# 2022秋-算法设计与分析

# 马的周游问题

|  |  |
| --- | --- |
| 姓名： |  |
| 学号： |  |

|  |
| --- |
| 2022年12月23日 |

目录

[2022秋-算法设计与分析 1](#_Toc124199689)

[马的周游问题 1](#_Toc124199690)

[1. 实验要求 2](#_Toc124199691)

[1.1 问题说明 2](#_Toc124199692)

[1.2 相关要求 2](#_Toc124199693)

[2. 实验报告 3](#_Toc124199694)

[2.1 分治法解决马的周游问题 3](#_Toc124199695)

[2.1.1 问题重述 3](#_Toc124199696)

[2.1.2 输入处理 3](#_Toc124199697)

[2.1.3 输出处理 4](#_Toc124199698)

[2.1.4 算法实现 4](#_Toc124199699)

[2.1.5 结果展示 8](#_Toc124199700)

[2.2 分治法时间复杂度分析 9](#_Toc124199701)

[2.2.1 时间复杂度分析 9](#_Toc124199702)

[2.2.2 时间复杂度验证 9](#_Toc124199703)

[2.3 程序编译运行指导 9](#_Toc124199704)

[2.3.1 程序编译指导 9](#_Toc124199705)

[2.3.2 程序运行指导 10](#_Toc124199706)

# 1. 实验要求

## 1.1 问题说明

1. 国际象棋的棋盘为8\*8的方格棋盘，现将“马”放在任意指定的方格中，按照“马”走棋的规则（与中国象棋规则一样，马走“日”字）将“马”进行移动。要求每个方格只能进入一次，最终使得“马”走遍棋盘64个方格，回到起点。

2. 编写代码，实现对棋盘的马踏棋盘操作，给定初始位置，用数字给出“马”移动的路径并格式化输出。

3. 必须实现的算法：分治法

## 1.2 相关要求

1. 要求提交程序和实验报告，打包成“学号-姓名-程序语言.zip”（如“12345678-张三-c.zip”、“18061234-李四-java.zip”、c/c++语言统一用“c”表示）提交。提交前把Debug, Release等编译文件删除，只保留代码源文件。

1）程序：不限语言，但不得用封装好的算法函数直接求解；一般不得依赖非标准库（c++可以使用stl、boost库），特殊情况需在实验报告中予以说明。

2）实验报告：解题思路；所实现算法的时间复杂度分析（结合程序统计关键步骤运行次数，以验证分析结果）；程序运行指导，包括程序编译说明、输入输出示例等。如果输入、输出信息较多，建议采用文件进行格式化输入、输出。实验报告命名方式为“实验报告.docx”。

3）一题多解：在实现要求的算法基础上，鼓励一题多解，可酌情加分。即在完成必须算法的基础上，用其他算法求解，请在实验报告中进行分析说明。

2. 评分说明：程序和实验报告等重要，综合得到本题分数，评分要点如下：

1）程序：程序正确性（至少实现对n\*n规格棋盘的处理）、注释完整性、关键函数的通用性、程序使用的方便性、边界处理。

2）实验报告：解题思路正确性与描述清晰程度、时间复杂度分析的正确性与完整性、运行指导的质量。

# 2. 实验报告

## 2.1 分治法解决马的周游问题

### **2.1.1 问题重述**

国际象棋的棋盘为8\*8的方格棋盘，现将“马”放在任意指定的方格中，按照“马”走棋的规则（与中国象棋规则一样，马走“日”字）将“马”进行移动。要求每个方格只能进入一次，最终使得“马”走遍棋盘64个方格，回到起点。

