Project 2 Readme Template

Version 1 9/11/24

A single copy of this template should be filled out and submitted with each project submission, regardless of the number of students on the team. It should have the name readme_"teamname"

Also change the title of this template to "Project x Readme Team xxx"

1	Team Name: cbowers4
2	Team members names and netids: Collin Bowers, cbowers4
3	Overall project attempted, with sub-projects: Program 1: Tracing NTM Behavior
4	Overall success of the project: Success
5	Approximately total time (in hours) to complete: 10 hours
6	Link to github repository: https://github.com/collinb424/cbowers4_theoryproj2
7	List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary)

File/folder Name	File Contents and Use
Code Files	
traceTM_cbowers4.py	Code for tracing the behavior of a given nondeterministic Turing Machine on user-specified input
Test Files	
check_cbowers4.csv	The same CSV file for the *a plus* machine as given in the project document.
Data Files	
data_a_plus_DTM_cbowers4.csv data_abc_star_cbowers4.csv data_abc_star_DTM_cbowers4.csv data_aplus_cbowers4.csv	CSV files for a variety of different machines that I used for testing my code.

data_check_random_cbowers4.csv data_end_with_0101_cbowers4.csv data_equal_01s_cbowers4.csv data_equal_01s_DTM_cbowers4.csv data_more_than_three_as_cbowers4.csv	
Output Files	
output_cbowers4.txt	Results of running traceTM_cbowers4.py on the above test/data files with a variety of inputs
Plots (as needed)	
plots_cbowers4.pdf	Reports the statistics for the results of running traceTM_cbowers4.py on the above test/data files with a variety of inputs

8 Programming languages used, and associated libraries: Python, csv, sys, collections

9 Key data structures (for each sub-project):

- The specified csv file is read into a dictionary called *ntm*. This dictionary keeps track of the name of the NTM, the start, accept, and reject states, as well as all of the transitions (the state and input are mapped to a list, where each element contains the new state, the new character, and the direction. A list is used because this is a nondeterministic machine: there could be several different new states, characters, or directions for the same state and input).
- The *Node* class contains information about the configuration (what the current state is, what is to the left of that state, and what is to the right of that state), the state's parent node, and the state's children.
- The variable *breadth_first_paths* is a list of lists that keeps track of the different Nodes (of type *Node*) on each level of the tree as we do the breadth-first traversal. As we go down to future levels, there could be several different nodes on each level, assuming the machine is nondeterministic. If the machine is deterministic, there will only be one node on each level.
- The variable *curr_level* is a list made up of *Node*s that simply keeps track of all of the nodes seen on the current level. Once the current level has been finished processing, *curr_level* is added to *breadth_first_paths*.

	- The variable *queue* is used for the breadth-first traversal. It offers First-In, First-Out functionality, which is needed for an iterative breadth-first traversal.
10	General operation of code (for each subproject) - The code begins by parsing the arguments the user passes in. - The CSV file is read into an internal representation (a dictionary, as described above) of a nondeterministic Turing Machine - A trace of the nondeterministic Turing machine is performed using breadth-first search based on the given input. The trace works by: - Creating the root node, based on the given input string and starting state. - Using a queue for the breadth-first traversal - Iterating over all configurations on the current level of the tree (so we can keep track of the different levels) - Iterating over the different possible transitions based on the current state and next character (which come from the current configuration). These transitions lead to new configurations, so we create a new node object for these configurations and add them to the queue to be considered next. - If we run into an accept state, we generate the accepting path and print the necessary output. - If the current depth exceeds the limit specified by the user, we stop the execution and print the necessary output. - If the queue becomes empty, that means we didn't hit an accept state or surpass the specified depth limit, so the input must have been rejected, and we then print the necessary output. - The output also computes the degree of nondeterminism, depth of the tree of configurations, and total number of transitions simulated.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. - I first used check_cbowers4.csv to test my code because that is the same CSV file that Professor Kogge gave us in the project document. Before coding, I manually created my own trace to ensure I had the right idea in mind for how to implement the breadth-first traversal. I then used this to generate my solution, and used this csv file to test my code along the way. After finishing my code, I verified that it had the same output as Professor Kogge specified, such as a depth of 6 for the input aaaaa (he specified this in class that the number given in the project document is wrong). - After verifying it worked for that, I created a couple of other files to test my code. I created data_aplus_cbowers4.csv, which is a similar but slightly different implementation of an NTM that accepts a$^+$, and I verified that this had the same results. Additionally, I created data_more_than_three_as_cbowers4.csv to see if my solution would properly accept strings if they have more than three a's. This worked, although I noticed there was an issue with going over the user-specified depth, so creating this file helped me debug that. Next, I added data_end_with_0101_cbowers.csv and data_check_random.csv because they were inspired by NFAs on the homework. - The other data_… files I used are the ones shared by another student that Professor Kogge sent to the class. I used these to test my code to ensure that the degree of nondeterminism was 1.0 on DTMs, and I also used them to ensure that my code worked for low and high degrees of nondeterminism.

