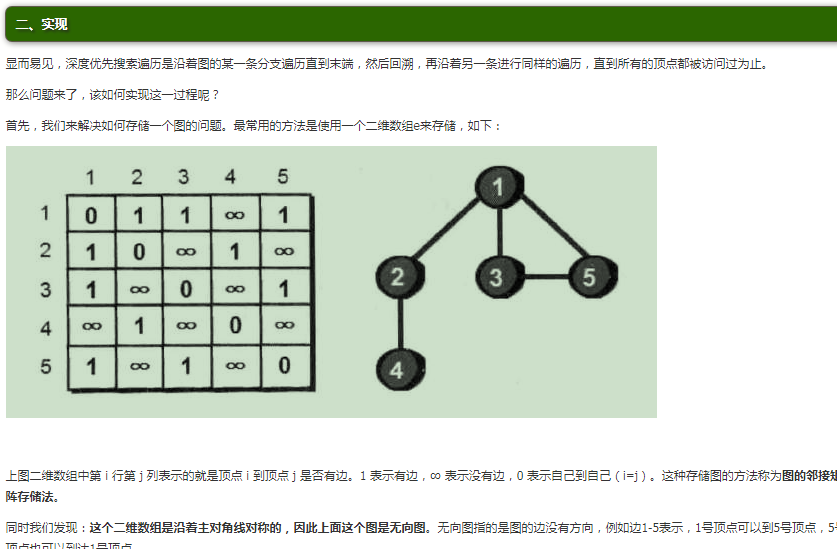
实验题目：深度优先遍历无向图

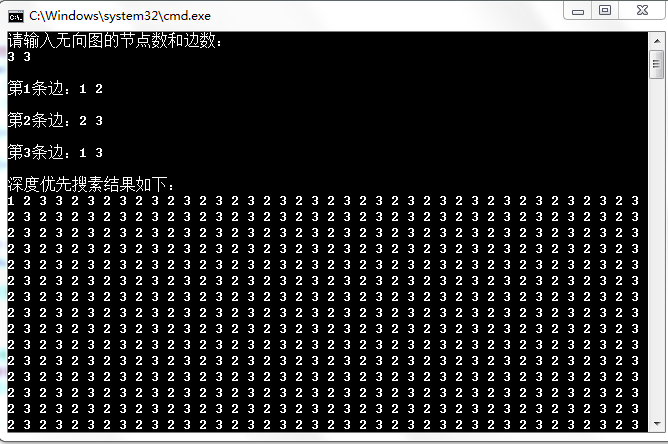
本次实验我结合在网上查阅的资料完成，原网页如下<https://www.cnblogs.com/OctoptusLian/p/8260173.html>这是一个深度优先遍历无向图的题目，是对深度优先算法的一个经典的应用，虽然代码很短，但很好的的体现了算法的思想.



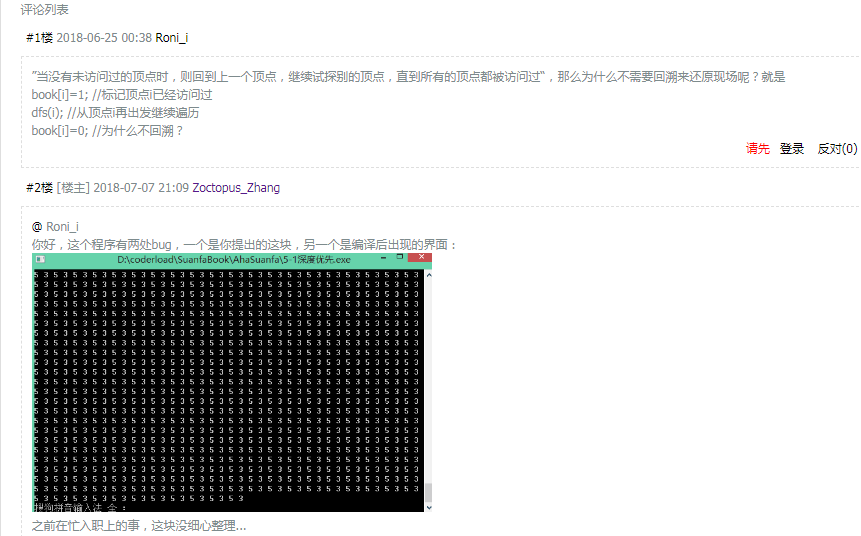
1. 实验中遇到了一些问题，网页中的源代码在运行之后，经过几次测试并不能得到想要的结果，由于我对深度优先算法的理解并不深入，我查阅了大二学习的数据结构课本，复习了深度优先遍历无向图的过程，对比之下发现网页的思路与课本一致，可见算法的大方向并没有错误，
2. 于是我仔细观察的出错的测试样例，当时以为是算法无法正确遍历含有闭合回路的无向图

