一、问题要求

【题目大意】
　　桌子上有一个m行n列的方格矩阵，将每个方格用坐标表示，行坐标从下到上依次递增，列坐标从左至右依次递增，左下角方格的坐标为(1,1)，则右上角方格的坐标为(m,n)且（0<m+n<=20）。

　　小明是个调皮的孩子，一天他捉来一只蚂蚁，不小心把蚂蚁的右脚弄伤了，于是蚂蚁只能向上或向右移动。小明把这只蚂蚁放在左下角的方格中，蚂蚁从左下角的方格中移动到右上角的方格中，每步移动一个方格。蚂蚁始终在方格矩阵内移动，请计算出不同的移动路线的数目。

【输入格式】
每一组第一行有两个数n、m表示m行n列。

【输出格式】
不同的移动路线的数目

例如
　　输入 10 10
　　输出 48620

二、问题分析

* 问题就描述为蚂蚁可以向上走或者向右走两种选择，所以自然而然想到
 1.一个格子可以由其左边格子走过来
 2.也可以由其下边格子走过来
* 然后就可以使用动态规划进行状态转移
注意：
* 该问题抽象出来是一个数学问题，可以用排列组合做
* 也可以用动态规划做，本实验报告是用的动态规划做

三、数据结构和算法设计

物理数据对象的设计

物理数据对象就是数组就足以存储所有数据
但如果想适应更大的数据就用long long int
　　开一个dp数组
```
const int maxn = 1000;
int dp[maxn][maxn];
```

算法思想的设计

```
【关键想法】
    使用动态规划
    定义dp[i][j]为蚂蚁走到第i行j列的路线数目
    状态转移方程为 dp[i][j] = dp[i-1][j] + dp[i][j-1] 即为其下和其左的路线之和

【注意点】
    动态规划开数组注意点就是下标要对应，还有初始化边界问题
    初始化dp[1][i] = 1 ( 1 =< i <= n n为列数)
         dp[j][1] = 1 ( 1 =< j <= m m为行数)
    我采用的编程下标是一一对应，所以注意数组要至少多开一位
```

关键功能的算法步骤

```
// m ：  为行数
// n ：  为列数
void count(int m,int n)
{
    //初始化
    for(int i = 1 ; i <= m ; i ++)
    {
        dp[i][1] = 1;
    }
    for(int i = 1 ; i <= n ; i ++)
    {
        dp[1][i] = 1;
    }
    //进行计算
    for(int i = 2 ; i <= m ; i ++)
    {
        for(int j = 2 ; j <= n ; j ++)
        {
            //状态转移方程
            dp[i][j] = dp[i-1][j] + dp[i][j-1];
        }
    }
}
```

三、算法性能分析

本算法主要空间消耗在开的数组中。

空间复杂度为 $O(n_2)$

时间主要消耗在状态转移中

时间复杂度为 $T(n^2)$

四、算法改进

改进方法一

该算法可以用组合数学做

即C（m,m+n）,进行实现即可

空间复杂度为O（1），时间复杂度为T（n^2）

改进方法二

滚动数组

可以将动态规划做的方法开的数组降为O(n)

因为每次计算只用到了其左边和上面的数据，而且其上面的数据在没更新之前还是之前的数据

```
int main()
{
    int m = 0;
    int n = 0;
    cin>>m>>n;
    long long int * dp = new long long int [n + 1];
    fill(dp,dp + n + 1,1);
    for(int i = 2; i <=m ; i++)
    {
        for(int j = 2; j <= n ; j ++ )
        {
            dp[j] = dp[j] + dp[j-1];
        }
    }
    cout<<dp[n];
}
```

时间复杂度为O（n^2）

空间复杂度为O（n）

五、日志

本道题还算比较简单，一眼看去就有几种方法，但是还是有可以优化的地方，动态规划的最简单的题莫过于此