Project Report

# Definition in complexity analysis

V: Vertices

E: Edges

C: Elementary circuits (simple cycles)

# Time complexity

Please refer the analysis of the time complexity of algorithm below:

* **If it is a Directed Acyclic Graph(dag):** it takes O(V+E) when deciding whether it is a dag, and it takes O(V+E) in topological sorting, therefore the overall time complexity in this scenario is O(V+E).
* **If it is a Directed Cycle Graph(Not a dag):** it takes O(V+E) when deciding whether it is a dag, and it takes O((V + E)(C + 1)) to find the simple cycles. So the overall time complexity in this scenario is O((V + E)(C + 1)).
* **As a whole**: the time complexity of this algorithm is O((V + E)(C + 1)).

# Space complexity

Please refer the analysis of the space complexity of algorithm below:

* **If it is a Directed Acyclic Graph(dag):** it needs O(V\*V) to install the adjacency matrix, and it needs O(V+E) in topological sorting, therefore overall space complexity in this scenario is O(V\*V).
* **If it is a Directed Cycle Graph(Not a dag):** it needs O(V\*V) to install the adjacency matrix, and it needs O(V+E) in finding the simples cycles. So the overall space complexity in this scenario is O(V\*V).
* **As a whole:** the space complexity of this algorithms is O(V\*V).

# Program and Analysis

"""

Project 2

Yuehua Duan

801006363

"""

import numpy as np

import networkx as nx

from collections import deque

from collections import defaultdict

------------------------------------------------------------------------------------------------------------------------------

**# Find the topological order of vertices**

**def topoSort(graph):**

"""

Time Complexity is O(V+E)

Space Complexity if O(V+E)

"""

# Determine all the in-degree for nodes

in\_degree = { u : 0 for u in graph }

for u in graph:

for v in graph[u]:

in\_degree[v] += 1

# Store the Zero in-degree into Q

Q = deque()

for u in in\_degree:

if in\_degree[u] == 0:

Q.appendleft(u)

# Use L to store the topological order of vertices

L = []

while Q:

u = Q.pop()

L.append(u)

for v in graph[u]:

in\_degree[v] -= 1

if in\_degree[v] == 0:

Q.appendleft(v)

if len(L) == len(graph):

return L

else:

return []

------------------------------------------------------------------------------------------------------------------------------

# Determine whether the graph is a directed acyclic graph or not

**def is\_directed\_acyclic\_graph**(G):

"""

Time Complexity is O(V+E)

Space Complexity if O(V+E)

"""

List = topoSort(G)

if List == []:

return False

else:

return True

------------------------------------------------------------------------------------------------------------------------------

# Find the cycles

**def simple\_cycles(G):**

"""

Time Complexity is O((V + E)(C + 1))

https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.algorithms.cycles.simple\_cycles.html

Space Complexity is O(V + E)

"""

def \_unblock(thisnode, blocked, B):

stack = set([thisnode])

while stack:

node = stack.pop()

if node in blocked:

blocked.remove(node)

stack.update(B[node])

B[node].clear()

# Find the strongly connected components

sccs = list(strongly\_connected\_components(G))

while sccs:

scc=sccs.pop()

startnode = scc.pop()

path=[startnode]

blocked = set()

closed = set()

blocked.add(startnode)

B=defaultdict(set)

stack=[ (startnode,list(G[startnode])) ]

while stack:

thisnode, nbrs = stack[-1]

if nbrs:

nextnode = nbrs.pop()

if nextnode == startnode:

yield path[:]

closed.update(path)

elif nextnode not in blocked:

path.append(nextnode)

stack.append( (nextnode,list(G[nextnode])) )

closed.discard(nextnode)

blocked.add(nextnode)

continue

if not nbrs:

if thisnode in closed:

\_unblock(thisnode,blocked,B)

else:

for nbr in G[thisnode]:

if thisnode not in B[nbr]:

B[nbr].add(thisnode)

stack.pop()

path.pop()

G.remove\_node(startnode)

H = G.subgraph(scc)

sccs.extend(list(nx.strongly\_connected\_components(H)))

def strongly\_connected\_components(graph):

index\_counter = [0]

stack = []

lowlink = {}

index = {}

result = []

def \_strong\_connect(node):

index[node] = index\_counter[0]

lowlink[node] = index\_counter[0]

index\_counter[0] += 1

stack.append(node)

successors = graph[node]

for successor in successors:

if successor not in index:

\_strong\_connect(successor)

lowlink[node] = min(lowlink[node],lowlink[successor])

elif successor in stack:

lowlink[node] = min(lowlink[node],index[successor])

if lowlink[node] == index[node]:

connected\_component = []

while True:

successor = stack.pop()

connected\_component.append(successor)

if successor == node: break

result.append(connected\_component[:])

for node in graph:

if node not in index:

\_strong\_connect(node)

return result

def remove\_node(G, target):

del G[target]

for nbrs in G.values():

nbrs.discard(target)

def subgraph(G, vertices):

return {v: G[v] & vertices for v in vertices}

------------------------------------------------------------------------------------------------------------------------------

def matrix2graph(fileName):

# Read n

with open(fileName, 'r') as f:

first\_line = f.readline()

print("Read a Graph Matrix with dimension", first\_line)

# Read an adjacency matrix of a directed graph with n vertices

"""

Space Complexity is O(V\*V)

"""

adjacenyMatrix = np.genfromtxt(fileName, skip\_header=2)

rows, cols = np.where(adjacenyMatrix == 1)

edges = zip(rows.tolist(),cols.tolist())

directGraph = nx.DiGraph()

directGraph.add\_edges\_from(edges)

if (is\_directed\_acyclic\_graph(directGraph) == True):

topoOrder = topoSort(directGraph)

print('This is a Directed Acyclic Graph, Please refer the topological order of the vertices')

for task in topoOrder:

print(task)

print('\n\n')

else:

cycles = simple\_cycles(directGraph)

print('This is a Directed Cycle Graph, please refer the direct cycle(s)')

for cycle in cycles:

print(cycle)

print('\n\n')

------------------------------------------------------------------------------------------------------------------------------

files = ['/users/zoe/desktop/Testcase1.txt', '/users/zoe/desktop/Testcase2.txt', '/users/zoe/desktop/Testcase3.txt', '/users/zoe/desktop/Testcase4.txt' ]

for file in files:

matrix2graph(file)