**Sicily 1153 马周游 解题报告**

*13331231*

*计应2班*

*孙圣*

1. **题目大意**

有一个8 \* 8的棋盘，指定一个起始位置。马需要将每一个格子走遍且不重复。马的走法为“日”字形。求出其中一条路径。

1. **算法思想及主要的数据结构**

这题是一道搜索问题。由于一开始行走的路线不一定能成功达到目标，因此需要进行回溯，考虑使用递归版本的DFS，降低实现难度。在每次递归调用之后，要将所修改的状态（即标记的已访问的格子和走过的格子的数目）复原。

1. **解题思路**

需要利用一个结构体保存马所在位置的坐标信息和可行格子的数目。将格子的编号转换为坐标要进行除法和取模的运算。之后进行初始化，开始搜索过程。最开始先判断是否已经完成搜索。搜索时先将8个方向的x, y坐标设定好，这样可以利用循环进行遍历。先得出下一跳可达的节点，之后统计从该节点可达其它节点的数目，并进行排序。之后挑选可达节点数最少的节点去访问，因为如果现在不去到那个节点，在将来很有可能就无法再次到达，这样搜索就会失败。这是利用启发式函数对算法进行优化，避免超时。

1. **算法描述**

定义结构体pos：

struct pos {

int x;

int y;

int count;

};

定义比较函数，根据可达节点的数目进行排序。

bool cmp(const struct pos& a, const struct pos& b) {

return a.count < b.count;

}

深度优先搜索函数，参数为init 初始位置，visit 节点是否被访问的布尔数组，v 节点访问顺序，flag 是否完成搜索， count已访问节点的数目：

void DFS(struct pos init, bool visit[8][8], int v[64], bool& flag, int& count);

1. **程序注释清单**

#include<iostream>

#include<cstring>

#include<algorithm>

using namespace std;

// 保存节点信息

struct pos {

int x;

int y;

int count;

};

// 比较函数，根据可达节点的数目进行比较

bool cmp(const struct pos& a, const struct pos& b) {

return a.count < b.count;

}

void DFS(struct pos init, bool visit[8][8], int v[64], bool& flag, int& count) {

// 搜索成功完成，退出

if ( flag == 1 ) {

return;

}

// 将该节点标记，并记录

visit[init.x][init.y] = 1;

v[count] = init.x \* 8 + init.y + 1;

++count;

// 如果所有节点都被访问，则直接返回，不再进行探索

if ( count == 64 ) {

flag = 1;

return;

}

// 8个方向的x, y坐标

int x[8] = {1, 2, 2, 1, -1, -2, -2, -1};

int y[8] = {2, 1, -1, -2, -2, -1, 1, 2};

struct pos a[8];

// 可达节点的数目

int valid = 0;

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

// 下一个方向上节点的坐标

a[valid].x = init.x + x[i];

a[valid].y = init.y + y[i];

// 如果下一个方向上的节点合法，则将其可达的节点初始化为0

if ( a[valid].x >= 0 && a[valid].x <= 7 &&

a[valid].y >= 0 && a[valid].y <= 7 &&

!visit[a[valid].x][a[valid].y] ) {

a[valid].count = 0;

} else {

continue;

}

// 计算下一个节点处可达节点的数目

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

if ( a[valid].x + x[j] >= 0 && a[valid].x + x[j] <= 7 &&

a[valid].y + y[j] >= 0 && a[valid].y + y[j] <= 7 &&

!visit[a[valid].x + x[j]][a[valid].y + y[j]] ) {

++a[valid].count;

}

}

++valid;

}

// 按可达节点数目从小到大排序

sort(a, a + valid, cmp);

// 逐个探索下一个节点，从可达数最小的开始

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

struct pos newPos;

newPos.x = a[i].x;

newPos.y = a[i].y;

DFS(newPos, visit, v, flag, count);

}

// 回溯，将设定的访问信息复原

visit[init.x][init.y] = 0;

--count;

}

int main() {

// freopen("a.txt", "r", stdin);

int n;

while ( cin >> n && n != -1 ) {

// 初始化

struct pos init;

init.x = (n - 1) / 8;

init.y = (n - 1) % 8;

bool visit[8][8];

bool flag = 0;

int count = 0;

memset(visit, 0, sizeof(visit));

int v[64];

DFS(init, visit, v, flag, count);

// 输出

for ( int i = 0; i < 63; ++i ) {

cout << v[i] << " ";

}

cout << v[63] << endl;

}

}

1. **测试数据**

由于棋盘一共只有64格，因此总共就只有64个测试数据。我们可以将棋盘按照水平、竖直和对角线分成8个部分。因此只要选取其中一个子部分，其他的解都是与之对称的。而且这题有很多种走法，因此只需要给出一个符合要求的即可。

(A).输入为1

1 11 17 2 12 6 16 31 48 63 53 59 49 34 51 57 42 25 10 4 14 8 23 40 55 61 46 56 62 52 58 41 26 9 19 36 21 38 32 15 5 22 7 24 39 29 44 27 33 50 60 43 28 45 35 18 3 13 30 20 37 47 64 54

(B).输入为2

2 17 34 49 59 42 57 51 61 55 40 23 8 14 24 7 13 3 9 19 25 10 4 21 6 16 31 48 63 53 36 46 56 62 52 58 41 26 11 1 18 33 43 60 50 35 45 28 38 44 27 12 29 39 54 64 47 37 20 30 15 32 22 5

(C).输入为3

3 9 26 41 58 52 62 56 39 24 7 13 23 8 14 4 10 25 42 57 51 61 55 40 46 63 48 31 16 6 21 36 19 2 17 34 49 59 53 38 32 15 30 47 64 54 60 45 35 50 44 29 12 27 33 43 37 20 5 22 28 18 1 11

(D).输入为4

4 10 25 42 57 51 61 55 40 23 8 14 24 7 13 3 9 19 2 17 34 49 59 44 50 60 54 64 47 62 56 39 45 30 15 32 38 48 63 53 36 46 29 35 52 58 41 26 20 5 11 1 18 28 43 33 27 37 22 12 6 16 31 21

(E).输入为10

10 25 42 57 51 61 55 40 23 8 14 4 19 9 3 13 7 24 39 56 62 52 58 41 26 36 46 63 48 31 16 6 21 15 32 38 53 59 49 34 17 2 12 29 44 27 33 50 60 43 28 45 35 18 1 11 5 22 37 54 64 47 30 20

(F).以此类推，我们还可以测试输入为11，12，19，20和28等样例。

1. **分析与优化**

一开始做这题时并没有利用启发式函数对搜索进行优化。所以在5 \* 6的那个问题中也发生了超时，但是经过改变访问的方向的先后次序就通过了测试。到这道8 \* 8的题目时，由于数据规模的加大，仅仅通过一个固定的访问顺序是很难保证程序的执行效率的。毕竟这题利用的是深度优先搜索，分枝因子b为8，而最大展开n层（n = 64），因此时间复杂度为O(n ^ b)，即64^8。

所以，要通过启发式函数来让程序先搜索更可能达到目的，即成功访问所有节点的路径。所以我将启发值设定为可达节点的数目，即比较对于下一跳的所有节点，它们可达节点的数目。如果可达节点数目较小，则优先去访问。最后将整个程序的运行时间大大降低。

对于空间复杂度，由于深度优先搜索本身所需要的空间就不是很大，为O(bn)，即64 \* 8，因此不需要进行过多的优化。