**Documentation**

**Documnetatuin of DFS:**

**1) the first thing is we should find the vertex from the graph and**

**Add this to stack**

parent = {} # this will keep track of every adjacent vertex

stack = Stack()

value = vertex

# start from any node and put it in the stack

try:

startnode = self.graph[vertex]

except KeyError:

return 'the vertex is not in the graph'

else:

stack.push(startnode)

startnode.visited = 0

**2. the vertex which is in the stack, take the vertex -> create a list -> put all its adjacent to**

**The list -> create a while loop -> iterate over the list -> find a random vertex -> but the vertex should**

**not be in the stack or not pop it from the stack (i.e the value is 1) -> once you get the vertex -> push it to the stack**

def findCircle(self,vertex):

parent = {} # this will keep track of every adjacent vertex

stack = Stack()

value = vertex

circle = None

# start from any node and put it in the stack

try:

startnode = self.graph[vertex]

except KeyError:

return 'the vertex is not in the graph'

else:

stack.push(startnode.data)

parent[startnode.data] = None

startnode.visited = 0

track\_cycle = []

while(stack.isEmpty() != -1):

# stack\_pop is parent

stack\_pop = self.graph[value]

# \*\* to find its adjacent \*\*

mylist = []

count = 0

i = 0

j = 0

next\_stack\_value = None

leaf\_node = None # keep track that the node is leaf node or not

for k in stack\_pop.neighbor:

mylist.append(k) # the adjacent values are in the mylist

count = count +1

# check its each adjacent vertex

# 1. find how many vertex are in the mylist

length\_list = len(mylist)

while(i<=length\_list-1):

random\_value = random.randint(0,count-1)

i = i+1

adjacent\_vertex = mylist[random\_value]

if self.graph[adjacent\_vertex].visited != 0 and self.graph[adjacent\_vertex].visited != 1:

next\_stack\_value = adjacent\_vertex

# maintain the child-parent relationship

parent[adjacent\_vertex] = stack\_pop.data

leaf\_node = "No" # the node is not a leaf node

break

else:

mylist.remove(adjacent\_vertex)

count = count - 1 # you remove a value from the list

# so you have to decrease one valur from count

if leaf\_node == 'No': # the node is not visited

stack.push(next\_stack\_value)

value = next\_stack\_value

self.graph[next\_stack\_value].visited = 0

**4. when you reach a ‘NODE’ or vertex and if it is the leaf node then you have to pop it from the stack**

else:

# what idf the stack\_pop is the leaf node

pop = stack.pop()

# pop == stack\_pop

self.graph[pop].visited = 1

find\_parent = parent[pop]

# \*\*\*\*\* once you pop the value make sure that you change the 'value' to the value of prev value of stack

value = find\_parent

**5. Now find a pattern (if there any cycle formed ) -> iterate through the leaf node adjacent -> find its any neighbor is visited or not except it’s parent -> when you find a adjacent vertex which is visited -> then there is a cycle -> the find the cycle**

for y in self.graph[pop].neighbor:

if y != find\_parent:

if self.graph[y].visited == 0:

circle = "found"

find\_cycle\_pattern = []

# we find cycle pattern pop -> y

find\_cycle\_pattern.append(y)

find\_cycle\_pattern.append(pop)

temp\_pop = pop

while(find\_cycle\_pattern.count(temp\_pop) <= 1):

previous\_value = parent[temp\_pop]

find\_cycle\_pattern.append(previous\_value)

temp\_pop = previous\_value

print("the cycle ->",find\_cycle\_pattern)