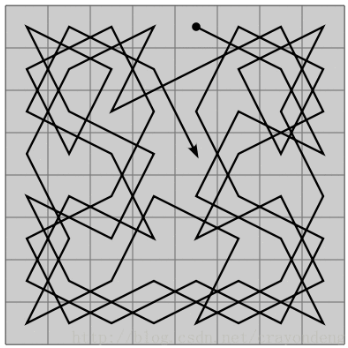


图1：马的周游问题示例

这个问题其实可以进行推广，棋盘大小可以是m\*n，满足：

① m≥6，n≥6；② m，n都是偶数；③ |m-n|≤2。

因此，为实现算法的通用性，该问题输入为棋盘大小m和n，输出为用数字表示的马的周游路线。

### **2.1.2 输入处理**

在main()函数中进行问题的输入。编写函数isStandard()来判断用户输入是否符合棋盘大小的3个标准，该函数接收用户输入宽度m、长度n、起始点横坐标sx和起始点纵坐标sy为参数，进行判断后，返回判断结果：

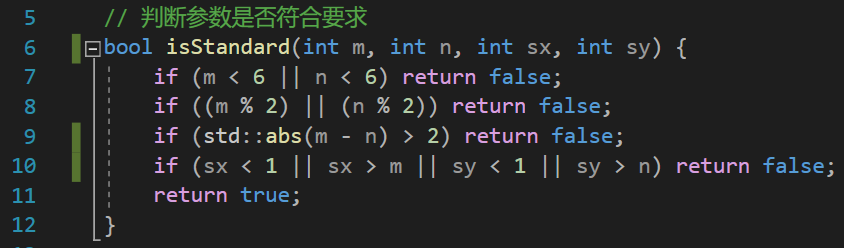


图2：isStandard()函数是实现

用户每次输入后，调用isStandard()函数判断输入是否符合标准，如果不符合，则需要再次输入，直到输入的参数符合标准为止。

### **2.1.3 输出处理**

为显示直观清晰的马的周游路线，输出为m\*n二维矩阵形式，矩阵中的每个元素代表马运动到此处所走过的步长，起始点步长为1，终点步长为m\*n。

由于数字位数不同，因此进行格式化输出，先计算出最大数字m\*n的位数，将其加1，作为所有数字输出时的宽度：

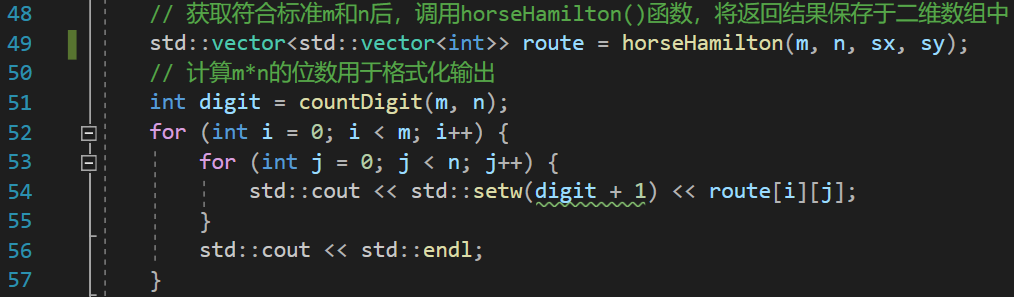


图3：输出格式处理

### **2.1.4 算法实现**

分治法解决马的周游问题的过程是，将大的棋盘分解为4个小的棋盘，每个小棋盘继续划分为4个小棋盘，每次划分时，确保棋盘长和宽为偶数且绝对值之差不大于2。

采用递归进行分治，递归的停止条件是，长和宽或宽和长为(6, 6)、(6, 8)、(8, 8)、(8, 10)、(10, 10)、(10, 12)中的一种即可，原因如下：首先，棋盘长和宽必须满足① m≥6，n≥6；② m，n都是偶数；③ |m-n|≤2。因此，棋盘长或宽最小为6，最小的可能是6\*6，那么，只要罗列出[6\*6, 12\*12)中的所有可能情况，即可用它们的组合表示出任意满足标准的m\*n的棋盘。

