C5-2021级航类-第五次练习赛题解

A 表决

表决

难度	考点
2	循环

题目解析

本题提供两种示例代码。注意本题是多组数据，每组数据结束后需要将计数变量重新清零。

示例代码 1：使用字符串做法，通过 `strlen` 获取投票总数。

示例代码 2：使用字符做法，由于最后一行行末没有换行符，需要在循环外再补一次判断和输出。

示例代码 1

```
#include<stdio.h>

int main()
{
    char in[200];
    int i,yes;
    while(scanf("%s",in)!=EOF)//按字符串读入
    {
        yes=0;
        for(i=0;i<strlen(in);i++)//循环统计票数
        {
            if(in[i]=='2') yes++;
        }
        if(1.0*yes/strlen(in)>0.5) printf("Pass\n");//注意整数除整数要转成double处理
        else printf("Veto\n");
    }
    return 0;
}
```

示例代码 2

```
#include<stdio.h>

int main()
{
    char in;
    int yes=0,count=0;
    while(scanf("%c",&in)!=EOF)//按字符读入
    {
```

```
            count++;
            if(in=='2') yes++;//统计投票数和赞成数
            if(in=='\n')//遇到换行符判断是否通过
            {
                if(1.0*yes/count>0.5) printf("Pass\n");
                else printf("Veto\n");
                yes=0;
                count=0;
            }
        }
        if(1.0*yes/count>0.5) printf("Pass\n");//最后补一次判断
        else printf("Veto\n");
        return 0;
    }
```

B 呱呱泡蛙一战成硕

难度	考点
2	排序

数据范围不大。普通的排序都可以通过。这里给一个使用了qsort的排序办法。

```
#include<stdio.h>
#include<stdlib.h>

int compare(const void *p1, const void *p2)
{
    int *a=(int *)p1;
    int *b=(int *)p2;
    if(*a>*b)
    {
        return 1;
    }
    else if(*a<*b)
    {
        return -1;
    }
    else
    {
        return 0;
    }
}

int a[1010];

int main()
{
    int n;
    scanf("%d",&n);
    int i;
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
```

```
    qsort(a,n,sizeof(int),compare);
    for(i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }
}
```

C 依法纳税

依法纳税

难度	考点
3	数组

题目解析

本题提供三种示例代码。

示例代码 1：排序法，对输入的两组整数分别排序，再逐一比较。

示例代码 2：线性查找法，对第一行输入的数字在第二行中逐一查找，看看是否存在。注意查找后需要清除第二行被找到的那个数。

示例代码 3：桶排序统计法。

示例代码 1

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

int comp(const void *a,const void *b)
{
    if((*(int*)a)>(*(int*)b)) return 1;
    if((*(int*)a)==(*(int*)b)) return 0;
    if((*(int*)a)<(*(int*)b)) return -1;
}

int main()
{
    int a,b,i,j,ain[305],bin[305];
    double sum=0;
    scanf("%d %d",&a,&b);
    for(i=0;i<a;i++)//输入第一行
    {
        scanf("%d",&ain[i]);
    }
    for(i=0;i<b;i++)//输入第二行
    {
        scanf("%d",&bin[i]);
    }
    qsort(ain,a,sizeof(int),comp);//对ain和bin进行排序
    qsort(bin,b,sizeof(int),comp);
    for(i=0,j=0;i<a;i++)//将排序后的ain元素和bin元素一一对应，按照归并排序的思路处理
```

```
    {
        if(ain[i]==bin[j])
        {
            sum+=ain[i]*0.5;
            j++;
        }
        else sum+=ain[i]*4;
    }
    printf("%.2lf",sum);
    return 0;
}
```

示例代码 2

```
#include<stdio.h>

int main()
{
    int a,b,i,j,ain[305],bin[305],flag=0;
    double sum=0;
    scanf("%d %d",&a,&b);
    for(i=0;i<a;i++)//输入第一行
    {
        scanf("%d",&ain[i]);
    }
    for(i=0;i<b;i++)//输入第二行
    {
        scanf("%d",&bin[i]);
    }
    for(i=0;i<a;i++)//线性查找，查找ain中的每个数是否在bin中出现过
    {
        flag=0;
        for(j=0;j<b;j++)
        {
            if(bin[j]==ain[i])
            {
                sum+=0.5*ain[i];
                flag=1;//找到了
                bin[j]=-1;//因为ain可以重复，避免多个ain找到同一个bin，需要把找到的bin清除
                break;
            }
        }
        if(!flag) sum+=4*ain[i];//没找到
    }
    printf("%.2lf",sum);
    return 0;
}
```

示例代码 3

