Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Huynh	Student ID:	20141439
Other name(s):	Van Phi Vu		
Unit name:	Data Structures and Algorithms	Unit ID:	COMP5008
Lecturer / unit coordinator:	Valerie Maxville		
Assessment:	Assignment		1

Signature: Huynh Van Phi Vu

Date of signature: 18/10/2021

(By submitting this form, you indicate that you agree with all the above text.)

Data Structures and Algorithms

Assignment v1.0

Semester 2, 2021

Report

1. User Guide

   Overview

   This program simulates the traverse time of an object from one node to another node. The program will calculate the time taken to complete the distance between 2 nodes based on the weight indexes. The time may vary depending on the weight index and the number of nodes along the way. In addition, obstacles or objects that attract attention can also affect the time to complete the route.

2. Description of Classes

   a. DSAGraph
      This class is the main class in this program because we will use graph to simulate the world of a particular object. The graph helps us to determine which node contain which information such as dogs, toys, food. Graph also help us to trace possible routes from one node to another node.

   b. DSAGraphEdge
      This class is part of the DSAGraph class that is used to store additional information such as label and value of edges appearing in the graph. This class contains some basic functions to help us communicate between the edge and the graph.

   c. DSAGraphVertex
      This class is part of class DSAGraph which is used to contain additional information such as label and value of nodes appearing in the graph. This class contains some basic functions to help us communicate between the node and the graph.

   d. DSAHashEntry
      This class is part of the DSAHashTable class that is used to hold information for each entry in the hash table. This class uses DSAHashEntryState to manage which items are

used, which are free, and which are previously used. This class also contains getter and setter function to get information from the entry.

e. DSAHashTable
This class will use the numpy array, which is allowed in this assignment, to store the entry within the input size. This class uses a good hash function to calculate the appropriate index for the entry. When the number of entries reaches the maximum size, this class will expand to double the current size for more storage. This class is used to get an entry faster.

f. DSALinkedList
Since the assignment prohibits us from using build-in Abstract Data Structure (ADT), therefore this class is used as a list to store sequence data. This class is also used in other ADT classes like DSAGraph and DSAGraphVertex.

g. DSAListNode
This class is part of DSALinkedList which to represent an element in a list. Each node contains a value or an object value to store information for that element.

h. DSAStack
This class works on the principle last-in, first-out (LIFO) which is very useful in this program. This class is used to create a route from one node to another node when the action of finding all possible routes is performed.

i. FileData
This class is used to store information from the input sample file. In this class, there are other 3 classes named CodeInfo, NodeInfo, and EdgeInfo.
CodeInfo is used to store data of node code and edge code including code label, weight, and description. NodeInfo is used to store data of node including node label and code label. EdgeInfo is used to store data of edge including from label, to label, and code label. These 3 classes contain getter functions to access to its data.
This class acts as a container for all the information of node code, edge code, node, edge, start label, target label by using the 3 classes defined above by create an instance and then save it in in Hash Table. This class also contains getter and setter functions to help access its data.

j. Route
This class is part of class RouteList and used to contain the information of one route from node A to node B with its cost(time). This class contains getter functions to get and display the route.

k. RouteList
This class is defined to find all possible routes from one node to another node with the option to include cost when perform method generate. DSAStack and DSALinkedList is

used to provide assistance in creating and storing route. In this class contains method called _findRoute() which will be called in recursive way until it reaches the target node. After generating, the class will return the list of all possible routes in LinkedList, user can choose to sort the routes by its cost by calling sort function.

3. Justification of Decisions
   a. DSAGraph
      The main purpose of this assignment is to extend the graph to find out all the possible routes from one node to another node. This class helps to display data as a graph with the nodes connected by edges. From here we can find all the possible routes between 2 nodes through the edges.
      In this class, the source code is taken from practical 6. An additional feature is visualize() which is used to display the world (actual graph) as the requirement bonus and to checking all the routes after generated.

   b. DSAHashTable
      This class was not going to be used initially; however, when creating all possible routes with cost, the process took a lot of time when the input data is too big. After online research, an article comparing element traversal performance between arrays, linked lists and hash tables concluded that if we want fast traversing, we need to use hash tables. Therefore, this class was added and the source code is taken from practical 7.