[6\*6, 12\*12)共有6种情况，用三维数组表示这6种棋盘的可行解。以6\*6棋盘为例，表示如下：

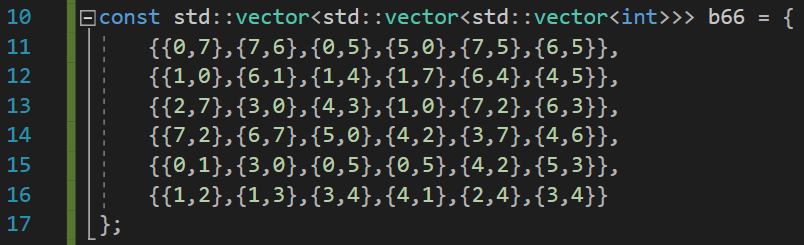


图4：6\*6棋盘的表示

其中，b66[i][j][0]和b66[i][j][1]为范围为0-7的整数，代表的是在(i, j)位置上的马，下一跳可以跳到(i+direct[b66[i][j][0]][0], j+direct b66[i][j][0][1])位置，或(i+direct[b66[i][j][1]][0], j+direct b66[i][j][1][1])位置。direct是二维数组，由8个一维数组组成，每个一维数组有两个元素，用以代表跳跃的方向。

route数组以数字表示步长的方式记录马的周游路线，board数组代表棋盘，以记录方向的方式记录马的周游路线。进入horseHamilton()函数后，首先将route数组和board数组初始化，然后调用divideConquer()分治函数，修改board数组，最后根据board数组修改route数组后，将route数组作为函数返回值返回：

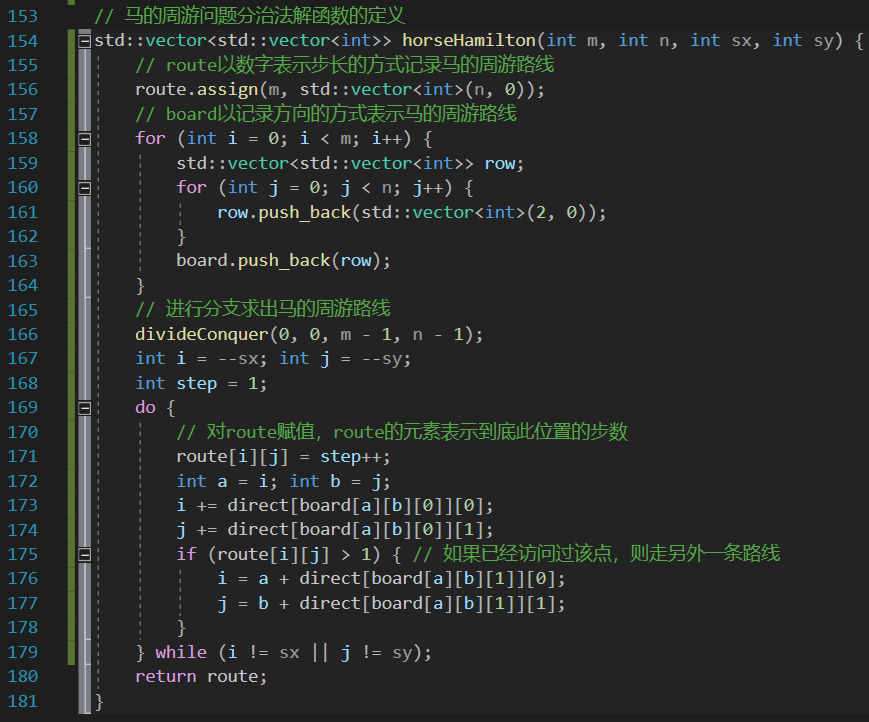


图5：horseHamilton()函数实现

divideConquer()是一个递归函数，首先计算出当前棋盘的长宽以及需要划分的棋盘的长宽；然后，根据长宽之间的关系确定是否对数组进行“转置”，保持长大于宽；再然后，判断是否满足递归停止条件，如果满足则调用fillBoard()函数填充棋盘后返回；最后，进行4个小棋盘的递归，之后进行合并即可：

