中北大学 ACM-ICPC 程序设计创新实验室
2015 年新生选拔赛

题解

2015 年 12 月 29 日

Problem A　签到题

Problem B　山门口

Problem C　大殿

Problem D　藏经阁

Problem E　迷雾森林

Problem F　莲花池

Problem G　密道入口

Problem H　包子铺

Problem I　京城戏院

Problem J　队列极值

Problem K　吃包子

Problem A　签到题

分析：

第一题是签到题，难度不高，读入字符串然后输出对应的小写字符串。

代码：

```
//Author:LiuYuan

#include<stdio.h>
int t;
char s[1000],*p;
int main()
{
    scanf("%d",&t);
    while(t--)
    {
        scanf("%s",s);
        p=s;//指针指向字符串首
        while(*p)//当p指针指向的不是'\0'时
        {
            putchar(*p-'A'+'a');//挨个输出小写
            p++;//p指针指向字符串的下一个字符
        }
        printf("\n");
    }
    return 0;
}
```

标程：

```
//Author:Andy

#include <cstdio>

int main()
{
    int T;
    char str[102];

    scanf( "%d", &T );
    while ( T-- )
    {
        scanf( "%s", str );
        for ( int i=0; str[i]!='\0'; i++ )
            str[i] |= 0x20;
        printf( "%s\n", str );
    }

    return 0;
}
```

Problem B　山门口

分析：

求一个 n 次方程的 m 阶导数，考察浮点数运算，没有什么高深的算法。不过这个题实现复杂，思路要特别清晰才能不犯错，十分考察基本功，而且高数要学好。

（编辑注：使用 pow()函数时应当注意计算精度。）

代码：

```
//Author:LiuYuan

#include<stdio.h>
#include<math.h>
int t,n,m,i,x,o,j;
double k,s,f[1000];
int main()
{
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d%d%lf",&n,&m,&k);
        for(i=0;i<=n;i++)scanf("%lf",&f[i]);//读入各次项的系数
        s=0;
        for(i=0;i<=n;i++)
        {
            x=i-m;//x是求导后的次数
            o=1;
            for(j=x+1;j<=i;j++)o*=j;
            //o是求导后的得到的系数，这里x<0的话o会为0;
            if(x>=0)s+=f[i]*o*pow(k,x);//求出这一项的结果并加到答案中
        }
        printf("%.2lf\n",s);
    }
    return 0;
}
```

Problem C　大殿

分析：

排序题，读入学号和名字，按学号排序输出名字，题目不难。

代码：

```
//Author:LiuYuan

#include<stdio.h>
#include<math.h>
struct cat
{
       int x;
       char n[100];//使用结构体存储数据，x是学号，n是名字
}a[10000];
int i,ppp,n;
int cmp(const void *a,const void *b)
{
//注意使用qsort函数要需要一个比较函数，而且两个参数是
//不带类型的指针const void *
//而且还要再类型转换成你的数据，转换出来还是个指针。
    return (*(struct cat *)a).x-(*(struct cat *)b).x;
}
int main()
{
    ppp=0;
    while(1)
    {
        scanf("%d",&n);
        if(n==0)break;
        ppp++;
        printf("Case %d:\n",ppp);
        for(i=0;i<n;i++)scanf("%d%s",&a[i].x,&a[i].n);
        qsort(a,n,sizeof(a[0]),cmp);//系统自带的排序函数qsort
        for(i=0;i<n;i++)printf("%s\n",a[i].n);
    }
    return 0;
}
```

Problem D　藏经阁

分析 1：

读入 x，问 x 在第几行第几列，这个题可以找规律，或者用累加公式。

代码：

```
//Author:LiuYuan

#include<stdio.h>
#include<math.h>
int t,n,x;
int main()
{
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d",&x);
        if(x&1)printf("Poor HuJie!\n");
        else
        {
            n=1;
            while(n*(n+1)<x)n++;
            n--;
            //第n行最后的数，是n*（n+1），然后找到小于x的最大的行
            printf("%d %d\n",n+1,(x-n*(n+1))/2);
            //n+1就是x所在的行，x-n*（n+1)就是所在的列
        }
    }
    return 0;
}
```

