**图搜索BFS算法及存储优化**

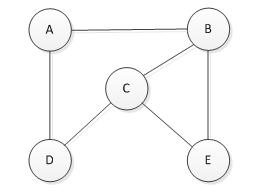
SA20225085 朱志儒

## 实验内容

**实验要求：**

针对无向图，通过邻接多重表方式进行存储。以节点A为起始节点，输出图的广度优先遍历的过程。

例如，对下图广度优先遍历的过程为：A-B-D-C-E。



邻接多重表的定义可参考：

<https://blog.csdn.net/bible_reader/article/details/71250117>

<https://book.itheima.net/course/223/1276707762369208322/1276709410189615105>

文件第一行为图的节点，接下来的行为边集。

**程序输入：**

文件data.txt。

**程序输出：**

图的广度优先遍历过程。

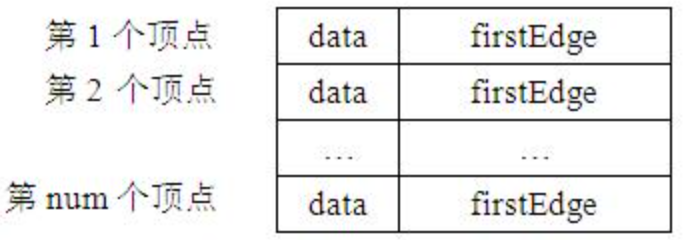
## 实验目的

熟练掌握广度优先搜索算法和邻接多重表。

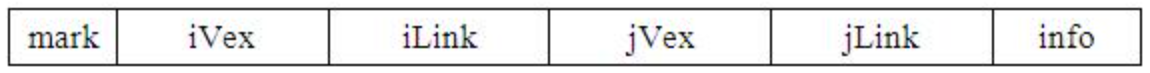
## 算法设计思路

数据结构：

顶点数组如下：



边结点数据结构如下：



其中mark表示标志位，用于标记该边是否已经被访问过；iVex和jVex表示该边的两个顶点在顶点数组中的位置；iLink和jLink分别表示指向依附于顶点iVex和jVex下一条边的指针。

广度优先遍历算法步骤：

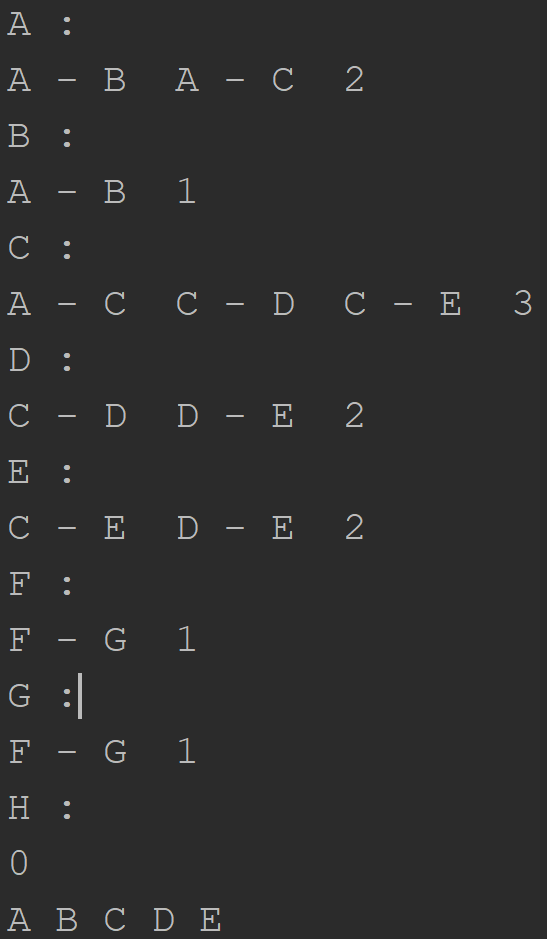
1. 首先将根节点放入队列中。
2. 从队列中取出第一个节点，访问它，将它所有尚未检验过的直接子节点加入队列中。
3. 若队列为空，表示整张图都遍历过了，结束算法。
4. 若队列不为空，则重复步骤2。

## 源码+注释

1. **class** Edge:
2. **def** \_\_init\_\_(self, mark = None, iVex = None, iLink = None, jVex = None, jLink = None):
3. self.mark = mark
4. self.iVex = iVex
5. self.iLink = iLink
6. self.jVex = jVex
7. self.jLink = jLink
9. **def** add\_edge(edges, edge, i):
10. **if** edges[i] **is** None:
11. edges[i] = edge
12. **else**:
13. ptr = edges[i]
14. **while** (ptr.iVex == i **and** ptr.iLink **is** **not** None) **or** (ptr.jVex == i **and** ptr.jLink **is** **not** None):
15. **if** ptr.iVex == i:
16. ptr = ptr.iLink
17. **else**:
18. ptr = ptr.jLink
19. **if** ptr.iVex == i:
20. ptr.iLink = edge
21. **else**:
22. ptr.jLink = edge
24. **def** read\_data():
25. file = open('data.txt', 'r')
26. vexs = file.readline().strip().split(',')
27. edges = [None **for** i **in** vexs]
28. **for** line **in** file.readlines():
29. i = vexs.index(line[0])
30. j = vexs.index(line[2])
31. edge = Edge(False, i, None, j, None)
32. add\_edge(edges, edge, i)
33. add\_edge(edges, edge, j)
34. **return** vexs, edges
36. **def** print\_edges(vex, edges):
37. **for** i **in** range(len(vexs)):
38. ptr = edges[i]
39. count = 0
40. **print**(vex[i], ':')
41. **while** ptr **is** **not** None:
42. count += 1
43. **print**(vexs[ptr.iVex], '-', vexs[ptr.jVex], end='  ')
44. **if** ptr.iVex == i:
45. ptr = ptr.iLink
46. **else**:
47. ptr = ptr.jLink
48. **print**(count)
50. **def** BFS(vexs, edges):
51. visited = [False **for** i **in** vexs]
52. queue = []
53. queue.append(0)
54. visited[0] = True
55. **while** len(queue) != 0:
56. index = queue.pop(0)
57. **print**(vexs[index], end=' ')
58. ptr = edges[index]
59. **while** ptr **is** **not** None:
60. **if** ptr.iVex == index:
61. **if** **not** visited[ptr.jVex]:
62. queue.append(ptr.jVex)
63. visited[ptr.jVex] = True
64. ptr = ptr.iLink
65. **else**:
66. **if** **not** visited[ptr.iVex]:
67. queue.append(ptr.iVex)
68. visited[ptr.iVex] = True
69. ptr = ptr.jLink
71. **if** \_\_name\_\_ == "\_\_main\_\_":
72. vexs, edges = read\_data()
73. print\_edges(vexs, edges)
74. BFS(vexs, edges)

## 算法测试结果

打印邻接多重表和广度优先搜索（从A开始）的结果：



从结果中可以看出，A没有与F，G，H连通。

## 实验过程中遇到的困难及收获

在本次实验中，唯一的难点在于构造邻接多重表。

邻接多重表与邻接表的不同之处是，对于无向图中的边，邻接表存储两个而邻接多重表只存储一个。