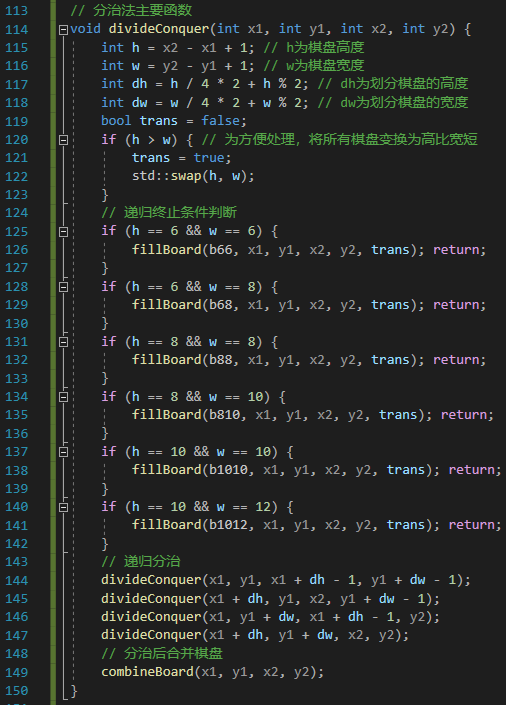


图6：divideConquer()函数实现

fillBoard()函数用于填充棋盘，即改变board数组，使用预定义好的6个可行解来修改board数组对应大小的部分。根据是否需要“转置”来确定赋值的顺序和方式，整体实现较为简单，如下图所示：

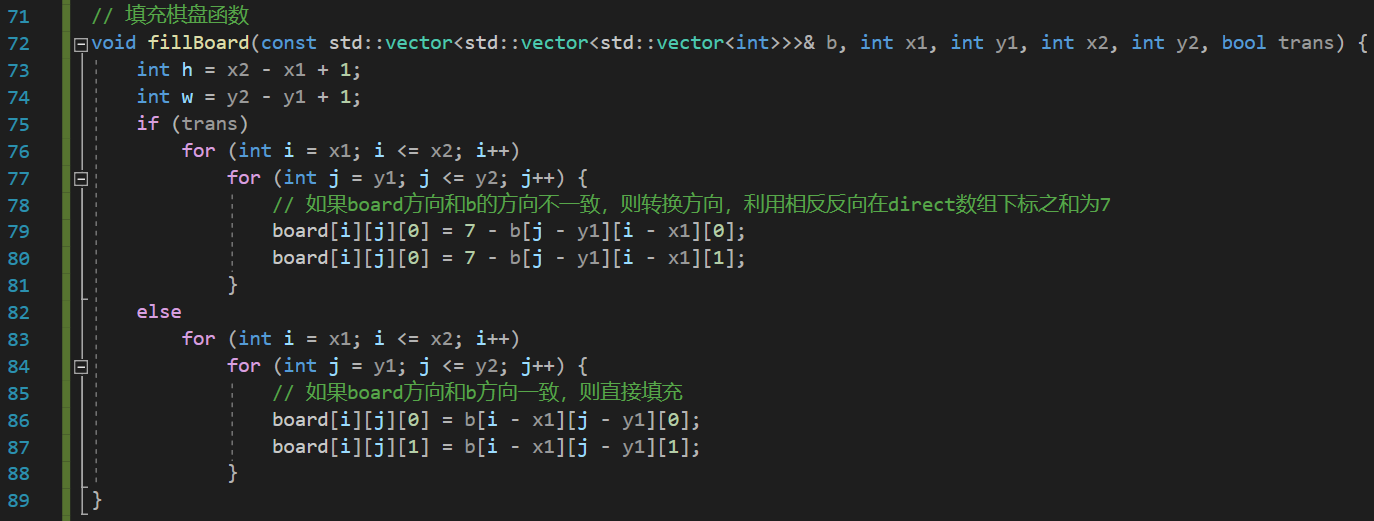
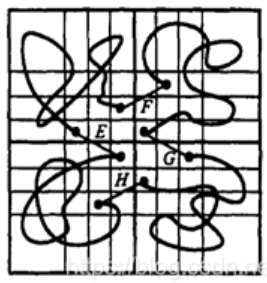
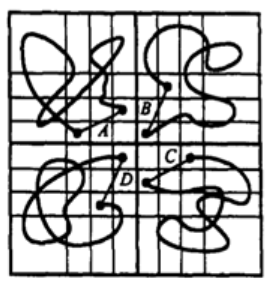


