**实习报告 2.4**

**马踏棋盘**

陈彦帆 2018K8009918002

李国峰 2018K8009922027

一、需求分析

1.以二维数组board[SIZE][SIZE]表示棋盘，其中board[i][j]表示棋盘的[i][j]位置上马经过的顺序，若尚未经过则为0。

2.程序开始时，用户输入一个起始位置，如”3 5”，程序计算从该位置起始的马踏棋盘路径。程序给出多条路径，用户每次可选择一条打印查看，并显示出该路径的回溯次数。

3.本题要求的SIZE为8，经设计，只需在源代码宏定义部分简单修改SIZE，可实现16路以内棋盘的游览路径查找。

4.题目要求采用非递归的回溯法，故需要用到栈结构。

5.8路马踏棋盘的暴力算法耗时过长，本程序主要采用Warnsdorff's rule算法。

6.本程序求出的大部分路径回溯次数为0，出于减少交互复杂度等考虑，演示回溯路径的板块不加入用户可执行程序中，而在报告和课堂展示中额外给出。

二、概要设计

**1.栈的数据类型定义**

typedef struct SqStack{

int sp; //栈指针

int stacksize;//栈当前的最大容量

ElemType \* base; //栈底指针

}Stack;

**基本操作：**

Stack \* InitStack(void)

创建一个栈并返回。

Status GetTop(Stack \*S,ElemType \*e)

获取栈顶元素，以指针的方式传到\*e。成功返回True。

Status Push(Stack \*S,ElemType e)

元素入栈；初始条件为栈未满。经过设计，所需栈的最大大小为4096字节，故栈无需扩容，也无需判断栈是否满。成功返回True。

Status Pop(Stack \*S, ElemType \*e)

元素出栈。初始条件为栈未空。在本程序中，其判断由调用者进行，以避免重复判空的浪费。成功返回True。

**2.棋盘与棋子的类型定义**

typedef unsigned char Board[SIZE][SIZE];

//棋盘类型定义

typedef struct pos{

char x;//坐标x

char y;//坐标y

char dir;//当前棋子到下一个棋子的方向

char cnt;//当前棋子的最佳路径数

char seq[8];//当前棋子最佳路径的顺序

}Pos; //棋子类型定义

**基本操作：**

void display(void)

打印当前棋盘。

void displayBoard(int cnt)

打印历史棋盘，cnt为棋盘号

inline void Undo(Stack \* S)

回撤一步。

Status leap(Stack \* S)

进行一次马行走。

Status isnOut(int i)

判断本次方向是否超出边界。

void SortSeq(Stack \* S)

对当前结点的行走顺序进行生成并排序。

**3.求解多条路径的算法描述**

（1）初始化棋盘和栈；

（2）选择初始位置并落子。

（3）若当前位置未生成下一步路径顺序，则生成下一步路径顺序。采用Warnsdorff's rule，即取下一步行动数最少的结点为最先。

（4）否则，按当前方向跳跃，回到（3）

（5）若（4）无法跳跃，需要回溯。出栈至下一个未被穷举过的局面，方向数+1。

（6）若回溯到初始状态，则未找到路径，返回当前找到的路径数。

（7）若跳跃至步数为SIZE\*SIZE，则踏遍棋盘。保存当前棋盘并回溯到上一个关键点，方向数+1继续寻找。（关键点：下一步行动数最少的结点不止一个）

（8）若总回溯步数过长，则运行时间过长，退出寻找。

三、详细设计

1，对下一步的方向进行排序。其中下一步行动点最少的方向排在最前。

void SortSeq(Stack \* S)

{//对下一步的方向进行排序

int num[8] = {0};//初始化下一步可行动点数组。

for(int i=0;i<8;i++){

pos.seq[i] = i;

pos.dir = i;

if(leap(S)==False){//按自然顺序进行跳跃。若无法跳跃，可行动点设为9（越界值）

num[i] = 9;

continue;