```
#include<stdio.h>

int main()
{
    int a,b,i,j,ain[105]={0},bin[105]={0};
```

```
    double sum=0;
    scanf("%d %d",&a,&b);
    for(i=0;i<a;i++)//输入第一行
    {
        scanf("%d",&j);
        ain[j]++;
    }
    for(i=0;i<b;i++)//输入第二行
    {
        scanf("%d",&j);
        bin[j]++;
    }
    for(i=1;i<=100;i++)//bin数组保存已补交的次数，ain数组保存已掌握的次数，ain-bin是已掌握但未补交的次数，i是金额
    {
        sum+=bin[i]*0.5*i+(ain[i]-bin[i])*4*i;
    }
    printf("%.2lf",sum);
    return 0;
}
```

D 复矩阵乘法

难度	考点
3	二维数组

这道题逻辑上与实矩阵乘法类似，只不过由于处理的是复矩阵，输入与输出的处理更加繁琐，而存储方式需要增加一个二维数组来存储。
输出上各种情况都需要考虑到位，尤其是虚部为正负一的情况，而数据范围上，虽然每个元素都在int范围内，但矩阵乘法会使数据规模急剧增大，因此需要long long类型进行存储。

```
#include<stdio.h>
long long a1[100][100], b1[100][100],a2[100][100],b2[100][100];
long long c[100][100],d[100][100];
int main(){
    int n=0;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            scanf("%lld%lldi ",&a1[i][j],&b1[i][j]);//输入复数
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            scanf("%lld%lldi ",&a2[i][j],&b2[i][j]);//输入复数
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
        for (int k = 0; k < n; k++) {
                c[i][j] += (a1[i][k] * a2[k][j]-b1[i][k]*b2[k][j]);//复数实部运算
                d[i][j] += (b1[i][k] * a2[k][j]+a1[i][k]*b2[k][j]);//复数虚部运算
            }
        }
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
```

```
            if(c[i][j]==0 && d[i][j]==0){
                printf("0 ");
            }else if(c[i][j]==0){
                if(d[i][j]==1){
                    printf("i ");
                }else if(d[i][j]==-1){
                    printf("-i ");
                }else{
                    printf("%lldi ",d[i][j]);
                }
            }else if(d[i][j]==0){
                printf("%lld ",c[i][j]);
            }else{
                if(d[i][j]==1){
                    printf("%lld+i ",c[i][j]);
                }else if(d[i][j]==-1){
                    printf("%lld-i ",c[i][j]);
                }else{
                    if(d[i][j]>0){
                        printf("%lld+%lldi ",c[i][j],d[i][j]);
                    }else{
                        printf("%lld%lldi ",c[i][j],d[i][j]);
                    }
                }
            }
        }
        printf("\n");
    }
}
```

E The Shy 的二分查找

考点	难度
二分查找	4

问题分析

很基本的二分下界二分查找，利用提示中给的lower_bound函数即可；注意在二分查找中，找到一个值之后不要线性查找第一个出现的，当测试数据中有很多很多个一样的数的时候，这样效率很低，容易超时。

参考代码

```
#include <stdio.h>
#include <stdlib.h>
#define N 1000005
typedef long long ll;
int arr[N];
int lower_bound(int a[],int n,int val)//提示中给的函数
{
    int mid;
    int l=0,r=n;
    while(l<r)
```

```
    {
        mid=(l+r)>>1;
        if(val<=a[mid])
        r=mid;
        else
        l=mid+1;
    }
    return l;
}

int n,m;
int main(void)
{
    scanf("%d%d",&n,&m);
    for(int i=0;i<n;i++)scanf("%d",&arr[i]);
    for(int i=0;i<m;i++)
    {
        int index,key;
        scanf("%d",&key);
        index=lower_bound(arr,n,key);//找到第一个大于key的下标index