图7：fillBoard()函数实现

combineBoard()函数是整个分治法的关键部分。分治法将棋盘划分为4个部分求出解后，情况如下图所示：



(a) (b)

图8：棋盘合并示意图

在8(a)中，对于左下棋盘的右上角点和右上棋盘的左下角点而言，无论周游路线如何，一定存在B和D两条路径，因此，只需针对左上棋盘和右下棋盘，构造出含有形如A和C路径的周游路线解，即可转变为8(b)所示情况。所以，预定义的6个可行解均满足8(a)所示情形，以保证可以成功转换为8(b)。

combineBoard()函数实现了从8(a)到8(b)的转换过程，即断开原有的4条边，连接新的4条边，具体实现如下图：

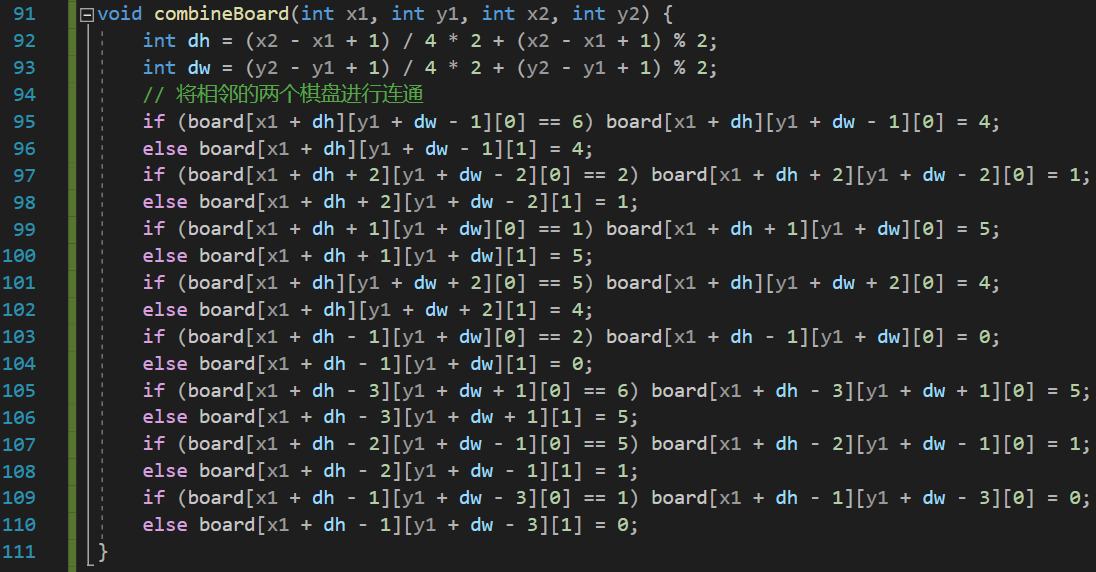


图9：combineBoard()函数实现