   c. DSALinkedList
      This assignment is aimed to practice using custom ADT and not using built-in one like list. The mechanism of LinkedList for storing and displaying data is roughly equivalent to using an array, I choose to use it to store sequence data.
      In this class, the source code is taken from practical 4. I added 2 more methods named remove() is to remove the node in the middle of the list. The way it works is similar to the removeFirst() and removeLast() methods, but with a little more customization to make sure the data nodes are always in sequence. Another method named sort() is used to sort the list in ascending order, this sort used selection sort algorithm cause this performs well on small list. The source is based on selection sort in practical 1 to make it works with linked list.

   d. DSAStack
      When creating the route from one node to another node, Queue is used to store all the visited nodes. However, after completing one route, it cannot go back to the parent node to find another route because of the first-in, first-out (FIFO) property.
      Due to this inconvenience, Stack was chosen since it has the last-in, first-out (LIFO) property and that makes it easier to go back to the parent node. The source code is taken from practical 3.

   e. FileData

Again, this assignment aims to practice using custom ADT and not using built-in one like list and dictionary. This class was created to act as dictionary to store all data from input file.

When reading line by line from input file, hash table is used to store all codes, nodes, edges, start label, and target label accordingly. This class also contains getter and setter functions to help accessing data.

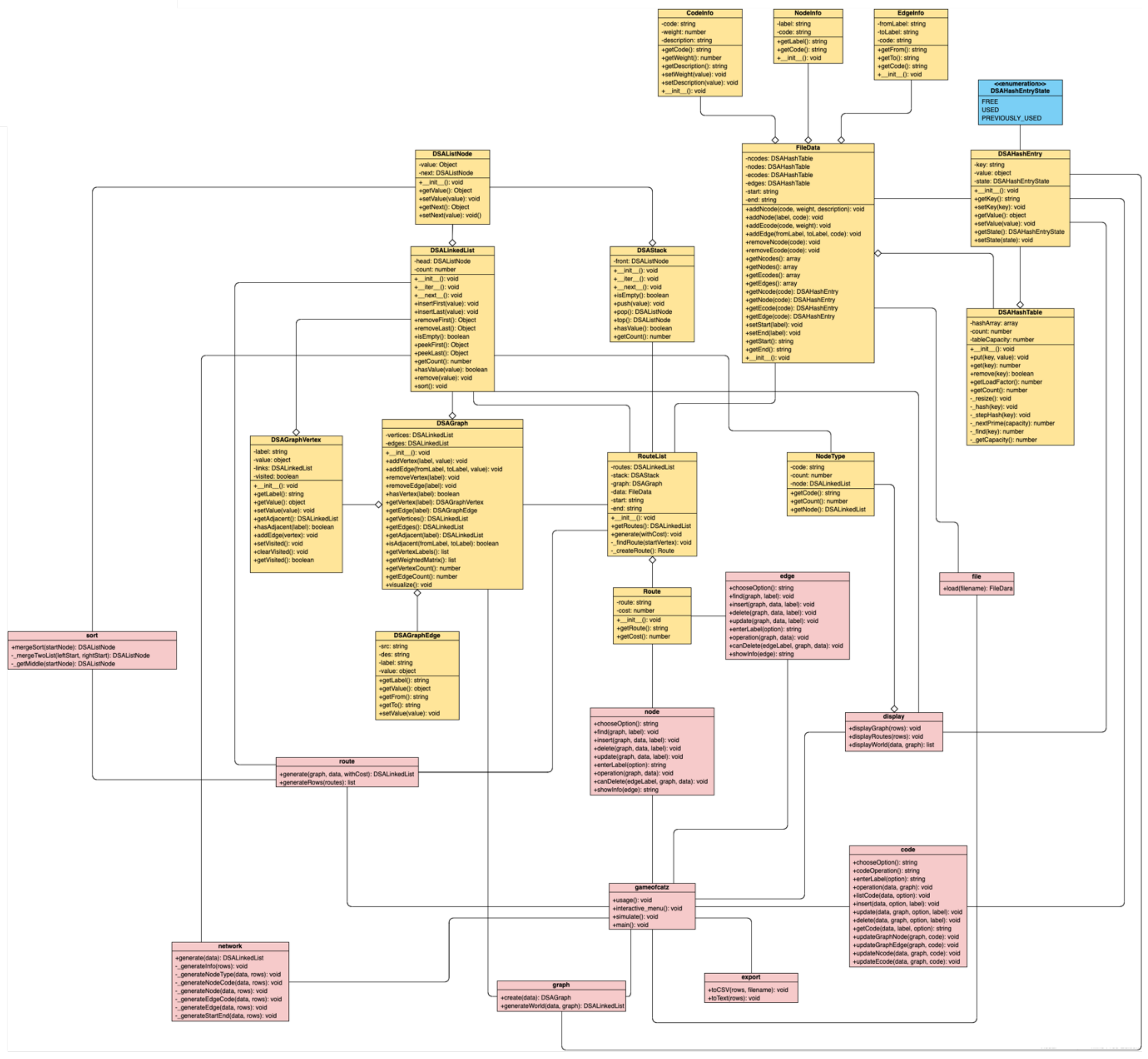
f. RouteList

This class was created to store all the routes after generating in LinkedList. As mentioned above, Stack is used in this class in to assist creation of route from one node to another node.

g. Merge sort

The sorting algorithm is used to sort the list with a large number of items. When creating routes with cost, I just push the item to the last index of the list which will result in an unsorted list after finish. The route list may contain a huge number of items, I used this merge sort to assist sorting the list because it will spit the list into half which can reduce the sorting time.

4. UML Class Diagram



5. Traceability Matrix

Menu	Requirements	Design/Code	Test