分析 2：

将偶数塔的所有标号都除以 2，则可以得到一个按正整数顺序排列的塔。此时我们发现，第n行最后一个卷轴的标号$x_{(n,n)}$为

$$x_{(n,n)} = 2\left[\sum_{i=1}^{n} i\right] = 2\left[\frac{n(n+1)}{2}\right] = n^2 + n$$

因此可以通过公式直接求得出编号为x所在的行号n和列号m

$$n = \left\lceil \frac{\sqrt{4x} - 1}{2} \right\rceil, \qquad m = \frac{-n^2 + n + x}{2}$$

其中“⌈ ⌉”符号为高斯记号，表示向上取整，与 ceiling()函数作用相同。

标程：

```
//Author:Andy

#include <cstdio>
#include <cmath>

int main()
{
    int T, x, hang, lie;

    scanf( "%d", &T );
    while ( T-- )
    {
        scanf( "%d", &x );
        if ( x & 1 )  //判断奇偶
            printf( "Poor HuJie!\n" );
        else
        {
            hang = ceil( (sqrt(x*4+1)-1.0)/2.0 );  //计算行
            lie  = ( -hang*hang + hang + x ) >> 1; //计算列
            printf( "%d %d\n", hang, lie );
        }
    }

    return 0;
}
```

Problem E　迷雾森林

分析：

读入一个 n,问多少个素数相加等于 n，这道题数据量小，可以直接暴力搜索。

代码：

```
//Author:LiuYuan

#include<stdio.h>
#include<math.h>
int t,x,sum,i;
int prime(int x)//prime函数是判断x是否为素数的
{
    int i;
    for(i=2;i<=sqrt(x);i++)
        if(x%i==0)return 0;
    return 1;
}
int main()
{
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d",&x);
        sum=0;
        //读入x，一个i从2到x/2循环，如果i和x-i都是素数，则计数器+1
        for(i=2;i<=x/2;i++)
            if(prime(i)&& prime(x-i))sum++;
        printf("%d\n",sum);
    }
    return 0;
}
```

Problem F　莲花池

分析：

题意是大鱼只会吃掉比自己小 1 的鱼，而且不是同时吃的，注意到数据 ai<=n，所以模拟的时候要从小鱼到大鱼轮着吃。读入的时候要记录，体重为 i 的鱼在数组中所在的位置，轮到它的时候可以直接找它，这道题应该使用双向链表，代码使用了数组模拟的链表，记录了左右节点，删除节点和链表的处理方法一样。

（编辑注：标程使用的是链表结构）

代码：

```
//Author:LiuYuan

#include<stdio.h>
#include<math.h>
struct
{
    int x,z,y;//x是鱼的体重，z是左边相邻的鱼，y是右边相邻的鱼
}a[1000001];
int op[1000001];//体重为i的鱼在数组中所在的位置
int n,i,j,l,r,sum,k;
int main()
{
    while(1)
    {
        scanf("%d",&n);
        if(!n)break;
        for(i=1;i<=n;i++)
        {
            scanf("%d",&a[i].x);
            a[i].z=i-1;
            a[i].y=i+1;
            op[a[i].x]=i;
        }
        a[1].z=0;
        a[n].y=0;
        //边界点特殊处理
        sum=n;//刚开始鱼塘里有n条鱼
        for(i=2;i<=n;i++)
        {
            k=op[i];//该体重为i序号为k的鱼吃小鱼了
            l=a[k].z;//看看k左边相邻的鱼能吃不?
            if(l>0 && a[l].x==a[k].x-1)
            {
                sum--;//左边的刚好比k小1，吃掉它，鱼的总数少一只
                a[k].z=a[l].z
                a[a[l].z].y=k;
                //因为左边的鱼被吃掉了，所以k变成了，被吃掉鱼左边的鱼的右边
                a[k].z=a[l].z;//k的左边也变成了，被吃掉鱼的左边
            }
            r=a[k].y;
```

```
            //r是k右边相邻的鱼，处理方法同上
            if(r>0 && a[r].x==a[k].x-1)
            {
                sum--;
                a[a[r].y].z=k;
                a[k].y=a[r].y;
            }
        }
        printf("%d\n",sum);
    }
    return 0;
}
```