}

for(int j=0;j<8;j++){

char nextx = pos.x + way\_x[j];

char nexty = pos.y + way\_y[j];

if(isnOut(nextx)&&isnOut(nexty)&&board[nexty][nextx]==0)

num[i]++; //计算下一步的可行动点数

}

if(!num[i] && S->sp<maxstep)

num[i] = 9; //若下一步可行动点数为0，把他设为9，方便排序。

Undo(S); //返回这一步。

}

pos.dir = 0;//初始化方向

pos.cnt = 0;//初始化最佳步数个数

int min0;

for(int i=0;i<8;i++){

int min = 9;

int tar = 0;

for(int j=0;j<8;j++){

if(num[j]<min){

min = num[j];

tar = j;

}

}//采用稳定的选择排序

num[tar] = 10;

if(!i)

min0=min;

if(min==9)

tar = 0xf;//此路不通

pos.seq[i] = tar;

if(min == min0 && min<9)

pos.cnt++;

}

}

2，非递归搜索函数

int Search(Stack \* S)

{

Again:

while(S->sp>=step && S->sp<maxstep){

if(!pos.dir)

SortSeq(S);//首次经过先排序

else if(pos.cnt>=0 && pos.dir>=pos.cnt){

Undo(S);//方向走完就回撤

backcnt++;

if(totalback+backcnt>600000)//回撤过多就返回

return history\_cnt;

pos.dir++;

continue;

}

if(!leap(S)){//前有路径就跳跃

pos.dir++;

continue;

}

}

if(S->sp<step)//回到原点就完成

return history\_cnt;

else {

if(history\_cnt>=MAXHISTORY)

return history\_cnt;//超过容量也返回

memcpy(history[history\_cnt],board,sizeof(board));//复制成功的棋盘

backtrace[history\_cnt] = backcnt;

history\_cnt++;

totalback += backcnt;//更新总回撤数

backcnt = 0;//初始化当前走法回撤数

do{

Undo(S);

pos.dir++;

}while(S->sp>=step&&pos.cnt<=pos.dir);//回撤到上一个分歧点

goto Again;//搜索下一条路径

}

}

3，main函数，提供交互

int main()

{

int x,y,cnt;

Stack \* S = InitStack();

clock\_t clockstart, clockend;

printf("Welcome to Knight's journey DEMO.\n");

while(1){

Init(S);

displayBoard(-1);//打印空棋盘

printf("Please enter the initial position(1-%d 1-%d):\n",SIZE,SIZE);

printf("For example:\"1 1\"\n");

scanf("%d%d",&x,&y);//读入初始位置

if(isnOut(--x)&&isnOut(--y)){

pos.y=y;

pos.x=x;

pos.dir=0;

board[y][x]=1;

Push(S,pos);//进行第一步落子

clockstart = clock();//统计运行时间

Search(S);

clockend = clock();

printf("%d paths found,enter a path number to view(0-%d)\nenter -1 to rechoose initial position\n",history\_cnt,history\_cnt-1);

printf("TIME:%.2fs\n", (double)(clockend - clockstart) / CLOCKS\_PER\_SEC);

while(scanf("%d",&cnt)==1){

if(cnt==-1)

break;

else if(cnt>=0&&cnt<history\_cnt)

displayBoard(cnt);//根据输入号数打印对应的路径

printf("%d paths found,enter a path number to view(0-%d)\nenter -1 to rechoose initial position\n",history\_cnt,history\_cnt-1);

