宽度优先搜索算法

171491103 米日阿地·买买提明 硬件一班

1. **实验题目**

广度优先搜索算法（BFS）

**二.算法简介**

宽度优先搜索算法（又称广度优先搜索）是最简便的图的搜索算法之一，这一算法也是很多重要的图的算法的原型。Dijkstra单源最短路径算法和Prim最小生成树算法都采用了和宽度优先搜索类似的思想。其别名又叫BFS，属于一种盲目搜寻法，目的是系统地展开并检查图中的所有节点，以找寻结果。换句话说，它并不考虑结果的可能位置，彻底地搜索整张图，直到找到结果为止。

三．**算法实现步骤**

（1）、把根节点放到队列的末尾。

（2）、每次从队列的头部取出一个元素，查看这个元素所有的下一级元素，把它们放到队列的末尾。并把这个元素记为它下一级元素的前驱。

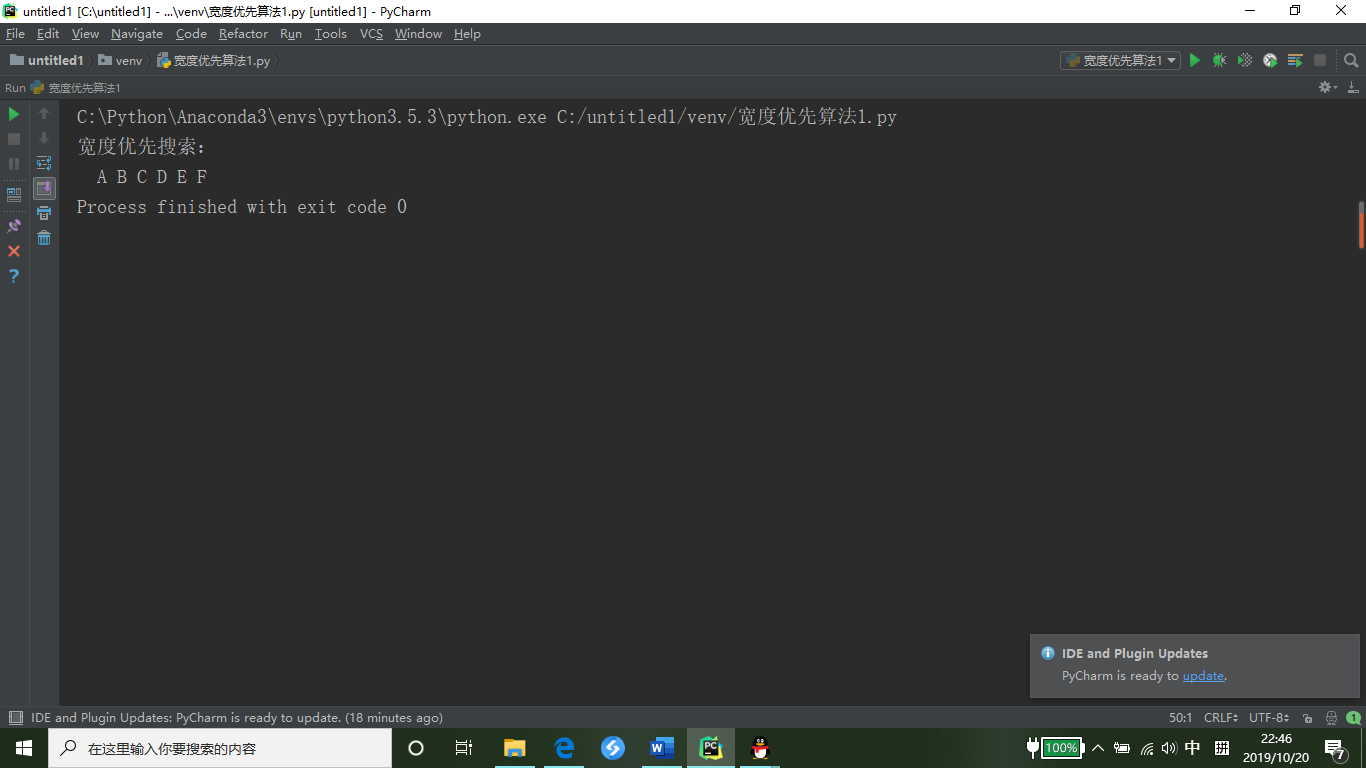
（3）、找到所要找的元素时结束程序。

（4）、如果遍历整个树还没有找到，结束程序。目标。结束搜索并回传“找不到目标”。

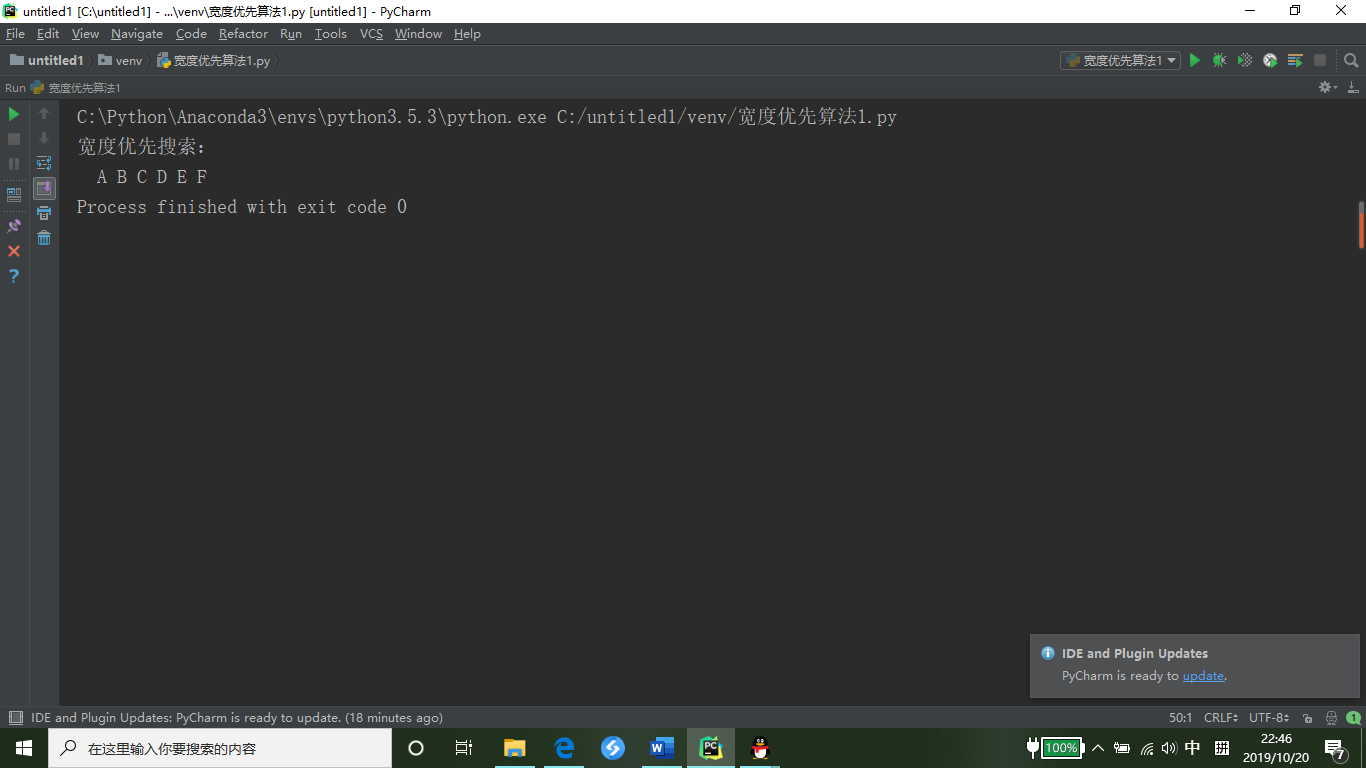
**三：实验代码**

from collections import deque  
class Graph(object):  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 self.order = []  
 self.neighbor = {}  
  
 def add\_node(self, node):  
 key, val = node  
 if not isinstance(val,list):  
 print('节点输入时应该为一个线性表')  
 self.neighbor[key] = val  
  
  
 def BFS(self, root):  
 if root !=None:  
 search\_queue = deque()  
 search\_queue.append(root)  
 visited = []  
 else:  
 print('root is None')  
 return -1  
  
 while search\_queue:  
 person = search\_queue.popleft()  
 self.order.append(person)  
  
 if(not person in visited) and (person in self.neighbor.keys()):  
 search\_queue += self.neighbor[person]  
 visited.append(person)  
  
 def clear(self):  
 self.order = []  
  
 def node\_print(self):  
 for index in self.order:  
 print(index, end=' ')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 g = Graph()  
 g.add\_node(('A',['B','C']))  
 g.add\_node(('B',['D','E']))  
 g.add\_node(('C',['F']))  
  
 g.BFS('A')  
 print('宽度优先搜索：')  
 print(' ', end=' ')  
 g.node\_print()  
 g.clear()

**四：实验截图**



**五：实验总结**



如果有两个节点同时指向下一个节点，如C1、C4都指向C3，这时C3再队列中会显示两次，但对于判断来说只需判断一次

如果图中存在无向图（无向图即双向关系），这种情况下会生成循环队列，使用此标记可以避免循环队列带来的冗余计算