

算法与数据结构设计报告

（ 2020 / 2021学年 第 一 学期）

题 目： 最短路径----拯救007

迷宫问题

|  |  |
| --- | --- |
| **专 业** | **计算机科学与技术** |
| **学 生 姓 名** | **吴雯** |
| **班 级 学 号** | **B18030322** |
| **指 导 教 师** | **李洋** |
| **指 导 单 位** | **计算机科学与技术系** |
| **日 期** | **2020.10.26-2020.11.6** |

|  |  |  |
| --- | --- | --- |
| **评价准则** | | **计分（每项10分）** |
| 1、能够掌握算法与数据结构设计的相关基础知识，并能够针对求解的工程问题，收集资料进行合理的分析与设计 | |  |
| 2、通过调研，能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行编程实现 | |  |
| 3、能够给出数据结构和算法的设计描述，给出关键算法的流程图或伪代码，并给出各算法之间的结构关系描述 | |  |
| 4、具备一定的人机交互设计意识，人机交互设计合理、友好，操作简便 | |  |
| 5、具备一定的算法与数据结构设计分析能力，能够完成课题要求的各项任务和指标 | |  |
| 6、能够结合计算机软硬件资源，合理选用算法、数据结构、数据存储方式等技术手段，对求解的工程问题进行有效建模和求解 | |  |
| 7、具备一定自学能力与探索创新意识，能够充分利用教科书及其资源（如网络等）自学新知识与新技能 | |  |
| 8、掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并能够进行充分测试 | |  |
| 9、能够正确、完整地回答指导教师关于课题的问询，反映其对课题内容，以及相关的工程基础知识具有较好的理解和掌握 | |  |
| 10、具备一定的语言表达能力与文字处理能力，能够结合复杂工程问题撰写报告，报告内容和实验数据详实，格式规范 | |  |
| 总 分 |  | |
| **指导教师： 年 月 日** | | |
| **备注：** | | |

**最短路径----拯救007**

**一、课题内容和要求**

**课题内容：**

本课题假设湖是100×100的正方形，设湖的中心在(0,0)，湖的东北角的坐标是(50,50)。湖中心的圆形小岛的圆心在(0,0)，直径是15。一些凶残的鳄鱼分布在湖中不同的位置。现已知湖中鳄鱼的位置（坐标）和James Bond可以跳的最大距离，请告诉James Bond一条最短的到达湖边的路径。他逃出去的路径的长度等于他跳的次数。

“最短路径----拯救007”的功能框架图如图1所示。

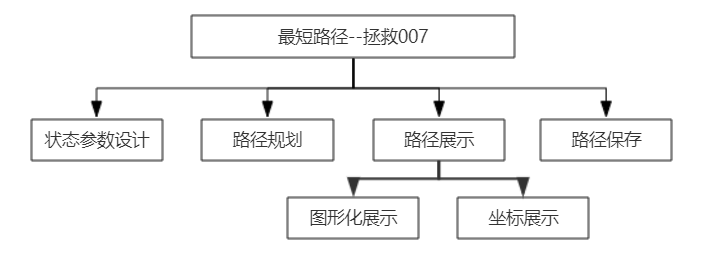


图1 功能框架图

**课题要求：**

（1）支持系统运行时的状态参数配置，如鳄鱼数量、位置、007跳的最大步长等；

（2）利用队列图的广度搜索算法实现最短路径的计算；

（3）支持求解路径的界面动态展示；

（4）输入要求：程序从文件中读取输入信息，文件中包括的多组输入数据。每组输入数据包括鳄鱼的数量、007可以跳的最大距离、鳄鱼的坐标（没有两只鳄鱼出现在同一个位置）；

（5）输出要求：程序结果输出到文件中。对于每组输入数据，如果007可以逃脱，则输出007必须跳的最小的步数，然后按照跳出顺序记录跳出路径上的鳄鱼坐标；如果007不能逃脱，则输出-1到文件；

（6）人机界面友好。

**二、数据结构说明**

1. 状态记录(007跳过的路径)结构体

1. **typedef** **struct** GraphNodeRecord {   //状态记录结构体
2. **int** X;                   //X轴坐标
3. **int** Y;                   //Y轴坐标
4. unsigned **int** Step;       //跳至该点的步数
5. Vertex Path;             //记录上一个点
6. }GraphNode;
7. **typedef** GraphNode \*Graph;

2. 链表形式表示的队列

1. //链表形式
2. **typedef** **struct** NodeRecord {
3. ElemType Element;
4. **struct** NodeRecord \*Next;    //指向下一个node
5. }\*Node;
7. **typedef** **struct** DequeRecord {
8. Node Front, Rear;           //分别指向Deque的前后两个点
9. }\*Deque;

**三、算法设计**

1 路径规划算法

基于队列图的广度优先思想：首先根据步长判断007是否能一步跳出，若不能跳出再考虑必须经过鳄鱼头上的情况；然后广度遍历所有的鳄鱼并判断007是否能从小岛跳到(x,y)处的鳄鱼头上，如果能从这个鳄鱼跳到岸上就只经过这条鳄鱼，反之将此鳄鱼坐标加入队列【Inject(i, D)】中；当队列不空时，取出队列【V = Pop(D)】中的第一只鳄鱼的位置，遍历判断能否从当前鳄鱼跳到其他鳄鱼头上，在这其中，如果能跳出则成功，反之将其加入队列；最后，返回保存007跳过的路径结构体G。