标程：

```
//Author:Andy

#include <cstdio>
#include <cstdlib>

struct FISH{
    int v;
    FISH *l;
    FISH *r;
} fish[1001000], *index[1001000];

void Q_sort( int left, int right )
{
    int l = left;
    int r = right;
    int flag = index[left+right>>1]->v;
    while ( l <= r )
    {
        while ( index[l]->v < flag )
            l++;
        while ( index[r]->v > flag )
            r--;
        if ( l <= r )
        {
            FISH *tmp;
            tmp = index[l];
            index[l] = index[r];
            index[r] = tmp;
            l++;
            r--;
        }
    }
    if ( left < r )
        Q_sort( left, r );
    if ( l < right )
        Q_sort( l, right );
}

int main()
{
    int n, ans;
    while ( scanf( "%d", &n ) )
    {
        if ( n == 0 )
            break;
```

```
        for ( int i=1; i<=n; i++ )
        {
            scanf( "%d", &fish[i].v );
            fish[i].l = fish + i-1;
            fish[i].r = fish + i+1;
            index[i]  = fish + i;
        }
        fish[0].v = fish[n+1].v = -1;
        Q_sort( 1, n );
        ans = n;
        for ( int i=1; i<=n; i++ )
        {
            if ( ( index[i]->l->v - index[i]->v == 1 ) || ( index[i]-
>r->v - index[i]->v == 1 ) )
            {
                index[i]->l->r = index[i]->r;
                index[i]->r->l = index[i]->l;
                ans--;
            }
        }
        printf( "%d\n", ans );
    }

    return 0;
}
```

Problem G　密道入口

描述：

题意是有 0 到 9 的 10 个数字，给一个 3 位数，然后用这 3 个数之外的 7 个数字组成一个成立的竖式。这道题数据量很小，可以直接枚举所有的可能。

代码：

```
//Author:LiuYuan

#include<stdio.h>
#include<math.h>
char v[100];//记录数是否被使用过
int t,ans,k1,k2,k3,k,i,x,yes,j;
char s[100];
int cat(int x)
{
    if(v[x])return 1;
    v[x]=1;
    return 0;
}
int main()
{
    scanf("%d",&t);
    ans=0;
    while(t--)
    {
        scanf("%s",s);
        k1=s[0]-'0';
        k2=s[1]-'0';
        k3=s[2]-'0';
        k=(k1)*100+(k2)*10+k3;//读入竖式上面的3位数
        ans=0;
        for(i=100;i<1000;i++)//竖式下面的从100枚举到999
            if(k+i>999)//如果上面的数加下面的数结果是4位数
            {
                for(j=0;j<10;j++)v[j]=0;
                x=k+i;
                v[k1]=1;
                v[k2]=1;
                v[k3]=1;
                yes=cat(i/100)+cat(i%10)+cat(i/10%10)+cat(x/1000)
                +cat(x%10)+cat(x/100%10)+cat(x/10%10);
                if(yes==0)ans++;
                    //然后判断这个竖式中的数字是否重复，如果没有重复，
                    //则是一种方案，计数器+1
            }
        printf("%d\n",ans);
    }
    return 0;
}
```

Problem H　包子铺

描述：

题意是有 n 种不同面值的钱币，每个面值为 m_i 的钱币数量有 c_i 个，问有多少种组成方案，能刚好组成包子的价格 p 这道题算法是搜索所有可能的组成方案，统计组成 p 价格的个数

代码：

```
#include<stdio.h>
#include<math.h>
int n,p,sum,i;
int m[100],c[100];

int cat(int money,int k)
{//序号1到k-1的钱币已经组成了价格money，现在来看第k的钱币使用的数量
    int i;
    if(k>n || money>p)
    {
        //递归出口，当所有的钱币方案已经枚举完成，或者组成的价格超过了答案
        if(money==p)sum++;//如果等于答案，方案+1
        return 0;
    }
    for(i=0;i<=c[k];i++)//i是第k种钱币使用的数量，从0到拥有的数量枚举
        cat(money+i*m[k],k+1);//递归进入下一层
    return 0;
}
int main()
{
    while(1)
    {
        scanf("%d%d",&n,&p);
        if(n+p==0)break;
        for(i=1;i<=n;i++)scanf("%d%d",&m[i],&c[i]);
        sum=0;//sum是方案总数
        cat(0,1);//现在开始枚举序号为1的钱币，已组成0元
        printf("%d\n",sum);
    }
    return 0;
}
```

