```
1  # test_graph = {'A': set(['B', 'C']),
2  #           'B': set(['A', 'D', 'E']),
3  #           'C': set(['A', 'F']),
4  #           'D': set(['B']),
5  #           'E': set(['B', 'F']),
6  #           'F': set(['C', 'E'])}
7
8  test_graph = {'1': set(['2', '3','6']),
9           '2': set(['4', '1']),
10          '3': set(['1', '5']),
11          '5': set(['3','6','7']),
12          '6': set(['4', '1','5']),
13          '4': set(['2', '6']),
14              '7': set(['5'])}
15
16
17 vertices = []
18 path_list = []
19
20
21 for key in test_graph:
22     vertices.append(key)
23 #this builds a list of vertices.only done once.
24
25 #this is pretty much a normal bfs that returns shortest and longest path
26
27 def bfs_paths(adj_list, begin, end):
28     queue = [(begin, [begin])]
29     #this is our original queue, with our begin node, and out list
30     while len(queue) != 0:
31        #while the queue still exists
32         (vertex, path) = queue.pop(0)
33         #vertex and path are set and the first vertex is popped off of the queu
   e
34         for next in adj_list[vertex] - set(path):
35                #this finds the next vertex that we are now at in the adj_list,
    subtracts the set path from it
36             if (next == end):
37                 yield path + [next]
38             #if we reached our final goal, yield the path
39             else:
40                 queue.append((next, path + [next]))
41
42 def diam():
43     diameter = 0
44     for x in vertices:
45     #iterates through one set of the vertices
46         for y in vertices:
47         #iterates throught the set of vertices
48             if (y != x):
49             #if we aren't calculating path to ourselves
50                 temp = list(bfs_paths(test_graph,x,y))[0]
51                 #calculate shorest path bewteen the two nodes.
52                 #path_list.append(temp)
53                 #this builds our path list for debugging purpoeses. Not necessa
   ry for final run
54                 if (len(temp) -1  > diameter):
55                     diameter = len(temp) -1
```

```
56                      #checks the length of the bfs optimal path between two node
   s, if the amount of edges are bigger than our largest diameter, we change our d
   imaeter to the new biggest edge length
57      return diameter
58
59  print diam()
60
61
```