1. Mode	1.1. Display usage if user does not provide enough command line argument.	gameofcatz.py main()	UI Test: Script 1
	1.2. System enters interactive mode with argument “-i”.	gameofcatz.py main()	N/A
	1.3. In interactive mode, display the menu for user to interact with program.	gameofcatz.py interactive_menu()	UI Test: Script 2
	1.4. System enters simulation mode with argument “-s”.	gameofcatz.py main()	N/A
	1.5. In simulation mode, system calls appropriate methods and export the result to file.	gameofcatz.py simulate()	UI Test: Script 3
2. Menu	2.1. Display the menu if user chooses interactive	gameofcatz.py	UI Test: Script 2

	mode.	interactive_menu()	
	2.2. System exits if user presses Enter.	gameofcatz.py main()	N/A
3. Load input file	3.1. System enters to option “1. Load input file”.	gameofcatz.py main()	N/A
	3.2. Ask user to input filename to read data.	gameofcatz.py main()	N/A
	3.3. Displays error message if user inputs non-exist filename.	gameofcatz.py main()	UI Test: Script 4
	3.4. System loads data from file and saves to the new instance of class FileData if user inputs exist filename.	file.py load()	testFile.py testCreateFileData
	3.5. System creates graph from file data.	graph.py create()	testGraph.py testCreateGraph
4. Node operations	4.1. Display error message if user does not choose option “Load input file” from the menu (2.1) first.	gameofcatz.py main()	UI Test: Script 5
	4.2. System enters to option “2. Node operations” if user loads input file already.	gameofcatz.py main()	N/A
	4.3. Display node menu operations including “Find”, “Insert”, “Update”, “Delete”.	node.py chooseOption()	UI Test: Script 6
	4.4. System back to display menu (2.1) if user presses Enter.	gameofcatz.py interactive_menu()	N/A
	4.5. Ask user to input node label if user chooses one of (4.3) options.	node.py enterLabel()	N/A

	4.6. Display error message if user chooses option “Find” or “Update” or “Delete” and inputs non-exist node label. System then displays node menu (4.3).	node.py
		find()
		update()
		delete()
			UI Test: Script 7

	4.7. Display error message if user chooses option “Insert” and inputs exist node label. System then displays node menu (4.3).	node.py insert()	UI Test: Script 7

	4.8. Display error message if user input non-exist code label. System then displays node menu (4.3).	node.py
		insert()
		update()
			UI Test: Script 9

	4.9. Display node information if user chooses option “Find” and inputs exist node label. System then displays node menu (4.3).	node.py
		find()
		showInfo()
			UI Test: Script 8