Problem I　京城戏院

分析 1：

题目的意思是，给出 2 个点的经纬坐标，求出他们之间球面距离。这道题读入是个难点，scanf 函数遇到空格和回车会停止，并把回车和空格留在输入缓冲区，所以要把回车和空格读入掉，才能够正确的读入数据通过经纬坐标求球面距离可以通过推导求得，比赛最好准备上笔和纸，经纬坐标类似极坐标，通过极坐标系转换成直角坐标系的公式为

$$x = r * \sin(a) * \cos(b)$$
$$y = r * \cos(a) * \cos(b)$$
$$z = r * \sin(b)$$

这个公式也可以自己推导出来，求出直角坐标系后，求出他们之间的直线距离然后分别连接原点，组成一个扇形，圆弧段就是所要求的答案，可以通过距离和 r 求出正弦，并用反三角函数，求出圆心角的弧度，最后利用求弧公式，求出的圆弧段就是答案，所以说数学很重要。

代码：

```
//Author:LiuYuan

#include<stdio.h>
#include<math.h>
const double pi=3.1415926;
struct cat
{
    double x,y,z;
} op1,op2;
char a1,a2,c1,c2,t;
double b1,b2,d1,d2,r,yye;

struct cat flycat(char a,double b,char c,double d)
{
    struct cat ans;
    double x,y,z;
    //经纬度是角度，必须先转化为弧度
    b=b/360*2*pi;
    d=d/360*2*pi;
    //如果是南纬或者西经弧度要变成负数
    if(a=='S')b=-b;
    if(c=='W')d=-d;
    x=r*sin(d)*cos(b);
    y=r*cos(d)*cos(b);
    z=sin(b)*r;
    ans.x=x;
    ans.y=y;
    ans.z=z;
    return ans;;
}
int main()
{
```

```
    while(1)
    {
        scanf("%lf",&r);
        if(r==0)break;
        getchar();//吃掉回车
        a1=getchar();
        scanf("%lf",&b1);
        getchar();//吃掉空格
        a2=getchar();
        scanf("%lf",&b2);
        getchar();//吃掉回车
        c1=getchar();
        scanf("%lf",&d1);
        getchar();//吃掉空格
        c2=getchar();
        scanf("%lf",&d2);
        //flycat函数，通过经纬度，求出直角坐标，返回类型是结构体
        op1=flycat(a1,b1,a2,b2);
        op2=flycat(c1,d1,c2,d2);
        //求2点的直线距离
        yye=sqrt((op1.x-op2.x)*(op1.x-op2.x)+(op1.y-op2.y)*(op1.y-
op2.y)+(op1.z-op2.z)*(op1.z-op2.z));
        yye=yye/2;
        yye=yye/r;
        //yye现在是正弦
        yye=asin(yye);//反三角函数求出圆心角的弧度
        yye=yye*2*r;//求出圆弧长即为答案
        printf("%.2lf\n",yye);
   }
    return 0;
}
```

分析 2：

完成此题，可以先求出两点到球心连线的夹角θ，然后乘以半径r得到最短球面路径距离s。其中夹角θ的求解是关键，标程解法是将经纬度$(LAT, LONG)$转换为向量$\langle x, y, z\rangle$，再通过向量点积和反三角函数求出夹角θ，最后乘以半径r求出距离s。需要注意的是，math.h 进行三角函数运算时的单位为弧度。公式如下：

$$\begin{cases} x = \cos(LAT)\sin(LONG) \\ y = \cos(LAT)\cos(LONG) \\ z = \sin(LAT) \end{cases}$$

$$\theta = \langle \vec{a}, \vec{b} \rangle = \cos^{-1}\left(\frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|}\right) = \cos^{-1}(x_a x_b + y_a y_b + z_a z_b)$$

