

哈尔滨工业大学（深圳）

数据结构实验报告

试探与回溯的应用

学 院:	_____ 计算机科学与技术 _____
姓 名:	_____ 胡博涵 _____
学 号:	_____ SZ170110113 _____
专 业:	_____ 计算机科学与技术 _____
日 期:	_____ 2018-05-03 _____

一、问题分析

任务一要求在指定的地图中寻找最多的可达点数目，对于计算机而言，可以通过暴力枚举的方法，不断地搜索，但这样效率很低。事实上，这个问题是一个连通图遍历的问题，对于计算机而言，我们要指定特定的行进规则，在给定的连通图内，进行不断地试探，直到不可继续前进的时候进行回溯。这就是所谓的深度优先搜索问题。

任务二要求在已给定的迷宫地图中，指定起点与终点坐标，寻找一条路径，这也是图的深度优先搜索问题。即对于计算机而言，要按照特定的行进规则，在某个特定的连通图内，寻找到一条符合要求的通路。

在这两个实验中，我们固然可以用递归解决问题，但是，我们还需要解决的问题是：如何保证已经被搜索过的部分不被重新搜索，其办法之一就是：在已经搜索过的地方留下标记。我们如何在已经搜索过的地方留下标记、如何记录搜索的路径，并保证不重不漏，这需要我们定义一个具体的搜索规则，同时选择合适的数据结构进行试探和回溯。

二、详细设计

2.1 设计思想

若要统计从点 A 出发可达点的数目，如果靠暴力枚举也可以解决，但是效率极低，因而，在思考问题的过程中，我发现这个问题可以分解为两个子问题，即寻找从 A 点走一步可到达的点的集合 P 中的点能到达的点的数目。这个问题具有自相似性，即由“减而治之”的思想，假设从点 A 出发，这个问题可以分解为从起点出发走一步后，寻找到达终点的路径。因此，问题就被简化为了一个常数规模的子问题和一个规模更小的子问题，我们就可以想到利用递归或是栈模拟递归调用来解决这个问题。（在后面的流程图中，可以观察出递归调用时，其每一个实例的流程图形状基本一致，对应的就是问题的自相似性）

若要解决 A 到 B 的路径问题，由试探和回溯的思想，我们很容易就可以想到，从上面可达点的数目问题进行再一次的具体概括，即沿某一特定的规则，按一定的优先级顺序沿各个方向不断试探，与任务 1 不同的是，只需多加一步：在每一次试探时，都判断一下这个点是否为我们所需要的终点，并在试探过程中，用一个辅助变量记录路径上顶点的顺序。我们可以将这个问题抽象为图的深度优先搜索问题后，用递归的方式去解决。

以上的“深度优先搜索”的思想，可以概括为：从某个点出发，优先向方向 D1 一直前行，若碰到障碍物或碰到已经访问的点，无法以 D1 方向继续前行时，尝

试改变方向为 D2 继续前行，每前行一步，都按 D1, D2, D3, D4 的优先级顺序依次检查是否能往该方向继续前行。直到无法继续前行时，进行回溯，后退一步，再从其他方向继续尝试，用一个形象的比喻就是“不撞南墙不回头”。

而递归的调用，可以用栈模拟，具体思路是：每一次递归调用保存的现场，都可以用定义好的数据结构存储在栈中。弹出栈顶的操作，对应于递归函数中的 return。入栈的操作，则代表递归中更深入的增加一个新的调用实例。

2.2 存储结构及操作

(1) 存储结构

二维数组	存储读入的地图	字符 (char)
结构体类型 Point	存储横纵坐标	横纵坐标 x,y (int)
	在每一轮的搜索中，标记是否被访问	访问标记 Status (enum)
	记录该点为障碍或是通路	节点类型 poinType (enum)
二维数组	存储所有的点	结构体 Point
非递归实现附加数据结构：		
栈	模拟递归调用实例	结构体 Point
	存储栈顶位置	top(int)

