E6-solution

A a^b problem Ver.⑨

难度	考点
1	函数调用

问题分析

直接调用函数即可。

参考代码

```c
#include <stdio.h>
#define LL long long

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

int main()
{
  LL a, b, p;
  scanf("%lld%lld%lld", &a, &b, &p);
  printf("%lld", KSM(a, b, p));
  return 0;
}
```

B ljh想对齐

难度	考点
2	数组，循环

问题分析

题目要求我们实现简单的区间标记，以及单点统计，考虑到n的范围很小，所以我们可以直接用一个数组来存储$1...n$所有页数被看过的次数，最后枚举一遍数组看看有没有那一页被看次数为0，并求出最大值即可。

参考代码

```c
#include<stdio.h>
int m,n,L,R,cnt[1010];
int main()
{
  scanf("%d%d",&m,&n);
  for(int i=1;i<=m;i++)
  {
    scanf("%d%d",&L,&R);
    for(int j=L;j<=R;j++) cnt[j]++; //统计L到R页被看过的次数
  }
  int flag=1;
  for(int i=1;i<=n;i++)
  if(!cnt[i]) //如果第i页一直没有被看过，则一定不可能看完
  {
    flag=0;
    break;
  }
  if(flag) printf("Yes\n");
  else printf("No\n");
  int max_cnt=0,x; //统计最大次数max_cnt，从而求得x
  for(int i=1;i<=n;i++) //从小到大枚举
  if(cnt[i]>max_cnt) //注意这里只能用大于，用大于等于的话就是求最大的x
  {
    max_cnt=cnt[i];
    x=i;
  }
  printf("%d",x);
  return 0;
}
```

C Meguru的最佳决策

题目分析

由题，只需要找到唯一的一个位置，使与该位置在同一行、同一列的所有元素之和取到最大即可。

可以先设置 3 个记录变量，分别记录和的最大值，以及取最大值时所选择的行、列位置（注意：行列位置都是从 1 开始，要将这 3 个记录变量**初始化**为合适的值，才能保证后续遍历的时候各变量正确的更新），然后通过两层 for 循环遍历整个区域的每个位置，对每个位置计算同行同列元素之和，并与当前记录的最大值作比较，若超过最大值则更新记录变量的值，否则不执行操作。最后根据记录变量中保存的值，按照题目要求输出答案即可。

示例代码1

```
#include <stdio.h>

int main() {
    int m, n;
    int mat[410][410];
    scanf("%d%d", &m, &n);
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            scanf("%d", &mat[i][j]);
        }
    }
    int max = 0, choose_i = 0, choose_j = 0;  //记录变量
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            int sum = 0;
            for (int k = 0; k < m; ++k) {
                sum = sum + mat[k][j];
            }
            for (int k = 0; k < n; ++k) {
                sum = sum + mat[i][k];
            }
            sum = sum - mat[i][j];  //mat[i][j]被计入两次，需要减去一次
            if (sum > max) {
                max = sum;
                choose_i = i;
                choose_j = j;
            }
        }
    }
    printf("%d %d %d", choose_i + 1, choose_j + 1, max);
    //choose_i与choose_j中保存的是下标值，从0开始，而题目中的行列位置是从1开始的，因此要记得给结果+1

    return 0;
}
```

然而，三层循环显而易见的比两层循环耗时。虽然本题使用三层循环可以通过，但是我们很有必要思考一下，怎样可以减少循环嵌套层数，提高代码效率。优化的思路可以从减少重复计算的角度入手考虑。

注意到：每次遍历到第 i 行（或第 j 列）的元素时，上面的代码都要把第 i 行（或第 j 列）所有元素的和重新计算一遍，效率是比较低的。

因此，我们可以预先将各行、各列元素的和计算出来存到数组里，这样，当我们遍历到第 i 行第 j 列的元素时，我们可以直接用已经算好的第 i 行元素的和与第 j 列元素的和加起来，再减去第 i 行第 j 列上元素的值（因为在求各行元素和与各列元素和时都计入了一次，要减去一次以免重复），得到的就是与第 i 行第 j 列的元素同行同列的所有元素之和。