$$s = r\theta$$

标程：

```
//Author:Andy

#include <cstdio>
#include <cmath>

#define ZERO 0.000000001
```

```
double READ_COORDINATE()
{
    char c, str[100];
    double l;
    scanf( "%s%lf", str, &l );
    c = str[0];
    switch( c & 0xdf )
    {
        case 'S': l = -l;        //南纬变负数
        case 'N': break;
        case 'W': l = -l;        //西经变负数
        case 'E': if ( l <= -180.0 )    //东西180度交线
                      l = 180.0; //变成正数
                  break;
    };
    l = l * M_PI / 180.0;    //转换为弧度
    if ( l >= M_PI )
        l = M_PI;
    if ( l <= -M_PI )
        l = -M_PI;
    return l;
}

int main()
{
    double r, s;
    double lat_1, lat_2, long_1, long_2;
    double x1, y1, z1, x2, y2, z2;
    while ( scanf( "%lf", &r ) )
    {
        if ( r < ZERO )
            break;

        lat_1  = READ_COORDINATE();
        long_1 = READ_COORDINATE();

        lat_2  = READ_COORDINATE();
        long_2 = READ_COORDINATE();

        x1 = sin(long_1) * cos(lat_1);
        y1 = cos(long_1) * cos(lat_1);
        z1 = sin(lat_1);

        x2 = sin(long_2) * cos(lat_2);
        y2 = cos(long_2) * cos(lat_2);
        z2 = sin(lat_2);

        s = acos( x1*x2 + y1*y2 + z1*z2 ) * r;

        printf( "%.2lf\n", s );
    }
    return 0;
}
```

Problem J　队列极值

分析：

有个队列，有 4 种队列操作，入队，出队，求队列中最大值，求队列中最小值。每次执行其中一种，但是执行操作的次数由 5x10^5，模拟的话时间是不够用的。可以利用 2 个栈实现队列，每次操作复杂度都是 O(1)，有兴趣的可以看看《编程之美》里有个类似的问题。

（编辑注：详见电子工业出版社《编程之美——微软技术面试心得》第三章第 7 节“队列中取最大值操作问题”）

标程 1：

```
//Author:Andy

//使用两个栈模拟一个队列
//使得每次询问与更新的平均时间复杂度为O(1)

#include <cstdio>

#define N   1001000
#define BIG 2000000000

#define max(a,b) (((a)>(b))?(a):(b))
#define min(a,b) (((a)<(b))?(a):(b))

#define ASK_MAX() max( S1->max_e[ S1->top ], S2->max_e[ S2->top ] )
#define ASK_MIN() min( S1->min_e[ S1->top ], S2->min_e[ S2->top ] )

#define DEBUG

struct STACK{
    int e[N];
    int max_e[N];
    int min_e[N];
    int top;
};

STACK s1, s2;
STACK *S1 = &s1, *S2 = &s2;

void INI_STACK( STACK *s )
{
    s->top = 0;
    s->max_e[0] = -1;
    s->min_e[0] = BIG;
}

void PUSH( STACK *s, int x )
{
    s->top++;
    s->e[ s->top ] = x;
    s->max_e[ s->top ] = max( x, s->max_e[ s->top - 1 ] );
```

```
    s->min_e[ s->top ] = min( x, s->min_e[ s->top - 1 ] );
}

int POP( STACK *s )
{
    int ret = -1;
    if ( s->top >= 1 )
    {
        ret = s->e[ s->top ];
        s->top--;
    }
    return ret;
}

void ENQUEUE( int x )
{
    PUSH( S1, x );
}

int DEQUEUE()
{
    if ( S2->top <= 0 )
        while ( S1->top > 0 )
            PUSH( S2, POP(S1) );
    return POP(S2);
}

#ifndef ASK_MAX()
inline int ASK_MAX()
{
//  if ( S1->top<=0 && S2->top<=0 )
//      return -1;
    return max( S1->max_e[ S1->top ], S2->max_e[ S2->top ] );
}
#endif

#ifndef ASK_MIN()
inline int ASK_MIN()
{
//  if ( S1->top<=0 && S2->top<=0 )
//      return -1;
    return min( S1->min_e[ S1->top ], S2->min_e[ S2->top ] );
}
#endif

int main()
{
    freopen( "J.in", "r", stdin );
    freopen( "J.out", "w", stdout );

    int n, x, cas = 1;
    char cmd[10];
    while ( scanf( "%d", &n ) )
    {
        if ( n == 0 )
            break;
        INI_STACK( S1 );
        INI_STACK( S2 );
        printf( "Case %d:\n", cas++ );
        while ( n-- )
```

```
        {
            scanf( "%s", cmd );
            if ( cmd[0] == 'E' )
            {
                scanf( "%d", &x );
                ENQUEUE( x );
            }
            else
            {
                switch( cmd[1] )
                {
                    case 'E':  x = DEQUEUE();    break;
                    case 'A':  x = ASK_MAX();    break;
                    case 'I':  x = ASK_MIN();
                };
                if ( x<0 || x>=BIG )
                    printf( "EMPTY!\n" );
                else
                    printf( "%d\n", x );
            }
        }
    }

    return 0;
}
```

