root[now] ] --;
        cov[ root[now] ] += cov[now];
        if (son[ root[now] ] == 0) sq[qe ++] = root[now];
    }
}
```

```
int get_val() {
    int ret = 0;
    char ch;
    while ((ch=getchar()) > '9' || ch < '0') ;
    do {
        ret = ret*10 + ch - '0';
    } while ((ch=getchar()) <= '9' && ch >= '0') ;
    return ret;
}

int main() {
    int i,j,a,b,rt;
    n = get_val();
    m = get_val();
    if (n == 1) {
        puts("0");
        return 0;
    }

    tree.tot = ENDFLAG +1;
    qf = qe = 0;
    for (i=0;i<n-1;i++) {
        a = get_val();
        b = get_val();
        tree.push_back(a,b);
        rt = a;
    }

    ordcnt = 0;
    dfs(rt,0);
    stn = ordcnt;
    make_st();

    for (i=0;i<m;i++) {
        a = get_val();
        b = get_val();
        cov[a] ++;
        cov[b] ++;
        cov[lca(a,b)] -= 2;
    }

    treedp();
    cnt[0] --;
```

```
    printf("%I64d\n",cnt[0]*m + cnt[1]);
}
```

24. LCA – Tarjan

```
//POJ 3417
//offline O(na(n))
#include <cstdio>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
typedef __int64 bigint;
const int MAX = 100010;

int n,m;
const int ENDFLAG = 0;
struct EDGELIST {
    int start[MAX];
    int last[MAX];
    int edge[MAX<<1][2];//pos,listnext
    int tot;

    void clear() {
        tot = ENDFLAG +1;
        memset(last,ENDFLAG,sizeof(last));
        memset(start,ENDFLAG,sizeof(start));
    }
    void push_back(int s,int t) {
        edge[tot][0] = t;
        edge[tot][1] = ENDFLAG;
        if (last[s] != ENDFLAG) {
            edge[ last[s] ][1] = tot;
        }
        else {
            start[s] = tot;
        }
        last[s] = tot;
        tot ++;
        //swap
        if (s == t) return;
        edge[tot][0] = s;
        edge[tot][1] = ENDFLAG;
```

```
        if (last[t] != ENDFLAG) {
            edge[ last[t] ][1] = tot;
        }
        else {
            start[t] = tot;
        }
        last[t] = tot;
        tot ++;
    }
}tree,newed;
int cov[MAX];
bool vis[MAX];
bigint cnt[2];

int father[MAX];
int ancestor[MAX];

int find_set(int x) {
    if (father[x] == x) return x;
    return father[x] = find_set(father[x]);
}

void union_set(int x,int y) {
    father[ find_set(y) ] = x;
}

void tarjan(int pos,int pre) {
    int i,j;

    father[pos] = pos;
    ancestor[pos] = pos;

    for (i=tree.start[pos]; i != ENDFLAG ;i=tree.edge[i][1]) {
        int next = tree.edge[i][0];
        if (next == pre) continue;
        tarjan(next,pos);
        union_set(pos,next);
        cov[pos] += cov[next];
    }

    vis[pos] = true;
    for (j=newed.start[pos]; j != ENDFLAG ;j=newed.edge[j][1]) {
        int next = newed.edge[j][0];
        if (vis[next]) cov[ ancestor[ find_set(next) ] ] -= 2;
```

```
    }

    if (cov[pos] <= 1) cnt[ cov[pos] ] ++;
}

int get_val() {
    int ret = 0;
    char ch;
    while ((ch=getchar()) > '9' || ch < '0') ;
    do {
        ret = ret*10 + ch - '0';
    } while ((ch=getchar()) <= '9' && ch >= '0') ;
    return ret;
}

int main() {
    int i,j,a,b,rt;
    n = get_val();
    m = get_val();
    if (n == 1) {
        puts("0");
        return 0;
    }

    tree.tot = newed.tot = ENDFLAG +1;
    for (i=0;i<n-1;i++) {
        a = get_val();
        b = get_val();
        tree.push_back(a,b);
        rt = a;
    }

    for (i=0;i<m;i++) {
        a = get_val();
        b = get_val();
        newed.push_back(a,b);
        cov[a] ++;
        cov[b] ++;
    }

    tarjan(rt,rt);
    cnt[0] --;
    printf("%I64d\n",cnt[0]*m + cnt[1]);
}
```

25. 指数型母函数

```
/*
HDOJ 1521 排列组合
有n种物品，并且知道每种物品的数量。要求从中选出m件物品的排列数。<=m,n<=10
数量较少时，直接用除法
*/
#include <cstdio>
#include <string>
#define MAX 100
double cal[2][MAX];
double *pre,*now,*pt;
int n,m;
int a[11];
double fac[100];

int main()
{
    int i,j,k,sum;
    fac[0] = fac[1] = 1;
    for (i=2;i<=20;i++) {
        fac[i] = fac[i-1] * i;
    }
    while (scanf("%d %d",&n,&m)==2) {
        memset(cal,0,sizeof(cal));
        for (i=0;i<n;i++) {
            scanf("%d",&a[i]);
        }
        pre = cal[0];
        now = cal[1];
        pre[0] = 1;
        for (i=1;i<=a[0];i++) {
            pre[i] = 1.0 / fac[i];
        }
        for (i=1;i<n;i++) {
            for (j=0;j<MAX;j++) {
                if (pre[j] > 0) {
                    for (k=0;k<=a[i];k++) {
                        now[k+j] += pre[j] / fac[k];
                    }
                }
            }
```

```
            pt = now;
            now = pre;
            pre = pt;
            memset(now,0,sizeof(cal[0]));
            pre[0] = 1;
        }
        printf("%.0lf\n",fac[m] * pre[m]);
    }
}
```

26. 指数型母函数（大数据）

```
#include<iostream>
using namespace std;
int mm[1000][100];
__int64 a[1000], b[1000];

inline __int64 gcd(__int64 x, __int64 y) //求公约数
{
    __int64 temp;
    while (x % y) {
        temp = x % y;
        x = y;
        y = temp;
    }
    return y;
}

int main() {
    int n, m, i, j, k;
    __int64 tmp, tmp1;
    while (scanf("%d %d", &n, &m) != EOF) {
        for (i = 0; i < n; i ++) {
            scanf("%d", &mm[i][0]);
            for (j = 1; j <= mm[i][0]; j ++)
                scanf("%d", &mm[i][j]);
        }
        memset(a, 0, sizeof(__int64)*(m + 1));
        for (i = 1; i <= mm[0][0]; i ++)
            a[mm[0][i]] = 1;
        for (i = 1; i < n; i ++) {
            memset(b, 0, sizeof(__int64)*(m + 1));
            for (j = 0; j <= m; j ++)
                for (k = 1;j + mm[i][k] <= m && k <= mm[i][0]; k ++) {
                    if (mm[i][k] != 0) {
```

```
                    tmp = 1; tmp1 = 1;
                    int w = j + mm[i][k];
                    int x = mm[i][k] < j ? mm[i][k] : j; //x是较小的数
                    int y = w - x;

                    __int64 z;
                    while (w > y) {
                        tmp *= w;
                        tmp1 *= x;
                        z = gcd(tmp, tmp1);
                        if (z > 1) {
                            tmp /= z;
                            tmp1 /= z;
                        }
                        w--;
                        x--;
                    }

                    b[j + mm[i][k]] += tmp * a[j];
                }
            }
        for (j = 0; j <= m; j ++)
            a[j] += b[j];
    }
    printf("%I64d\n", a[m]);
  }
  return 0;
}
```

27. AC 自动机（字典树+KMP）

```
const int NMAX = 10000;
const int LMAX = 1000001;
const int MMAX = 51;
const int MEMMAX = 500000;

char s[LMAX];
char p[MMAX];
int n, m;

struct NODE
{
    int nsuffix;
    char chword;
    NODE * next, * father;
```

```
    NODE * son[26];
}mem[MEMMAX];
int total;
NODE * root;

NODE * new_node()
{
    NODE * ret = &mem[total ++];
    memset(ret, 0, sizeof(NODE));
    return ret;
}

// O(n MMAX)
void insert(NODE * rt, char * p)
{
    //puts(p);
    if (*p == 0)
    {
        rt->nsuffix ++;
        return;
    }
    if (rt->son[*p - 'a'] == NULL)
    {
        rt->son[*p - 'a'] = new_node();
        rt->son[*p - 'a']->father = rt;
        rt->son[*p - 'a']->chword = *p;
    }
    insert(rt->son[*p - 'a'], p+1);
}

// O(n MMAX)
void bfs()
{
    int i, j;
    queue <NODE *> sq;
    sq.push(root);

    while (!sq.empty())
    {
        NODE * now = sq.front();
        sq.pop();

        if (now->father == root)
            now->next = root;
```

```
        else
        {
            NODE * shift = (now->father)->next;
            while (shift != root && shift->son[now->chword -'a'] == NULL)
                shift = shift->next;
            now->next = shift->son[now->chword -'a'];
            if (now->next == NULL)
                now->next = root;
        }

        for (i=0; i<26; i++)
        {
            if (now->son[i] != NULL)
                sq.push(now->son[i]);
        }
    }
}

// O(LMAX)
int solve()
{
    int i,j;
    int ret = 0;
    NODE * now = root;
    NODE * psuffix;

    root->father = root;
    bfs();

    for (i=0; s[i]; i++)
    {
        while (now != root && now->son[ s[i] - 'a' ] == NULL)
            now = now->next;
        now = now->son[ s[i] - 'a' ];
        if (now == NULL)
            now = root;
        // add same suffix
        psuffix = now;
        while (psuffix != root && psuffix->nsuffix != -1)
        {
            ret += psuffix->nsuffix;
            psuffix->nsuffix = -1;
            psuffix = psuffix->next;
        }
```

```
    }
    return ret;
}
```

28. FFT（大数乘法）

```
const int BASE = 100000;
const int N_DIGIT = 5;
const int N = 32768;
const double PI = acos(-1.0);

struct Complex
{
    double real, imag;
};

Complex omega[N >> 1];
Complex a[N];
Complex b[N];

char s[1000003];
int  d[N], len;

void bitReverse(Complex a[])
{
    int i, j = 1, k;
    Complex t;
    for (i = 1; i < len; ++ i)
    {
        if (i < j)
        {
            t.real = a[j - 1].real;
            t.imag = a[j - 1].imag;
            a[j - 1].real = a[i - 1].real;
            a[j - 1].imag = a[i - 1].imag;
            a[i - 1].real = t.real;
            a[i - 1].imag = t.imag;
        }
        k = len >> 1;
        while (k < j)
        {
            j -= k;
            k >>= 1;
        }
        j += k;
```

```
    }
}

void calOmega()
{
    double unit = 2 * PI / len;
    int n = len >> 1;
    for (int i = 0; i < n; ++ i)
    {
        double t = unit * i;
        omega[i].real = cos( t );
        omega[i].imag = sin( t );
    }
}

void fft(Complex a[], bool inverse = false)
{
    bitReverse( a );

    int s = len >> 1;
    int m, k, j;
    int up, t, step;
    int i1, i2;
    Complex tmp;

    if ( inverse )
    {
        for (j = 0; j < s; ++ j)
            omega[j].imag = - omega[j].imag;
    }
    s = 1;
    for (m = 2; m <= len; m <<= 1)
    {
        up = m >> 1, t = len >> s;          // 2^(log2(n) - s) != n - 2^s   !!!!!!!
        for (k = 0; k < len; k += m)
        {
            step = 0;
            for (j = 0; j < up; ++ j)
            {
                i1 = k + j;
                i2 = i1 + up;
                tmp.real = omega[step].real * a[i2].real - omega[step].imag *
a[i2].imag;
                tmp.imag = omega[step].real * a[i2].imag + omega[step].imag *
```

