**实验二 广度优先搜索**

一.实验目的

编写实现广度优先搜索的算法。

二.实验准备

如下二叉树

8

6

5

7

4

2

3

1

三.实验内容

源代码：python实现

**from** collections **import** deque  
**from** collections **import** namedtuple  
  
  
**def** bfs(start\_node, end\_node, graph): *# 开始节点 目标节点 图字典* node = namedtuple(**'node'**, **'name, from\_node'**) *# 使用namedtuple定义节点，用于存储前置节点* search\_queue = deque() *# 使用双端队列，这里当作队列使用，根据先进先出获取下一个遍历的节点* name\_search = deque() *# 存储队列中已有的节点名称* visited = {} *# 存储已经访问过的节点* search\_queue.append(node(start\_node, **None**)) *# 填入初始节点，从队列后面加入* name\_search.append(start\_node) *# 填入初始节点名称* path = [] *# 用户回溯路径* path\_len = 0 *# 路径长度* print(**'开始搜索...'**)  
 **while** search\_queue: *# 只要搜索队列中有数据就一直遍历下去* print(**'待遍历节点: '**, name\_search)  
 current\_node = search\_queue.popleft() *# 从队列前边获取节点，即先进先出，* name\_search.popleft() *# 将名称也相应弹出* **if** current\_node.name **not in** visited: *# 当前节点是否被访问过* print(**'当前节点: '**, current\_node.name, end=**' | '**)  
 **if** current\_node.name == end\_node: *# 退出条件，找到了目标节点，接下来执行路径回溯和长度计算* pre\_node = current\_node

**while True**: *# 开启循环直到找到开始节点* **if** pre\_node.name == start\_node: *# 退出条件：前置节点为开始节点* path.append(start\_node) *# 退出前将开始节点也加入路径，保证路径的完整性* **break  
 else**:  
 path.append(pre\_node.name) *# 不断将前置节点名称加入路径* pre\_node = visited[pre\_node.from\_node] *# 取出前置节点的前置节点，依次类推* path\_len = len(path) - 1 *# 获得完整路径后，长度即为节点个数-1* **break  
 else**:  
 visited[current\_node.name] = current\_node *# 如果没有找到目标节点，将节点设为已访问，并将相邻节点加入搜索队列，继续找下去* **for** node\_name **in** graph[current\_node.name]: *# 遍历相邻节点，判断相邻节点是否已经在搜索队列* **if** node\_name **not in** name\_search: *# 如果相邻节点不在搜索队列则进行添加* search\_queue.append(node(node\_name, current\_node.name))  
 name\_search.append(node\_name)print('搜索完毕') *# 打印搜索结果*

**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
  
 graph = dict() graph[1] = [3, 2]  
 graph[2] = [5]  
 graph[3] = [4, 7]  
 graph[4] = [6]  
 graph[5] = [6]  
 graph[6] = [8]  
 graph[7] = [8]  
 graph[8] = []  
 bfs(1, 8, graph) *# 执行搜索*

实验结果：

