class VERTEX: //the vertex class

\_\_INIT\_\_(SELF) // list hold the neighbors, and the node is equal to n#

self.name<- n

self.neighbors<- list()

ADD\_NEIGHBORS(SELF, V) // the function to add neighbors, it checks that if v is not in self. neighbors, if it is it appends it then sort it

if v not in self.neighbors

add v to self.neighbors

sort self.neighbors

GETID(SELF)

return self.name

GET\_CONNECTIONS(SELF) //method returns all of the vertices in the adjacency list#

return self.neighbors.keys()

class GRAPH

\_\_INIT\_\_(SELF)

self.vertices<- {} #the vertices list that holds the ke value pairs of vertices and their neighbors#

ADD\_VERTEX(SELF, VERTEX)

if isinstance(vertex, Vertex) and vertex.name not in self.vertices

self.vertices[vertex.name] <- vertex

return True

else:

return False

//checks to see if the instances of vertex and vertex is not in the list, then returns true#

ADD\_EDGE(SELF, U, V)

if u in self.vertices and v in self.vertices: //checks to see if u and v key value pairs are in the verties list and then return true or false

for key, value in self.vertices.items()

if key=u

value.add\_neighbor(v)

if key=v

value.add\_neighbor (u)

return True

else:

return False

GET\_NEIGHBORS(SELF) // for each key(vertex) prints the values(neighbors)

for i in self.vertices.keys()

print(i,self.vertices[i].neighbors)

PRINT\_GRAPH(SELF)

for key in sorted(list(self.vertices.keys()))

print(key,str(self.vertices[key].neighbors))

def depth\_first\_search(self, v):

s=[]

visited=[]

s.append(v)

while s != []:

u=s.pop()

if u not in visited:

visited.append(u)

for edges in range(len(self.vertices[u].neighbors)):

n=self.vertices[u].neighbors[edges-1]

s.append(n)

return visited

def breadth\_first\_search(self, v):

q=[]

visited=[]

q.insert(0,v)

while q != []:

u=q.pop()

if u not in visited:

visited.append(u)

for edges in range(len(self.vertices[u].neighbors)):

n=self.vertices[u].neighbors[edges-1]

if n not in visited:

q.insert(0,n)

return visited add u to visited

for edges in self.vertices[u].neighbors

add edges-1 to q

return visited

g<- Graph()

for i in range(6)

g.add\_vertex(Vertex(i))

g.vertices

g.add\_edge(0,1)

g.add\_edge(0,5)

g.add\_edge(1,2)

g.add\_edge(2,3)

g.add\_edge(3,4)

g.add\_edge(3,5)

g.add\_edge(4,0)

g.add\_edge(5,4)

g.add\_edge(5,2)

g.get\_neighbors()

print(g.depth\_first\_search(0))

print(g.breadth\_first\_search(0))