程序中对应主要函数为：

1. Graph read\_case(**FILE**\* InFile, **int** num, Vertex\* Bank, Deque D){
2. ……
3. }

其功能为：读一个case返回一个Graph，\*Bank记录最短到达河岸的路径。该算法流程图如图2所示。

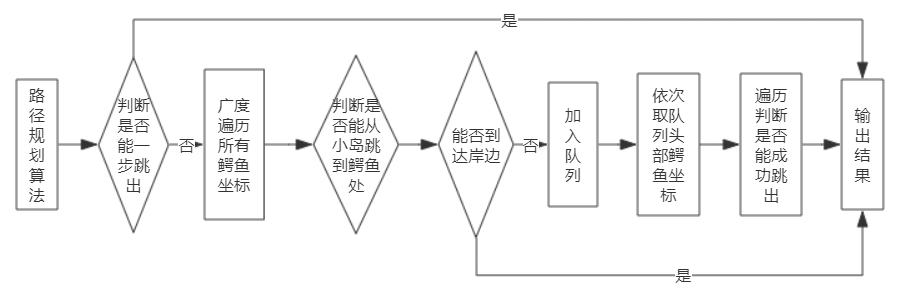


图2 路径规划算法

2 路径展示算法

根据路径规划算法中求得的逃离鳄鱼潭需跳过的鳄鱼位置序列，在设定的的展示区域依次连线，并通过适当的Sleep()过程，实现路径的动态展示。流程图如图3所示。

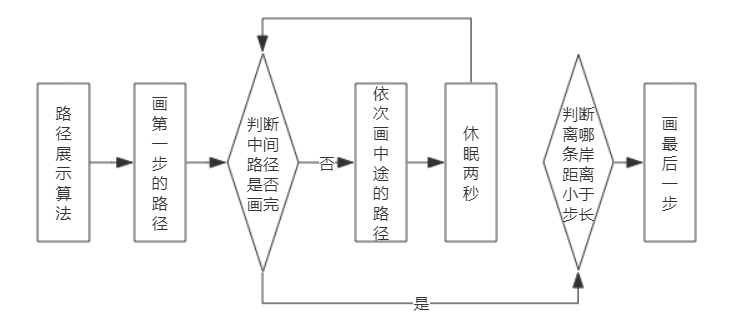


图3 路径展示流程

3 路径保存算法

根据路径规划算法得到的\*Bank（记录了最短到河岸的路径）判断是哪种情形；0是无法跳出，1是可以直接跳出，其他需要借助鳄鱼跳出；将路径规划算法返回的路径结构体G中的G[Bank].Path值push到队列中，然后依次出队列写入文件中，将界面保存至image文件夹中。

程序中对应主要函数为：

1. **void** write\_result(**FILE**\* OutFile, Vertex Bank, Graph G, Deque D){
2. **……**
3. }

其功能为：写出结果，即最短路径。该算法流程图如图4所示。

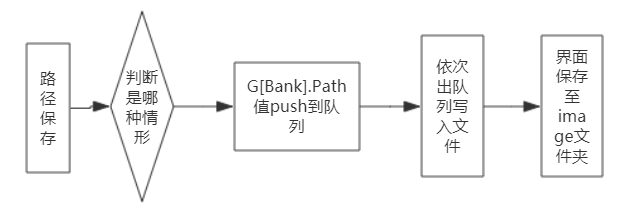


图4 路径保存算法

**四、详细设计**

1. Deque.cpp

1. //创建新的Deque
2. Deque DequeNew()
3. {
4. Deque D;
5. D = (Deque)malloc(**sizeof**(**struct** DequeRecord));
6. CHECK(D);
7. D->Front = D->Rear = (Node)malloc(**sizeof**(**struct** DequeRecord));  //空的头
8. CHECK(D->Front);
9. D->Front->Element = 0;    //初始化
10. D->Rear->Next = NULL;
11. **return** D;
12. }
14. //删除Deque
15. **void** DequeDelete(Deque D)
16. {
17. **if** (D)
18. {
19. **while** (D->Front)
20. {
21. D->Rear = D->Front->Next;
22. free(D->Front);
23. D->Front = D->Rear;
24. }
25. free(D);
26. }
27. }
29. //删除所有的节点除了头节点
30. **void** DequeClear(Deque D)
31. {
32. **if** (D)
33. {
34. **while** (D->Front->Next)     //删除第一个节点
35. {
36. D->Rear = D->Front->Next->Next;
37. free(D->Front->Next);
38. D->Front->Next = D->Rear;
39. }
40. D->Rear = D->Front;
41. }
42. }
44. //判断Deque是否为空
45. **int** IsEmpty(Deque D)
46. {
47. **return** D->Front == D->Rear;
48. }
50. //将X元素压栈到D中
51. **void** Push(ElemType X, Deque D)
52. {
53. Node NewNode;
54. NewNode = (Node)malloc(**sizeof**(**struct** NodeRecord));   //建立新的节点
55. CHECK(NewNode);
56. NewNode->Element = X;
57. NewNode->Next = D->Front->Next;
58. **if** (D->Front == D->Rear)                   //如果D为空
59. D->Rear = NewNode;
60. D->Front->Next = NewNode;       //压栈
61. }
63. //将第一个元素出栈
64. ElemType Pop(Deque D)
65. {
66. Node Temp;
67. ElemType Item;
68. **if** (D->Front == D->Rear)
69. {
70. Error("Deque is empty");
71. **return** 0;
72. }
73. **else**
74. {
75. Temp = D->Front->Next;    //得到第一个元素
76. D->Front->Next = Temp->Next;  //重置第一个元素
77. **if** (Temp == D->Rear)    //如果只有第一个元素
78. D->Rear = D->Front;   //将D置空
79. Item = Temp->Element;
80. free(Temp);
81. **return** Item;
82. }
83. }
85. //插入元素X至D末尾
86. **void** Inject(ElemType X, Deque D)
87. {
88. Node NewNode;
89. NewNode = (Node)malloc(**sizeof**(**struct** NodeRecord));   //创建新节点
90. CHECK(NewNode);
91. NewNode->Element = X;
92. NewNode->Next = NULL;
93. D->Rear->Next = NewNode;
94. D->Rear = NewNode;
95. }