	- I also manually calculated the degree of nondeterminism for several of my test files to ensure that my calculation was working properly for that.
12	How you managed the code development - I first read carefully over the project document and did a breadth-first trace of the “a plus” NTM myself to ensure I understood how it functioned. - After getting a good understanding, I approached the code by breaking it down into chunks of different functions. I began by reading the specified CSV file into the dictionary, and then expanded that to properly parsing the user’s arguments. - After this, I played around with a variety of examples and developed a general algorithm using a queue and breadth-first search to simulate the trace of an NTM. I ran into many errors on this part, but using the Python debugger on Visual Studio Code was incredibly helpful for tracking the state of variables and figuring out where the problem was. Consequently, I was able to spend a couple dozen minutes on each error, rather than a couple hours on each error. I also originally did not have the Node class, but added that in after completing the algorithm because it made it a lot easier to track the parent and children of each node in the tree. - After getting the algorithm working, I created several output functions to compute and report the necessary information for if the string was accepted, rejected, or had its execution terminated. - Along the way, and after completing the above, I created and used a variety of test files to ensure my algorithm was working properly.
13	Detailed discussion of results: - The primary thing I noticed after doing dozens of runs is how the average nondeterminism impacts the relationship between the depth of the tree (to reach an accept or reject state) and the number of configurations explored. It does not explicitly impact either of these on their own, but rather the *relationship* between them. For example, when I ran the string 010101 on data_equal_01s_DTM_cbowers4.csv, the depth was 25 and number of configurations was 25 (which are both relatively high numbers), and because it was a DTM, the average nondeterminism was 1.0. However, if we compare this to running the string aabc on data_abc_star_cbowers4.csv, we get a depth of 5 and number of configurations of 29. This is a large discrepancy, and is explained by the fact that the average nondeterminism was 1.63158. This intuitively makes sense, because if there is a high degree of nondeterminism, the machine has to explore many different possibilities even if the accepting state isn’t very far down the tree. Thus, a high degree of nondeterminism makes the code less performant because it has to explore many different possibilities which may lead to reject states. - Again, to reiterate, the average nondeterminism impacts <u>the relationship</u> between the depth of the tree (to reach an accept or reject state) and the number of configurations explored. What impacts each of these <u>individually</u> is the complexity of the Turing Machine, which is why the short string input of 010101 on data_equal_01s_DTM_cbowers4.csv led to a depth of 25 and number of configurations explored of 25, even though the average nondeterminism was 1.0. This is much larger than the result one would get from running the input aabc on data_abc_star_DTM_cbowers4.csv (depth of 5 and number of configurations explored of 5), even though the average nondeterminism is also

	1.0. The reason for this is the complexity of the Turing Machine.
14	How team was organized - I did all of the work since it was an individual team (see section 12 for more on how I organized the development process). I took into account how I could improve from the last project and started earlier on this project. This paid off because I was less stressed and able to think more creatively when solving any problems that arose.
15	What you might do differently if you did the project again - In the future, I hope to improve on organizing my thoughts when trying to solve a problem. I often run into the scenario where I sort of get paralyzed for several minutes and continually turn over the same ideas in my head, causing me to not make much progress. Instead, I should work more on writing my thoughts out in an organized fashion to ensure I am always making progress on the goal.
16	Any additional material: N/A