标程 2：

```
//Author:HuJie

//用一个双端队列来模拟创建队堆
//维护最大最小值的复杂度是常数级别

#pragma comment(linker, "/STACK:1677721600")
#include <map>
#include <set>
#include <cmath>
#include <queue>
#include <stack>
#include <vector>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <climits>
#include <cassert>
#include <iostream>
#include <algorithm>
#define pb push_back
#define mp make_pair
#define LL long long
#define lson lo,mi,rt<<1
#define rson mi+1,hi,rt<<1|1
#define Min(a,b) ((a)<(b)?(a):(b))
#define Max(a,b) ((a)>(b)?(a):(b))
#define mem(a,b) memset(a,b,sizeof(a))
#define FIN freopen("J.in", "r", stdin)
#define FOUT freopen("myJ.out", "w", stdout)
#define rep(i,a,b) for(int i=(a); i<=(b); i++)
```

```
#define dec(i,a,b) for(int i=(a); i>=(b); i--)

using namespace std;
const int mod = 1e9 + 7;
const double eps = 1e-8;
const double ee = exp(1.0);
const int inf = 0x3f3f3f3f;
const int maxn = 1e6 + 10;
const double pi = acos(-1.0);

int readT()
{
    char c;
    int ret = 0,flg = 0;
    while(c = getchar(), (c < '0' || c > '9') && c != '-');
    if(c == '-') flg = 1;
    else ret = c ^ 48;
    while( c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c ^
48);
    return flg ? - ret : ret;
}

LL readTL()
{
    char c;
    int flg = 0;
    LL ret = 0;
    while(c = getchar(), (c < '0' || c > '9') && c != '-');
    if(c == '-') flg = 1;
    else ret = c ^ 48;
    while( c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c ^
48);
    return flg ? - ret : ret;
}

class listNode
{
public:
    int data;
    int counter;
    listNode* pred;
    listNode* succ;
    listNode() {}
};

class Deque
{
public:
    Deque()
    {
        head = new listNode;
        tail = new listNode;
        head -> succ = tail;
        tail -> pred = head;
        head -> pred = NULL;
        tail -> succ = NULL;
    }
    ~Deque() {}
    listNode* first()
    {
```

```
        return head -> succ;
    }
    listNode* last()
    {
        return tail -> pred;
    }
    void insertFront(int e)
    {
        listNode* now;
        now = new listNode;
        now -> data = e;
        listNode* to = head -> succ;
        head -> succ = now;
        now -> pred = head;
        now -> succ = to;
        to -> pred = now;
        return;
    }
    void insertRear(int e)
    {
        listNode* now;
        now = new listNode;
        now -> data = e;
        listNode* fr = tail -> pred;
        tail -> pred = now;
        now -> succ = tail;
        now -> pred = fr;
        fr -> succ = now;
        return;
    }
    listNode* removeFront()
    {
        listNode* now = head -> succ;
        head -> succ = now -> succ;
        head -> succ -> pred = head;
        return now;
    }
    listNode* removeRear()
    {
        listNode* now = tail -> pred;
        tail -> pred = now -> pred;