2. Graph.cpp

1. //创建新的Graph
2. Graph GraphNew(**int** NodeNum)
3. {
4. Graph G;
5. **int** i;
6. **if** (NodeNum <= 0)
7. **return** NULL;
8. G = (Graph)malloc(NodeNum \* **sizeof**(GraphNode));    //分配空间
9. CHECK(G);
10. **for** (i = 0; i < NodeNum; i++)                             //初始化
11. {
12. G[i].X = 0;
13. G[i].Y = 0;
14. G[i].Step = INFINITY;
15. G[i].Path = 0;
16. }
17. **return** G;
18. }

21. //删除一个Graph
22. **void** GraphDelete(Graph G)
23. {
24. **if** (G)
25. free(G);
26. }
28. //判断007是否能从起点跳至该点(x,y)，步长是d
29. **int** CheckForStart(**int** x, **int** y, Distance d)
30. {
31. **double** t;
32. //x^2+y^2<=(ISLAND\_DIAMETER/2.0+d)^2
33. t = (ISLAND\_DIAMETER + (d \* 2.0));
34. **return** (x \* x + y \* y) <= t \* t / 4.0;
35. }
37. //判断007能否从该点跳至河岸，步长是d
38. **int** CheckForEnd(**int** x, **int** y, Distance d)
39. {
40. **if** (x < 0)
41. x = -x;
42. **if** (y < 0)
43. y = -y;
44. //因为湖是正方形，只需要检查这两距离
45. **return** (d >= LAKE\_BOUNDARY\_X - x) || (d >= LAKE\_BOUNDARY\_Y - y);
46. }
48. //判断007能否从点i跳至点j，步长是d
49. **int** CheckForConnect(Graph g, Vertex i, Vertex j, Distance d)
50. {
51. **int** x, y;
52. x = g[i].X - g[j].X;
53. y = g[i].Y - g[j].Y;
54. **return** x \* x + y \* y <= d \* d;
55. }

3. main.cpp

1. //读一个case返回一个Graph，\*Bank记录最短到达河岸的路径
2. Graph read\_case(**FILE**\* InFile, **int** num, Vertex\* Bank, Deque D)
3. {
4. Graph G = NULL;
5. Distance JamesJump;
6. Vertex V;
7. **int** x, y;
8. **int** i, Times;
9. \*Bank = 0;
10. fscanf\_s(InFile, "%lf", &JamesJump);
11. **if** (CheckForEnd(0, 0, JamesJump + ISLAND\_DIAMETER / 2.0))                  //如果能一步跳出
12. {
13. **for** (i = 0; i < (num << 1); i++)                                       //此循环是为了保证屏蔽鳄鱼位置，否则输出鳄鱼个数多的-1
14. fscanf\_s(InFile, "%d", &x);
15. \*Bank = 1;                                                             //步数为1
16. }
17. **else** **if** (num > 0)     //007必须经过鳄鱼头上的情况
18. {
19. num += 2;
20. G = GraphNew(num);
21. **for** (i = 2; i < num; i++)   //第三个node开始是鳄鱼（前两个分别是鳄鱼个数和步长）
22. {
23. fscanf\_s(InFile, "%d", &x);  //读取第i个鳄鱼的xy坐标
24. fscanf\_s(InFile, "%d", &y);
25. G[i].X = x;
26. G[i].Y = y;
27. **if** (CheckForStart(x, y, JamesJump))  //判断是否能跳上该点，如果能跳上去
28. {
29. G[i].Path = 1;   //007可以跳到
30. G[i].Step = 1;   //一步
31. **if** (CheckForEnd(x, y, JamesJump))//判断是否能跳出，如果能从这个点跳出去
32. {
33. \*Bank = i; //007可以跳出
34. Times = (num - i - 1) << 1;
35. **for** (i = 0; i < Times; i++)   //不必检验其他鳄鱼
36. fscanf\_s(InFile, "%d", &y);
37. DequeClear(D);
38. **break**;
39. }
40. **else**                 //不能跳出就将此鳄鱼放入队列中
41. Inject(i, D);   //插入该点，并开始下一个检测
42. }
43. }
44. **while** (!IsEmpty(D))   //只经过一个鳄鱼无法跳出，必须还要跳到其他鳄鱼的情况
45. {
46. V = Pop(D);       //取出队列中的头鳄鱼
47. **for** (i = 2; i < num; i++) //从这只鳄鱼跳到其他各个鳄鱼
48. {
49. **if** ((G[i].Step > G[V].Step + 1) && CheckForConnect(G, V, i, JamesJump))
50. //如果能从当前鳄鱼跳到其他鳄鱼
51. {
52. G[i].Path = V;
53. G[i].Step = G[V].Step + 1; //步数加一
54. **if** ((G[i].Step < G[\*Bank].Step) && CheckForEnd(G[i].X, G[i].Y, JamesJump))
55. //如果能从当前鳄鱼跳到岸边就结束
56. \*Bank = i;
57. **else**
58. Inject(i, D); //不能跳出就将鳄鱼放入队列
59. }
60. }
61. }
62. }
63. **return** G;
64. }
66. //写出结果，即最短路径
67. **void** write\_result(**FILE**\* OutFile, Vertex Bank, Graph G, Deque D)
68. {
69. unsigned **int** Times, i;
70. Vertex V;
71. **switch** (Bank) {
72. **case** 0:   //007无法跳出
73. fprintf(OutFile, "%d\n", -1);
74. **break**;
75. **case** 1:    //007可以直接跳出
76. fprintf(OutFile, "%d\n", 1);
77. **break**;
78. **default**:
79. Times = G[Bank].Step + 1;    //跳的步数
80. **while** (Bank != 1)     //跟踪路径
81. {
82. Push(Bank, D);
83. Bank = G[Bank].Path;
84. }
85. fprintf(OutFile, "%d\n", Times);   //输出
86. **for** (i = 1; i < Times; i++)
87. {
88. V = Pop(D);
89. fprintf\_s(OutFile, "%d ", G[V].X);
90. fprintf\_s(OutFile, "%d\n", G[V].Y);
91. }
92. }
93. }