示例代码2

```
#include <stdio.h>

int main() {
    int m, n;
    int mat[410][410], sum_row[410] = {0}, sum_column[410] = {0};
    //sum_row和sum_column数组分别待存二维数组每行的和与每列的和
    scanf("%d%d", &m, &n);
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            scanf("%d", &mat[i][j]);
            sum_row[i] = sum_row[i] + mat[i][j];
            sum_column[j] = sum_column[j] + mat[i][j];
            //读入时就计算每行每列的和
            //注意：如果使用这样的写法，直接用sum_row、sum_column两数组中的元素做累加的话，两数组必须先将所有位置的值初始化为0!
        }
    }
    int max = 0, choose_i = 0, choose_j = 0;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (sum_row[i] + sum_column[j] - mat[i][j] > max) { //只做一步运算就可以代替一层for循环，提高了代码效率
                max = sum_row[i] + sum_column[j] - mat[i][j];
                choose_i = i;
                choose_j = j;
            }
        }
    }
    printf("%d %d %d", choose_i + 1, choose_j + 1, max);

    return 0;
}
```

与示例代码 1 相比，示例代码 2 的循环层数要少一层。通过在 OJ 上的测试时间反馈可以看出，示例代码 2 的运行时间比 1 要短很多，效率更高。

D 回转吧！字符串！

题目理解

题意很容易理解，将字符串首尾相接之后，对于从每一个位置开始的一段长度为k的字符串，各判断一次是不是回文串，输出总共有几个是回文串。回文串的定义也很容易理解，就是轴对称的字符串。

题目解析

关于如何判断首尾相接——既可以对数组中的下标取模，也可以直接把字符串复制一份到原字符串后面。回文串判断只需要逐个比对正序和倒序的字符即可（为了简化计算，显然只比较一半的字符就可以验证是否是回文串）。本题没有什么思维难度，但是需要注意细节。

注：strcat不能自己对自己使用，属于未定义行为，在不同平台上会有不同表现，即使你这么使用后AC了，也不代表这样在所有平台都是正确的，请尽量避免未定义行为。

标准代码

```c
#include <stdio.h>
#include <string.h>

int main() {
    char s[2005];
    // 因为要把字符串复制一份到后面，所以要开两倍大小的字符串
    int k;
    scanf("%s%d",s,&k);
    int len = strlen(s);
    for (int i = 0; i < len; i++) {
        s[i+len] = s[i];
        // 将字符串复制一份接在后面
    }
    s[len+len] = '\0';
    // 复制后的字符串没有结尾'\0'，单独补上
    int ans = 0;
    for (int i = 0; i < len; i++) {
    // 遍历每个开始位置
        int good = 1;
        // 记录这个字串是不是回文串
        for (int j = 0; j < k; j++) {
        // 比较该字符串和倒转字符串是否对应位置相同
            if (s[i+j] != s[i+k-j-1]) {
                good = 0;
                break;
                // 只要有一个不等就退出
            }
        }
        ans += good;
    }
    printf("%d", ans);
    return 0;
}
```

E 樱花满地集于我心

题目分析

本题主要考察读入格式的问题，附带扩展了一个小的数学知识点：海伦公式

当我们读完第一行的整数n之后，还有一个换行符\n没有读入

然后我们如果需要读入括号中的数，就要这样写：

```
for(int i=1;i<=n;i++)scanf("(%lf,%lf)",&x[i],&y[i]);
```

用格式字符串中的(,)来读取输入中的这些字符

但是此时没有读入的这个\n会造成影响，它是无法被（匹配的，那么读入就失败了，程序可能在读入循环这里就直接结束

那么为了匹配掉这个\n和其他可能的一些空白字符，我们就要在格式字符串前面加上一个\n或者一个空格

或者，读入还有另外一种方法，适用于整型，我称之为快读，它的工作原理如下：

读取数字字符之前的所有非数字字符，如果读到负号则标记这个数是负数

遇到第一个数字字符之后，将它和后续的数字字符转化成十进制数字

再次遇到非数字字符说明读入结束，返回这个数，具体见代码

代码示例

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
#include<string.h>
#include<time.h>
#define REP(i,a,b) for(register int i=a;i<=b;i++)
#define PER(i,a,b) for(register int i=a;i>=b;i--)
typedef long long LL;
int n,m,A,B,C,opt;
double x[10010],y[10010];
const double eps=1e-4;
const double pi=3.1415926;
double dis(int p,int q){
  return sqrt((x[p]-x[q])*(x[p]-x[q])+(y[p]-y[q])*(y[p]-y[q]));
}
int check(int p,int q,int r){//判断是否三点共线
  double x1=x[p]-x[q],x2=x[q]-x[r];
  double y1=y[p]-y[q],y2=y[q]-y[r];
  return (fabs(x1*y2-x2*y1)<eps);
}
int read(){//读取整数的函数
  int res=0,f=1;//f表示符号
  char c=getchar();
```