	4.10. Ask user to input code label if user chooses option “Insert” and inputs non-exist node label.	node.py insert()	N/A
	4.11. System adds new node to graph if user inputs exist code label. System then displays node menu (4.3).	node.py insert()	testNode.py testInsertNode

	4.12. Ask user to input another code label if user chooses option “Update” and inputs non-exist node label.	node.py update()	N/A
	4.13. System updates another code label to node if user inputs exist code label. System then displays node menu (4.3).	node.py update()	testNode.py testUpdateNode
	4.14. Display error message if user inputs node label of start or target. System then displays node menu (4.3).	node.py delete()	testNode.py testDeleteStart testDeleteTarget

	4.15. Display error message if user inputs node label that cannot create a route from node Start to node Target. System then displays node menu (4.3).	node.py
		delete()
		canDelete()
			testNode.py
			testCantDeleteNode

	4.16. System deletes edges that has the node and itself if user inputs exist node label and not in case (4.14, 4.15). System then displays node menu (4.3).	node.py delete()	testNode.py testDeleteNode
5. Edge operations	5.1. Display error message if user does not choose option “Load input file” from the menu (2.1) first.	gameofcatz.py main()	UI Test: Script 4
	5.2. System enters to option “3. Edge operations” if user loads input file already.	gameofcatz.py main()	N/A
	5.3. Display edge menu operations including “Find”, “Insert”, “Update”, “Delete”.	edge.py chooseOption()	UI Test: Script 6
	5.4. System back to display menu (2.1) if user presses Enter.	gameofcatz.py interactive_menu()	N/A
	5.5. Ask user to input edge label if user chooses one of (5.3) options.	edge.py enterLabel()	N/A
	5.6. Display error message if user inputs incorrect format of edge label. System then displays edge menu (5.3).	edge.py enterLabel()	UI Test: Script 13

	5.7. Display error message if user chooses option “Find” or “Update” or “Delete” and inputs non-exist edge label. System then displays edge menu (5.3).	edge.py
		find()
		update()
		delete()
			UI Test: Script 7

	5.8. Display error message if user chooses option “Insert” and inputs exist edge label. System then displays edge menu (5.3).	edge.py insert()	UI Test: Script 7

	5.9. Display error message if user input non-exist code label. System then displays edge menu (5.3).	edge.py
		insert()
		update()
			UI Test: Script 9

	5.10. Display edge information if user chooses option “Find” and inputs exist edge label. System then	edge.py find()	UI Test: Script 8

	displays edge menu (5.3).	showInfo()	
	5.11. Ask user to input code label if user chooses option "Insert" and inputs non-exist edge label.	edge.py insert()	N/A
	5.12. System adds new edge to graph if user inputs exist code label. System then displays edge menu (5.3).	edge.py insert()	testEdge.py testInsertEdge
	5.13. Ask user to input another code label if user chooses option "Update" and inputs non-exist edge label.	edge.py update()	N/A
	5.14. System updates another code label to edge if user inputs exist code label. System then displays edge menu (5.3).	edge.py update()	testEdge.py testUpdateEdge

	5.15. Display error message if user inputs edge label that cannot create a route from node Start to node Target. System then displays edge menu (5.3).	edge.py
		delete()
		canDelete()
			testEdge.py
			testCantDeleteEdge

	5.16. System deletes edge if user inputs exist edge label and not in case (5.15). System then displays edge menu (5.3).	edge.py delete()	testEdge.py testDeleteEdge
6. Parameter tweaks	6.1. Display error message if user does not choose option "Load input file" from the menu (2.1) first.	gameofcatz.py main()	UI Test: Script 4
	6.2. System enters to option "4. Parameter tweaks" if user loads input file already.	gameofcatz.py main()	N/A
	6.3. Display code menu operations including "Node code operation" and "Edge code operation".	code.py chooseOption()	UI Test: Script 14
	6.4. System back to display menu (2.1) if user presses Enter.	gameofcatz.py interactive_menu()	N/A
	6.5. Display "Node code operation" menu including "Insert", "Update", "Delete" if user chooses node code operation from menu (6.3).	code.py codeOperation()	UI Test: Script 14
	6.6. Display "Edge code operation" menu including "Insert", "Update", "Delete" if user chooses edge code operation from menu (6.3).	code.py codeOperation()	UI Test: Script 14
	6.7. Ask user to input code label if user chooses one of (6.5 or 6.6) options.	code.py enterLabel()	N/A