但检查了算法的递归过程之后发现这部分也没有bug

1. 然后我观察结果中可能存在死循环

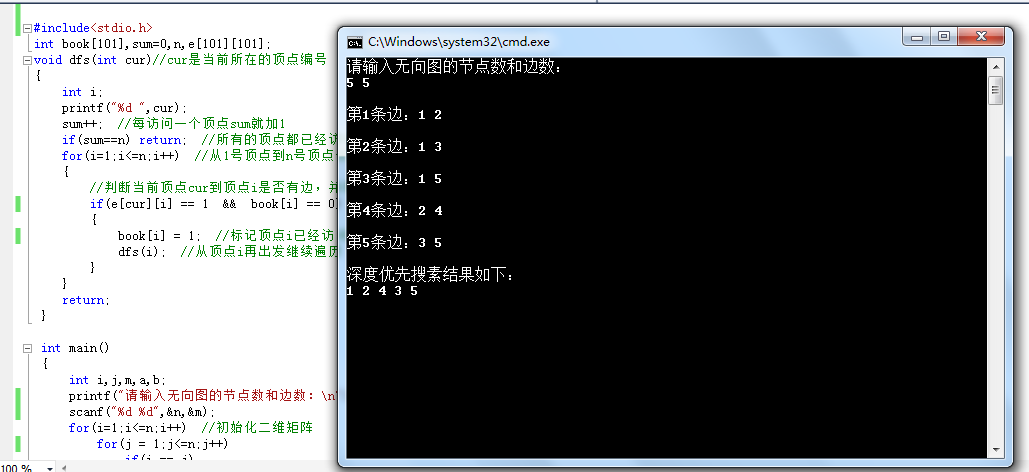


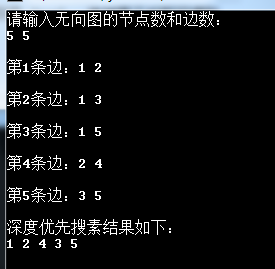
于是检查了函数的边界，sum，n等参数都没有问题，至此经过看书，查资料我已经对dfs算法有了更深入的理解，确信问题出在代码中存在死循环，而原网页中也有人评论存在bug，如下



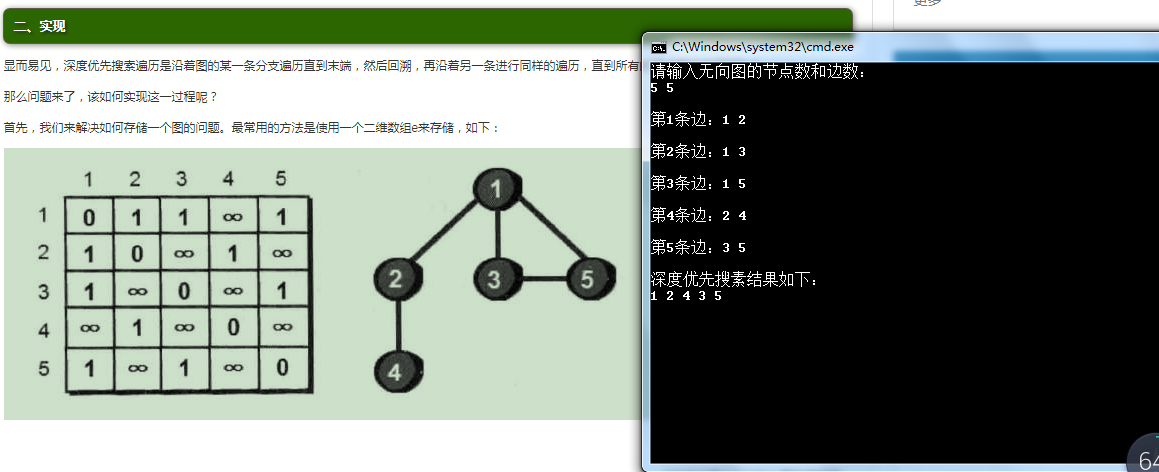
但都没有给出解决方案

4.于是我使用编译器的调试功能一步一步调试，发现死循环的出现与边界无关，而是存在于原网页内容作者 的笔误，在递归函数中访问标记数组book[i]=1赋值符号“=”写成了判断相等符号“==”，将其更正后即可得到正确结果如下





与题目对比如下



1. 至此本次实验成功完成，通过debug的过程我学习了更多的关于深度优先搜索的知识
2. 实验的源程序如下：
3. #include<stdio.h>
4. int book[101],sum=0,n,e[101][101];
5. void dfs(int cur)//cur是当前所在的顶点编号
6. {
7. int i;
8. printf("%d ",cur);
9. sum++; //每访问一个顶点sum就加1
10. if(sum==n) return; //所有的顶点都已经访问过则直接退出
11. for(i=1;i<=n;i++) //从1号顶点到n号顶点依次尝试，看哪些顶点与当前顶点cur有边相连
12. {
13. //判断当前顶点cur到顶点i是否有边，并判断顶点i是否已访问过
14. if(e[cur][i] == 1 && book[i] == 0)
15. {
16. book[i] = 1; //标记顶点i已经访问过
17. dfs(i); //从顶点i再出发继续遍历
18. }
19. }
20. return;
21. }
22. int main()
23. {
24. int i,j,m,a,b;
25. printf("请输入无向图的节点数和边数：\n");
26. scanf("%d %d",&n,&m);
27. for(i=1;i<=n;i++) //初始化二维矩阵
28. for(j = 1;j<=n;j++)
29. if(i == j)
30. e[i][j]=0;
31. else
32. e[i][j]=99999999; //在这里假设99999999为正无穷
34. //读入顶点之间的边
35. for(i=1;i<=m;i++)
36. {
37. printf("\n第%d条边：",i);
38. scanf("%d %d",&a,&b);
39. e[a][b] = 1;
40. e[b][a] = 1;//这里是无向图，所以需要将e[b][a]也赋为1
41. }
43. //从1号顶点出发
44. book[1]=1; //标记1号顶点已被访问
45. printf("\n深度优先搜素结果如下：\n");
46. dfs(1); //从1号顶点开始遍历
48. getchar();getchar();
49. return 0;
50. }