100. **int** main(**int** argc,**char** \*argv[])
101. {
102. **FILE**\* in, \* out;
103. Deque D;
104. **int** VertexNum;
105. Graph G = NULL;
106. Vertex Bank = 0;
107. fopen\_s(&in, "input.txt", "r");
108. **if** (NULL == in)
109. {
110. fprintf(stderr, "Can not open input.txt");
111. exit(-1);
112. }
113. fopen\_s(&out, "output.txt", "w");
114. **if** (NULL == out)
115. {
116. fprintf(stderr, "Can not open output.txt");
117. fclose(in);
118. exit(-1);
119. }
120. D = DequeNew();
121. **while** ((EOF != fscanf\_s(in, "%d", &VertexNum)) && (0 <= VertexNum))
122. {
123. G = read\_case(in, VertexNum, &Bank, D); //读文件直到结尾
124. write\_result(out, Bank, G, D);
125. **if** (G)
126. GraphDelete(G);
127. }
128. fclose(in);
129. fclose(out);
130. DequeDelete(D);
131. **return** 0;
132. }

**五、测试数据及其结果分析**

对于控制台程序，我们对三种情况分别进行测试：

情形一：一步跳出，如表1所示

表1 一步跳出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 输入 | |  | 理想输出 | 实际输出 |
| 3 | 43 |  | 1 | 1 |
| 1 | 5 |  |  |  |
| 10 | 40 |  |  |  |
| -25 | -19 |  |  |  |

情形二：不能跳出，如表2所示

表2 不能跳出

|  |  |  |  |
| --- | --- | --- | --- |
| 输入 | | 理想输出 | 实际输出 |
| 3 | 10 | -1 | -1 |
| 1 | 5 |  |  |
| 10 | 40 |  |  |
| -25 | -19 |  |  |

情形三：借助鳄鱼跳出，如表3所示

表3 借助鳄鱼跳出

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 输入 | | 理想输出 | | 理想输出 | |
| 8 | 20 | 3 |  | 3 |  |
| 18 | -1 | 18 | -1 | 18 | -1 |
| 16 | 14 | 33 | -2 | 33 | -2 |
| 5 | 26 |  |  |  |  |
| -19 | 22 |  |  |  |  |
| -34 | -11 |  |  |  |  |
| -5 | -24 |  |  |  |  |
| 27 | -15 |  |  |  |  |
| 33 | -2 |  |  |  |  |

对于图形化界面程序：

1. 测试实例一

当步长大于42.5时，007可以一步跳至岸边；在程序运行后在步长获取框中输入步长：43如图5所示，程序继续运行得到结果弹框：can jump to the side without the help of crocodile!!! 如图6所示； 其中我们把输入信息（鳄鱼数量、007的步长、鳄鱼坐标）保存在input.txt文件中，输出结果按题目要求保存在output.txt文件中如图7所示。

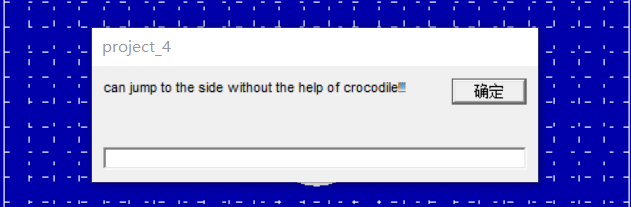


图5 测试步长实例一 图6 测试结果实例一

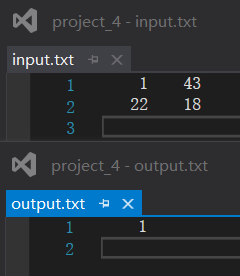
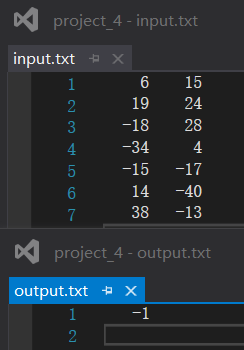
 

图7 文件内容实例一 图8 文件内容实例二

1. 测试实例二

当步长小于42.5时，因鳄鱼位置分布具有随机性，007存在不能借助鳄鱼跳出的情况；在程序运行后在步长获取框中输入步长：15，如图9所示，程序继续运行，按照一定规则放置几条鳄鱼使得007不能成功逃脱，继续运行得到结果弹框：can not jump to the side!!!如图10所示；其中我们把输入信息（鳄鱼数量、007的步长、鳄鱼坐标）保存在input.txt文件中，输出结果按题目要求保存在output.txt文件中如图8所示。

图9 测试步长实例二 图10 测试结果实例二

1. 测试实例三

当步长小于42.5时，因鳄鱼位置分布具有随机性，007存在可以借助鳄鱼跳出的情况；在程序运行后在步长获取框中输入步长：20，如图11所示，程序继续运行，按照一定规则放置几条鳄鱼使得007可以成功逃脱，继续运行得到结果弹框：can jump to the side with the help of crocodile!!!如图12所示；最后将经过的鳄鱼坐标给标识出并且开始一步一步的画出007所经过的路径如图13所示；其中我们把输入信息（鳄鱼数量、007的步长、鳄鱼坐标）保存在input.txt文件中，输出结果按题目要求保存在output.txt文件中如图14所示，最终路径展示界面保存在image文件夹中。