```
a[i2].real;
                a[i2].real = a[i1].real - tmp.real;
                a[i2].imag = a[i1].imag - tmp.imag;
                a[i1].real += tmp.real;
                a[i1].imag += tmp.imag;
                step += t;
            }
        }
        ++ s;
    }
    if ( inverse )
    {
        double t = 1.0 / len;
        for (j = 0; j < len; ++ j)
            a[j].real *= t;
    }
}

int convert(int d[], char s[])
{
    int sLen = strlen( s );
    int dLen = ((sLen - 1) / N_DIGIT) + 1, i = 0, n;
    char *pRight = s + sLen - 1, *pLeft = pRight - (N_DIGIT - 1);
    memset(d, 0, sizeof(int) * dLen);

    while (i < dLen && pRight >= s)
    {
        if (pLeft < s)      pLeft = s;
        n = 0;
        while (pLeft <= pRight)
        {
            n = n * 10 + (*pLeft & 15);
            ++ pLeft;
        }
        d[i ++] = n;
        pRight -= N_DIGIT;
        pLeft = pRight - (N_DIGIT - 1);
    }
    return dLen;
}

bool init()
{
    int i, j;
```

```
    //read a
    if (scanf("%s", s) != 1)
        return false;
    int aLen = convert(d, s);          //length of a
    for (i = 0; i < aLen; ++ i)
    {
        a[i].real = d[i];
        a[i].imag = 0;
    }

    //read b
    scanf("%s", s);
    int bLen = convert(d, s);          //length of b
    for (j = 0; j < bLen; ++ j)
    {
        b[j].real = d[j];
        b[j].imag = 0;
    }

    len = 1;                           //length of product who uses int
    while (len < aLen + bLen)     len <<= 1;

    memset(a + i, 0, sizeof(Complex) * (len - i));
    memset(b + j, 0, sizeof(Complex) * (len - j));

    calOmega();
    return true;
}

void mul()
{
    for (int i = 0; i < len; ++ i)
    {
        double real = a[i].real * b[i].real - a[i].imag * b[i].imag;
        double imag = a[i].real * b[i].imag + a[i].imag * b[i].real;
        a[i].real = real;
        a[i].imag = imag;
    }
}

void print()
{
    double carry = 0, t;
```

```
    static char format[10];
    int i;

    for (i = 0; i < len; ++ i)
    {
        t = carry + a[i].real;
        carry = floor((t + 0.5) / BASE);
        d[i] = int(t - carry * BASE + 0.5);
    }
    for (i = len - 1; i > 0 && d[i] == 0; -- i);
    sprintf(format, "%%.%dd", N_DIGIT);
    printf("%d", d[i]);
    for (-- i; i >= 0; -- i)
        printf(format, d[i]);
    printf("\n");
}

int main()
{
    while ( init() )
    {
        fft( a );
        fft( b );
        mul();
        fft(a, true);
        print();
    }

    return 0;
}
```

29. 二分图网络最大流最小割

```
// PKU 2125 Destroying The Graph
// 二分图最小点权覆盖集，求割集
// 1.设置一个集合A，最开始A={s},按如下方法不断扩张A:
// 2.若存在一条边(u,v)，其流量小于容量，且u属于A,则v加入A
// 3.若存在(v,u)，其流量大于0，且u属于A,则v加入A
// 4.A计算完毕，设B=V-A，最小割集E={(u,v) | u∈A,v∈B}
// Character '+' means that Bob removes all arcs incoming into the specified vertex
and '-' that Bob removes all arcs outgoing from the specified vertex.
bool S[MAX];
void dfs(int pos) {
    int i;
    S[pos] = 1;
```

```
    for(i=1;i<=m;i++) {
        if(!S[i] && net[pos][i]) dfs(i);
    }
}

struct node {
    int num;
    char sign;
}cs[MAX];
void find_cut() {
    int i,ps = 0;
    memset(S, 0, sizeof(S));
    dfs(1);
    for(i=2;i<=n+1;i++) {
        if(!S[i] && net[1][i] == 0) {
            //printf("%d -\n", i-1);
            cs[ps].num = i-1;
            cs[ps].sign = '-';
            ps ++;
        }
    }
    for(i=n+2;i<=2*n+1;i++) {
        if(S[i] && net[i][m] == 0) {
            //printf("%d +\n", i-n-1);
            cs[ps].num = i-n-1;
            cs[ps].sign = '+';
            ps ++;
        }
    }
    printf("%d\n", ps);
    for(i=0;i<ps;i++) printf("%d %c\n", cs[i].num, cs[i].sign);
    //puts("------");
    //for(i=1;i<=m;i++) if(S[i]) printf("%d ",i);
    //puts("\n------");
}

int win[MAX], wout[MAX];
int main() {
    int i,j;
    while(scanf("%d %d", &n,&m)==2) {
        memset(net, 0, sizeof(net));
        for(i=2;i<=n+1;i++) scanf("%d", win+i);
        for(i=2;i<=n+1;i++) scanf("%d", wout+i);
        while(m --) {
```

```
            int x,y;
            scanf("%d %d", &x,&y);
            x ++; y += n+1;
            net[x][y] = INT_MAX;
        }
        for(i=2;i<=n+1;i++) {
            net[1][i] = wout[i];
        }
        for(i=n+2;i<=2*n+1;i++) {
            net[i][2*n+2] = win[i-n];
        }
        m = 2*n + 2;
        printf("%d\n", Edmonds_Karp());
        find_cut();
    }
}
```

30. 混合图欧拉回路

```
// 1637 PKU
bool solve()
{
    int i, j;
    for (i=2; i<n; i++)
    {
        x[i] = indeg[i] - outdeg[i];
        if (x[i] % 2)
            return false;
        if (x[i] > 0)
            net[i][m] += x[i] >> 1;
        else if (x[i] < 0)
            net[1][i] += (-x[i]) >> 1;
    }
    int cap = 0;
    for (i=2; i<n; i++)
        cap += net[1][i];
    int flow = Edmonds_Karp();
    // when flow==cap, say it exist euler circuit
    /*
    // print the undirected edge's direction
    for (i=2; i<m; i++)
        for (j=2; j<m; j++)
            if (net[i][j] != 0)
                printf("%d -> %d\n", i-1,j-1);
    */
```

```
    return (flow == cap);
}

int main()
{
    int i, j, cas;
    scanf("%d", &cas);
    while (cas --)
    {
        memset(indeg, 0, sizeof(indeg));
        memset(outdeg, 0, sizeof(outdeg));
        memset(net, 0, sizeof(net));
        scanf("%d %d", &m, &s);
        for (i=0; i<s; i++)
        {
            int x, y, d;
            scanf("%d %d %d", &x, &y, &d);
            x ++; y ++;
            // if d=0, make x->y
            indeg[y] ++;
            outdeg[x] ++;
            if (d == 0)
                net[x][y] ++;
        }
        n = m + 2;
        puts(solve() ? "possible" : "impossible");
    }
}
```

31. 无源汇上下界网络流

```
/*
*   2314 ZJU Reactor Cooling
*   无源汇上下界网络流
*   (1) 新建S，T
*   (2) D(u) = ΣB(i,u) - ΣB(u,i)
*       D(u) > 0，建弧(S,u)，权值为D(u)
*       D(u) < 0，建弧(u,T)，权值为-D(u)
*   (3) 求最大流，判定是否满流
*/
struct NODE
{
    int x, y;
    int b, c;
    NODE (int _x = 0, int _y = 0, int _b = 0, int _c = 0)
```

```
        : x(_x), y(_y), b(_b), c(_c) {}
};
vector <NODE> nodes;
int make_net()
{
    int i, j;
    int D[NMAX] = {0};
    memset(net, 0, sizeof(net));
    n += 2;
    vector <NODE>::iterator iter;
    foreach (iter,nodes)
    {
        i = iter->x;
        j = iter->y;
        net[i][j] = iter->c - iter->b;
        D[j] += iter->b;
        D[i] -= iter->b;
    }
    int ret = 0;
    for (i=2; i<n; i++)
    {
        if (D[i] > 0)
            net[1][i] = D[i];
        else if (D[i] < 0)
            net[i][n] = - D[i];
        ret += net[1][i];
    }
    return ret;
}

void solve()
{
    int i, j;
    int cap = make_net();
    int flow = Edmonds_Karp();
    if (flow != cap)
        puts("NO\n");
    else
    {
        puts("YES");
        vector <NODE>::iterator iter;
        foreach (iter,nodes)
            printf("%d\n", iter->c - net[iter->x][iter->y]);
    }
```

```
}

int main()
{
    int i, j, cas;
    scanf("%d", &cas);
    while (cas --)
    {
        scanf("%d %d", &n, &m);
        nodes.clear();
        for (i=0; i<m; i++)
        {
            int x, y, l, cap;
            scanf("%d %d %d %d", &x, &y, &l, &cap);
            x ++; y ++;
            nodes.push_back(NODE(x,y,l,cap));
        }
        solve();
    }
}
```

32. 二分图最小点权覆盖

```
// 3308 PKU Paratroopers
// 2874 ZJU
double R[MAX], C[MAX];
// 二分图最小点权覆盖-> 网络最大流
void make_net()
{
    int i, j;
    memset(net, 0, sizeof(net));
    // C(S,x) = W[x]
    for (i=0; i<n; i++)
        net[0][i+1] = log(R[i]);
    // C(y,T) = W[y]
    for (i=0; i<m; i++)
        net[n+i+1][n+m+1] = log(C[i]);
    // C(x,y) = inf
    for (i=0; i<l; i++)
    {
        int x, y;
        scanf("%d %d", &x, &y);
        net[x][y+n] = 1e99;
    }
    n = n + m + 1;
```

```
}

double solve()
{
    int i, j;
    double ret;
    make_net();
    ret = Edmonds_Karp();
    return exp(ret);
}
```

33. 带约束的轨道计数(Burnside 引理)

```
// PKU 2888
#include <stdio.h>
#include <math.h>

const int MOD = 9973;
bool A[32000];
int prim[3500],T[10][10];
int total,n,m;

void init()
{
    int i,j;
    total = 0;
    for(i = 2;i < 32000;i++)
        if(!A[i])
        {
            prim[total++] = i;
            for(j = 2*i;j < 32000;j += i)
                A[j] = true;
        }
}

int phi(int x)
{
    int temp,i,num;
    if(x == 1) return 1;
    temp = 1;
    for(i = 0;i < total;i++)
    {
        num = prim[i];
        if(x % num == 0)
        {
```

```
            temp *= num-1;
            temp %= MOD;
            x /= num;
            while(x % num == 0)
            {
                x /= num;
                temp *= num;
                temp %= MOD;
            }
            if(x == 1) break;
        }
    }
    if(x != 1)
    {
        temp *= x - 1;
        temp %= MOD;
    }
    return temp;
}

void GT(int TT[][10],int p)
{
    int i,j,sum,k;
    if(p == 1)
    {
        for(i = 0;i < m;i++)
            for(j = 0;j < m;j++)
                TT[i][j] = T[i][j];
        return;
    }
    int t2[10][10];
    GT(t2,p/2);
    for(i = 0;i < m;i++)
        for(j = 0;j < m;j++)
        {
            sum = 0;
            for(k = 0;k < m;k++)
                sum += t2[i][k]*t2[k][j];
            TT[i][j] = sum % MOD;
        }
    if(p % 2 == 0) return;
    int t[10][10];
    for(i = 0;i < m;i++)
        for(j = 0;j < m;j++)
```