}

}

}

四、调试分析

**1，求出多条路径**

更改main函数，检验全部64个点的搜索情况。测试结果如下：

x=1,y=1 137300 paths found TIME:0.19s

x=1,y=2 205491 paths found TIME:0.26s

x=1,y=3 14570 paths found TIME:0.02s

x=1,y=4 52469 paths found TIME:0.06s

x=1,y=5 52469 paths found TIME:0.06s

x=1,y=6 14570 paths found TIME:0.02s

x=1,y=7 205491 paths found TIME:0.24s

x=1,y=8 137300 paths found TIME:0.16s

x=2,y=1 205491 paths found TIME:0.24s

x=2,y=2 69388 paths found TIME:0.09s

x=2,y=3 84653 paths found TIME:0.11s

x=2,y=4 148423 paths found TIME:0.19s

x=2,y=5 148423 paths found TIME:0.22s

x=2,y=6 84653 paths found TIME:0.11s

x=2,y=7 69388 paths found TIME:0.09s

x=2,y=8 205491 paths found TIME:0.24s

x=3,y=1 14570 paths found TIME:0.02s

x=3,y=2 84653 paths found TIME:0.11s

x=3,y=3 60488 paths found TIME:0.08s

x=3,y=4 92424 paths found TIME:0.12s

x=3,y=5 92424 paths found TIME:0.12s

x=3,y=6 60488 paths found TIME:0.08s

x=3,y=7 84653 paths found TIME:0.11s

x=3,y=8 14570 paths found TIME:0.02s

x=4,y=1 52469 paths found TIME:0.06s

x=4,y=2 148423 paths found TIME:0.19s

x=4,y=3 92424 paths found TIME:0.13s

x=4,y=4 510410 paths found TIME:0.69s

x=4,y=5 510410 paths found TIME:0.68s

x=4,y=6 92424 paths found TIME:0.12s

x=4,y=7 148423 paths found TIME:0.19s

x=4,y=8 52469 paths found TIME:0.06s

x=5,y=1 52469 paths found TIME:0.06s

x=5,y=2 148423 paths found TIME:0.19s

x=5,y=3 92424 paths found TIME:0.12s

x=5,y=4 510410 paths found TIME:0.67s

x=5,y=5 510410 paths found TIME:0.67s

x=5,y=6 92424 paths found TIME:0.13s

x=5,y=7 148423 paths found TIME:0.19s