(2) 涉及的操作

对于栈的操作：

函数名	传入参数	函数功能	返回值
push	栈指针 S、操作元素 A	将 A 压入栈 S	无
pop	栈指针 S	将 S 顶部元素弹出	弹出的元素
top	栈指针 S	无	S 顶部的元素

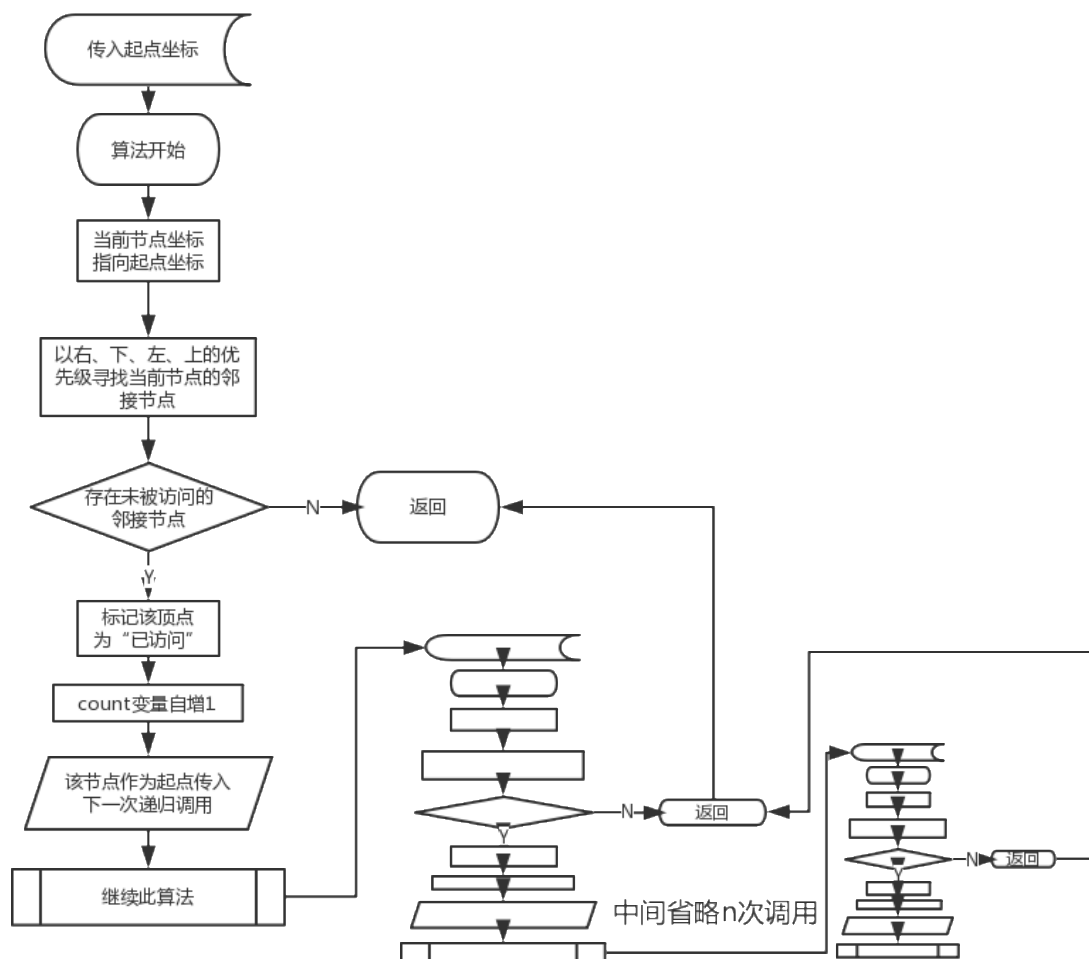
通用操作：

nextNbr: 按照顺时针的原则，即右、下、左、上的优先级，寻找相对于当前位置的下一个没有被访问的邻接节点，并返回其在地图上的坐标，需要注意的是，判断的顺序是：在判断邻接节点是否被访问之前，先要判断函数给出的下一个邻接节点的坐标是否有越界，若没有越界，才能进行下一步判断：该坐标对应的点是否被访问过。若没有找到满足条件的邻接节点，就返回一个特定坐标(-1, -1)。