```
            {
                sum = 0;
                for(k = 0;k < m;k++)
                    sum += TT[i][k]*T[k][j];
                t[i][j] = sum % MOD;
            }
        for(i = 0;i < m;i++)
            for(j = 0;j < m;j++)
                TT[i][j] = t[i][j];
}

int Tr(int p)
{
    int sum = 0,i;
    int TT[10][10];
    GT(TT,p);
    for(i = 0;i < m;i++)
        sum += TT[i][i];
    return sum % MOD;
}

int gn()
{
    int temp,sum = 0,i,all;
    temp = (int)(sqrt(1.0*n)+0.4) + 1;
    sum = (phi(1)*Tr(n)%MOD + phi(n)*Tr(1)%MOD) % MOD;
    for(i = 2;i < temp;i++)
    {
        if(n % i == 0)
        {
            if(i*i == n)
            {
                sum += phi(i)*Tr(i)%MOD;
                sum %= MOD;
            }
            else
            {
                sum += phi(i)*Tr(n/i)%MOD + phi(n/i)*Tr(i)%MOD;
                sum %= MOD;
            }
        }
    }
    all = sum / n;
    sum %= n;
```

```
    while(sum)
    {
        all++;
        sum = n - sum;
        sum %= MOD;
        if(sum == 0) break;
        sum = MOD - sum;
    }
    return all;
}

int main()
{
    int cas,k,x,y,i,j;
    scanf("%d",&cas);
    init();
    while(cas--)
    {
        scanf("%d %d %d",&n,&m,&k);
        for(i = 0;i < m;i++)
            for(j = 0;j < m;j++)
                T[i][j] = 1;
        while(k--)
        {
            scanf("%d %d",&x,&y);
            x--;
            y--;
            T[x][y] = 0;
            T[y][x] = 0;
        }
        printf("%d\n",gn());
    }
    return 0;
}
```

34. 三分法求函数波峰

```
// linle专场考研路茫茫——早起看书
const int MMAX = 11000;
const double EPS = 1e-4;
int x[MMAX], y[MMAX];
int n, m;

#define f(dt) k*(dt-x[p-1]) + y[p-1] + 1.0*n/dt/dt;
```

```
double triple_search(int p)
{
    double mmin = 1e99;
    double k = 1.0 * (y[p]-y[p-1]) / (x[p]-x[p-1]);
    double xl = x[p-1], xr = x[p];
    double lm, rm, flm, frm;

    while (fabs(xr-xl) > EPS)
    {
        lm = (2.0*xl + xr) / 3.0;
        rm = (2.0*xr + xl) / 3.0;
        flm = f(lm);
        frm = f(rm);
        if (frm > flm)
            xr = rm, mmin = min(mmin, flm);
        else
            xl = lm, mmin = min(mmin, frm);
    }
    return mmin;
}

double solve()
{
    int i, j, k;
    double mmin = 1e99;
    for (i=1; i<m; i++)
    {
        double ret = triple_search(i);
        mmin = min(mmin, ret);
    }
    return mmin;
}

int main()
{
    int i, j;
    while (scanf("%d %d", &m, &n) == 2)
    {
        for (i=0; i<m; i++)
            scanf("%d %d", &x[i], &y[i]);
        printf("%.3lf\n", solve());
    }
}
```

35. 单词计数，DFA 自动机，Trie 图

```
// linle专场考研路茫茫——单词情结
// 由正则表达式构造NFA，NFA转DFA，最小化DFA
// 构造状态转移矩阵，矩阵乘法
typedef unsigned long long ULL;
#define foreach(it,c) for (it=(c).begin(); it!=(c).end(); it++)
#define forsize(it,c) for (it=0; it<(c).size(); it++)

const int NMAX = 6;
int n, l;
char rt[NMAX][6];

const int SMAX = 80;

#define ADD(a,x) ((a)=((a)+(x)))

struct MATRIX
{
    ULL mat[SMAX][SMAX];
    int n;

    MATRIX (int _n = SMAX)
    {
        n = _n;
        memset(mat, 0, sizeof(mat));
    }
    void to_E(int nn)
    {
        int i;
        n = nn;
        memset(mat, 0, sizeof(mat));
        for (i=0; i<n; i++)
            mat[i][i] = 1;
    }
    void fill(const MATRIX & mt, int x, int y)
    {
        int i, j;
        for (i=0; i<mt.n; i++)
            for (j=0; j<mt.n; j++)
                mat[i+x][j+y] = mt.mat[i][j];
    }
    MATRIX operator * (const MATRIX & mt)
    {
        MATRIX ret;
        int i, j, k;
```

```
        for (i=0; i<n; i++)
            for (j=0; j<n; j++)
            {
                ret.mat[i][j] = 0;
                for (k=0; k<n; k++)
                    ADD(ret.mat[i][j], mat[i][k] * mt.mat[k][j]);
            }
        ret.n = n;
        return ret;
    }
    MATRIX operator ^ (int ex)
    {
        int i;
        MATRIX ret, tmp;
        ret = *this;
        tmp.to_E(this->n);
        while (ex > 1)
        {
            if (ex & 1)
                tmp = tmp * ret;
            ret = ret * ret;
            ex >>= 1;
        }
        ret = ret * tmp;
        return ret;
    }
};

const int NFAMAX = 60;
struct EDGE
{
    char ch;
    int next;
    EDGE (char _c = 0, int _n = 0)
        : ch(_c), next(_n) {}
};
vector <EDGE> nfa[NFAMAX];
vector <EDGE> dfa[NFAMAX];
vector <EDGE> mindfa[NFAMAX];
int nfact;
int dfasn;
int mindfasn;
vector <int> dfact;
vector <int> mindfact;
```

```
#define BADD(x,p) ((x) |= ((ULL)1<<(p)))
#define BSUB(x,p) ((x) &= ~((ULL)1<<(p)))
#define BGET(x,p) ((x) & ((ULL)1<<(p)))

void make_nfa()
{
    int i, j, k;
    for (i=0; i<NFAMAX; i++)
        nfa[i].clear();

    for (i='a'; i<='z'; i++)
        nfa[0].push_back(EDGE(i,0));

    nfact = 1;
    int lend[NMAX];
    for (i=0; i<n; i++)
    {
        nfa[0].push_back(EDGE('$',nfact++));
        for (j=0; rt[i][j]; j++)
        {
            nfa[nfact-1].push_back(EDGE(rt[i][j],nfact));
            nfact ++;
        }
        lend[i] = nfact - 1;
    }

    for (i=0; i<n; i++)
        nfa[ lend[i] ].push_back(EDGE('$',nfact));
    for (i='a'; i<='z'; i++)
        nfa[nfact].push_back(EDGE(i,nfact));
    nfact ++;
}

bitset <NFAMAX> vis;
ULL e_closure(int now)
{
    int i, j;
    ULL ret = 0;
    vector <EDGE>::iterator iter;

    BADD(ret, now);
    if (vis[now])
        return ret;
```

```
    vis[now] = true;

    foreach (iter, nfa[now])
        if (iter->ch == '$')
            ret |= e_closure(iter->next);
    return ret;
}

ULL e_closure2(ULL now)
{
    int i, j;
    ULL ret = now;

    vis.reset();
    for (i=0; i<nfact; i++)
        if (BGET(now, i))
            ret |= e_closure(i);
    return ret;
}

map < ULL, int > hash;

void dfs(ULL now)
{
    int i, j;
    vector <EDGE>::iterator iter;
    vector <int>::iterator iter2;
    vector <int> nxt[30];

    for (i=0; i<nfact; i++)
    {
        if (BGET(now,i))
        {
            foreach (iter, nfa[i])
            {
                if (iter->ch == '$')
                    continue;
                nxt[iter->ch - 'a'].push_back(iter->next);
            }
        }
    }

    int stag = hash[now];
    for (i='a'; i<='z'; i++)
```

```
    {
        if (nxt[i-'a'].empty())
            continue;
        ULL next = 0;
        foreach (iter2, nxt[i-'a'])
            BADD(next, *iter2);
        next = e_closure2(next);
        bool flag = false;
        int ntag = hash[next];
        if (ntag == 0)
            ntag = hash[next] = dfasn ++, flag = true;
        dfa[stag-1].push_back(EDGE(i,ntag-1));
        if (flag)
        {
            if (BGET(next, nfact-1))
                dfact.push_back(ntag-1);
            dfs(next);
        }
    }
}

void nfa_dfa()
{
    int i, j, k;

    dfasn = 1;
    vis.reset();
    hash.clear();
    dfact.clear();
    for (i=0; i<NFAMAX; i++)
        dfa[i].clear();

    ULL bs = e_closure(0);
    hash[bs] = dfasn ++;
    dfs(bs);
}

void min_dfa()
{
    int i, j, k;
    vector < vector <int> > split;
    vector <EDGE>::iterator iter;
    int belg[NFAMAX];
```

```
for (i=0; i<dfasn; i++)
{
    vector <int> newi;
    newi.push_back(i);
    split.push_back(newi);
    belg[i] = i;
}

bool flag = true;
while (flag)
{
    flag = false;
    for (i=0; i<split.size(); i++)
    {
        for (j=i+1; j<split.size(); j++)
        {
            vector < pair <char, int> > ibel, jbel;
            for (k=0; k<split[i].size(); k++)
                foreach (iter, dfa[ split[i][k] ])
                    ibel.push_back(make_pair(iter->ch, belg[iter->next]));
            for (k=0; k<split[j].size(); k++)
                foreach (iter, dfa[ split[j][k] ])
                    jbel.push_back(make_pair(iter->ch, belg[iter->next]));
            sort(ibel.begin(), ibel.end());
            sort(jbel.begin(), jbel.end());
            if (ibel == jbel)
            {
                flag = true;
                break;
            }
        }
        if (flag)
            break;
    }
    if (flag)
    {
        int s1 = belg[ split[i][0] ], s2 = belg[ split[j][0] ];
        for (k=0; k<dfasn; k++)
            if (belg[k] == s2)
                belg[k] = s1;
        split[i].insert(split[i].end(), split[j].begin(), split[j].end());
        split.erase(split.begin() + j);
    }
}
```

```
    for (i=0; i<split.size(); i++)
        for (j=0; j<split[i].size(); j++)
            belg[ split[i][j] ] = i;

    bitset <NFAMAX> acts;
    for (i=0; i<dfact.size(); i++)
        acts[ dfact[i] ] = true;
    mindfact.clear();

    mindfasn = split.size();
    for (i=0; i<mindfasn; i++)
    {
        int go[30];
        memset(go, -1, sizeof(go));
        mindfa[i].clear();
        flag = false;

        for (j=0; j<split[i].size(); j++)
        {
            if (acts[ split[i][j] ])
                flag = true;
            foreach (iter, dfa[ split[i][j] ])
                go[iter->ch - 'a'] = belg[iter->next];
        }
        for (j='a'; j<='z'; j++)
            if (go[j-'a'] != -1)
                mindfa[i].push_back(EDGE(j,go[j-'a']));
        if (flag)
            mindfact.push_back(i);
    }
}

MATRIX T;
MATRIX TT;
MATRIX BT;
MATRIX E;

void make_matrix()
{
    int i, j;
    vector <EDGE>::iterator iter;