```
  while((c>'9')||(c<'0')){if(c=='-')f=-1;c=getchar();}
  while((c>='0')&&(c<='9')){res=res*10+c-'0';c=getchar();}
  return res*f;
}
int main(){
  scanf("%d",&n);
  for(int i=1;i<=n;i++)scanf(" (%lf,%lf)",&x[i],&y[i]);//读入格式这里的空格很关键，没有它就不能正常读入了
  //或者：
  /*  for(int i=1;i<=n;i++){
    int res;
    res=read();
    x[i]=res;
    res=read();
    y[i]=res;
  }*/
  scanf("%d",&m);
  while(m--){
    scanf("%d%d%d%d",&A,&B,&C,&opt);
    if(check(A,B,C)){printf("Yuzuriha Inori\n");continue;}//三点共线
    double a=dis(B,C);
    double b=dis(A,C);
    double c=dis(A,B);
    double p=(a+b+c)/2.0;//海伦公式中的p
    double S=sqrt(p*(p-a)*(p-b)*(p-c));//海伦公式计算面积
    if(opt==1)printf("%.3lf\n",S);
    if(opt==2){//计算内切圆半径
      double r=S/p;
      printf("%.3lf\n",pi*r*r);
    }
  }
}
```

F-璎花的汉诺塔时间

考点	难度
递归，汉诺塔	3

题目分析

经典递归问题之汉诺塔——课本上就有源代码

让我们再回顾其思路

假设有一个 n 层的汉诺塔在左侧的柱子上，要将其移动到右侧的柱子，需要怎么移动最快?

首先，至少，我们需要将最大的，最底下的第 n 圆盘移动到最右侧的柱子上。

由于圆盘只能放在更大的圆盘上，因此为了将第 n 号圆盘移动到最右侧的柱子上，我们需要:

> 第一步，将前 $n-1$ 个圆盘组成的汉诺塔移动到中间的柱子上

随后我们就能移动第 n 号圆盘

> 第二步，将第 n 个圆盘移动到最右侧柱子上

最后，将放在中间柱子上的前 $n-1$ 层圆盘组成的汉诺塔移到最右侧柱子上

> 第三步，将前 $n-1$ 个圆盘组成的汉诺塔移动到右侧柱子上。

第二步可以直接输出，接下来需要处理的就是第一步和第三步，注意到第 n 层圆盘的存在完全不会影响前 $n-1$ 层圆盘的移动，那么第一步和第三步本质就是 $n-1$ 层的汉诺塔的问题。

那么这 $n-1$ 层的汉诺塔问题又可以再一次化归成 $n-2$ 层的汉诺塔，化归成 $n-3$ 层的汉诺塔.....直到变成最基本的情况：只有一个圆盘，直接移动即可。

由此我们就得到了递归关系和初始状态，就能解决整个问题了。

另外，其实输出的字符串存在错别字（毕竟璎花只是个夭折的不识字的水子），希望大伙可以复制题目给出的输入/输出而不是手敲一遍，又慢又容易打错

示例代码

```c
#include <stdio.h>
void Hanoi(int, char, char, char);
int main(void) {
    int n;
    char L, M, R;
    scanf(" %c %c %c %d", &L, &M, &R, &n);
    Hanoi(n, L, M, R);
    return 0;
}
void Hanoi(int n, char L, char M, char R) {
    //将n层的汉诺塔从L柱经过M柱移动到R柱上
    if (n == 1) {
        printf("Eika moved Koishi %02d form the %c to the %c\n", 1, L, R);
        //只有一层的初始状态，直接输出即可
    } else {
        Hanoi(n - 1, L, R, M);
        //第一步，将前n-1层组成的汉诺塔从L柱经过R柱移动到M柱上
        printf("Eika moved Koishi %02d form the %c to the %c\n", n, L, R);
        //第二步，移动第n层圆盘至R柱上
        Hanoi(n - 1, M, L, R);
        //第三步，将前n-1层组成的汉诺塔从M柱经过L柱移动到R柱上
    }
}
```

G 朝田诗乃的朋友圈

题目分析

对于此题，可以直接倒序处理操作，现在假设有两个数组Receive、Send，表示一个人发出和收到了多少朋友圈·。对每个操作，我们有：

1、对于`! x`型：直接 `++Send[x]`

2、对于`- x y`型：`Receive[x]-=Send[y], Receive[y]-=Send[x]`

3、对于`+ x y`型：`Receive[x]+=Send[y], Receive[y]+=Send[x]`

这样做的正确性是不难理解的，在一减一加的过程中，x、y无法看到的朋友圈数量被抵消了，但能看到的朋友圈数量被保留了下来。

代码的时间复杂度为 $O(m)$

示例代码