        printf("%d\n",arr[index]==key?index+1:-1);//根据arr[index]和key的值来判断是否存在key
    }
    return 0;
}
```

F 方阵取数

难度	考点
4	函数应用

题目分析

本题目涉及了难度更高的函数递归应用，可以选择使用深度优先搜索算法，动态递归求取所取每一组满足题目要求的数之和，在这些值中比较输出最小值即可。

同时本题目也可以选择设出一个二维数组b[m][m]，通过递归的方式以b[i][j]表示该处数据与其上若干行数据相加和的最小值，对b数组的最后一行数据进行比较得到题解。

示例代码1

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int num[12][12];
int book[12][12];
int min,line;
int available(int i,int j) {
    if(book[i-1][j] == 1) return 0;
    if(book[i+1][j] == 1) return 0;
```

```
    if(book[i][j+1] == 1) return 0;
    if(book[i][j-1] == 1) return 0;
    if(book[i+1][j+1] == 1) return 0;
    if(book[i+1][j-1] == 1) return 0;
    if(book[i-1][j+1] == 1) return 0;
    if(book[i-1][j-1] == 1) return 0;
    return 1;
}
void DFS(int col,int i,int sum) {
    if(col == line + 1) {
        if(sum < min) min = sum;
        return;
    }
    if(available(col,i)) {
        if(sum > min) return;

        for(int k = 1; k <= line; k++) {
            book[col][i] = 1;
            DFS(col + 1, k ,sum + num[col][i]);
            book[col][i] = 0;
        }
    }
    return;
}
int main()
{
    int tot;
    scanf("%d",&tot);
    while(tot--) {
        min = 0xffffffff;
        memset(book,0,sizeof(book));
        scanf("%d",&line);
        for(int i = 1; i <= line ; i++)
            for(int j = 1; j <= line ;j++)
                scanf("%d",&num[i][j]);
        for(int i = 1 ; i <= line ; i++)
            DFS(1,i,0);
        printf("%d\n",min);
    }
    return 0;
}
```

示例代码2

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define MAX( x, y ) ( ((x) > (y)) ? (x) : (y) )
#define MIN( x, y ) ( ((x) < (y)) ? (x) : (y) )

int main()
{
    int n, m, i, j, k;
    scanf("%d", &n);
    while (n --)
    {
```

```
        scanf("%d", &m);
        int a[30][30];
        for (i = 0; i < m; i ++)
            for (j = 0; j < m; j ++)
                scanf("%d", &a[i][j]);
        int b[m][m];
        for (i = 0; i < m; i ++)
            b[0][i] = a[0][i];
        for (i = 1; i < m; i ++)
            for (j = 0; j < m; j ++)
            {
                int min = INT_MAX;
                for (k = 0; k < m; k ++)
                    if ((j > k + 1) || j < (k - 1))
                        min = MIN(min, (a[i][j] + b[i - 1][k]));
                b[i][j] = min;
            }
        int min = INT_MAX;
        for (i = 0; i < m; i ++)
            min = MIN(min, b[m - 1][i]);
        printf("%d\n", min);
    }
    return 0;
}
```

G rlx教你写代码（一）

难度	考点
4	字符串

题目解析

本题中表达式所包含的基本元素其实有一个统称：token。根据本题的规定，token的种类可以进一步归纳为：

- 数字：`int`类型常量（无尾缀）；
- 变量：合法命名的变量；
- 运算符：二元运算符：`+ - * / = > < <= <= ==`、一元运算符：`-`和括号：`( )`.

不难发现，不同类型的token，它们的起始的第一个字符一定不一样，因此通过检查第一个字符即可确定token的类型。但是此题有个小难点，即`-`可以是一元或二元运算符，判断的手段是：如果`-`前面是除了`)`之外的运算符，那么它是个一元运算符；否则是个二元运算符。要想实现这一点，就需要我们在读取新的token的时候保存好上一个token，用于辅助判断。整体的解题流程大致如下：

- 第一步：如果表达式中有字符，读取一个表达式中的字符，根据这个字符判断当前token的类型；
- 第二步：读取字符直到当前字符不属于可以构成当前token的字符集合，至此得到新的token；
- 第三步：根据题目要求输出token，如果当前token是`-`，需要根据上一个token来判断是一元还是二元运算符。对于变量类型token，还要检查其长度；
- 第四步：将当前token保存，准备读取下一个token，回到第一步。

示例代码

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

char token[110], old_token[110], s[110], v[110][10];
int type, old_type, i, p, cnt;
// s用于存储表达式，v用于存储长度小于3的变量名
// type: -1->END 0->数字类型 1->变量类型 2->运算符类型 or START
// i为下一个应该在token的哪个位置存储字符
// p为下一个应该读取的表达式中字符的下标（即下一个读取s[p]）
// cnt记录长度小于3的变量名个数

void get_number(void) {

    // 读一个数字类型token并存储进token数组
    while ('0' <= s[p] && s[p] <= '9') {
        token[i++] = s[p++];
    }
    type = 0;
}

void get_variable(void) {

    // 读一个变量类型token并存储进token数组
    while (('0' <= s[p] && s[p] <= '9') || ('a' <= s[p] && s[p] <= 'z') || ('A' <= s[p] && s[p] <= 'Z') || s[p] == '_') {
        token[i++] = s[p++];
    }
    type = 1;
}

void get_operator(void) {

    // 读一个运算符类型token并存储进token数组
    token[i++] = s[p++];
    if (token[0] == '<' || token[0] == '>' || token[0] == '=') {
        if (s[p] == '=') {
            token[i++] = s[p++];
        }
    }

    type = 2;
}

void get_token(void) {

    // 存储上一个token及其类型
    old_type = type;
    strcpy(old_token, token);

    // 跳过空格
    while (s[p] == ' ') {
        p++;
    }