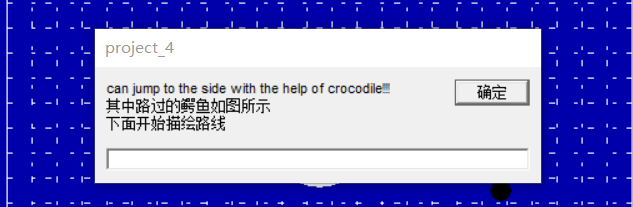
 

图11 测试步长实例三 图10 测试结果实例三

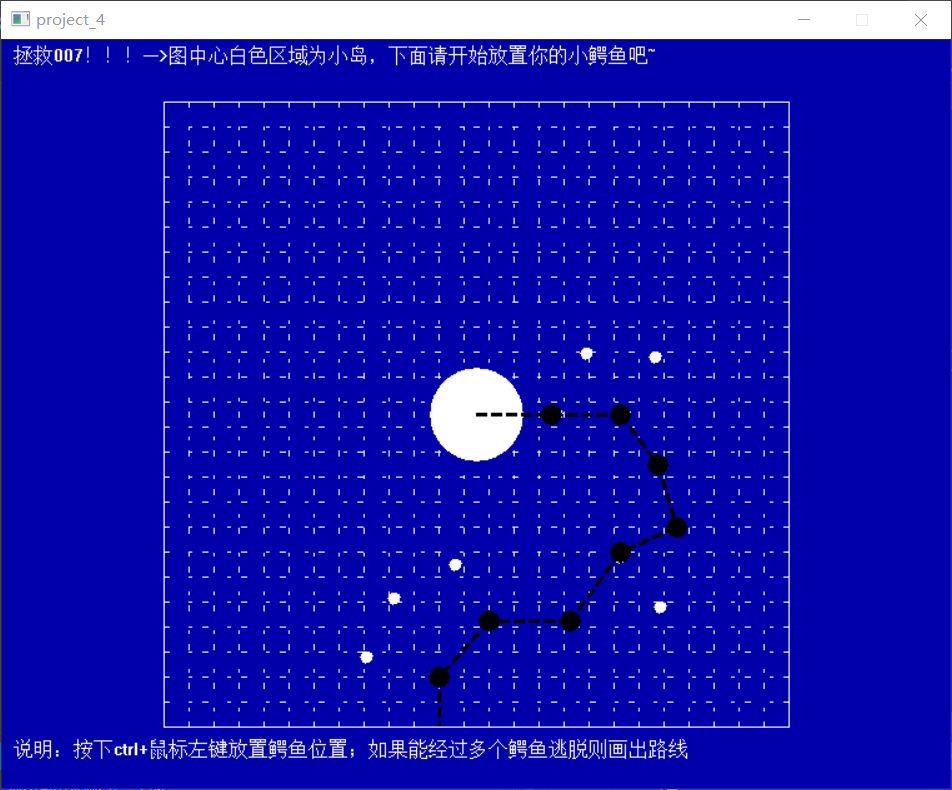
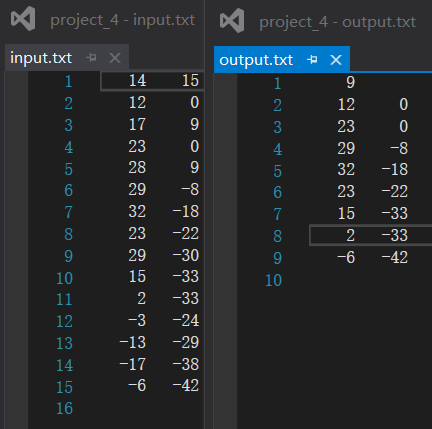
 

图11 测试实例三 图12 文件内容实例三

**六、算法设计和程序调试过程中的问题**

**问题1**：在获取鼠标在池塘上放置的鳄鱼的位置坐标时，当鳄鱼个数为个位数的时候保存到input.txt文件中的各项参数都正常；但是当鳄鱼个数到两位数以上时，input.txt文件中的参数就会发生错误，比如：第一行的应该只是鳄鱼数目和步长，实际上是鳄鱼数目和第一个鳄鱼的x坐标参数，而第二行的鳄鱼坐标只剩下y坐标；

**解决方法**：数据写入文件中的是追加的形式，因为一开始没有指定每个数据的宽度，当最后在第一行写入鳄鱼数目和步长的时候，因为修改后的长度大于修改前的长度，将会影响文件中的下一行数据。最后指定文件中每个数据的宽度是5个字符，这样的话就不会影响下面的内容，成功解决。

**问题2**：在绘制007跳过的路径时，所经过的鳄鱼的位置标识错误以及所画的路径不对，调试了好多次，对各个象限进行调试后发现只有第一象限能正常显示；

**解决方法**：经过printf相关参数后发现，部分参数比例变换存在问题（因为画界面的时候单位是像素，地图太小我放大了5倍，但是在获取output.txt文件中007路过的鳄鱼位置坐标变换到图形界面时忘记相关变换倍数，导致显示位置出错；最终改完后发现其实变换规则对于任何一个区域都适用，不需要分象限讨论。