```
#include<stdio.h>
#include<string.h>
int Send[200004],Receive[200004];
int op[500004],opx[500004],opy[500004];
int main(){
  int n,m;scanf("%d %d",&n,&m);
  for(register int i=1; i<=m; i++){
    char ope[3];
    scanf("%s",ope);
    if(ope[0]=='!'){
      op[i] = 0;
      scanf("%d",&opx[i]);
    }
    else if(ope[0]=='+'){
      op[i] = 1;
      scanf("%d %d",&opx[i],&opy[i]);

    }
    else{
      op[i]=2;
      scanf("%d %d",&opx[i],&opy[i]);
    }
  }
  for(register int i=m; i>=1; i--){
    if(op[i]==0){
      Send[opx[i]]++;
    }
    else{
      if(op[i]==1){
        Receive[opx[i]]+=Send[opy[i]];
        Receive[opy[i]]+=Send[opx[i]];
      }
```

```
      else{
        Receive[opx[i]]-=Send[opy[i]];
        Receive[opy[i]]-=Send[opx[i]];
      }
    }
  }
  for(register int i=1; i<=n; i++) printf("%d ",Receive[i]);puts("");
  return 0;
}
```

H 哪吒的区间合并（1）

题目分析

基本思路是用数组存储记录区间，有两种思路来做这道题，具体思路如下：

思路1

初始化数组为 0，每次读入区间，左端点为数组元素的下标，右端点为数组元素的值，最后遍历数组进行合并。

在读入时，如果存在左端点相同的区间，则取右端点较大者存入数组。

读入结束后遍历数组，在合并时，如果遍历到的数组元素的值为 0，说明没有以该位置为左端点的区间，则应跳过。

如果遇到值不为 0 的数组元素，开始合并该区间：遍历的区间的左端点应该小于等于当前正在合并的区间右端点，每次合并，更新当前正在合并的区间右端点为遍历到的区间右端点和当前正在合并的区间右端点的较大者，遍历到的区间不满足左端点小于当前合并区间右端点时，结束该区间的合并，输出后继续遍历区间。

思路1-示例代码

```
#include <stdio.h>
int range[1000005]; //全局数组初始化均为0
int main() {
  int l, r; //分别表示区间左、右端点
  while(~scanf("%d%d", &l, &r)) //不定组输入
    if(r > range[l]) range[l] = r; //读入时，左端点相同，取右端点较大者
  for(l = 0; l <= 1000000; ++l) {
    if(!range[l]) continue; //数组该位置元素值为0，说明没有以该位置为左端点的区间，跳过
    printf("%d ", l); //输出区间左端点
    for(r = range[l]; l <= r; ++l) //继续遍历数组，合并区间，注意用相同的计数变量，注意循环条件中的r是随着合并过程变化的
      if(range[l] > r) r = range[l]; //更新正在合并的区间右端点
    printf("%d\n", --l); //最后依次++l后不满足l<=r，此时l=r+1，因此应--l，再输出右端点
  }
  return 0;
}
```

思路2

初始化数组为 0，每次读入区间，标记数组下标分别为区间左端点和右端点的元素，最后遍历一遍输出区间。

标记方式可以选择标记左端点为 1，右端点为 -1，由于区间可能存在相同端点，将标记方式改为左端点的元素值自增 1，右端点的元素值自减 1。即，读入之后的数组某位置的元素值表示以该位置为左端点的区间数量减去以该位置为右端点的区间数量。

读入结束后遍历数组，将数组元素值更改为原数组前 i 项值的累和（前缀和），则若在某位值数组元素由 0 变为正数，则说明该位置为合并后区间左端点，在某为值数组元素由正数变为 0 则说明该位置为合并后区间右端点。按输出要求输出即可。

思路2-示例代码