    E.to_E(mindfasn);
```

```
    T.n = mindfasn;
    memset(T.mat, 0, sizeof(T.mat));
    for (i=0; i<mindfasn; i++)
    {
        foreach (iter, mindfa[i])
            T.mat[i][iter->next] ++;
    }
// 构造等比矩阵
    BT.n = mindfasn<<1;
    memset(BT.mat, 0, sizeof(BT.mat));
    BT.fill(T, 0, 0);
    BT.fill(E, 0, mindfasn);
    BT.fill(E, mindfasn, mindfasn);
}

ULL solve()
{
    int i, j;
    ULL ret = 0;
    vector <EDGE>::iterator iter;

    make_nfa();
    nfa_dfa();
    dfasn = hash.size();
    min_dfa();
    make_matrix();

    BT = BT ^ l;
    TT.n = mindfasn;
    for (i=0; i<mindfasn; i++)
        for (j=0; j<mindfasn; j++)
            TT.mat[i][j] = BT.mat[i][j+mindfasn];
    T = T * TT;
    for (i=0; i<mindfact.size(); i++)
        ret += T.mat[0][ mindfact[i] ];
    return ret;
}

int main()
{
    int i, j;
    while (scanf("%d %d", &n, &l) == 2)
    {
        for (i=0; i<n; i++)
```

```
            scanf("%s", rt[i]);
        printf("%I64u\n", solve());
    }
}
// HDU 2471 History of Languages
// DFA同构判断
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

const int NMAX = 2010;
const int TMAX = 26;
const int FAIL = -1;

int sigma; // 字符集大小
bool dis[NMAX][NMAX];
int lx[NMAX*NMAX], ly[NMAX*NMAX];

struct DFA {
    int X[NMAX][TMAX];
    bool F[NMAX];
    int n;

    void read() {
        int i, j, k;
        scanf("%d", &n);
        for (i = 0; i < n; i++) {
            scanf("%d", &k);
            F[i] = (k == 1);
            for (j = 0; j < sigma; j++) {
                scanf("%d", &X[i][j]);
                if (X[i][j] == -1) X[i][j] = n;
            }
        }
        // 虚拟一个非接受态节点，再补边
        F[n] = false;
        for (i = 0; i < sigma; i++)
            X[n][i] = n;
        //n++;
    }

    void Minization(DFA &dst) {
        memset(dst.X, -1, sizeof(dst.X));
```

```
int i, j, ch;
int p, q;

bool vis[NMAX] = {false};
vis[0] = true;
fill(vis, vis + n, true);
fill(dis[0], dis[n], false);

int Q[NMAX], Qf, Qr;
Qf = Qr = 0;
Q[Qr++] = 0;
while (Qf < Qr) {
    p = Q[Qf++];
    for (ch = 0; ch < sigma; ch++) {
        q = X[p][ch];
        if (q != -1 && !vis[q]) {
            vis[q] = true;
            Q[Qr++] = q;
        }
    }
}

for (i = 0; i < n; i++) {
    if (!vis[i]) continue;
    for (j = i + 1; j < n; j++) {
        if (!vis[j]) continue;
        if (F[i] != F[j]) dis[i][j] = dis[j][i] = true;
    }
}

while (1) {
    bool update = false;
    for (i = 0; i < n; i++) {
        if (!vis[i]) continue;
        for (j = i + 1; j < n; j++) {
            if (!vis[j]) continue;
            if (dis[i][j] == true) continue;

            for (ch = 0; ch < sigma; ch++) {
                p = X[i][ch];
                q = X[j][ch];

                if (p == -1 && q == -1) continue;
                if (p == -1 || q == -1 || dis[p][q])  {
```

```
                        dis[i][j] = dis[j][i] = true;
                        update = true;
                        break;
                    }
                }//update
            }//for j
        }//for i
        if (!update) break;
    }//while

    int id[NMAX], cnt = 0;
    fill(id, id + n, -1);
    for (i = 0; i < n; i++) {
        if (!vis[i]) continue;
        if (id[i] != -1) continue;

        for (j = 0; j < n; j++) {
            if (!vis[j]) continue;
            if (dis[i][j] == false) id[j] = cnt;
        }
        cnt++;
    }
    dst.n = cnt;

    for (i = 0; i < n; i++) {
        if (!vis[i]) continue;
        if (id[i] != -1) continue;

        p = id[i];
        dst.F[p] = F[i];
        for (ch = 0; ch < sigma; ch++) {
            q = X[i][ch];
            if (q != FAIL) q = id[q];
            dst.X[p][ch] = q;
        }
    }
}

void show() {
    int i, j;
    for (i = 0; i < n; i++) {
        cout << F[i] << " ";
        for (j = 0; j < sigma; j++)
            cout << X[i][j] << " ";
```

```
            cout << endl;
        }
        cout << endl;
    }

    bool equals(DFA &dfa2) {
        memset(dis, 0 ,sizeof(dis));
        int p, q;

        p = q = 0;
        lx[q] = 0;
        ly[q] = 0;

        dis[0][0] = true;
        q ++;
        while (p < q) {
            if (F[ lx[p] ] != dfa2.F[ ly[p] ]) break;

            for (int k = 0; k < sigma;k++)
                if (! dis[ X[lx[p]][k] ][ dfa2.X[ly[p]][k] ]) {
                    dis[ X[lx[p]][k] ][ dfa2.X[ly[p]][k] ] = true;
                    lx[q] = X[ lx[p] ][k];
                    ly[q] = dfa2.X[ ly[p] ][k];
                    q ++;
                }
            p ++;
        }
        return p == q;
    }
};

DFA a, b;
int main() {
    int cas = 1;
    while (scanf("%d", &sigma), sigma) {
        a.read();
        b.read();
        printf("Case #%d: ", cas++);
        if (a.equals(b)) puts("Yes");
        else puts("No");
    }
    return 0;
}
/*
```

```
Trie图
在Tire树的基础上补边（类似AC自动机）
可用于多字符串匹配和自动机的构造
图可转换为矩阵，或拓扑排序，用作统计或动态规划之用
*/
typedef pair <int, int> PII;
typedef vector <PII> TRANS;
const int VMAX = 200;
const int SIGMAX = 50;
char d2c[110] = "ACGT"; // 字符集
struct TrieGraph
{
    struct NODE
    {
        int suffix; // 后缀节点指针
        int father; // 父节点指针
        int next[SIGMAX]; // 儿子节点指针
        bool mark; // 标记是否出现过
        char ch; // 入边信息
    };
    NODE mem[VMAX];
    int vn, root;
    char c2d[300]; // 字符集hash表
    int siglen; // 字符集大小

    int new_node(char ch, int fat) {
        memset(mem+vn, 0, sizeof(NODE));
        mem[vn].ch = ch;
        mem[vn].father = fat;
        return vn ++;
    }
    // 初始化(字符集)
    void init(char * pstr) {
        vn = 0;
        siglen = strlen(pstr);
        for (int i=0; pstr[i]; i++)
            c2d[pstr[i]] = i;
        // 初始化节点，用作安全转移
        root = new_node('$', 0);
        for (int i=0; i<siglen; i++)
            mem[root].next[i] = new_node(pstr[i], root);
    }
    // 构造Trie
    void insert(char * pstr) {
```

```
        int i;
        for (i=root; *pstr; pstr++) {
            int x = c2d[*pstr];
            if (mem[i].next[x] == 0)
                mem[i].next[x] = new_node(*pstr, i);
            i = mem[i].next[x];
        }
        mem[i].mark = true;
    }
    int get_suffix(int idx) {
        int fat = mem[idx].father;
        if (fat == root) return root;
        int ich = c2d[mem[idx].ch];
        for (fat=mem[fat].suffix; fat!=root && mem[fat].next[ich]==0;
fat=mem[fat].suffix) ;
        if (mem[fat].next[ich] == 0) return root;
        return mem[fat].next[ich];
    }
    // 构造Trie Graph
    void construct() {
        queue <int> sq;
        for (int i=0; i<siglen; i++) {
            if (mem[root].next[i] == 0) continue;
            NODE & son = mem[ mem[root].next[i] ];
            son.suffix = get_suffix(mem[root].next[i]);
            sq.push(mem[root].next[i]);
        }
        while (! sq.empty()) {
            int idx = sq.front(); sq.pop();
            NODE & now = mem[idx];
            for (int i=0; i<siglen; i++) {
                int sonidx = now.next[i];
                NODE & son = mem[sonidx];
                if (sonidx == 0) continue;
                sq.push(sonidx);
                son.suffix = get_suffix(sonidx);
                son.mark = son.mark || mem[son.suffix].mark;
            }
            for (int i=0; i<siglen; i++) {
                if (now.next[i] != 0) continue;
                now.next[i] = mem[now.suffix].next[i];
            }
        }
    }
```

```
    // 构造安全图
    TRANS make_safe_graph() {
        queue <int> sq;
        bitset <VMAX> vis;
        TRANS ret;
        sq.push(root);
        vis[root] = true;
        while (! sq.empty()) {
            int idx = sq.front(); sq.pop();
            NODE & now = mem[idx];
            for (int i=0; i<siglen; i++) {
                int sonidx = now.next[i];
                if (sonidx==0 || mem[sonidx].mark) {
                    now.next[i] = 0; // 更新Trie Graph
                    continue;
                }
                ret.push_back(PII(idx, sonidx));
                if (vis[sonidx]) continue;
                sq.push(sonidx);
                vis[sonidx] = true;
            }
        }
        return ret;
    }
    void print() {
        printf("%6s%6s%6s%6s...\n", "Node", "Suff", "Mark", "Son");
        for (int i=0; i<vn; i++) {
            printf("%6d%6d%6d", i, mem[i].suffix, mem[i].mark);
            for (int j=0; j<siglen; j++)
                printf("%6d", mem[i].next[j]);
            puts("");
        }
    }
};
/*
PKU 1625 Censored!
求长度为m，字符集为n且不含p个不良单词的字符串的数目，
就是求在安全图中从根结点出发走m步有多少种走法。
用count[step,x]表示从根结点出发走step步到结点x的走法数。
fillchar(count,sizeof(count),0);
count[0,根]:=1;
for step:=1 to m do
    for 安全图中每条边(i,j) do
        inc(count[step,j],count[step-1,i]);
```

```
ans:=0;
for 安全图中每个结点x do
    inc(ans,count[m,x]);
*/
/*
PKU 2778 DNA Sequence
用矩阵做状态转移，矩阵二分求答案
*/
int n, m, p;
xnum dp[2][VMAX];
TrieGraph tg;

void solve()
{
    int i, j;
    xnum ret = 0;

    TRANS tr = tg.make_safe_graph();
    for (i=0; i<tg.vn; i++) dp[0][i] = dp[1][i] = 0;
    dp[0][0] = 1;
    for (i=1; i<=m; i++)
    {
        for (j=0; j<tr.size(); j++)
            dp[i&1][tr[j].second] = dp[i&1][tr[j].second] +
dp[(i&1)^1][tr[j].first];
        for (j=0; j<tg.vn; j++) dp[(i&1)^1][j] = 0;
    }

    for (i=0; i<tg.vn; i++)
        ret = ret + dp[m&1][i];
    ret.print();puts("");
}

int main()
{
    int i, j;
    char str[110];
    while (scanf("%d %d %d", &n, &m, &p) == 3)
    {
        scanf("%s", d2c);
        tg.init(d2c);
        for (i=0; i<p; i++)
        {
            scanf("%s", str);
```