```

```
    // 如果表达式中已经没有字符
    if (s[p] == 0) {
        type = -1;
        return;
    }

    // 判断起始字符来确定当前token类型
    if ('0' <= s[p] && s[p] <= '9') {
        get_number();
    } else if (('a' <= s[p] && s[p] <= 'z') || ('A' <= s[p] && s[p] <= 'Z') || s[p] == '_') {
        get_variable();
    } else {
        get_operator();
    }

    // 添加'\0'
    token[i++] = 0;
}

int main() {

    gets(s);

    old_type = type = 2;

    while (1) {
        i = 0;
        get_token();
        if (type == -1) {
            break;
        }
        if (type == 2) {
            // 括号或一元运算符-的两边不需要输出空格
            if (token[0] == '(' || token[0] == ')' || (token[0] == '-' && old_type == 2 && old_token[0] != ')')) {
                printf("%s", token);
            } else {
                // 二元运算符的两边需要输出空格
                printf(" %s ", token);
            }
        } else {
            // 变量两边不需要输出空格
            printf("%s", token);
        }
        if (type == 1) {
            // token类型是变量时需要检查其长度
            if (strlen(token) < 3) {
                int check = 0;
                for (int j = 0; j < cnt; j++) {
                    if (strcmp(token, v[j]) == 0) {
                        check = 1;
                    }
                }
                if (check == 0) {
                    strcpy(v[cnt++], token);
                }
            }
```

```
        }
    }

    printf("\n");

    for (int j = 0; j < cnt; j++) {
        printf("%s: Is it meaningful?\n", v[j]);
    }

    return 0;
}
```

H 草蛇灰线

难度	考点
5	递归

题目分析

设函数 $match(i,j,k)$ 表示判断以网格的 (i,j) 位置出发，能否搜索到单词 $word[k..]$，其中 $word[k..]$ 表示字符串 $word$ 从第k 个字符开始的后缀子串。如果能搜索到，则返回 `True.`，反之返回 `False.`。函数$match(i,j,k)$ 的执行步骤如下：

如果

$$board[i][j] \neq s[k]$$

，当前字符不匹配，直接返回 $false$。
如果当前已经访问到字符串的末尾，且对应字符依然匹配，此时直接返回 $true$。
否则，遍历当前位置的所有相邻位置。如果从某个相邻位置出发，能够搜索到子串 $word[k+1..]$，则返回 $true$，否则返回$false$。
这样，我们对每一个位置 (i,j) 都调用函数 $match(i,j,0)$ 进行检查：只要有一处返回 $true$，就说明网格中能够找到相应的单词，输出 `True.`，否则说明不能找到输出 `False.`。

为了防止重复遍历相同的位置，需要额外维护一个与 $martrix$ 等大的 $martrix\,flag$ 数组，用于标识每个位置是否被访问过。每次遍历相邻位置时，需要跳过已经被访问的位置。

示例代码

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

int increment[4][2] = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
char str[105];
int martrixflag[105][105];
char martrix[105][105];

void idInitialize()
{
    int i, j;
    for (i = 0; i < 105; i++)
        for (j = 0; j < 105; j++)
```