x=5,y=8 52469 paths found TIME:0.06s

x=6,y=1 14570 paths found TIME:0.02s

x=6,y=2 84653 paths found TIME:0.11s

x=6,y=3 60488 paths found TIME:0.08s

x=6,y=4 92424 paths found TIME:0.13s

x=6,y=5 92424 paths found TIME:0.12s

x=6,y=6 60488 paths found TIME:0.08s

x=6,y=7 84653 paths found TIME:0.11s

x=6,y=8 14570 paths found TIME:0.02s

x=7,y=1 205491 paths found TIME:0.24s

x=7,y=2 69388 paths found TIME:0.09s

x=7,y=3 84653 paths found TIME:0.11s

x=7,y=4 148423 paths found TIME:0.19s

x=7,y=5 148423 paths found TIME:0.19s

x=7,y=6 84653 paths found TIME:0.11s

x=7,y=7 69388 paths found TIME:0.09s

x=7,y=8 205491 paths found TIME:0.24s

x=8,y=1 137300 paths found TIME:0.15s

x=8,y=2 205491 paths found TIME:0.24s

x=8,y=3 14570 paths found TIME:0.02s

x=8,y=4 52469 paths found TIME:0.06s

x=8,y=5 52469 paths found TIME:0.06s

x=8,y=6 14570 paths found TIME:0.02s

x=8,y=7 205491 paths found TIME:0.24s

x=8,y=8 137300 paths found TIME:0.15s

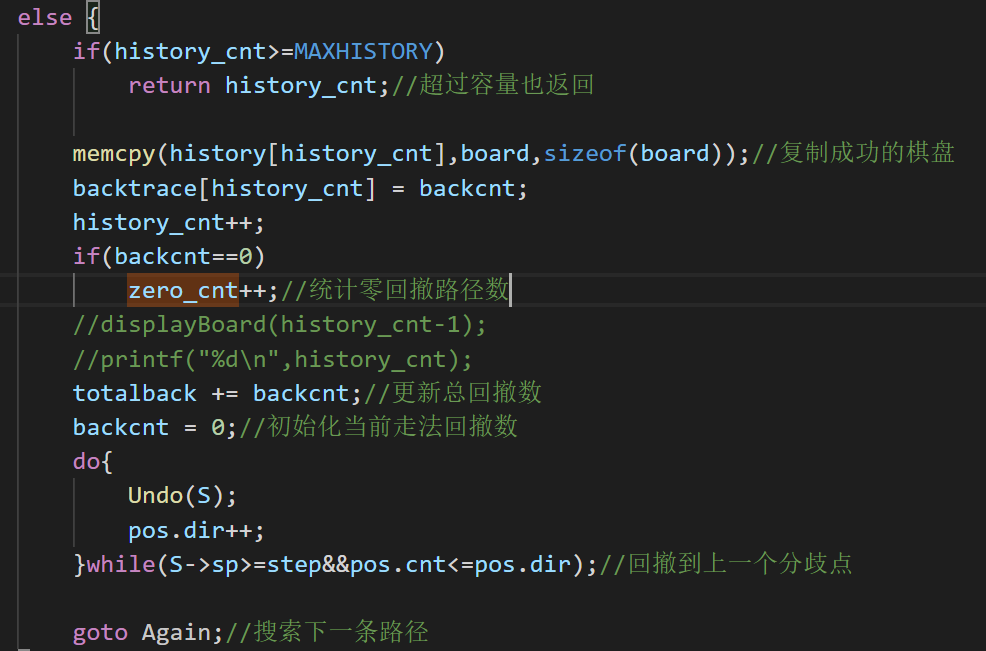
可见，本程序在8\*8棋盘中可求出的路径条数从14570-510410不等，视初始位置而定。

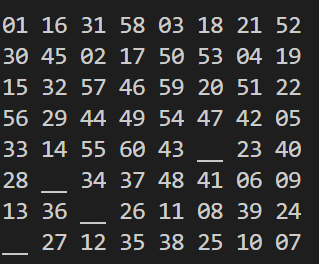
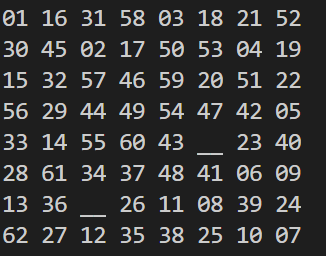
在笔记本电脑上的运行时间从0.02秒到0.68秒，可以令人满意。

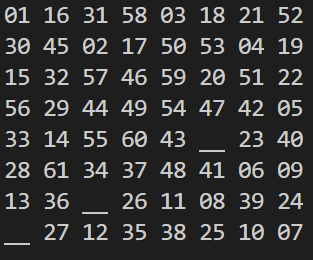
**2.减少回撤次数**

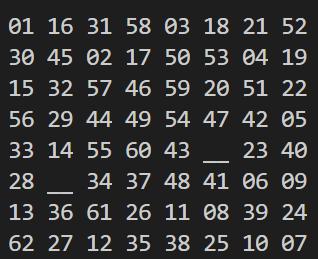
本程序在试探时采用Warnsdorff's rule，即下一步行动点最少的方向排在最前。可以有效减少回撤步数

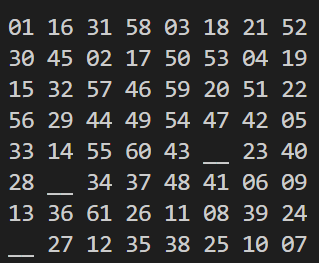
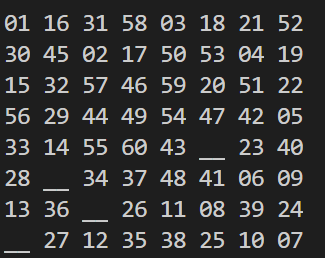
简单修改代码，打印出总条数和0回撤的条数。

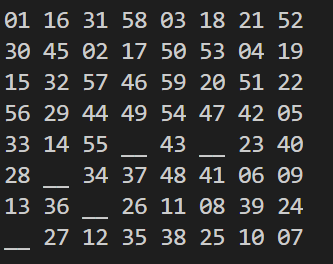
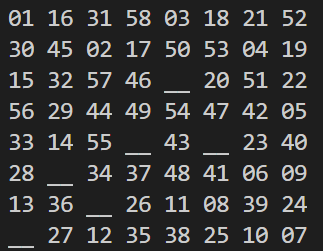
 64个选点总共求得7894584条路径，其中0回撤的路径有7609835条。占比96.4%。几乎无回撤。