        tail -> pred -> succ = tail;
        return now;
    }
    bool empty()
    {
        return head -> succ == tail;
    }
    void print()
    {
        listNode* now = head -> succ;
        while (now != tail)
        {
            printf("%d ", now -> data);
            now = now -> succ;
        }
        printf("\n");
        return;
    }
private:
```

```
    listNode* head;
    listNode* tail;
};

class Queap
{
public:
    Queap();
    void enqueap(int);
    int dequeap();
    int min();
    int max();
    bool empty();
private:
    listNode* head;
    listNode* tail;
    Deque maxDQ;
    Deque minDQ;
};

Queap::Queap()
{
    head = new listNode;
    tail = new listNode;
    head -> succ = tail;
    head -> pred = NULL;
    tail -> pred = head;
    tail -> succ = NULL;
}

bool Queap::empty()
{
    return head -> succ == tail;
}

void Queap::enqueap(int value)
{
    listNode* now;
    now = new listNode;
    now -> data = value;
    listNode* fr = tail -> pred;
    tail -> pred = now;
    now -> succ = tail;
    now -> pred = fr;
    fr -> succ = now;
    int cnt = 1;
    while (!minDQ.empty() && (minDQ.last() -> data >= value))
    {
        cnt += (minDQ.removeRear() -> counter);
    }
    minDQ.insertRear(value);
    minDQ.last() -> counter = cnt;

    cnt = 1;
    while (!maxDQ.empty() && (maxDQ.last() -> data <= value))
    {
        cnt += (maxDQ.removeRear() -> counter);
    }
    maxDQ.insertRear(value);
    maxDQ.last() -> counter = cnt;
```

```
    return;
}

int Queap::dequeap()
{
    listNode* now = head -> succ;
    head -> succ = now -> succ;
    head -> succ -> pred = head;
    if (!(--(minDQ.first() -> counter)))
        minDQ.removeFront();
    if (!(--(maxDQ.first() -> counter)))
        maxDQ.removeFront();
    return now -> data;
}

int Queap::min()
{
    return minDQ.first() -> data;
}

int Queap::max()
{
    return maxDQ.first() -> data;
}

int main()
{
#ifdef LOCAL
    FIN;
    FOUT;
#endif // LOCAL
    int n;
    int ca = 1;
    while (~scanf("%d", &n) && n)
    {
        Queap q;
        printf("Case %d:\n", ca++);
        while (n--)
        {
            char op[10];
            scanf("%s", op);
            if (op[0] == 'E')
            {
                int x = readT();
                q.enqueap(x);
            }
            if (q.empty())
            {
                puts("EMPTY!");
                continue;
            }
            if (op[0] == 'D')
            {
                printf("%d\n", q.dequeap());
            }
            if (op[0] == 'M')
            {
                if (op[1] == 'A')
                {
```

```
                printf("%d\n", q.max());
            }
            else
            {
                printf("%d\n", q.min());
            }
        }
    }
    return 0;
}
```