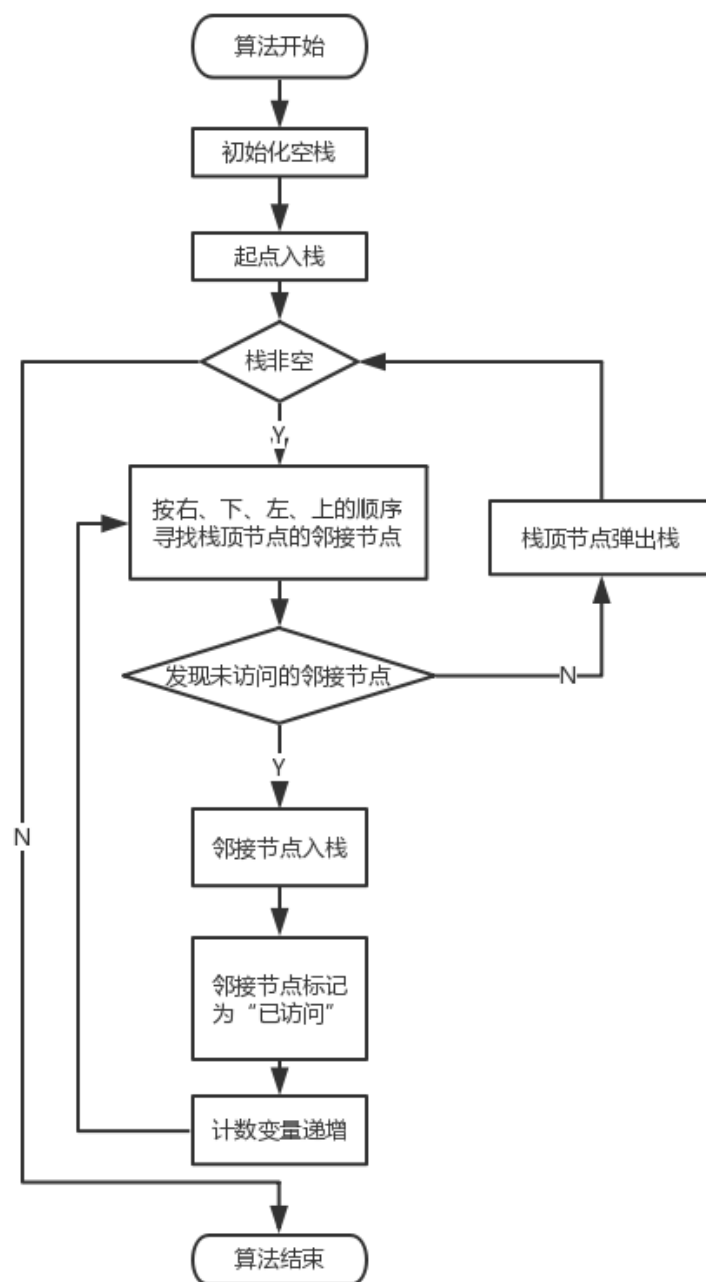
2.3 程序整体流程

任务 1:

递归调用流程图:

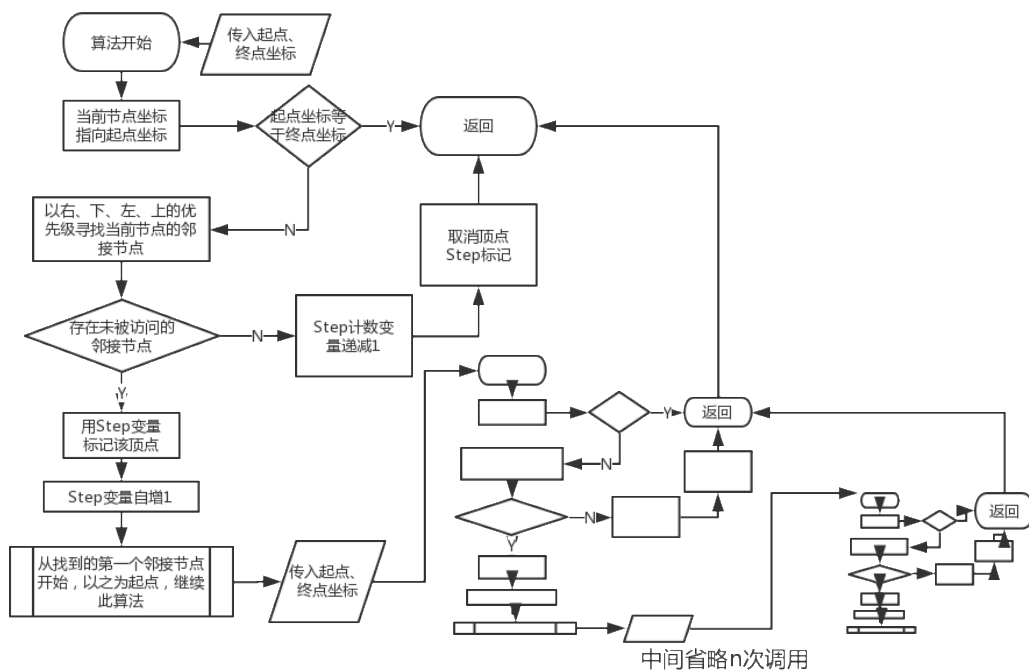


非递归调用流程图：

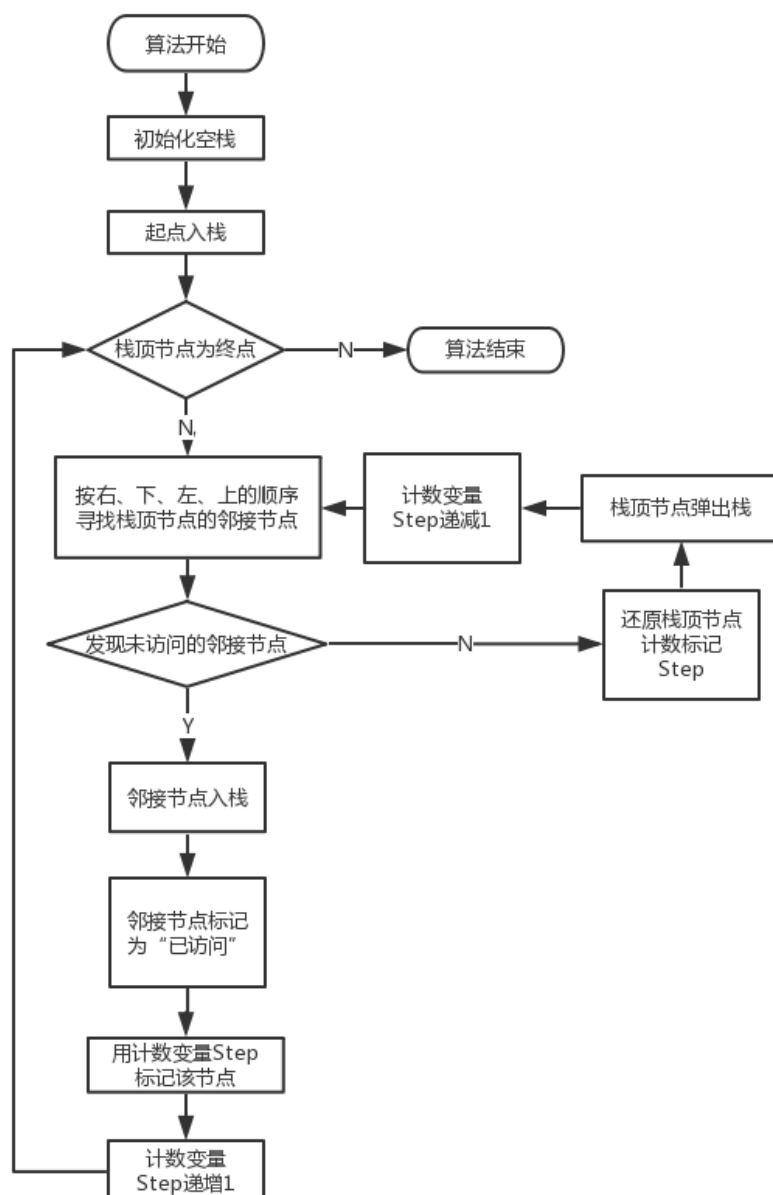


任务 2:

递归调用流程图:



非递归调用流程图：



三、用户手册

从文件读入迷宫数据

在输入中，首先输入两个数，用空格隔开，代表迷宫的行和列。用#代表墙壁，0 代表空地，?和*分别表示起点和终点，迷宫只会有一个起点和一个终点。每个点可以往上下左右四个方向走。

在输出中，用数字代表路径经过点的顺序，大于 9 的时候重新从 1 开始计数。若不可能，直接输出”Impossible”。

四、总结

本次实验考查了试探与回溯的思想，以及减而治之的解决问题的策略，还有递归和非递归的转换问题。

从以上的递归调用流程图可以看出，每一次调用实例的流程图形状基本类似，可以总结出：凡递归可以解决的问题，其有一定的自相似性，即可以分解为一个常数规模的问题和一个与之相似的、但规模更小的问题。

试探与回溯的思想可以适用于此类穷举问题，这个问题的本质就是穷举从点 A 出发的路径，或是统计路径点的数目，或是找出这些路径中可以到达特定终点 B 的某一条。

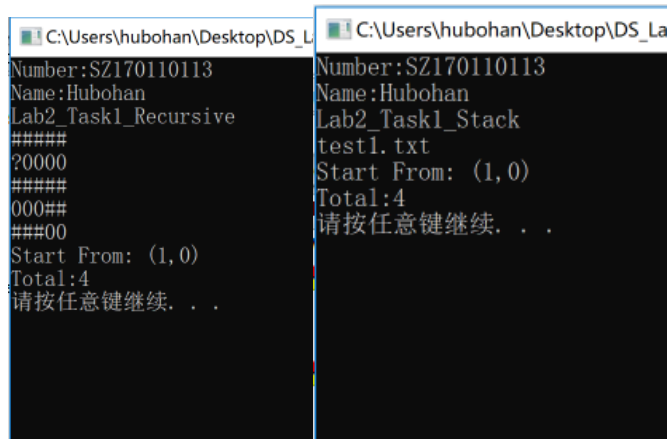
试探与回溯的方法很容易证明是正确的，因为这个算法会尝试与当前格点相邻的所有格点，通过数学归纳的思想，我们如果在到达了终点之后，不停止算法，继续访问终点四周的节点，则该算法可以到达和起始格点相同的所有格点，真正做到“不重不漏”，只要目标节点和起始节点之间连通，就一定可以找到一条对应的路径。

对于这个算法的复杂度分析：算法的每一步迭代（或每调用一个递归实例）需要常数时间，故算法的复杂度正比于试探、回溯操作的总数，而试探回溯操作的总数的数量级又相当于访问过的格点的总数。

递归与非递归的转换，最简单的方案就是用栈模拟递归调用的工作，递归的调用实际上就是函数压栈、出栈的过程，理解递归调用栈的变化情况和函数中对应的操作，就不难将非递归改写为递归。

五、结果

程序正确运行的结果截图（左图为递归，右图为用栈模拟递归）



```
C:\Users\hubohan\Desktop\DS_L
Number:SZ170110113
Name:Hubohan
Lab2_Task1_Recursive
#####
?0000
#####
000##
###00
Start From: (1,0)
Total:4
请按任意键继续. . .

C:\Users\hubohan\Desktop\DS_La
Number:SZ170110113
Name:Hubohan
Lab2_Task1_Stack
test1.txt
Start From: (1,0)
Total:4
请按任意键继续. . .
```



```

C:\Users\hubohan\Desktop\DS_Lab2\Lab2_1_Recursion\bin\De
Number:SZ170110113
Name:Hubohan
Lab2_Task1_Recursive
#####
#?0#000#0#
#00#0000#0#
#00#00000#
#####0#
###00*000#
#####
Start From: (1,1)
Total:26
请按任意键继续. . .

Process returned 0 (0x0)   execution time : 4.4
Press any key to continue.