熟悉了邻接矩阵为储存结构的深度优先搜索之后，我在网上找到了以邻接表为存储结构的深度优先搜索算法，并进行了编译和理解，c++代码如下

1. #include <iostream>
3. using namespace std;
5. //访问标志的数组,为1表示访问过，为0表示未被访问
6. int visted[100];
7. //边表结点
8. typedef struct EdgeNode {
9. //顶点对应的下标
11. int adjvex;
13. //指向下一个邻接点
15. struct EdgeNode \*next;
16. } edgeNode;
17. //顶点表结点
18. typedef struct VertexNode {
19. //顶点数据
21. char data;
23. //边表头指针
25. edgeNode \*firstedge;
26. } VertexNode, AdjList[100];
28. //集合
29. typedef struct {
30. AdjList adjList;
31. //顶点数和边数
32. int numVertexes, numEdges;
33. } GraphAdjList;
34. class AdjacencyList {
35. public:
37. void CreateALGraph(GraphAdjList \*G);
39. void ShowALGraph(GraphAdjList \*G);
41. void DFS(GraphAdjList \*G, int i);
43. void DFSTraverse(GraphAdjList \*G);
45. void Test();

48. };
49. void AdjacencyList::CreateALGraph(GraphAdjList \*G) {
50. int i, j, k;
51. edgeNode \*e;
52. cout << "输入顶点数和边数" << endl;
53. //输入顶点数和边数
54. cin >> G->numVertexes >> G->numEdges;
55. //读入顶点信息，建立顶点表
56. for (i = 0; i < G->numVertexes; i++)
57. {
58. //输入顶点信息
59. cin >> G->adjList[i].data;
60. //将边表置为空表
61. G->adjList[i].firstedge = NULL;
63. }
64. //建立边表（头插法）
66. for (k = 0; k < G->numEdges; k++)
67. {
68. cout << "输入边（vi,vj）上的顶点序号" << endl;
69. cin >> i >> j;
70. e = new EdgeNode;
71. e->adjvex = j;
72. e->next = G->adjList[i].firstedge;
73. G->adjList[i].firstedge = e;
75. e = new EdgeNode;
77. e->adjvex = i;
78. e->next = G->adjList[j].firstedge;
79. G->adjList[j].firstedge = e;
81. }
82. }
83. void AdjacencyList::Test() {
84. cout << "ALL IS OK." << endl;
85. }
86. void AdjacencyList::ShowALGraph(GraphAdjList \*G) {
87. for (int i = 0; i < G->numVertexes; i++)
88. {
89. cout << "顶点" << i << ": " << G->adjList[i].data << "--firstedge--";
90. edgeNode \*p = new edgeNode;
91. p = G->adjList[i].firstedge;
92. while (p)
93. {
94. cout << p->adjvex << "--Next--";
95. p = p->next;
97. }
98. cout << "--NULL" << endl;
99. }
101. }
102. void AdjacencyList::DFS(GraphAdjList \*G, int i) {
103. EdgeNode \*p;
104. visted[i] = 1;
105. cout << G->adjList[i].data << "--";
106. p = G->adjList[i].firstedge;
107. while (p)
108. {
109. if (!visted[p->adjvex])
110. {
111. //递归访问
112. DFS(G, p->adjvex);
114. }
115. p = p->next;
116. }
118. }
119. void AdjacencyList::DFSTraverse(GraphAdjList \*G) {
120. //初始化所有顶点都没有访问过
121. cout<<"深度优先遍历结果为："<<endl;
122. for (int i = 0; i < G->numVertexes; i++)
123. {
124. visted[i] = 0;
125. }
126. for (int i = 0; i < G->numVertexes; i++)
127. {
128. if (visted[i] == 0)
129. {
130. DFS(G, i);
131. }
132. }
133. }
134. int main() {
136. AdjacencyList adjacencyList;
137. GraphAdjList \*GA = new GraphAdjList;
138. adjacencyList.Test();
139. adjacencyList.CreateALGraph(GA);
140. adjacencyList.ShowALGraph(GA);
141. adjacencyList.DFSTraverse(GA);
142. return 0;
143. }