```
            tg.insert(str);
        }
        tg.construct();
        //tg.print();
        solve();
    }
}
```

36. 字符串和数值 hash

```
// 整数hash
// 104729, 224737, 350377, 479909, 611953, 882377
// 1020379, 1299709, 1583539, 1870667, 2015177
// 4256233,5800079,7368787, 10570841, 15485863
const int MOD = 20023;
bool bhash[MOD];
int vhash[MOD];
int cnt[MOD];

bool find_hash(int & pos) {
    int val = pos;
    pos %= MOD;
    for (; bhash[pos]; pos=(pos+1)%MOD) {
        if (vhash[pos] == val)
            return true;
    }
    return false;
}

int make_hash(int val) {
    int pos = val;
    if (! find_hash(pos)) {
        bhash[pos] = true;
        vhash[pos] = val;
        cnt[pos] = 0;
    }
    cnt[pos] ++;
    return pos;
}

//字符串hash
const int MOD = 20023;
bool bhash[MOD];
char vhash[MOD][45];
char str[45];
```

```
int cal_str() {
    int i, j, pos;
    for (i=pos=0,j=1; str[i]; i++,j=(j*27)&INT_MAX,pos&=INT_MAX) {
        int num = str[i] - 'a';
        if (str[i] == ' ')
            num = 26;
        pos += j*num;
    }
    return pos % MOD;
}

bool find_hash(int & pos) {
    pos = cal_str();
    for (; bhash[pos]; pos=(pos+1)%MOD) {
        if (strcmp(vhash[pos], str) == 0)
            return true;
    }
    return false;
}

int make_hash() {
    int pos;
    if (! find_hash(pos)) {
        bhash[pos] = true;
        strcpy(vhash[pos], str);
    }
    return pos;
}
```

37. 滚动队列，前向星表示法

```
int que[2][2000];
int qf[2],qe[2],qnow;

#define push_que(a) (que[qnow][ qe[qnow]++ ] = (a))
#define pop_que2    (que[qnow^1][ qf[qnow^1]++ ])
#define switch_que  qnow ^= 1; \
                    qf[qnow] = qe[qnow] = 0;
#define empty_que2      (qf[qnow^1] >= qe[qnow^1])
#define size_que2   (qe[qnow^1] - qf[qnow^1])
/*
前向星表示法
空间O(E+N)
存储所有边，并用链表来实现读取s为起点的有向边
```

```
方便插入和遍历所有边,删除是O(E)
*/
const int ENDFLAG = -1;
struct EDGELIST {
    int start[NMAX];
    int last[NMAX];
    int edge[MMAX][2];//pos,listnext
    int tot;

    void clear() {
        tot = ENDFLAG +1;
        memset(last,ENDFLAG,sizeof(last));
    }
    void push_back(int s,int t) {
        edge[tot][0] = t;
        edge[tot][1] = ENDFLAG;
        if (last[s] != ENDFLAG) edge[ last[s] ][1] = tot;
        else start[s] = tot;
        last[s] = tot;
        tot ++;
    }
    int get_start(int s) {
        return start[s];
    }
    int get_next(int & p) {
        p = edge[p][1];
        return edge[p][0];
    }
    void erase(int s, int t) {
        int i,pre = ENDFLAG;
        int p,v;
        for (p=start[s]; p!=ENDFLAG; p=edge[p][1]) {
            v = edge[p][0];
            if (v == t) {
                if (pre == ENDFLAG) start[s] = edge[p][1];
                else edge[pre][1] = edge[p][1];
            }
            else pre = p;
        }
        last[s] = pre;
    }
};
```

38. 最小点基，最小权点基

```
// HDOJ 1827 Summer Holiday
// 点基：通过点基的点，能够到达有向图全部点
// 最小权点基：有向图顶点有权值
void Gabow()
{
    int i,j,l;
    //dfs in original graph
    memset(id, 0, sizeof(id));
    memset(vis, 0, sizeof(vis));
    scc = step = 1;
    order_pos = order2_pos = 0;
    for (i=1; i<=n ;i++) {
        if (vis[i] == 0) {
            dfs(i);
        }
    }
    scc --;
}

void top_sort()
{
    int i,j,k,l,m = 0;
    memset(out_degre,0,sizeof(out_degre));
    memset(sel, 0x7f,sizeof(sel));
    l = SQ.size();
    while (l --) {
        SQ.pop();
    }
    for (i=1;i<=n;i++) {
        int id1 = id[i];
        l = path[i].size();
        for (j=0;j<l;j++) {
            int id2 = id[ path[i][j] ];
            if (id1 != id2) {
                out_degre[id2] ++;
                dag[id1].push_back(id2);
            }
        }
    }
    for (i=1;i<=scc;i++) {
        if (out_degre[i] == 0) {
            SQ.push(i);
        }
    }
```

```
    while (!SQ.empty()) {
        int now = SQ.front();
        SQ.pop();
        l = dag[now].size();
        for (i=0;i<l;i++) {
            int next = dag[now][i];
            out_degre[next] --;
            if (out_degre[next] == 0) {
                SQ.push(next);
                m --;//find non-hightest scc
                sel[next] = -1;
            }
        }
    }
    for (i=1;i<=n;i++) {
        if (sel[ id[i] ] != -1) {//selection mininum cost in the highest scc
            sel[ id[i] ] = sel[ id[i] ] > cost[i] ? cost[i] : sel[ id[i] ];
        }
    }
    min_cost = 0;
    for (i=1;i<=scc;i++) {
        if (sel[i] != -1) {
            min_cost += sel[i];
        }
    }
    min_num = scc+m;
}

int main()
{
    int i,x,y;
    path.resize(NMAX);
    dag.resize(NMAX);
    while (scanf("%d %d",&n, &m)==2) {
        for (i=0;i<=n;i++) {
            path[i].clear();
            dag[i].clear();
        }
        for (i=1;i<=n;i++) {
            scanf("%d", cost+i);
        }
        for (i=1;i<=m;i++) {
            scanf("%d %d", &x,&y);
            path[x].push_back(y);
```

```
        }
        Gabow();
        top_sort();
        //min_num : mininum vertex number
        //min_cost : mininum cost
        printf("%d %d\n", min_num, min_cost);
    }
}
```

39. LCSubsequence O(N^2/logN)

```
// 1210 WHU
/*
*   LCSubsequence O(N^2/logN)
*   这个解法是在字符集不大的情况下，先预处理，再用位运算做状态转移。
*/
typedef UL data_type; // 变量存储类型
const int NMAX = 31000; // 字符串长度
const int BITLEN = sizeof(data_type)*8; // 变量存储位长度
const int BINLEN = 5; // 2^BINLEN = BITLEN
const int MMAX = (NMAX/BITLEN) +1; // 申请空间长度

// ((x)/BITLEN)
#define GETBLOCK(x) ((x)>>BINLEN)

char str1[NMAX];
char str2[NMAX];

struct BITSET
{
    data_type dat[MMAX];
    int len;
    int bs;

    BITSET (int l = 0) {
        len = l;
        bs = GETBLOCK(l+BITLEN-1);
        memset(dat, 0, sizeof(dat));
    }
    bool operator [] (int p) {
        return (dat[GETBLOCK(p)] & ((data_type)1<<(p%BITLEN)));
    }
    void set(int p, bool flag) {
        if (! flag)
            dat[GETBLOCK(p)] &= ~((data_type)1<<(p%BITLEN));
```

```
        else
            dat[GETBLOCK(p)] |= ((data_type)1<<(p%BITLEN));
    }
    void reset(int l) {
        len = l;
        bs = GETBLOCK(l+BITLEN-1);
        memset(dat, 0, sizeof(dat));
    }
    BITSET operator ~ () {
        BITSET ret = *this;
        int i;
        for (i=0; i<bs; i++)
            ret.dat[i] = ~ret.dat[i];
        return ret;
    }
    BITSET operator & (const BITSET & a) {
        BITSET ret = *this;
        int i;
        for (i=0; i<bs; i++)
            ret.dat[i] &= a.dat[i];
        return ret;
    }
    BITSET operator | (const BITSET & a) {
        BITSET ret = *this;
        int i;
        for (i=0; i<bs; i++)
            ret.dat[i] |= a.dat[i];
        return ret;
    }
    BITSET operator ^ (const BITSET & a) {
        BITSET ret = *this;
        int i;
        for (i=0; i<bs; i++)
            ret.dat[i] ^= a.dat[i];
        return ret;
    }
    BITSET & operator <<= (int l) {
        int i, j;
        int ll = l % BITLEN;
        l /= BITLEN;
        for (i=bs-l; l && i>=0; i--)
            dat[i] = dat[i-l];
        for (i=bs-l; ll && i>0; i--)
            dat[i] = (dat[i]<<ll) | (dat[i-1]>>(BITLEN-ll));
```

```
        dat[0] <<= ll;
        return *this;
    }
    BITSET operator - (const BITSET & a) {
        BITSET ret = *this;
        int i, borw = 0, tborw;
        for (i=0; i<bs; i++) {
            if (ret.dat[i] < a.dat[i] + borw)
                tborw = 1;
            else
                tborw = 0;
            ret.dat[i] -= a.dat[i] + borw;
            borw = tborw;
        }
        return ret;
    }
    int count(int l = 0) {
        int i, j, ret = 0;
        l = (l==0) ? len : l;
        for (i=0; i<bs && l>0; i++,l-=BITLEN) {
            data_type tmp = dat[i];
            int tl = l;
            for (; tmp && tl; tmp>>=1,tl--)
                ret += (tmp & 1);
        }
        return ret;
    }
};

BITSET ext[300];
BITSET row, X;

int BitLCS(char * s1, char * s2)
{
    int i, j;
    int len1 = strlen(s1);
    int len2 = strlen(s2);

    for (i=0; i<300; i++)
        ext[i].reset(len1);
    row.reset(len1);
    X.reset(len1);

    for (i=0; i<len1; i++)
```

```
        ext[ s1[i] ].set(i, 1);

    for (i=0; i<len2; i++)
    {
        X = row | ext[ s2[i] ];
        row <<= 1;
        row.set(0, 1);
        row = X & ((X-row) ^ X);
    }
    return row.count(len1);
}
int main()
{
    while (scanf("%s %s", str1, str2) == 2)
        printf("%d\n", BitLCS(str1, str2));
}
```

40. 伸展树

```
/*
伸展树
二叉查找树的改进，平摊复杂度都是O(log n)
维护序列，适用于统计对象次序发生大规模变化
有翻转和移动时，线段树不适用，且效率高于块状链表
HDOJ 1890 Robotic Sort
*/
const int MMAX = 101000;
struct NODE
{
    int key, cnt;// 键值,重复次数
    NODE * pl, * pr;
    NODE * pf;
};
NODE mem[MMAX];
int mempos;
NODE * root;

inline NODE * new_node()
{
    NODE * pt = &mem[mempos ++];
    memset(pt, 0, sizeof(NODE));
    return pt;
}

// x = L[y]
```

```
inline void Zig(NODE * y)
{
    NODE * x = y->pl;
    NODE * z = y->pf;
    y->pl = x->pr;
    if (y->pl) y->pl->pf = y;
    x->pr = y; y->pf = x;

    if (! z)
    {
        x->pf = NULL;
        return;
    }
    if (z->pl == y) z->pl = x;
    else z->pr = x;
    x->pf = z;
}