```
            martrixflag[i][j] = 0;
}
void Input(int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            scanf("%s", &martrix[i][j]);
}
int match(int m, int n, int m_i, int m_j, int str_i, int str_len)
{
    int i, j, k, flag = 0;
    for (i = 0; i < 4; i++)
    {
        if (m_i + increment[i][0] < 0 || m_j + increment[i][1] < 0 || m_i + increment[i][0] >= m || m_j + increment[i][1] >= n)
            continue;
        if (martrix[m_i + increment[i][0]][m_j + increment[i][1]] == str[str_i + 1] && martrixflag[m_i + increment[i][0]][m_j + increment[i][1]] == 0)
        {
            if (str_i + 1 == str_len - 1)
                return 1;
            martrixflag[m_i + increment[i][0]][m_j + increment[i][1]] = 1;
            flag = match(m, n, m_i + increment[i][0], m_j + increment[i][1], str_i + 1, str_len);
            if (flag == 1)
                return 1;
        }
    }
    martrixflag[m_i][m_j] = 0;
    return 0;
}
int match0(int m, int n)//首先寻找到一个首字符相匹配的位置，传输给match函数
{
    int i, j, k, flag = 0;
    int str_len = strlen(str);
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (martrix[i][j] == str[0])
            {
                martrixflag[i][j] = 1;
                if (match(m, n, i, j, 0, str_len))
                {
                    flag = 1;
                    break;
                }
                martrixflag[i][j] = 0;
            }
        }
        if (flag)
            break;
    }
    return flag;
}
int main()
{
```

```
    int m, n, num;
    int i, j, k, str_len, flag;
    scanf("%d%d%d", &m, &n, &num);
    Input(m, n);
    while (num--)
    {
        idInitialize();
        flag = 0;//学习指针知识后可以将flag封装到idinitialize函数中一起初始化
        scanf("%s", str);
        str_len = strlen(str);
        flag = match0(m, n);
        if (flag)
            printf("True.\n");
        else
            printf("False.\n");
    }
    return 0;
}
```

I 繁星若尘

难度	考点
5	二维数组

题目分析

本题的目标比较清晰，即在大图中找到小图的全等图案。

关于图形的表示，可采用`点+向量`的表示方式，例如`(1,1)(1,2)(2,1)(2,2)`表示为点`(1,1)`与向量`(0,1),(1,0),(1,1)`一方面，向量的引入可以让我们在循环遍历时只以一点为基准点；另一方面，本题引入了旋转，向量在旋转的计算中较有优势。

示例代码

```
#include <stdio.h>

int starMap[120][120];

int inMap(int x, int y);
int checkStar(int x0, int y0, int v1x, int v1y, int v2x, int v2y, int v3x, int
v3y);

int main()
{
    int starNum = 0, i = 0, j = 0;
    int x1 = 0, y1 = 0, x2 = 0, y2 = 0, x3 = 0, y3 = 0, x4 = 0, y4 = 0;
    scanf("%d", &starNum);

    for (i = 0; i < starNum; i++) {
        int sx = 0, sy = 0;
        scanf("%d %d", &sx, &sy);
        starMap[sx][sy] = 1;
    }
```

```
19
20      scanf("%d %d %d %d %d %d %d %d", &x1, &y1, &x2, &y2, &x3, &y3, &x4, &y4);
21
22      int vector1_x = x2 - x1;
23      int vector1_y = y2 - y1;
24      int vector2_x = x3 - x1;
25      int vector2_y = y3 - y1;
26      int vector3_x = x4 - x1;
27      int vector3_y = y4 - y1;
28
29      int found = 0;
30
31      for (i = 0; i <= 100; i++) {
32          for (j = 0; j <= 100; j++) {
33              //点是否自己在星图上
34              if (!inMap(i, j)) {
35                  continue;
36              }
37              //原角度
38              if (checkStar(i, j, vector1_x, vector1_y, vector2_x, vector2_y,
    vector3_x, vector3_y)) {
39                  found = 1;
40              }
41              //顺时针90
42              if (checkStar(i, j, vector1_y, -vector1_x, vector2_y, -vector2_x,
    vector3_y, -vector3_x)) {
43                  found = 1;
44              }
45              //顺时针180
46              if (checkStar(i, j, -vector1_x, -vector1_y, -vector2_x, -vector2_y, -
    vector3_x, -vector3_y)) {
47                  found = 1;
48              }
49              //顺时针270
50              if (checkStar(i, j, -vector1_y, vector1_x, -vector2_y, vector2_x, -
    vector3_y, vector3_x)) {
51                  found = 1;
52              }
53
54              if (found) {
55                  break;
56              }
57          }
58
59          if (found) {
60              break;
61          }
62      }
63
64      if (!found) {
65          printf("NOT EXISTS!");
66      }
67
68      return 0;
69  }
70
71  int inMap(int x, int y)
72  {
```