```
#include <stdio.h>
int range[1000005];
int main() {
  int l, r;
  while(~scanf("%d%d", &l, &r)) {
    ++range[l]; //左端点的元素值自增1
    --range[r]; //右端点的元素值自减1
  }
  for(int i = 1; i <= 1000000; ++i) {
    range[i] += range[i - 1]; //更改数组为前i项值的累和
    if(range[i] > 0 && range[i - 1] == 0) printf("%d ", i); //若上一个位置值为0，此位置值大于0，则该位置为合并后区间左端点
    if(range[i] == 0 && range[i - 1] > 0) printf("%d\n", i); //若上一个位置值大于0，此位置值为0，则该位置为合并后区间右端点
  }
  return 0;
}
```

补充

更推荐大家学习理解思路 1。思路 1 的精髓在于用数组下标存储左端点，自动就将所有的左端点排好序了，仅需按照左端点递增顺序即数组遍历顺序根据右端点情况进行合并。这也启示我们，当无法按照这样存储区间时，可以将区间按照左端点递增进行排序，然后根据右端点情况进行合并。

思路 2 很巧妙，用了类似差分和前缀和的方式来记录区间，但是适用范围可能较小。

Author: 哪吒

繁琐的代码

题目分析

本题主要考察字符串的处理。首先建议逐个字符读入，逐个字符处理，因为多行注释可能同时影响多行。剩下就是仔细的分类讨论：

- 单、双引号：因为题目保证匹配，所以一直读入直至匹配。
- 斜杠`/`，判断下一个字符：
  - `/`：单行注释，一直读入至换行，要**保留换行**。
  - `*`：多行注释，一直读入至`*`，判断下一个字符：
    - `/`：多行注释结束。
    - 其他：继续进行判断和读入，仍为注释状态。
  - 其他：输出`/`。
- 其他：直接输出。

示例代码

```c
#include <stdio.h>

char c;
int main() {
  while (~scanf("%c", &c)) {
    if (c == '\"') {
      putchar(c);
      while (~scanf("%c", &c)) {
        putchar(c);
        if (c == '\"')
          break;
      }
    } else if (c == '\'') {
      putchar(c);
      while (~scanf("%c", &c)) {
        putchar(c);
        if (c == '\'')
          break;
      }
    } else if (c == '/') {
      scanf("%c", &c);
      if (c == '/') {
        while (~scanf("%c", &c)) {
          if (c == '\n') {
            putchar(c);
            break;
          }
        }
      } else if (c == '*') {
        while (~scanf("%c", &c)) {
          if (c == '*') {
            scanf("%c", &c);
            if (c == '/')
```

```
              break;
            ungetc(c, stdin);
          }
        }
      } else {
        putchar('/');
        putchar(c);
      }
    } else
      putchar(c);
  }
  return 0;
}
```

代码中使用的`ungetc()`函数，可以将字符输入到缓冲区，以便作为下一次的读入。

J 鼠标坏啦!

题目描述

Jerydeak 的鼠标坏了，但他要输入 n 个字符 `0`，他可以进行以下操作：

- 输入 `0`；
- `Ctrl+A`，进入全选状态；
- `Ctrl+C`，复制全部内容（在全选状态才可以使用）；
- `Ctrl+V`，输入复制的内容；

注意，每次输入时，如果在全选状态，会将用输入**覆盖**当前内容并退出全选状态。

Jerydeak 的初始文档是空白的，请问他最少需要几步操作，才能输入**恰好** n 个 `0`？

题目分析

这个题的关键就是搞清楚状态转移，并以此进行递推。

假设所有状态都建立在**非全选的状态**下，状态为一个二元组：(m, c)，其中 m 表示当前文档字符数，c 表示当前剪切板字符数。题目求的就是 $(n, *)$ 所需要操作次数的最小值。

对于某一个状态，有几种可能转换到下一个状态：

- 直接输入一个 0；
- 直接粘贴当前剪切板内容；
- 全选、复制、粘贴；

示例代码

```
#include<stdio.h>

#define MAXN 1201
#define MAXNN (MAXN * 2 + 5)
```

```c
int dp[MAXNN][MAXNN]; // 第一维表示当前文档字符数，第二维表示剪切板字符数

void initialize() {
    for (int i = 0;i < MAXNN;i++) {
        for (int j = 0;j < MAXNN;j++) {
            dp[i][j] = 0x3f3f3f3f;
        }
    }
    dp[0][0] = 0;
    for (int k = 0;k < MAXN;k++) {
        for (int t = 0;t <= k;t++) {
            int value;
            // 输入一个单字符: (k, t) -> (k + 1, t)
            value = dp[k][t] + 1;
            if (value < dp[k + 1][t]) dp[k + 1][t] = value;
            // 粘贴: (k, t) -> (k + t, t)
            value = dp[k][t] + 1;
            if (value < dp[k + t][t]) dp[k + t][t] = value;
            // Ctrl+A Ctrl+C Ctrl+V: (k, t) -> (k, k)
            value = dp[k][t] + 3;
            if (value < dp[k][k]) dp[k][k] = value;
        }
    }
}

int solve(int n) {
    int minvalue = 0x3f3f3f3f;
    for (int i = 0;i <= n;i++) {
        if (dp[n][i] < minvalue) minvalue = dp[n][i];
    }
    return minvalue;
}

int main() {
    initialize();
    int n;
    while (scanf("%d", &n) != EOF) {
        printf("%d\n", solve(n));
    }
}
```