Problem K　吃包子

分析：

有 n 个节点的图，有 m 条边连接图之间的节点，问从节点 1 到节点 n 可以有多少条路径，不通过相同节点（每个节点只能通过 1 次）。这道题是个图论中的网络流问题（最大流），可以使用增广路算法或预流推进算法。关于图论，笔者也接触不多，以下是标程。

（编辑注：此题为网络流中的最大流问题，比较特殊的一点是需要求出最多的顶点不相交路径数，也就是说流量限制在节点上，除源点和汇点外其余节点流量不能超过 1。有兴趣的同学可以参考机械工业出版社《算法导论（原书第二版）》中网络流部分的内容以及习题）

（编辑另注：你们有木有发现，本题和上一题的标程的作者这就是题目的主角 HuJie 啊啊~~~这么牛的代码快来膜拜!!!）

标程：

```
//Author:HuJie

#pragma comment(linker, "/STACK:1677721600")
#include <map>
#include <set>
#include <cmath>
#include <queue>
#include <stack>
#include <vector>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <climits>
#include <cassert>
#include <iostream>
#include <algorithm>
#define pb push_back
#define mp make_pair
#define LL long long
#define lson lo,mi,rt<<1
#define rson mi+1,hi,rt<<1|1
#define Min(a,b) ((a)<(b)?(a):(b))
#define Max(a,b) ((a)>(b)?(a):(b))
#define mem(a,b) memset(a,b,sizeof(a))
#define FIN freopen("in.txt", "r", stdin)
#define FOUT freopen("out.out", "w", stdout)
#define rep(i,a,b) for(int i=(a); i<=(b); i++)
#define dec(i,a,b) for(int i=(a); i>=(b); i--)

using namespace std;
const int mod = 1e9 + 7;
const double eps = 1e-8;
const double ee = exp(1.0);
const int inf = 0x3f3f3f3f;
const int maxn = 2e3 + 10;
```

```
const double pi = acos(-1.0);

int readT()
{
    char c;
    int ret = 0,flg = 0;
    while(c = getchar(), (c < '0' || c > '9') && c != '-');
    if(c == '-') flg = 1; else ret = c ^ 48;
    while( c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c ^
48);
    return flg ? - ret : ret;
}

LL readTL()
{
    char c;
    int flg = 0;
    LL ret = 0;
    while(c = getchar(), (c < '0' || c > '9') && c != '-');
    if(c == '-') flg = 1; else ret = c ^ 48;
    while( c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c ^
48);
    return flg ? - ret : ret;
}

struct Edge
{
    int fr, to, cap, flow;
    Edge(){}
    Edge(int _fr, int _to, int _cap, int _flow)
    {
        fr = _fr;
        to = _to;
        cap = _cap;
        flow = _flow;
    }
};

int n, f, d;
int flow[maxn][maxn];
int cap[maxn][maxn];
int lev[maxn];
int a[maxn];

void init()
{
    mem(cap, 0);
    mem(flow, 0);
    mem(lev, 0);
    mem(a, 0);
}

void addEdge(int fr, int to, int c)
{
    cap[fr][to] = c;
    cap[to][fr] = c;
}

//bfs找层次网络，一次寻找多条增广路径
```

```
//st最小顶点编号，ed最大顶点编号，s源点，t汇点
bool bfs(int st, int ed, int s, int t)
{
    memset(lev, inf, sizeof(lev));
    queue<int> q;
    q.push(s);
    lev[s] = 0;
    while (!q.empty())
    {
        int t = q.front();
        q.pop();
        for (int i = st; i <= ed; i++)
        {
            if (flow[t][i] < cap[t][i])
            {
                if (lev[t] + 1 < lev[i])
                {
                    lev[i] = lev[t] + 1;
                    q.push(i);
                }
            }
        }
    }
    return lev[t] < inf;
}

//利用层次网络进行增广，每次dfs寻找的是从该点出发进行dfs增加的总量
//a表示从源点到该节点可增广流量
int dfs(int v, int st, int ed, int s, int t)
{
    int res = 0;
    if (v == t)
        return a[t];
    for (int i = st; i <= ed; i++)
    {
        if (a[v] == 0)
            break;
        if (flow[v][i] < cap[v][i] && lev[v] + 1 == lev[i])
        {
            a[i] = min(a[v], cap[v][i] - flow[v][i]);
            int temp = dfs(i, st, ed, s, t);
            res += temp;
            a[v] -= temp;
            flow[v][i] += temp;
            flow[i][v] -= temp;
        }
    }
    if (res == 0)
    {
        lev[v] = inf;
    }
    return res;
}

int dinic(int st, int ed, int s, int t)
{
    int res = 0;
    memset(flow, 0, sizeof(flow));
    while (bfs(st, ed, s, t))
```

```
    {
        memset(a, inf, sizeof(a));
        int temp = dfs(s, st, ed, s, t);
        if (temp == 0)
            break;
        res += temp;
    }
    return res;
}

int main()
{
    #ifdef LOCAL
    FIN;
    FOUT;
    #endif // LOCAL
    int n, m;
    while (~scanf("%d%d", &n, &m))
    {
        if (n == 0 && m == 0)
        {
            break;
        }

        init();

        int s = 1, t = n;

        rep(i, 1, m)
        {
            int fr = readT();
            int to = readT();
            if (fr == to)
                continue;
            if (fr == s)
            {
                addEdge(fr, to, 1);
            }
            else if (to == n)
            {
                int tmp = fr + n - 1;
                addEdge(fr, tmp, 1);
                addEdge(tmp, t, 1);
            }
            else
            {
                int tFr = fr + n - 1;
                int tTo = to + n - 1;
                addEdge(fr, tFr, 1);
                addEdge(to, tTo, 1);
                addEdge(tFr, to, 1);
            }
        }
        int ans = dinic(1, 2 * n - 2, s, t);
        printf("%d\n", ans);
    }
    return 0;
}
```