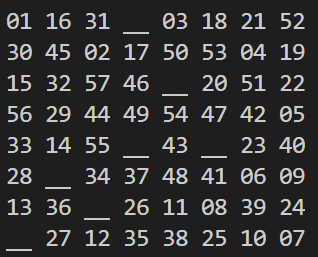
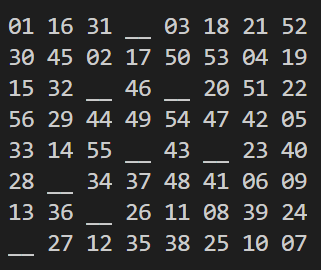
**3.演示回撤过程**

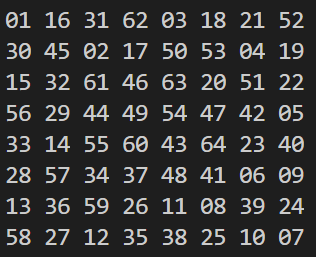
 以下是x=1，y=1，第118条路径的所有回撤过程。











五、用户手册

1.本程序运行环境为Windows系统，界面为控制台。若在LINUX下重新编译也可在LINUX下运行。

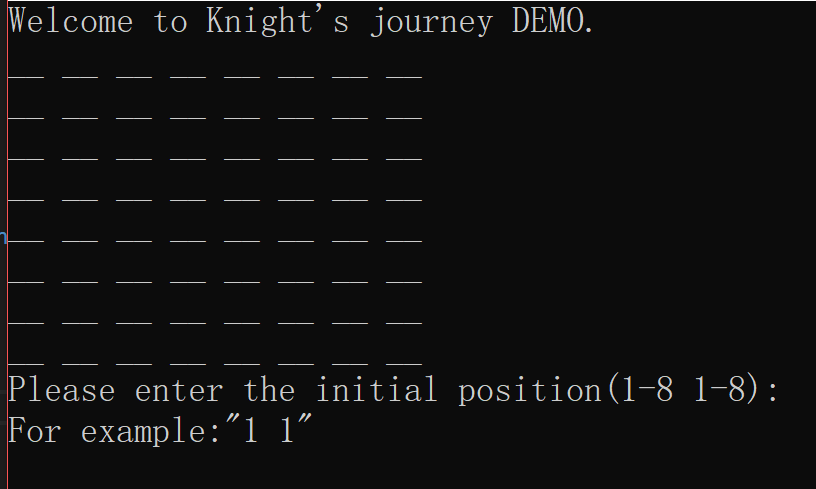
2.进入程序后，显示用户界面。