**问题3**：调用结果提示框的时候Windows弹框总是显示在湖的界面的后面，人机交互体验感下降；尝试过n种办法后还是无效；

**解决方法**：经过查阅相关资料，发现easyx库下的InputBox可以一直显示在最上层，虽然多了一个输入框，但是可以起到弹框提示的作用，只好将就着用用了。

**七、课程设计总结**

最短路径----拯救007一题中，我采用的是队列图的广度优先算法来遍历图的，可广度那也可以考虑深度优先，采用堆栈的数据结构来记录所经过的路径。

在迷宫问题中，我采用的是默认向右转的思想来判断是不是能够走出迷宫，但是这样带来的问题就是不能以最短路径走出迷宫；迷宫求解利用的是栈结构，即深度优先，探索一个位置就标记，通则走，不同则后退寻找下一个位置，可以求出通路，简单但是不一定是最短路径。

改进设想：求最短路径利用广度优先的思想，利用队列实现，一个元素出对列，然后访问这个元素相邻的所有元素，原理是，一个二维数组，0表示墙，1表示路，利用随机数生成0和1，4个方向，在广度优先算法的思想下，队头元素出队，然后广度依次访问他的4个方向，依次入队，并记下他们的前一个坐标在队列中的位置，重复直到出对列的是终点，在找到终点后，利用每一个位置都有前一个坐标在队列中的下标进行回访，访问到起点即走了一遍找到的路径，此时便可正向输出路径即为最短路径。

此次课程设计对数据结构和算法的能力要求比较高，加强了我对算法与数据结构的了解。我可以通过此次课程设计很好的巩固堆栈、队列、广度搜索、深度搜索等知识。懂得了理论与实践相结合的重要性，有理论知识是远远不够的，理论学的再好，考试考的再高，如果不能将其应用到编程中都是徒劳的，实践动手更加重要。

通过课程设计我看到自己的实际编码能力的严重不足，理论知识不够深刻，掌握不够牢固，编程基础薄弱。这次课程设计使我发现自己的知识是如此贫乏，顿时感到了压力的存在，从而也激发了学习热情。

根据我在课程设计中遇到的问题，我将在以后的学习过程中认真上好专业实验课，多在实践中锻炼自己，在做设计的时候要有信心、有耐心，切忌浮躁/认真学习课本知识，掌握课本中的知识点，并在此基础上灵活运用。

**迷宫问题**

**一、课题内容和要求**

**课题内容：**

以一个m\*n的长方形表示迷宫，0和1分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的迷宫，求出从入口到出口的通路，或得到没有通路的结论。

“迷宫问题”的功能框架图如图1所示。

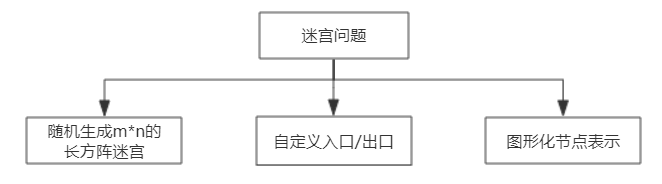


图1 功能框架图

**课题要求：**

（1）随机生成m\*n的长方阵迷宫，m、n>8；

（2）任意输入入口坐标点和出口坐标点，输出通行过程坐标点，或者不存在通行路径；

（3）使用图形化节点表示。

**二、数据结构说明**

1. 迷宫坐标位置的结构体

1. **typedef** **struct** {
2. **int** x;     //x坐标
3. **int** y;     //y坐标
4. }PosType;          //迷宫中坐标的位置

2. 栈中元素类型结构体

1. **typedef** **struct** {
2. **int** ord;
3. PosType seat;
4. **int** di;
5. }SElemType;//栈中元素类型

3. 栈类型结构体

1. **typedef** **struct** {
2. SElemType\* base;
3. SElemType\* top;
4. **int** staticsize;
5. }Stack;//栈类型

**三、算法设计**

1 路径规划算法

基于堆栈的思想，首先判断入口是否是通路；如果是通路就将此节点压入到栈中，并判断是否到达出口，没有到达出口默认向右转；如果当前位置不通，从栈中取出栈顶元素，当其遇到障碍后就撞墙回头，如果栈顶位置有其他方向可以选择，依照我们的方法默认右转，将其下一步压入栈中。

程序中对应主要函数为：

1. Status PassMaze(MazeType& maze, PosType start, PosType end, Stack& S) {
2. }

该函数主要功能是实现走出迷宫的算法。

2 路径展示算法

由算法1得到的堆栈中保存了走出迷宫经过路径的具体信息，只需要将堆栈中保留的信息取出来便可得到如何走出迷宫的解。

程序中对应的代码段为：

1. Status StackTraverse(Stack S, Status(\*visit)(SElemType)) {
2. //从栈底元素到栈顶元素依次执行visit函数，用于输出栈中元素
3. SElemType\* p = S.base;
4. **if** (S.base == S.top) printf\_s("空栈\n");
5. **else**
6. **while** (p < S.top) {
7. (\*visit)(\*p);
8. ++p;
9. }
10. **return** OK;
11. }