// x = R[y]
inline void Zag(NODE * y)
{
    NODE * x = y->pr;
    NODE * z = y->pf;
    y->pr = x->pl;
    if (y->pr) y->pr->pf = y;
    x->pl = y; y->pf = x;

    if (! z)
    {
        x->pf = NULL;
        return;
    }
    if (z->pl == y) z->pl = x;
    else z->pr = x;
    x->pf = z;
}

// y = L[z], x = L[y]
inline void ZigZig(NODE * z)
{
    NODE * y = z->pl;
    NODE * x = y->pl;
    NODE * gz = z->pf;
    y->pl = x->pr;
```

```
    if (y->pl) y->pl->pf = y;
    z->pl = y->pr;
    if (z->pl) z->pl->pf = z;
    x->pr = y; y->pf = x;
    y->pr = z; z->pf = y;

    if (! gz)
    {
        x->pf = NULL;
        return;
    }
    if (gz->pl == z) gz->pl = x;
    else gz->pr = x;
    x->pf = gz;
}

// y = R[z], x = L[y]
inline void ZigZag(NODE * z)
{
    NODE * y = z->pr;
    NODE * x = y->pl;
    NODE * gz = z->pf;
    y->pl = x->pr;
    if (y->pl) y->pl->pf = y;
    z->pr = x->pl;
    if (z->pr) z->pr->pf = z;
    x->pl = z; z->pf = x;
    x->pr = y; y->pf = x;

    if (! gz)
    {
        x->pf = NULL;
        return;
    }
    if (gz->pl == z) gz->pl = x;
    else gz->pr = x;
    x->pf = gz;
}

// y = R[z], x = R[y]
inline void ZagZag(NODE * z)
{
    NODE * y = z->pr;
    NODE * x = y->pr;
```

```
    NODE * gz = z->pf;
    y->pr = x->pl;
    if (y->pr) y->pr->pf = y;
    z->pr = y->pl;
    if (z->pr) z->pr->pf = z;
    x->pl = y; y->pf = x;
    y->pl = z; z->pf = y;

    if (! gz)
    {
        x->pf = NULL;
        return;
    }
    if (gz->pl == z) gz->pl = x;
    else gz->pr = x;
    x->pf = gz;
}

// y = L[z], x = R[y]
inline void ZagZig(NODE * z)
{
    NODE * y = z->pl;
    NODE * x = y->pr;
    NODE * gz = z->pf;
    y->pr = x->pl;
    if (y->pr) y->pr->pf = y;
    z->pl = x->pr;
    if (z->pl) z->pl->pf = z;
    x->pl = y; y->pf = x;
    x->pr = z; z->pf = x;

    if (! gz)
    {
        x->pf = NULL;
        return;
    }
    if (gz->pl == z) gz->pl = x;
    else gz->pr = x;
    x->pf = gz;
}


NODE * splay_slow(NODE * x)
{
```

```
    if (! x) return NULL;
    while (x->pf)
    {
        NODE * y = x->pf;
        if (y->pl == x) Zig(y);
        else Zag(y);
    }
    return x;
}

NODE * splay(NODE * x)
{
    if (! x) return NULL;
    while (x->pf)
    {
        NODE * y = x->pf;
        NODE * z = y->pf;
        if (z)
        {
            if (z->pl == y)
            {
                if (y->pl == x) ZigZig(z);
                else ZagZig(z);
            }
            else
            {
                if (y->pr == x) ZagZag(z);
                else ZigZag(z);
            }
        }
        else
        {
            if (y->pl == x) Zig(y);
            else Zag(y);
        }
    }
    return x;
}

NODE * find(int val, NODE * rt)
{
    NODE * x = rt;
    NODE * pre = rt;
    while (x)
```

```
    {
        if (x->key == val) return x;
        pre = x;
        if (val < x->key) x = x->pl;
        else x = x->pr;
    }
    return pre;
}


// make sure all_elem(rt1) <= all_elem(rt2)
NODE * join(NODE * rt1, NODE * rt2)
{
    if (rt1) rt1->pf = NULL;
    if (rt2) rt2->pf = NULL;
    if (! rt1) return rt2;
    if (! rt2) return rt1;
    NODE * x = find(INT_MAX, rt1);
    rt1 = splay(x);
    rt1->pr = rt2;
    rt2->pf = rt1;
    return rt1;
}

NODE * split(int val)
{
    NODE * x = find(val, root);
    if (x == NULL || x->key != val) return NULL;
    root = splay(x);
    NODE * newroot = root->pr;
    newroot->pf = NULL;
    root = root->pl;
    root->pf = NULL;
    return newroot;
}

void insert(int val)
{
    if (root == NULL)
    {
        root = new_node();
        root->key = val;
        root->cnt = 1;
        return;
```

```
    }
    NODE * x = find(val, root);
    if (x->key == val) x->cnt ++;
    else
    {
        NODE * pnew = new_node();
        pnew->key = val;
        pnew->cnt = 1;
        pnew->pf = x;
        if (val < x->key) x->pl = pnew;
        else x->pr = pnew;
    }
}

void remove(int val)
{
    NODE * x = find(val, root);
    root = splay(x);
    if (root && root->key == val)
    {
        root->cnt --;
        if (root->cnt == 0)
            root = join(root->pl, root->pr);
    }
}

void print_tree(NODE * rt)
{
    if (rt == NULL) return;
    printf(" (");
    print_tree(rt->pl);
    if (rt->pf == NULL)
        printf(" [%d] ", rt->key);
    else
        printf(" %d ", rt->key);
    print_tree(rt->pr);
    printf(") ");
}

void test_tree()
{
    int v;
    char cmd[2];
    root = NULL;
```

```
    mempos = 0;
    while (scanf("%s %d", cmd, &v) == 2)
    {
        if (cmd[0] == 'i')
            insert(v);
        else if (cmd[0] == 'r')
            remove(v);
        else if (cmd[0] == 's')
            root = splay(find(v, root));
        print_tree(root);
        puts("");
    }
}

int main()
{
    test_tree();
    return 0;
}
```

41. Treap

```
/*
Treap
是有随机数满足堆的性质的二叉搜索树
其结构相当于以随机顺序插入的二叉搜索树
其基本操作的期望复杂度为O(log n)
其特点是实现简单，效率高于伸展树并且支持大部分基本功能，性价比很高
*/
#define MAX 100
typedef struct
{
    int l,r,key,fix;
}node;

class treap
{
public:
    node p[MAX];
    int size,root;
    treap()
    {
        srand(time(0));
        size=-1;
        root=-1;
```

```
}

void rot_l(int &x)
{
    int y=p[x].r;
    p[x].r=p[y].l;
    p[y].l=x;
    x=y;
}

void rot_r(int &x)
{
    int y=p[x].l;
    p[x].l=p[y].r;
    p[y].r=x;
    x=y;
}

void insert(int &k,int tkey)
{
    if (k==-1)
    {
        k=++size;
        p[k].l=p[k].r=-1;
        p[k].key=tkey;
        p[k].fix=rand();
    }
    else
        if (tkey<p[k].key)
        {
            insert(p[k].l,tkey);
            if (p[ p[k].l ].fix>p[k].fix)
                rot_r(k);
        }
        else
        {
            insert(p[k].r,tkey);
            if (p[ p[k].r ].fix>p[k].fix)
                rot_l(k);
        }

}

void remove(int &k,int tkey)
```

```
{
    if (k==-1) return;
    if (tkey<p[k].key)
        remove(p[k].l,tkey);
    else if (tkey>p[k].key)
        remove(p[k].r,tkey);
    else
    {
        if (p[k].l==-1 && p[k].r==-1)
            k=-1;
        else if (p[k].l==-1)
            k=p[k].r;
        else if (p[k].r==-1)
            k=p[k].l;
        else
            if (p[ p[k].l ].fix < p[ p[k].r ].fix)
            {
                rot_l(k);
                remove(p[k].l,tkey);
            }
            else
            {
                rot_r(k);
                remove(p[k].r,tkey);
            }
    }
}

int find(int k,int r)
{
    if (r<=p[p[k].l].size)
        return find(p[k].l,r);
    else if (r> p[p[k].l].size+p[k].cnt)
        return find(p[k].r,r-(p[p[k].l].size+p[k].cnt));
    return p[k].key;
}

void print(int k)
{
    if (p[k].l!=-1)
        print(p[k].l);
    cout << p[k].key << " : " << p[k].fix << endl;
    if (p[k].r!=-1)
        print(p[k].r);
```

```
    }
};

treap T;

int main()
{

    int i;
    for (i=3;i>=1;i--)
        T.insert(T.root,i);
    T.print(T.root);
    for (i=3;i>=1;i--)
    {
        cout << endl;
        T.remove(T.root,i);
        T.print(T.root);
    }
    return 0;
}
```

42. 0/1 分数规划 K 约束

```
// 2976 PKU Dropping tests
// 0-1 Fractional Programing with K drop limit
/*
(Σa_i) / (Σb_i) >= x
        <=> Σa_i >= Σ(x*b_i)
        <=> Σ(a_i - x*b_i) >= 0.

let w_i = a_i - x*b_i.
max Q(x) = max (Σw_i).
so drop K smallest w_i is ok.
*/
const int NMAX = 1100;
int n, k;
int A[NMAX], B[NMAX];
struct NODE
{
    double w;
    int a, b;
    bool operator < (const NODE & nt) const
    {
        return w < nt.w;
    }
```

```
}W[NMAX];

#define EQ(a,b) (fabs((a)-(b))<1e-4)
// Dinkelbach iterative algorithm
int solve()
{
    int i, j;
    double x = 1.0;
    for (i=0; i<100; i++)
    {
        for (j=0; j<n; j++)
        {
            W[j].w = 1.0*A[j] - x*B[j];
            W[j].a = A[j];
            W[j].b = B[j];
        }
        sort(W, W+n);
        double sum = 0;
        double sa, sb;
        sa = sb = 0;
        for (j=k; j<n; j++)
        {
            sum += W[j].w;
            sa += W[j].a;
            sb += W[j].b;
        }
        if (EQ(sum, 0)) break;
        x = 1.0 * sa / sb;
    }
    return (x*100+0.5);
}

// binary enum
int solve2()
{
    int i, j;
    double lb = 0, ub = 1;
    double x = 1, prex;
    for (i=0; i<100; i++)
    {
        prex = x;
        x = (lb+ub) / 2;
        if (EQ(x, prex)) break;
        for (j=0; j<n; j++)
```

```
            W[j].w = 1.0*A[j] - x*B[j];
        sort(W, W+n);
        double sum = 0;
        for (j=k; j<n; j++)
            sum += W[j].w;
        if (sum >= 0) lb = x;
        else ub = x;
    }
    return (x*100+0.5);
}

int main()
{
    int i, j;
    while (scanf("%d %d", &n, &k), n|k)
    {
        for (i=0; i<n; i++)
            scanf("%d", A+i);
        for (i=0; i<n; i++)
            scanf("%d", B+i);
        printf("%d\n", solve());
    }
}
```

43. 表达式求值

```
// HDU 2127 Polish notation
#include <cstdio>
#include <string>
#include <algorithm>
using namespace std;
typedef __int64 int64;

const int MAX = 1100;
char exp[MAX];
int priority[MAX];
int len;
bool output;

int64 dfs(int spos, int epos) {
    int i;
    int64 op1,op2,ans;
    char opr;
    int minv, minp = -1;
    if (spos > epos || spos >= len || epos < 0) {
```