	6.8. Display error message if user chooses option "Update" or "Delete" and inputs non-exist code label. System then displays code menu (6.5 or 6.6).	code.py
		update()
		delete()
			UI Test: Script 9

	6.9. Display error message if user chooses option "Insert" and inputs exist code label. System then displays code menu (6.5 or 6.6).	code.py insert()	UI Test: Script 9

	6.10. Ask user to input weight and description if user chooses option "Insert" and inputs non-exist code label.	code.py insert()	N/A
	6.11. System adds new code to file data. System then displays code menu (6.5 or 6.6).	code.py insert()	testCode.py testInsertEcode
	6.12. Ask user to input new weight if user chooses option "Update" and inputs exist code label.	code.py update()	N/A
	6.13. System updates new weight to code in file data and graph. System then displays code menu (6.5 or 6.6).	code.py update()	testCode.py testUpdateEcode
	6.14. Display error message if user inputs code "–". System then displays code menu (6.5 or 6.6)	code.py delete()	testCode.py testDeleteMinus
	6.15. System deletes code in file data and update nodes/edges in graph to code "–" if user inputs code label except for code "–". System then displays code menu (6.5 or 6.6).	code.py delete()	testCode.py testDeleteEcode
7. Display graph	7.1. Display error message if user does not choose option "Load input file" from the menu (2.1) first.	gameofcatz.py main()	UI Test: Script 4
	7.2. System enters to option "5. Display graph" if user loads input file already.	gameofcatz.py main()	N/A
	7.3. Display weighted adjacency matrix of the graph.	display.py displayMatrix()	UI Test: Script 15
	7.4. Ask if user wants to export the matrix or not.	gameofcatz.py main()	N/A
	7.5. Ask user to input filename if user input "y" or "yes" from 7.4.	export.py toCSV()	N/A
	7.6. System exports the matrix to filename from 7.5.	export.py toCSV()	UI Test: Script 10
	7.7. System back to display menu (2.1) if user inputs "n" or presses Enter.	gameofcatz.py interactive_menu()	N/A
8. Display world	8.1. Display error message if user does not choose option "Load input file" from the menu (2.1) first.	gameofcatz.py main()	UI Test: Script 4
	8.2. System enters to option "6. Display world" if user loads input file already.	gameofcatz.py main()	N/A
	8.3. Display world with the count of node types in the graph.	display.py displayWorld()	UI Test: Script 16
	8.4. Ask if user wants to export the world or not.	gameofcatz.py main()	N/A
	8.5. Ask user to input filename if user input "y" or "yes" from 8.4.	export.py toCSV()	N/A
	8.6. System exports the world to filename from 8.5.	export.py	UI Test: Script 10

		toCSV()	
	8.7. System back to display menu (2.1) if user inputs “n” or presses Enter.	gameofcatz.py interactive_menu()	N/A
9. Generate routes	9.1. Display error message if user does not choose option “Load input file” from the menu (2.1) first.	gameofcatz.py main()	UI Test: Script 4
	9.2. System enters to option “7. Generate routes” if user loads input file already.	gameofcatz.py main()	N/A
	9.3. Generate all possible routes with the data from graph and file data. The process time depends on how large the input file is.	route.py generate()	testRoute.py testGenerateRoute
	9.4. Sort all the routes based on their weight (cost) from nodes and edges.	route.py generate() sort.py mergeSort()	testRoute.py testGenerateRoute
	9.5. System back to display menu (2.1).	gameofcatz.py interactive_menu()	N/A
10. Display routes	10.1. Display error message if user does not choose option “Load input file” from the menu (2.1) first.	gameofcatz.py main()	UI Test: Script 4
	10.2. System enters to option “8. Display routes” if user loads input file already.	gameofcatz.py main()	N/A
	10.3. Display error message if user does not choose option “Generate routes” from the menu (2.1) first.	gameofcatz.py main()	UI Test: Script 17
	10.4. Display all the possible routes from low-cost to high-cost order.	display.py displayRoutes()	UI Test: Script 18
	10.5. Ask if user wants to export the routes or not.	gameofcatz.py main()	N/A
	10.6. Ask user to input filename if user input “y” or “yes” from 10.5.	export.py toCSV()	N/A
	10.7. System exports all the routes to filename from 10.6	export.py toCSV()	UI Test: Script 10
	10.8. System back to display menu (2.1) if user inputs “n” or presses Enter	gameofcatz.py interactive_menu()	N/A
11. Save network	11.1. Display error message if user does not choose option “Load input file” from the menu (2.1) first.	gameofcatz.py main()	UI Test: Script 4
	11.2. System enters to option “9. Save network” if user loads input file already.	gameofcatz.py main()	N/A
	11.3. Generate the network with the data from the graph and file data.	network.py generate()	UI Test: Script 11
	11.4. Ask user to input filename for the network.	export.py	N/A