```
    if (x < 0 || x > 100) {
        return 0;
    }

    if (y < 0 || y > 100) {
        return 0;
    }

    if (starMap[x][y] == 1) {
        return 1;
    }

    return 0;
}

int checkStar(int x0, int y0, int v1x, int v1y, int v2x, int v2y, int v3x, int
v3y)
{
    if (inMap(x0, y0) && inMap(x0 + v1x, y0 + v1y) && inMap(x0 + v2x, y0 + v2y)
&& inMap(x0 + v3x, y0 + v3y)) {
        printf("%d %d %d %d %d %d %d %d\n", x0, y0, x0 + v1x, y0 + v1y, x0 + v2x,
y0 + v2y, x0 + v3x, y0 + v3y);
        return 1;
    }

    return 0;
}
```

J zhnの储蓄

难度	考点
6	期望dp

确定一个词，理解好了，这个题就好了。应该怎样理解确定呢：

如果现在钱数不超过T，取一块钱都取不出来了，那么就确定存折里面没有钱了

如果现在钱数为1，取1块钱取出来了，那么就确定存折里面没有钱了（因为所有钱都已经取出来了）

确定一词，是用来确定边界情况的。

所以，这个题的方法就是枚举：枚举所有的可能猜测的值（从1枚举到k），当然，有的取得出来，有的取不出来。

设置如下的动态转移方程：

$dp[i][j]$为现在钱数不超过i，还有j次猜测的情况下的期望

如果假设当前情况下钱数我要取K，那么有两种情况：

第一种是：取得出来：说明钱数上限比K大，那么是由$dp[i-k][j]$转移过来的，概率是$(i-k+1)/(i+1)$。

取不出来：说明钱数上限不足K，那么上限为$K-1$（还是不确定有多少钱），用过了一次猜测的机会，那么是由$dp[k-1][j-1]$转移过来的，概率是$k/(i+1)$。

所以可以得出总的状态转移方程如下：

$dp[i][j]=dp[i-k][j]*(i-k+1)/(i+1)+dp[k-1][j-1]*k/(i+1)+1$

上面等式中第一部分是取的数k不超过真实余额的概率，右面部分是取的存款k超过真实余额的概率。后面的$+1$代表本次操作，所以期望应该$+1$。

k是多少呢？不知道，所以需要枚举

这样看，i，j，k三个变量，三重循环，都是需要枚举的，复杂度是$O(K^2*W)$，会超时

那么，想想，W是不需要那么搞的，因为最好的方法是二分，就可以减去一半的枚举量，也就是说：

最多的枚举次数是：$O(logK)$，K最大是2000，那么j最大是12咯

所以可以用上面的方法打表，而且减小时间复杂度。

```
#include <stdio.h>
const int M=2005;
double dp[2005][15];
double min(double x,double y){
    if(x-y>=1e-8) return y;
    else return x;
}
void init(){
    int sum=1;
    for(int i=1;i<M;i++){
        sum+=i+1;
        dp[i][1]=(double)(sum-1)/(i+1);
    }
    for(int i=1;i<M;i++)
    {
        for(int j=2;j<15;j++)
        {
            dp[i][j]=1e18;
            for(int k=1;k<=i;k++)
                dp[i][j]=min(dp[i][j],dp[i-k][j]*(i-k+1)/(i+1)+dp[k-1][j-1]*k/(i+1)+1);
        }
    }
}
int main()
{
    int k,w;
    init();
    while(scanf("%d%d",&k,&w)!=EOF)
    {
        w=min(12,w);
        printf("%.6lf\n",dp[k][w]);
    }
    return 0;
}

```

*ps:*本题出题的时候没有考虑到大家没有系统的学过动态规划，所以本题可能给大家带来了不是很良好的体验，毒瘤出题人在这里表示歉意，之后出题会注意的*orz*