**四、详细设计**

main.cpp

1. //函数状态码定义
2. #define TRUE 1
3. #define FALSE 0
4. #define OK 1
5. #define ERROR 0
6. #define INFEASIBLE -1
7. #define NULL 0
9. //墙或通路及前进方向符合定义
10. #define WALL 0//当前格子是墙
11. #define PATH 1//当前是通路且未走过
12. #define RIGHT -1//代表是通路且从其向右走
13. #define DOWN -2//代表是通路且从其向下走
14. #define LEFT -3//代表是通路且从其向左走
15. #define UP -4//代表是通路且从其向上走
16. #define BACK -5//代表是通路且从其后退一步
17. #define DESTINATION -6//代表当前格子是通路且是目标位置
19. #define M 10
20. #define N 10
21. **typedef** **int** MazeType[M + 2][N + 2];//最外遭初始化墙，实际含(M-2)(N-2)个格子
22. **typedef** **int** Status;
23. **typedef** **int** ElemType;//迷宫数组中的元素类型
25. //栈的定义和实现，采用顺序存储结构
26. #define STACK\_INIT\_SIZE 100
27. #define STACKINCREMENT 10
29. **typedef** **struct** {
30. **int** x;
31. **int** y;
32. }PosType;//迷宫中坐标的位置
34. **typedef** **struct** {
35. **int** ord;
36. PosType seat;
37. **int** di;
38. }SElemType;//栈中元素类型
40. **typedef** **struct** {
41. SElemType\* base;
42. SElemType\* top;
43. **int** staticsize;
44. }Stack;//栈类型
46. Status InitStack(Stack& S) {
47. //构造空栈S
48. S.base = (SElemType\*)malloc(STACK\_INIT\_SIZE \* **sizeof**(SElemType));
49. **if** (!S.base) exit(OVERFLOW);//存储分配失败
50. S.top = S.base;//空栈
51. S.staticsize = STACK\_INIT\_SIZE;
52. **return** OK;
53. }//初始化栈
55. Status Push(Stack& S, SElemType e) {
56. //插入e为栈顶元素
57. **if** (S.top - S.base >= S.staticsize)//栈满则重新分配空间
58. {
59. S.base = (SElemType\*)realloc(S.base, (S.staticsize + STACKINCREMENT) \* **sizeof**(SElemType));
60. **if** (!S.base) exit(OVERFLOW);
61. S.top = (S.base + S.staticsize);//使得S.top重新指向原栈顶，因为realloc
62. S.staticsize += STACK\_INIT\_SIZE;
63. }
64. \*S.top++ = e;//top指向待插入位置
65. **return** OK;
66. }
67. Status Pop(Stack& S, SElemType& e) {
68. //若栈不空则栈顶元素出栈并用e带回其值
69. **if** (S.top == S.base) **return** ERROR;
70. **else**
71. e = \*(--S.top);//栈顶元素为\*(S.top-1)
72. **return** OK;
73. }
74. Status GetTop(Stack S, SElemType& e) {
75. **if** (S.top == S.base) **return** ERROR;
76. e = \*(S.top - 1);//注意top指向待插入位置
77. **return** OK;
78. }
79. Status StackEmpty(Stack S) {//判空
80. **if** (S.top == S.base) **return** TRUE;
81. **else**
82. **return** FALSE;
83. }
84. Status StackTraverse(Stack S, Status(\*visit)(SElemType)) {
85. //从栈底元素到栈顶元素依次执行visit函数，用于输出栈中元素
86. SElemType\* p = S.base;
87. **if** (S.base == S.top) printf\_s("空栈\n");
88. **else**
89. **while** (p < S.top) {
90. (\*visit)(\*p);
91. ++p;
92. }
93. **return** OK;
94. }
95. //输出走出迷宫的路径
96. Status PrintSElem(SElemType e) {
97. printf\_s("step:%d to (%d,%d)\n", e.ord, e.seat.x, e.seat.y);
98. **return** OK;
99. }

102. //迷宫求解具体算法
103. Status MakeMaze(MazeType& maze) {//生成迷宫，0表示PATH，1表示WALL
104. PosType m;
105. srand(time(NULL));
106. **for** (m.y = 0; m.y <= N + 1; m.y++) { maze[0][m.y] = WALL; maze[M + 1][m.y] = WALL; }//表示上下的墙
107. **for** (m.x = 1; m.x <= M; m.x++) { maze[m.x][0] = WALL; maze[m.x][N+1] = WALL; }//表示左右的墙
108. **for** (m.x = 1; m.x <= M; m.x++)
109. **for** (m.y = 1; m.y <= N; m.y++)
110. maze[m.x][m.y] = rand() % 2;                                        //随机生成迷宫
111. **return** OK;
112. }
114. //■◎
115. //图形化节点表示迷宫
116. **void** PrintMaze(MazeType maze) {
117. **int** x, y;
118. **for** (x = 0; x <= M+1; x++) {
119. **for** (y = 0; y <= N+1; y++) {
120. **switch** (maze[x][y]) {
121. **case** WALL:printf\_s("■"); **break**;
122. **case** PATH:printf\_s("  "); **break**;
123. **case** RIGHT:printf\_s("→"); **break**;
124. **case** DOWN:printf\_s("↓"); **break**;
125. **case** LEFT:printf\_s("←"); **break**;
126. **case** UP:printf\_s("↑"); **break**;
127. **case** BACK:printf\_s("@ "); **break**;
128. **case** DESTINATION:printf\_s("◎"); **break**;
129. **default**:printf\_s("error\n"); exit(-1);
130. }
131. }
132. printf\_s("\n");
133. }
134. }
136. //表示下一步往哪走
137. PosType Nextpos(PosType position, ElemType direction) {
138. PosType result;
139. result = position;
140. **switch** (direction)
141. {
142. **case** 1:result.y++; **break**;
143. **case** 2:result.x++; **break**;
144. **case** 3:result.y--; **break**;
145. **case** 4:result.x--; **break**;
146. }
147. **return** result;
148. }
150. //主要算法
151. Status PassMaze(MazeType& maze, PosType start, PosType end, Stack& S) {
152. PosType curpos;
153. SElemType e;
154. **int** curstep = 1;
155. curpos = start;
156. **if** (maze[curpos.x][curpos.y] != PATH) {             //判断入口是否通
157. printf\_s("当前迷宫没有入口\n");
158. **return** FALSE;
159. }
160. **do** {
161. **if** (maze[curpos.x][curpos.y] == PATH) {            //当前位置是通道
162. e.ord = curstep;
163. e.seat = curpos;
164. e.di = 1;
165. Push(S, e);                                    //压栈
166. **if** (curpos.x == end.x && curpos.y == end.y) {  //判断是否到达出口
167. maze[curpos.x][curpos.y] = DESTINATION;
168. **return** OK;
169. }
170. **else** {
171. maze[curpos.x][curpos.y] = RIGHT;          //向右
172. curpos = Nextpos(curpos, 1);
173. curstep++;
174. }
175. }
176. **else** {                                         //当前位置不通
177. GetTop(S, e);                              //取栈顶元素
178. **while** (!StackEmpty(S) && e.di == 4) {
179. maze[e.seat.x][e.seat.y] = BACK;       //撞墙回头
180. Pop(S, e);  //出栈
181. curstep--;
182. **if** (StackEmpty(S)) **break**;
183. GetTop(S, e);
184. }
185. **if** (e.di < 4) {                           //栈顶位置有其他方向可以选择
186. Pop(S, e);
187. e.di++;
188. Push(S, e); maze[e.seat.x][e.seat.y] = -e.di;//注意前进方向踪迹与e.di互为相反数，因为是向右转
189. curpos = Nextpos(e.seat, e.di);
190. }
191. }
192. } **while** (!StackEmpty(S));
193. **return** FALSE;
194. }