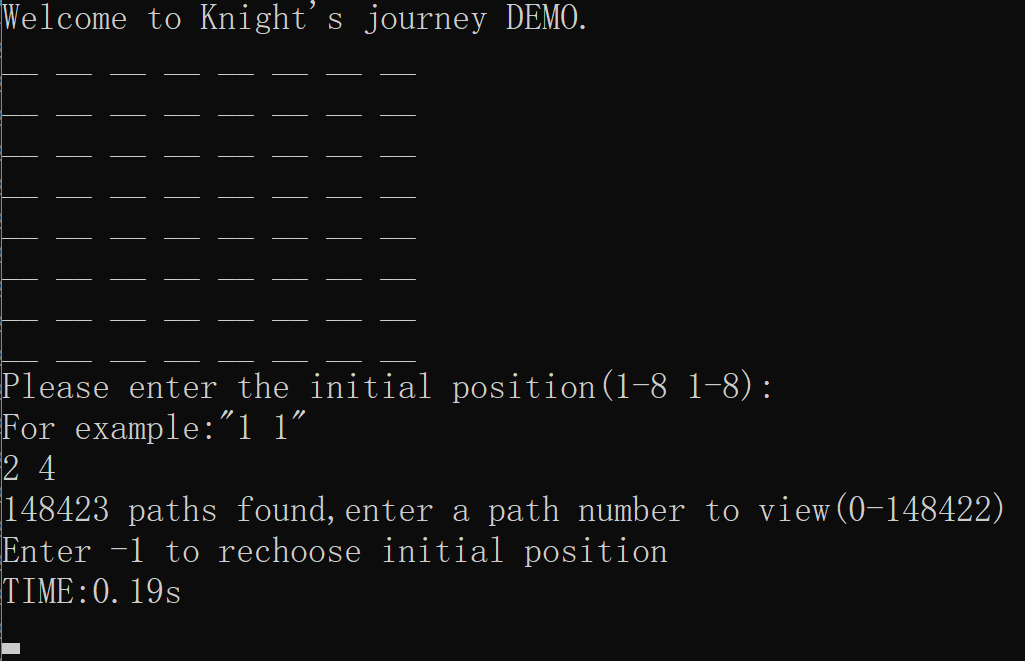
3.输入初始点，并回车。如“1 2”。

4.此时程序计算出结果。输出想查看的路径号查看路径。输入-1重新指定起始位置。

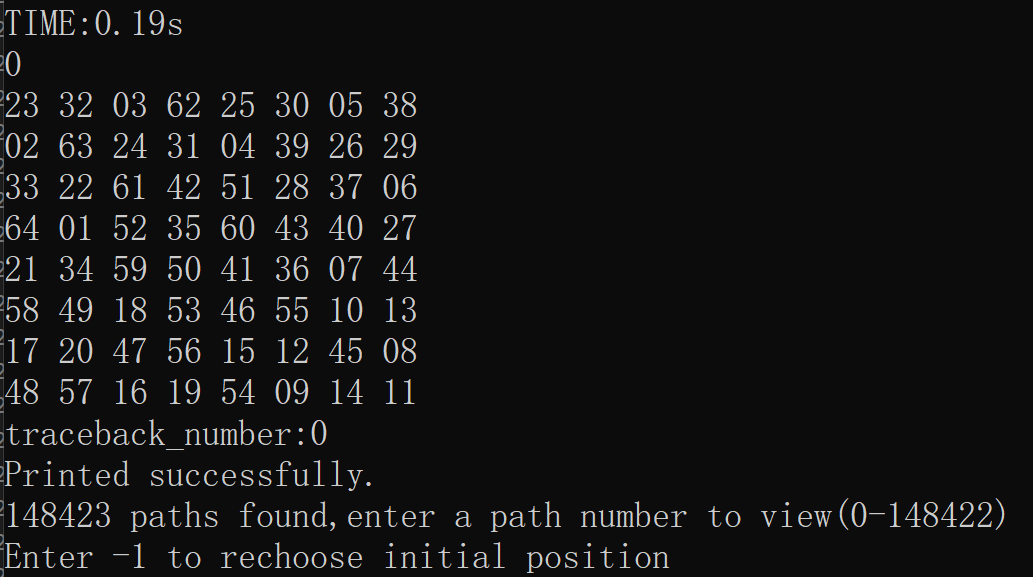
5.程序设定最多求出1000000条路径，存储这些路径需要约150MB内存容量。考虑到目前浏览器内存占用经常超过1GB，150MB的内存可以接受。若机器过于古老无法承载，可以从源代码中修改MAXHISTORY至100000，即可最多求100000条路径并大量减少内存占用。