```
        return 0;
    }
    for (i=epos;i>=spos;i--) {
        if (priority[i] != 0) {
            if (minp == -1) {
                minp = i;
                minv = priority[i];
            }
            else if (minv > priority[i]) {
                minv = priority[i];
                minp = i;
            }
        }
    }
    ans = 0;
    if (minp == -1) {
        for (i=spos;i<=epos;i++) {
            ans = ans * 10 + exp[i] - '0';
        }
        if (output) putchar(' ');
        printf("%d",ans);
        output = true;
    }
    else {
        opr = exp[minp];
        if(opr != '(' && opr != ')') {
            if (output) putchar(' ');
            putchar(opr);
            output = true;

            op1 = dfs(spos,minp-1);
            op2 = dfs(minp+1,epos);
            switch(opr) {
            case '+':
                ans = op1 + op2;
                break;
            case '-':
                ans = op1 - op2;
                break;
            case '*':
                ans = op1 * op2;
                break;
            }
        }
```

```
            else {
                ans = dfs(spos,minp-1) + dfs(minp+1,epos);
            }
        }
        return ans;
}

int main() {
    int i,pre,t = 1;
    while(gets(exp)) {
        len = strlen(exp);
        pre = 0;
        // +,- 1
        // *,/ 2
        // - 3
        // () 4
        for (i=0;i<len;i++) {
            if (exp[i] == '*' || exp[i] == '/') priority[i] = pre + 2;
            else if (exp[i] == '+') priority[i] = pre + 1;
            else if (exp[i] == '(') priority[i] = pre = pre + 4;
            else if (exp[i] == ')') priority[i] = pre, pre -= 4;
            else if (exp[i] == '-') {
                if (exp[i-1]>='a' && exp[i-1]<='z' || exp[i-1]==')') priority[i] =
pre + 1;
                else priority[i] = pre + 3;
            }
            else priority[i] = 0;
        }
        printf("Case %d:\n", t ++);
        output = false;
        printf("\n%I64d\n", dfs(0,len-1));
    }
}
```

44. 乘除法博弈,Wythoff 博弈

```
/*
PKU 2633 Funny Games
给定f[1..n]和x
两人轮流选择一个f[i],使得x=x*f[i]
当x<=1.0时胜利
普通的博弈搜索难于x过大,又是浮点数
*/
typedef pair <double, double> pdd;
typedef pair <double, double> * ppdd;
```

```
#define EQ(a,b) (fabs((a)-(b)) <= 1e-8)
#define LES(a,b) ((a) < (b))
#define LEQ(a,b) ((a)+1e-8 <= (b))
pdd win[10000];

ppdd interval_union(ppdd begin, ppdd end, ppdd dest)
{
    sort(begin, end);
    for (; begin != end; dest++)
    {
        *dest = *begin;
        for (begin++; begin != end; begin++)
        {
            if (LEQ(dest->second, begin->first))
                break;
            dest->second = max(dest->second, begin->second);
        }
    }
    return dest;
}

char * solve2()
{
    int i, j;
    pdd lose;
    double maxf;
    lose.second = maxf = 0;

    for (i=0; i<k; i++)
    {
        win[i] = make_pair(1, 1.0/f[i]);
        maxf = max(maxf, f[i]);
    }
    int nwin;
    for (i=0,nwin=k; LES(lose.second, x); i++)
    {
        // union the interval
        nwin = interval_union(win+i, win+nwin, win+i) - win;
        // lose <- win
        lose.first = win[i].second;
        lose.second = win[i].second / maxf; // it's min, and must
        // if the win have many interval
        if (i < nwin-1)
            lose.second = min(lose.second, win[i+1].first);
```

```
            // win <- lose
            for (j=0; j<k; j++)
                win[nwin ++] = make_pair(lose.first/f[j], lose.second/f[j]);
        }
        if (LES(x, lose.first)) return "Nils";
        return "Mikael";
}
// 有 2 堆石子，一次可以取任意个在一堆中或者任意个在两堆中取相同数目，取完者胜利
int swap(int &x,int &y)
{
        int t;
        t=x;
        x=y;
        y=t;
}
int main()
{
        double alpha = (1.0 + sqrt(5.0)) / 2.0;
        double beta  = (3.0 + sqrt(5.0)) / 2.0;
        int big, small, n, temp1, temp2;
        while(cin>>big>>small)
        {
            if(big < small)
                swap(big, small);
            n = ceil(big / beta);
            temp1 = alpha * n;
            temp2 = beta  * n;
            if(small == temp1 && big == temp2)
                cout<<0<<endl;
            else cout<<1<<endl;
        }
        return 0;
}
```

45. 状态压缩的积木型 DP

```
// 1038 PKU Bugs Integrated, Inc.
// DP with state compression
#define MAX(a,b) ((a)>(b)?(a):(b))
int bad[160];
int n, m, k;
int e3[] = {
    1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683, 59049, 177147
};
short dp[2][60000];
```

```
/*
--→X(N)
|
↓Y(M)
0: [ ][ ]
1: [#][ ]
2: [#][#]
当维数扩展时,可以相应的扩展进制数.
此题积木可分解为1*3影响列,则记为3进制数.
当M小时,也可用进制直接保存列.
基于状态压缩的积木填充型DP，有种实现方法:
1)直接构造fm和gn状态,如此题.
2)for(fm)再构造gn状态.
3)预处理,保存<fm,gn>为边.
优劣:
1)2)易实现,但状态量大时会比较耗时
3)保存边信息后,状态转移的时间消耗少
但边分布不均,需要动态数组或链表,且要处理去重操作
*/
void dfs(int x, int y, int fm, int gn, short v)
{
    if (y > m) return;
    if (y == m)
    {
        dp[(x&1)^1][gn] = MAX(v + dp[(x&1)][fm], dp[(x&1)^1][gn]);
        return;
    }

    int mask23 = 7 << y;
    int mask32 = 3 << y;
    if (x+1 < n)
    {
        if (!(bad[x]&mask23) && !(bad[x+1]&mask23))
            dfs(x, y+3, fm, gn+13*e3[y], v+1); // 2*3
        if (x+2 < n)
        {
            if (!(bad[x]&mask32) && !(bad[x+1]&mask32)
&& !(bad[x+2]&mask32))
                dfs(x, y+2, fm, gn+8*e3[y], v+1); // 3*2
        }
    }

    dfs(x, y+1, fm, gn, v); // 0->0
    if (!(bad[x]&(1<<y)))
```

```
    {
        dfs(x, y+1, fm+1*e3[y], gn, v); // 1->0
        if (!(bad[x+1]&(1<<y)))
            dfs(x, y+1, fm+2*e3[y], gn+1*e3[y], v); // 2->1
    }
}

int solve()
{
    int i, j;
    memset(dp, 0, sizeof(dp));
    for (i=0; i<n; i++)
    {
        dfs(i, 0, 0, 0, 0);
        memset(dp[i&1], 0, sizeof(dp[0]));
    }
    return dp[i&1][0];
}

int main()
{
    int i, j;
    int cas;
    for (scanf("%d", &cas); cas; cas--)
    {
        scanf("%d %d %d", &n, &m, &k);
        memset(bad, 0, sizeof(bad));
        for (i=0; i<k; i++)
        {
            int x, y;
            scanf("%d %d", &x, &y);
            x --; y --;
            bad[x] |= 1 << y;
        }
        printf("%d\n", solve());
    }
}
```

46. 解一般线性方程组(消元法)

```
typedef int INT;
INT gcd(INT a, INT b) {
    return (b == 0)?a:gcd(b, a % b);
}
struct Fraction {
```

```
    INT up, down;
    Fraction():up(0), down(1) {};
    Fraction(INT a, INT b = 1):up(a), down(b) {};
    Fraction(const Fraction& a) {
        up = a.up;
        down = a.down;
    }
    Fraction operator - () const {
        return Fraction(-up, down);
    }
    Fraction& operator = (const Fraction& a) {
        up = a.up;
        down = a.down;
        return *this;
    }
    void reduce() {
        INT g = gcd(abs(up), abs(down));
        up /= g; down /= g;
    }
};

Fraction abs(const Fraction& a) {
    return Fraction(abs(a.up), abs(a.down));
}
Fraction operator + (Fraction a, Fraction b) {
    INT u1 = a.up * b.down + a.down * b.up;
    INT u2 = a.down * b.down;
    INT g = gcd(abs(u1), abs(u2));
    return Fraction(u1 / g, u2 / g);
}
Fraction operator - (Fraction a, Fraction b) {
    return a + (-b);
}
Fraction operator * (Fraction a, Fraction b) {
    INT u1 = a.up * b.up;
    INT u2 = a.down * b.down;
    INT g = gcd(abs(u1), abs(u2));
    return Fraction(u1 / g, u2 / g);
}
Fraction operator / (Fraction a, Fraction b) {
    INT u1 = a.up * b.down;
    INT u2 = a.down * b.up;
    if (u2 < 0) u1 = - u1, u2 = -u2;
    int g = gcd(abs(u1), abs(u2));
```

```
    return Fraction(u1 / g, u2 / g);
}
bool operator > (const Fraction& a, const Fraction& b) {
    return (a - b).up > 0;
}
bool operator == (const Fraction& a, const Fraction& b) {
    return (a - b).up == 0;
}
bool operator != (const Fraction& a, const Fraction& b) {
    return !(a == b);
}
bool operator < (const Fraction& a, const Fraction& b) {
    return (b - a).up > 0;
}
ostream& operator << (ostream& out, const Fraction& a) {
    if (a.down == 1) out << a.up;
    else out << a.up << '/' << a.down;
    return out;
}

const int nSize = 101;
const int mSize = 26;
typedef Fraction fEquation[nSize];
typedef fEquation fMatrix[mSize];
/*
解一般形式的线性方程组(只输出其中一组解)
neqn个方程，nvar个变量
矩阵表示如下
B0 = A0,0 * X0 + A0,1 * X1 + ... + A0,n-1 * Xn-1
B1 = A1,0 * X0 + A1,1 * X1 + ... + A1,n-1 * Xn-1
... ...
Bm = Am,0 * X0 + Am,1 * X1 + ... + Am,n-1 * Xn-1
*/
struct EqnGauss {
    int neqn, nvar;
    fMatrix f;
    Fraction avail[nSize];

    EqnGauss(): neqn(0), nvar(0) {}
    EqnGauss(const EqnGauss& a): neqn(a.neqn), nvar(a.nvar) {
        memcpy(f, a.f, sizeof(fMatrix));
    }
    void build() {
        for (int i = 0; i < neqn; ++ i)
```

```
            for (int j = 0; j <= nvar; ++ j) {
                int x;
                cin >> x;
                f[i][j] = Fraction(x);
            }
    }
    bool rebuild() {
        int cur = 0;
        for(int i = 1; i <= nvar; ++ i) {
            bool found = false;
            for(int j = cur; j < neqn; ++ j)
                if (f[j][i] != 0) {
                    found = true;
                    fEquation tmp;
                    memcpy(tmp, f[cur], sizeof(fEquation));
                    memcpy(f[cur], f[j], sizeof(fEquation));
                    memcpy(f[j], tmp, sizeof(fEquation));
                }
                if (!found) continue;
                f[cur][0] = f[cur][0] / f[cur][i];
                for(int j = i + 1; j <= nvar; ++ j)
                    f[cur][j] = f[cur][j] / f[cur][i];
                f[cur][i] = 1;
                for(int j = 0; j < neqn; ++ j)
                    if (j != cur && f[j][i] != 0) {
                        f[j][0] = f[j][0] - f[j][i] * f[cur][0];
                        for(int k = i + 1; k <= nvar; ++ k)
                            f[j][k] = f[j][k] - f[j][i] * f[cur][k];
                        f[j][i] = 0;
                    }
                ++ cur;
        }
        return (cur != nvar);
    }
    void solve() {
        int cur = 0;
        for(int i = 1; i <= nvar; ++ i)
            if (f[cur][i] == 0) {
                INT ulcm = 1;
                for(int j = 0; j < cur; ++ j)
                    ulcm = ulcm / gcd(ulcm, f[j][i].down) * f[j][i].down;
                avail[i] = ulcm;
            } else ++ cur;
        cur = 0;
```