197. **int** main()
198. {
199. MazeType maze;
200. PosType start, end;
201. Stack S;
202. InitStack(S);
203. MakeMaze(maze);
204. printf\_s("迷宫初始为：\n");
205. PrintMaze(maze);
206. printf\_s("输入迷宫的入口位置坐标从(0,0)到(%d,%d)：", M, N);
207. scanf\_s("%d %d", &start.x, &start.y);
208. printf\_s("输入迷宫的出口位置坐标从(0,0)到(%d,%d)：", M, N);
209. scanf\_s("%d %d", &end.x, &end.y);
210. **if** (PassMaze(maze, start, end, S)) {
211. printf\_s("迷宫可通，路径踪迹如下：\n");
212. PrintMaze(maze);
213. printf\_s("具体路径为：\n");
214. StackTraverse(S, PrintSElem);
215. }
216. **else** {
217. printf\_s("\n迷宫不可通\n");
218. //PrintMaze(maze);
219. }
220. **return** 0;
221. }

**五、测试数据及其结果分析**

1. 入口位置坐标有障碍

当迷宫的入口位置坐标处对应的是障碍时，程序会提示“当前迷宫没有入口”；如图2所示。

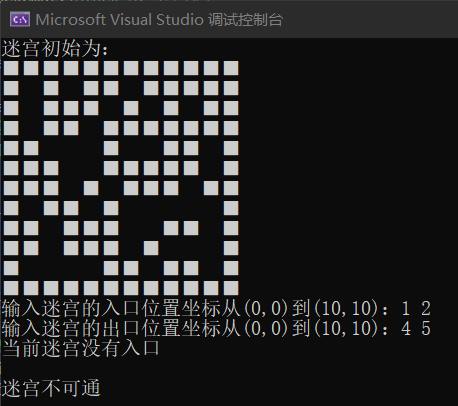
 

图2 入口有障碍 图3 出口有障碍

2. 出口位置坐标有障碍

当迷宫的出口位置坐标处对应的是障碍时，程序会提示“迷宫不可通”；如图3所示。

3. 入口出口位置坐标均合理且有通路

当迷宫的入口和出口位置坐标处均是通路，且入口出口相通的时候，输入入口出口位置坐标后程序运行，得到图形化节点表示的迷宫以及所走的路径，如图4所示；其中箭头代表所走的方向，@表示遇到障碍回头，◎表示出口位置；

在迷宫下方还会打印出具体走出迷宫经过的路径，如图5所示。

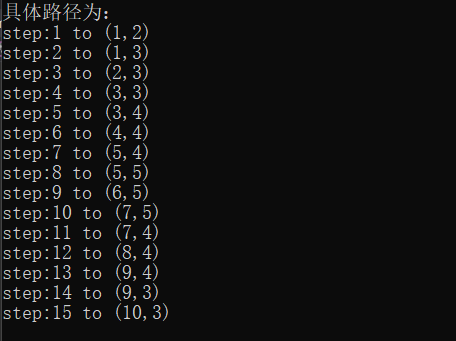
 

图4 迷宫路径 图5 具体路径

4. 入口出口位置坐标均合理但无通路

当迷宫的入口和出口位置坐标处均是通路，但是入口出口不相通的时候，输入入口出口位置坐标后程序运行，程序会提示“迷宫不可通”；如图6所示。



图6 入出口不相通

**六、算法设计和程序调试过程中的问题**

**问题1**： 在将迷宫用图形化节点表示的时候，发现在控制台输出迷宫时会出现错位的情况，导致不能直观观察迷宫的行和列；

**解决方法**： 在PrintMaze函数中，我将几种情况对应的输出符号（"■"、" "、"→"、"↓"、"←"、"↑"、"@ "、"◎"）对齐了观察，发现空格和@符合只占了一个字符，其他的均占了两个字符，因此通过将这两个改为两字符大小解决了错位的问题。

**问题2**： 在前进路上遇到障碍时，栈顶有其他方向可以选择的时候，因为方向不定，导致不便确认先往哪个方向转，不能产生合适的结果；

**解决方法**： 由于之前定义了迷宫内部右转优先原则，我定义了各个方向对应的状态值，当右转时该方向状态值的基础上+1就可以成功代表转换后的方向，有效解决冲突。

**问题3**： 一开始没有采用动态分配栈空间的方式，导致运行报错栈的问题；

**解决方法**： 使用动态空间分配方法为堆分配内存空间

S.base = (SElemType\*)realloc(S.base, (S.staticsize + STACKINCREMENT)

\* sizeof(SElemType));从而解决了栈的分配问题。