A picture containing logo

Description automatically generated

# **ICS2210 - Data Structures and Algorithms 2**

# Nathan Bonavia Zammit,

Bachelor of Science in

Information Technology (Honours)

(Artificial Intelligence)

*May 2022*

**Statement of Completion**

|  |  |
| --- | --- |
| **Item** | **Completed (Yes/No/Partial)** |
|  | |
| Created a random DFA | Yes |
| Correctly computed the depth of the DFA | Yes |
| Correctly implemented DFA minimization | Yes |
| Correctly computed the depth of the minimized DFA | Yes |
| Correctly implemented Tarjan’s algorithm | Yes |
| Printed number and size of SCCs | Yes |
| Provided a good discussion on Johnson’s algorithm | Yes |
| Included a good evaluation in your report | Yes |

**Question 1**

The classes State and Graph are used in order to construct the required deterministic finite state automata (DFSA) that are used in this assignment.

Starting with the State class, one constructor is present, which takes 3 parameters. These parameters being, ‘accepting’, ‘transitions’ and ‘stateid’. The ‘accepting’ parameter is set as a Boolean value which will specify whether or not the state is either an accepting state or a final state. The ‘transitions’ parameter is expected to be a dictionary which maps the characters in the alphabet of the DFSA, in the following code used this will consist of ‘a’ and ‘b’, to the ids of the states which will transition to form the given state when given the respective character. The ‘stateid’ parameter is used to assign the numerical id of the state within the DFSA it is used in. The use of dictionaries to map characters in the alphabet to their corresponding transitions is a form of an implementation of an adjacency list. The automaton that was requested is specified to have only 2 edges for each state with a limit of up to 64 states. Therefore, the graph is sparsely connected and as such the use of an adjacency list instead of an adjacency matrix is further more justified.

Besides the 3 mentioned parameters, the State class also contains a Boolean flag ‘onstack’, and two integers ‘order’ and ‘link’. These will be used for when it comes to implementing Tarjan’s algorithm further on.

The Graph class takes up two more parameters, these being ‘numofstates’ and ‘name’. The ‘name’ parameter is a String parameter which is used to assign a name to the graph in order to make it far easier to differentiate between automata A and M. The ‘numofstates’ parameter is an integer parameter which is used to generate a number of states with randomized properties via the python library ‘random’. A state is determined whether it is accepting or not via the use of the ‘random.getrandbits’ function to generate either a 0 or a 1 value randomly which will in turn be converted into a Boolean. The transitions are then generated via the use of ‘random.randint’ in order to have a random integer within the range of states in the graph, this range being between 0 and ‘numofstates’ -1, for each character in our alphabet. After the generation of 2 random numbers, if both ‘a’ and ‘b’ end up leading to the same state, then the number for the transition that is leading from ‘b’ will be regenerated through the same method, until a different number will be generated. This was done assuming that states with both characters transitioning into the same state were not desired. After a state’s properties are generated randomly and it is assigned its index based on the number of states generated before it, it is then added to the graph’s ‘states’ list. If the state is marked as accepting, then it will be also added to the graph’s ‘finalstates’ list. After all the required states are generated, a random number between 0 and numofstates -1 is determined to pick a state to be the starting state. The state whose id is the number generated will have it’s start property set to True and therefore, the graph’s ‘startid’ property is then set to said number.

The ’calculatedepth’ method which is looked into further detail later on, is called and will return a list of Booleans, with each corresponding to whether the state in the graph’s ‘states’ list is reachable in any possible way from the starting state. Then the state list is iterated through and each each state which is reachable is added to the list ‘reachablestates’. The ‘depth’ property is used to store the depth of the automaton which is calculated by the ‘calculatedepth’ method. This was required for questions 2 & 4.

The 5 properties ‘index’, ‘stack’, ‘largestSCC’, ‘smallestSCC’ and ‘sccs’ are used for Tarjan’s algorithm in question 5. When the graph constructor is called, the ‘numofstates’ parameter is randomly generated between 16 and 64, as requested.

