图搜索 BFS 算法及存储优化

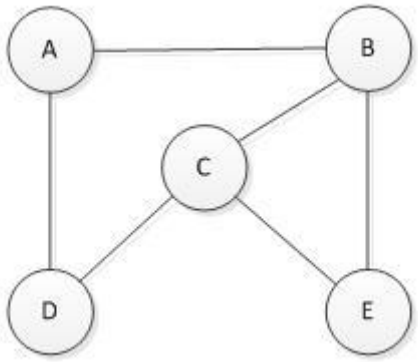
SA20225085 朱志儒

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/223/127670776236920832 2/1276709410189615105

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：

第 1 个顶点	data	firstEdge
第 2 个顶点	data	firstEdge
	…	…
第 num 个顶点	data	firstEdge

边结点数据结构如下：

mark	iVex	iLink	jVex	jLink	info

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
8.
9.  def add_edge(edges, edge, i):
10.     if edges[i] is None:
11.         edges[i] = edge
```

```
    else:
        ptr = edges[i]
        while (ptr.iVex == i and ptr.iLink is not None) or (ptr.jVex == i an
d ptr.jLink is not None):
            if ptr.iVex == i:
                ptr = ptr.iLink
            else:
                ptr = ptr.jLink
        if ptr.iVex == i:
            ptr.iLink = edge
        else:
            ptr.jLink = edge

def read_data():
    file = open('data.txt', 'r')
    vexs = file.readline().strip().split(',')
    edges = [None for i in vexs]
    for line in file.readlines():
        i = vexs.index(line[0])
        j = vexs.index(line[2])
        edge = Edge(False, i, None, j, None)
        add_edge(edges, edge, i)
        add_edge(edges, edge, j)
    return vexs, edges

def print_edges(vex, edges):
    for i in range(len(vexs)):
        ptr = edges[i]
        count = 0
        print(vex[i], ':')
        while ptr is not None:
            count += 1
            print(vexs[ptr.iVex], '-', vexs[ptr.jVex], end='  ')
            if ptr.iVex == i:
                ptr = ptr.iLink
            else:
                ptr = ptr.jLink
        print(count)

def BFS(vexs, edges):
    visited = [False for i in vexs]
    queue = []
    queue.append(0)
    visited[0] = True
```

```
    while len(queue) != 0:
        index = queue.pop(0)
        print(vexs[index], end=' ')
        ptr = edges[index]
        while ptr is not None:
            if ptr.iVex == index:
                if not visited[ptr.jVex]:
                    queue.append(ptr.jVex)
                    visited[ptr.jVex] = True
                ptr = ptr.iLink
            else:
                if not visited[ptr.iVex]:
                    queue.append(ptr.iVex)
                    visited[ptr.iVex] = True
                ptr = ptr.jLink

if __name__ == "__main__":
    vexs, edges = read_data()
    print_edges(vexs, edges)
    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：

```
A :
A - B  A - C  2
B :
A - B  1
C :
A - C  C - D  C - E  3
D :
C - D  D - E  2
E :
C - E  D - E  2
F :
F - G  1
G :
F - G  1
H :
0
A B C D E
```

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。