```
        for(int i = 1; i <= nvar; ++ i)
            if (f[cur][i] == 1) {
                avail[i] = f[cur][0];
                for(int j = i + 1; j <= nvar; ++ j)
                    avail[i] = avail[i] - f[cur][j] * avail[j];
                ++ cur;
            }
    }
};
ostream& operator << (ostream& out, const EqnGauss& a) {
    for (int i = 1; i <= a.nvar; ++ i)
        out << a.avail[i] << endl;
    return out;
}

EqnGauss eqns;
int main() {
    while (true) {
        cin >> eqns.nvar >> eqns.neqn;
        eqns.build();
        if (eqns.rebuild()) {
            eqns.solve();
            cout << eqns;
        }
        else
            cout << "no solution" << endl;
    }
    return 0;
}
/*
4 2
10 1 1 1 1
3 1 1 0 0
4 2
10 1 1 0 0
3 1 1 0 0
*/
```

47. 块状链表

```
//块状链表
#include<iostream>
#include<cmath>
#define MAX 2900
struct block {//Type of block
```

```
    int nos;//Number of elements in this block
    char em[MAX];//elements
    block *be,*su;//previous&successor
}
*first;
struct cursor {//Type of cursor
    int n;//The postion of cursor(in one block)
    block *ll;//The block the cursor in
}
cur;
int tot,n;//tot:the total number of elements,n:the number of operations
inline void clean(block *op) {//Clean a new block
    op->nos=0;
    memset(op->em,0,sizeof op->em);
    op->be=NULL;
    op->su=NULL;
}
inline void Spilt(block *a,int newsize) {//Break a block into two blocks,one's size
equal to 'newsize',and another's equal to 'a->nos-newsize'
    if(!newsize)return;
    int tmp=a->nos;
    a->nos=newsize;
    block *tps=a->su;
    a->su=new(block);
    clean(a->su);
    a->su->be=a;
    if(tps) {
        a->su->su=tps;
        a->su->su->be=a->su;
    }
    a->su->nos=tmp-newsize;
    block *tt=a->su;
    for(int i=newsize+1;i<=tmp;i++)tt->em[i-newsize]=a->em[i];
}
inline void Merge(block *a) {//Merge a & a->su
    if(a->su==NULL)return ;
    int tmp=a->nos;
    for(int i=1;i<=a->su->nos;i++)a->em[a->nos+i]=a->su->em[i];
    a->nos+=a->su->nos;
    block *oo=a->su;
    if(cur.ll==oo) {
        cur.ll=cur.ll->be;
        cur.n+=tmp;
    }
```

```
    a->su=a->su->su;
    delete(oo);
    if(a->su)a->su->be=a;
}
inline void Balance() {//Make these blocks' size balance
    block *k=first;
    int kk=(int)sqrt(tot);
    for(;k!=NULL;) {
        for(;k->nos < kk/2 || k->nos > 2*kk;) {
            if(k->nos<kk/2) {//the block is too small?
                if(k->su)Merge(k);
                else break;
            } else if(k->nos>kk*2) {//the block is too big?
                Spilt(k,(k->nos)>>1);
                if((cur.ll==k)&&(cur.n>k->nos)) {
                    cur.ll=k->su;
                    cur.n-=k->nos;
                }
                k=k->su;
            }
        }
        k=k->su;
    }
}
inline void Insert(block *lk,int x,int k) {//Insert text behind the cursor
    block *oo=new(block);
    clean(oo);
    block *gg=oo;
    int rr=k;
    tot+=rr;
    int bt=(int)sqrt(tot);
    for(int i=0;i<rr;i++) {
        char gg;
        scanf("%c",&gg);
        for(;(gg>126)||(gg<32);scanf("%c",&gg));
        oo->em[++oo->nos]=gg;
        if((oo->nos>=bt)&&(i<rr-1)) {
            oo->su=new(block);
            clean(oo->su);
            oo->su->be=oo;
            oo=oo->su;
        }
    }
    if(x) {
```

```
            Spilt(lk,x);
        } else {
            if(!cur.ll->be) {
                block *jj;
                jj=first;
                first=new(block);
                clean(first);
                first->su=jj;
                jj->be=first;
            }
            cur.ll=cur.ll->be;
            cur.n=cur.ll->nos;
            lk=lk->be;
        }
        block *tmp=lk->su;
        lk->su=gg;
        oo->su=tmp;
        if(oo->su)oo->su->be=oo;
        lk->su->be=lk;
        cur.ll=lk->su;
        cur.n=0;
        Balance();
}
inline void Remove(block *lk,int x,int num) { //Delete 'num' elements behind the
cursor
        if(x) {
            Spilt(lk,x);
            lk=lk->su;
        } else {
            if(!cur.ll->be) {
                block *jj;
                jj=first;
                first=new(block);
                clean(first);
                first->su=jj;
                jj->be=first;
            }
            cur.ll=cur.ll->be;
            cur.n=cur.ll->nos;
        }
        tot-=num;
        int ttt=num;
        block *tmp;
        block *ii;
```

```
    for(tmp=lk;tmp&&((num-tmp->nos)>=0);tmp=ii) {
        ii=tmp->su;
        num-=tmp->nos;
        delete(tmp);
    }
    if(num&&tmp) {
        Spilt(tmp,num);
        cur.ll->su=tmp->su;
        cur.ll->su->be=cur.ll;
        if(cur.ll->be)cur.ll->be->su=cur.ll;
    } else {
        cur.ll->su=tmp;
        if(cur.ll->be)cur.ll->be->su=cur.ll;
        if(cur.ll->su)cur.ll->su->be=cur.ll;
    }
    Balance();
}
inline void Print(block *lk,int x,int num) {//print 'num' elements behind the cursor
    block *cp=lk;
    for(;num-(cp->nos-x)>0;cp=cp->su) {
        for(int i=x+1;i<=cp->nos;i++)printf("%c",cp->em[i]);
        num-=(cp->nos-x);
        x=0;
    }
    for(int i=1;i<=num;i++)printf("%c",cp->em[i+x]);
    printf("\n");
}
inline void prev() {//cursor move forward
    cur.n--;
    if(cur.n<0) {
        cur.ll=cur.ll->be;
        cur.n=cur.ll->nos-1;
    }
}
inline void next() {//cursor move backward
    cur.n++;
    if((cur.n>=cur.ll->nos)&&(cur.ll->su)) {
        cur.ll=cur.ll->su;
        cur.n-=cur.ll->be->nos;
    }
}
inline void move(int k) {//move the cursor to the postion 'k'
    cur.ll=first;
    for(;(cur.ll)&&(k-cur.ll->nos>0);cur.ll=cur.ll->su)
```

```
            k-=cur.ll->nos;
        cur.n=k;
}
/*
NOI2003 editor O(sqrt(n))
MOVE(k)     Move k      将光标移动到第k个字符之后，如果k=0，将光标移到文本开头
INSERT(n, s)    Insert n S   在光标处插入长度为n的字符串s，光标位置不变，n >= 1
DELETE(n)   Delete n    删除光标后的n个字符，光标位置不变，n >= 1
GET(n)      Get n       输出光标后的n个字符，光标位置不变，n >= 1
PREV()      Prev        光标前移一个字符
NEXT()      Next        光标后移一个字符
*/
int main() {
    freopen("editor.in","r",stdin);
    freopen("editor.out","w",stdout);
    first=new(block);
    clean(first);
    char str[100];
    cur.ll=first;
    cur.n=0;
    scanf("%d\n", &n);
    int k;
    for (int i=0;i<n;i++) {
        scanf("%s", str);
        switch (str[0]) {//deal with operations
        case 'M' : scanf("%d", &k); move(k); break;
        case 'I' : scanf("%d", &k); Insert(cur.ll,cur.n,k); break;
        case 'D' : scanf("%d", &k); Remove(cur.ll,cur.n,k); break;
        case 'G' : scanf("%d", &k); Print(cur.ll,cur.n,k);; break;
        case 'P' : prev(); break;
        case 'N' : next(); break;
        }
    }
    fclose(stdin);
    fclose(stdout);
}
```

48. Factor Oracle

```
/*
Factor Oracle
后缀自动机构造过于复杂
可利用Factor Oracle实现基于子串的搜索
实现两个串公共最长子串/单串最长重复子串的O(n)算法
(1)能在O(|u|)识别p的子串u
```

```
(2)可以识别p的所有子串,可能会误识别长度小于|p|的子串!!
(3)在O(|p|)时间内构造
*/
/*
abaaabbabaa
abab
Find, it's wrong. (2)
*/
const int INIT = 1;
const int FAIL = 0;
const int MMAX = 201000;
const int SIGMAX = 30;
/*
2774 PKU Long Long Message
求出lrs和S以后，问题就好解决了
对于公共子串，扫描后半部分的lrs，加上S的限制，防止重复串在同一个串中
若是单串，直接取lrs中的最大值
*/
char str[MMAX];
struct ORACLE
{
    int T[MMAX][SIGMAX];
    int S[MMAX];
    int LRS[MMAX];
    int SN;
    int c2d[256];

    void init() {
        SN = INIT + 1;
        S[INIT] = FAIL;
        LRS[INIT] = 0;
//      memset(T, FAIL, sizeof(T)); // 节省清空的时间开销
//      memset(c2d, FAIL, sizeof(c2d));
    }
    void add(char ch) {
        int m = SN - 1;
        SN ++;
        ch = c2d[ch];
        T[m][ch] = m + 1;
        int k = S[m];
        int pre = m;
        while (k != FAIL && T[k][ch] == FAIL) {
            T[k][ch] = m + 1;
            pre = k;
```

```
            k = S[k];
        }
        int shift;
        if (k == FAIL) shift = INIT;
        else shift = T[k][ch];
        S[m+1] = shift;
        LRS[m+1] = len_repeat_suffix(pre, S[m+1]);
    }
    void construct(char * p) {
        for (int i=0,j=FAIL+1; p[i]; i++) {
            if (c2d[ p[i] ] == FAIL)
                c2d[ p[i] ] = j ++;
            add(p[i]);
        }
    }
    int len_common_suffix(int p1, int p2) {
        if (S[p1] == p2) return LRS[p1];
        while (S[p1] != S[p2]) p2 = S[p2];
        return min(LRS[p1], LRS[p2]);
    }
    int len_repeat_suffix(int p1, int m) {
        if (m == INIT) return 0;
        return len_common_suffix(p1, m-1) + 1;
    }
};
ORACLE fo;

int main()
{
    gets(str);
    int len = strlen(str);
    str[len] = '$';
    gets(str+len+1);

    fo.init();
    fo.construct(str);
    int ans = 0;
    for (int i=len+INIT+2; i<fo.SN; i++)
    {
        if (fo.S[i] >= len+INIT+2) continue;
        ans = max(ans, fo.LRS[i]);
    }
    printf("%d\n", ans);
}
```

49.