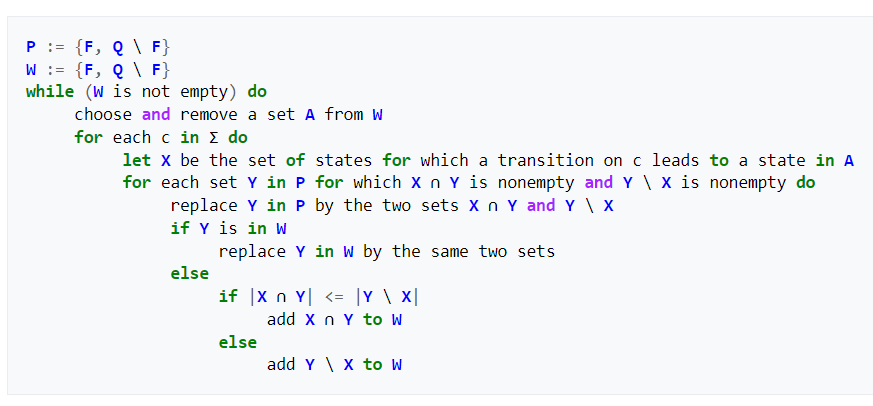
**Question 2 & 4**

The ‘calculatedepth’ function is defined in the Graph class in order to calculate the depth of the graph as required. This was done via the breadth first search algorithm which is explained down below [1].

1. ‘visitednodes’, a list of Boolean flags, is initialized with a number of elements which are equal to the number of states in the graph, and each index is set to false
2. ‘searchqueue’ is declared which will store the ids of states in queue to be visited. The initial depth of the graph is also set to 0.
3. The index of the starting node of the graph is added to the ‘searchqueue’, followed by a value of -1, and the node corresponding to the index of the starting node in the ‘visitednodes’ list is set to true. The value of -1 in the ‘searchqueue’ is used to indicate that all the nodes in the current depth level of the graph have been visited.
4. The first state in the queue is popped from the queue and its neighbours which haven’t yet been visited are pushed to the end of the queue and marked as visited. If the value popped is a -1, another -1 is popped to the of end of the queue. The next element in the queue is then checked and if it is found to be a -1, then the loop will be broken and the list of ‘visitednodes’ is returned. Otherwise, the depth is increased by 1 and the loop will not break.
5. Step 4 will be repeated until the aforementioned condition is reached which will indicate that all reachable nodes have been visited.

**Question 3**

For the minimization of the automata Hopcroft’s algorithm was used instead of Moore’s algorithm. This was because even though both Moore’s & Hopcroft’s algorithms have an average time complexity of O(n log log n), the worst case complexity of Moore’s ( O(n log n)) is worse than that of Hopcroft’s ( O(n log log n)) [2]. The code used was based off following pseudocode [3]:



The new graph, in this case ‘M’ starts as a deepcopy of the graph which is required to be minimized. This was done as creating a new graph equal to it would simply act as a pointer and therefore lose the properties of the initial graph. The set difference and intersection were done using python’s innate set functions. This algorithm returns a list of sets whereby each set represents one or more states which are deemed equivalent to each other in the scope of the automata.

Each of the sets were then iterated through in order to check if the states inside are final states and if the starting state is present as well. It is also determined which of the other sets contains the states which the states inside should transition to. After these properties are checked, the new state corresponding to that set is created with the respective properties and then added to a list ‘newstates’. The startid of the new graph is set to the index of the set containing the starting state in the partition and all the new properties of the new graph are set based on the ‘newstates’ list.

**Question 5**

{to be added tomorrow}

**Question 6**

{To be added tomorrow}

**References:**

[1] - <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

[2] J. David, Average Complexity of Moore’s and Hopcroft’s Algorithms. 2010. pgs. 22-23. [Accessed: 26/05/2022]

[3] - <https://en.wikipedia.org/wiki/DFA_minimization>