```

```

C:\Users\hubohan\Desktop\DS_Lab2\Lab2_1_Recursion\bin\De
Number:SZ170110113
Name:Hubohan
Lab2_Task1_Recursive
#####
#?0#000#0#
#00#0000#0#
#00#00000#
#####0#
###00*000#
#####
Start From: (1,1)
Total:5
请按任意键继续. . .

C:\Users\hubohan\Desktop\DS_Lab2\Lab2_1_Stack\bin\I
Number:SZ170110113
Name:Hubohan
Lab2_Task1_Stack
test3.txt
Start From: (1,1)
Total:5
请按任意键继续. . .

```

```

选择C:\Users\hubohan\Desktop\DS_Lab2\Lab2_1_Recursion\bin
Number:SZ170110113
Name:Hubohan
Lab2_Task1_Recursive
#####00
#00#000#0#
#00#00#0#
#00#0#00?#
#####0#
###00#000#
#####
Start From: (3,8)
Total:10
请按任意键继续. . .

C:\Users\hubohan\Desktop\DS_Lab2\Lab2_1_Stack\bin\Debug\La
Number:SZ170110113
Name:Hubohan
Lab2_Task1_Stack
test4.txt
Start From: (3,8)
Total:10
请按任意键继续. . .

```

```

C:\Users\hubohan\Desktop\DS_Lab2\Lab2_2_Recursion\bin\Debug\Lab2_2_Recursic
Number:SZ170110113
Name:Hubohan
Lab2_Task2_Recursive
test1.txt
Impossible.
请按任意键继续. . .

C:\Users\hubohan\Desktop\DS_Lab2\Lab2_2_Stack\bin\Debug\L
Number:SZ170110113
Name:Hubohan
Lab2_Task2_Stack
test1.txt
Impossible.
请按任意键继续. . .

```

```
C:\Users\hubohan\Desktop\DS_Lab2\Lab2_2_Recursion\bin\Debug
Number:SZ170110113
Name:Hubohan
Lab2_Task2_Recursive
test2.txt
#####
#?1#000#0#
#023456#0#
#00#00789#
#####1#
###00*432#
#####
请按任意键继续. . .
```

```
3 char_fileadd[1000] = "test2.txt";
C:\Users\hubohan\Desktop\DS_Lab2\Lab2_2_Stack\bin\Debug\Lab2_2
Number:SZ170110113
Name:Hubohan
Lab2_Task2_Stack
test2.txt
#####
#?1#000#0#
#023456#0#
#00#00789#
#####1#
###00*432#
#####
请按任意键继续. . .
```

```
C:\Users\hubohan\Desktop\DS_Lab2\Lab2_2_Recursion\
Number:SZ170110113
Name:Hubohan
Lab2_Task2_Recursive
test3.txt
Impossible.
请按任意键继续. . .
```

```
C:\Users\hubohan\Desktop\DS_Lab2\Lab2_2_Stack\bin\Debug\Lab2_2_Stack.exe
Number:SZ170110113
Name:Hubohan
Lab2_Task2_Stack
test3.txt
Impossible.
请按任意键继续. . .
```

```
C:\Users\hubohan\Desktop\DS_Lab2\Lab2_2_Recursion\
Number:SZ170110113
Name:Hubohan
Lab2_Task2_Recursive
test4.txt
Impossible.
请按任意键继续. . .
```

```
C:\Users\hubohan\Desktop\DS_Lab2\Lab2_2_Stack\bin\De
Number:SZ170110113
Name:Hubohan
Lab2_Task2_Stack
test4.txt
Impossible.
请按任意键继续. . .
```