### **2.1.5 结果展示**

各函数逻辑实现完毕后，即可编译运行，输入棋盘宽度m和长度n，以及起始点sx和sy的坐标，运行结果如下图：

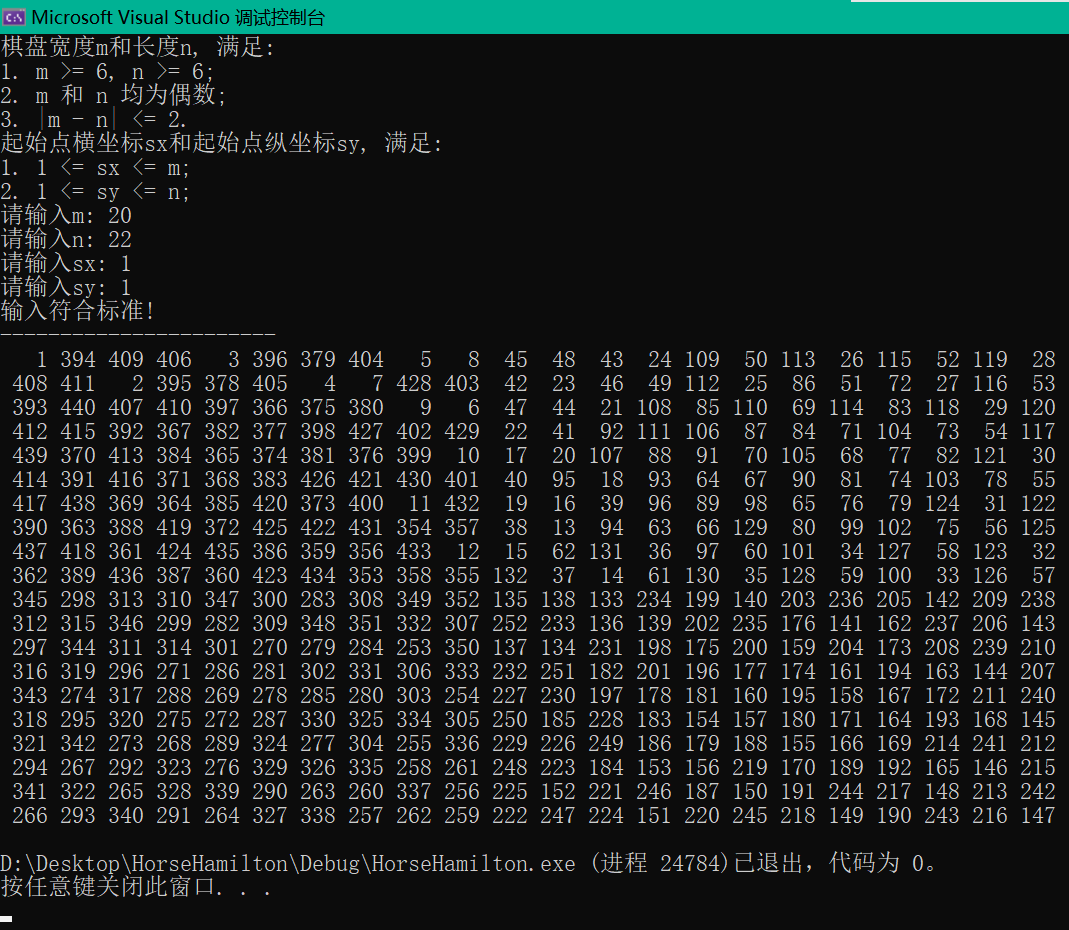


图10：运行结果示例

## 2.2 分治法时间复杂度分析

### **2.2.1 时间复杂度分析**

该算法时间复杂度分为两部分，分别是算法执行时间复杂度和输出时间复杂度。首先是算法执行时间复杂度，分治过程中划分了4个区域，规模下降为n/2，由主定理可知，k=4，m=2，d=1，显然，k>md，时间复杂度O(nlog2(4))=O(n2)。其次是输出时间复杂度，输出时，将棋盘遍历一遍，以确定每个位置的步数，所以时间复杂度为O(m\*n)，由于m与n之差的绝对值不超过2，因此时间复杂度近似为O(n2)。综上来看，该算法时间复杂度为O(n2)。