六、测试结果

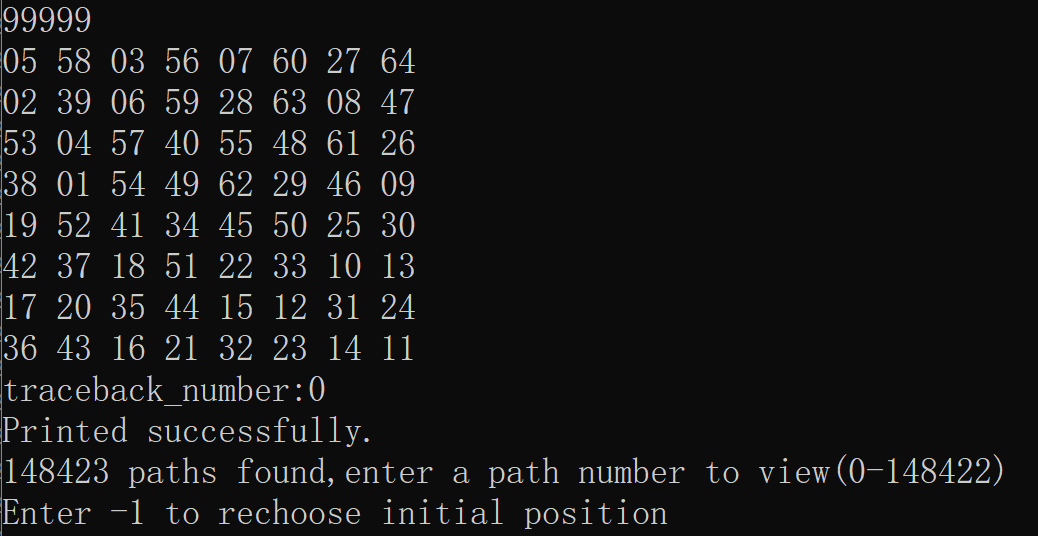
 欢迎界面

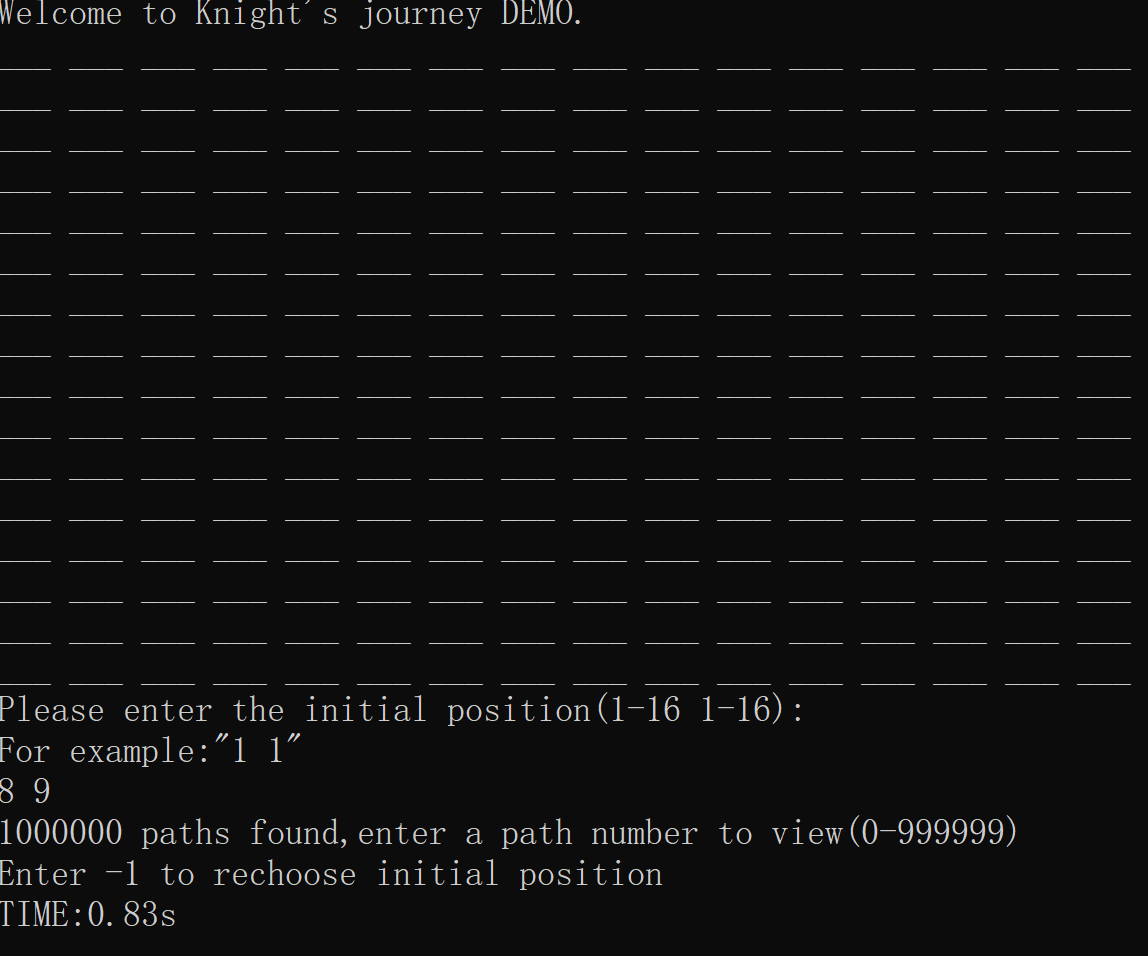
输入起始点并回车：x=2,y=4

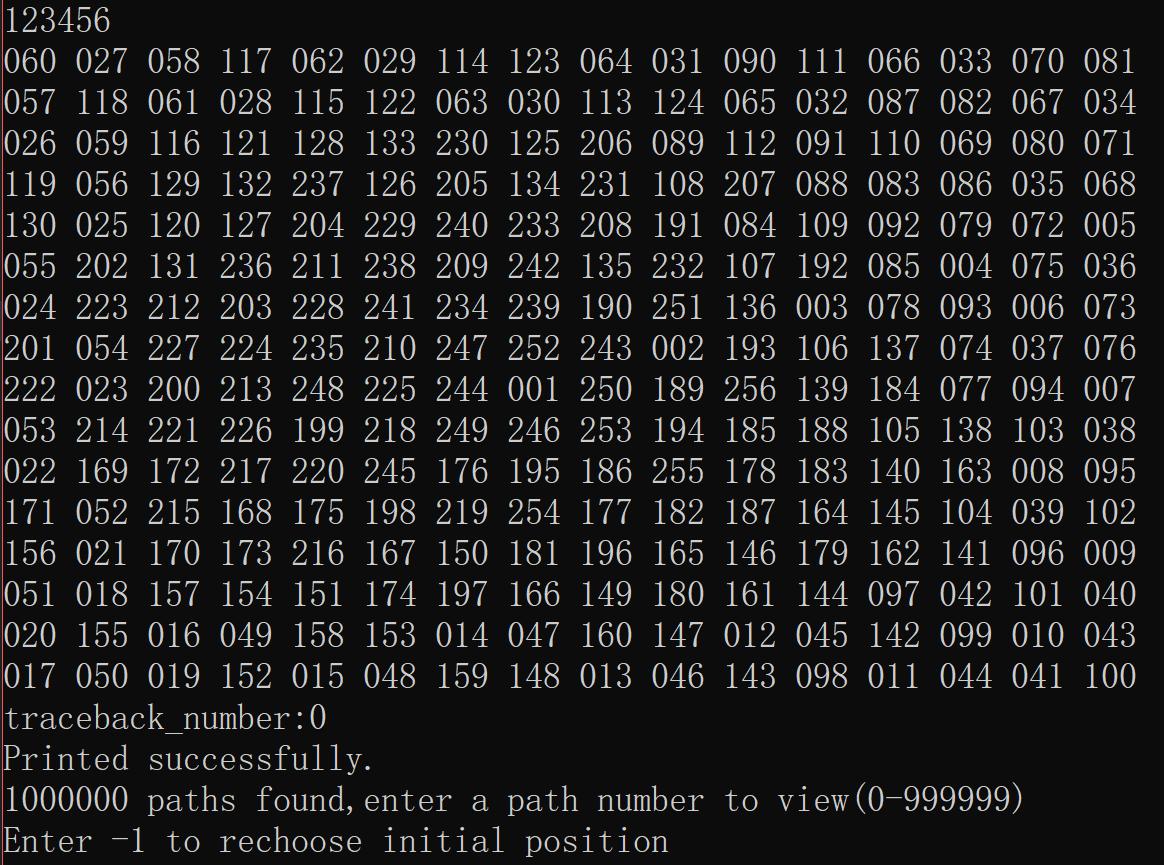
查看路径：

0号：

99999号：



16路棋盘周游（修改源代码SIZE为16即可）：

打印第123456号路径

七、附录

源代码文件清单：

knight\_journey.c

可执行程序文件清单：

knight\_journey.exe