		toText()	
	11.5. System exports all the network to filename from 11.4.	export.py toText()	UI Test: Script 11
	11.6. System back to display menu (2.1).	gameofcatz.py interactive_menu()	N/A
12. Visualize	12.1. Display error message if user does not choose option “Load input file” from the menu (2.1) first	gameofcatz.py main()	UI Test: Script 4
	12.2. System enters to option “10. Visualize” if user loads input file already	gameofcatz.py main()	N/A
	12.3. Display graph visualization	DSAGraph.py visualize()	UI Test: Script 12

6. Showcase
   a. Introduction
      The program ‘gameofcatz’ simulates a ‘cat’ object trying to go from one node to another node. However, not every path is easy to follow. Along the way, it might face some obstacles that make the cat spend more time than expected. The aim of this program is to help the cat can find the shortest path to take.
      In this program, user can choose between interactive mode or simulation mode. In simulation mode, the program will output all the possible routes from node A to node B with the ranking of the route. The order is from the shortest to longest, this is based on how long the cat can go from node A to node B and how long it is affected by the node it is passing by.
      In the interactive mode, the user can manually change the indicators in the program. User can manipulate add, edit, or delete node, edge and code. These changes to node, edge, or code can make a significant impact to finding the shortest path for the cat. The program provides options like display graph or display world for user to check all the changes has made to the world. User can export the result from the above options after viewing. Besides, user can request to generate all the route the cat can go from one node to another node. Once generated, user can choose to display all the routes with the cost in order from shortest to longest. The route generating process is fast or slow depends on number of nodes and number of edges appear in this world.

   b. Scenarios
      The first scenario script I use is a sample file called gameofcatz2.txt. In this sample file, the amount of nodes and edges is huge. When all possible routes are generated from one node to another without cost calculation, it takes about 16 seconds. In the case of creating routes with calculating the cost, the completion time is about 45 seconds which is very slow in performance. This is because the process of finding the respective costs of nodes and edges take so much time to complete. When calculating the cost, the program needs to find the cost many times for the same node and edge and with the amount of routes that this scenario can have, this can take ages if we do not use the

proper ADT. Besides the route generation method, other functions return results immediately.

The next script is awesome.txt. In this sample file, like the previous scenario, the amount of nodes and edges is huge. However, since this scenario is like a maze, there is only one route from start to finish, the process of creating routes and calculating cost is very short, and we do not need to sort with only one route.

The last scenario is snakes&ladders.txt. In this scenario, the amount of nodes and edges are also huge. This scenario is like we are playing snakes and ladders, we move from bottom to the top and in some cases, we can fall off the snake or climb up the ladder, then we can continue until we reach the final node. This scenario will make the graph in one direction, this will reduce the time when creating routes and calculating cost. When generating routes including cost and ranking, it gives the result instantly.

c. Conclusion

In conclusion, in three scenarios, we can see that with the graph with one possible route or directed graph, the performance is fast. However, with the undirected graph, depending on the amount of nodes and edges, the performance can be either instant or slow.

In the future, I hope to reduce the time when calculating the cost by caching the previous found cost of node and edge for the next use. This will skip the calling find method of the system and only use the cached item.