### **2.2.2 时间复杂度验证**

对函数插桩，在输入结束后，算法执行前设置start\_time，在算法执行后设置end\_time，相减后输出。改变问题的规模并统计时间，结果如下图所示：

图11：时间复杂度验证

显然，运行时间随着输入规模的增大呈指数增长。

## 2.3 程序编译运行指导

### **2.3.1 程序编译指导**

使用g++编译器，在终端输入以下命令：g++ main.cpp horseHamilton.cpp -o main，生成可执行程序main。

### **2.3.2 程序运行指导**

在终端输入命令./main即可运行程序，输入宽度m、长度n、起始点横纵坐标sx和sy，即可看到运行结果：

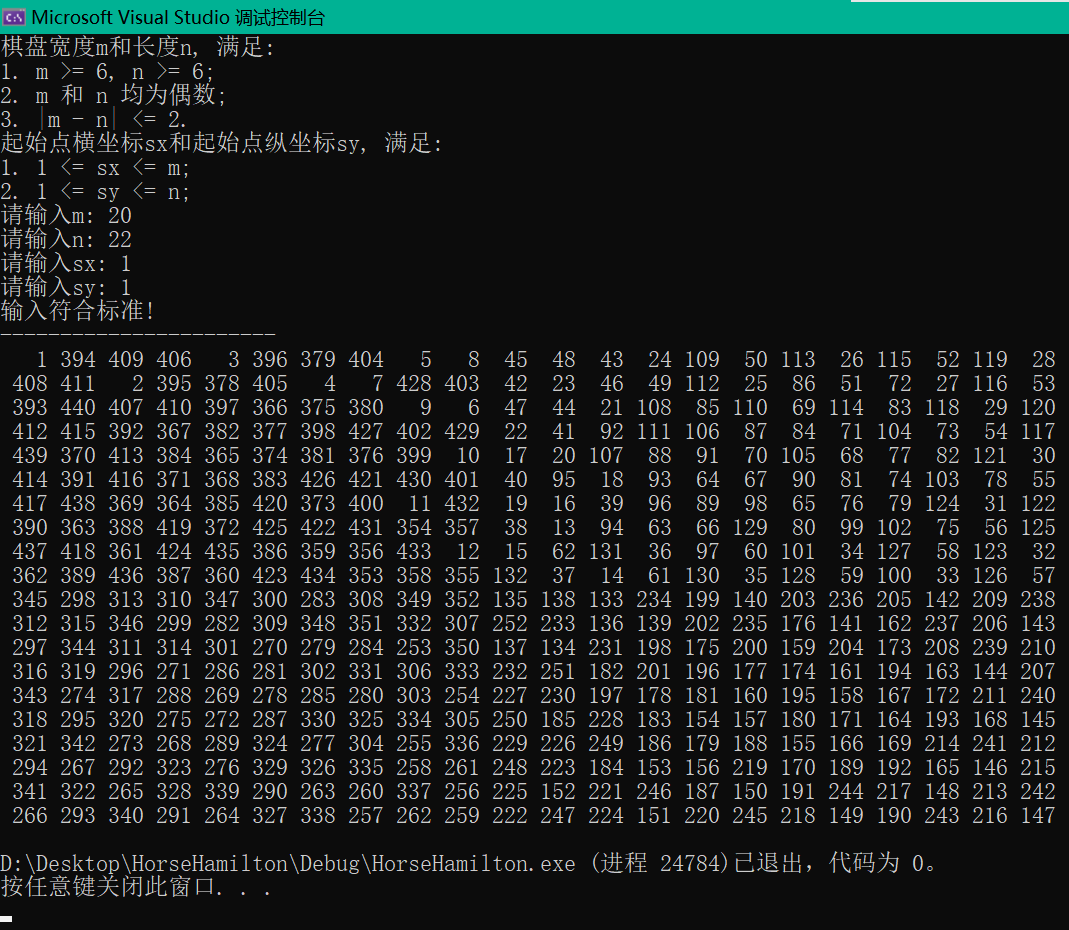


图12：程序运行示例