Project 2 Readme Team djohan

1	Team Name:djohan
2	Team members names and netids: Daniel Johan, djohan
3	Overall project attempted, with sub-projects: Tracing NTM Behavior
4	Overall success of the project: Successful
5	Approximately total time (in hours) to complete: 20 hours
6	Link to github repository: https://github.com/djohan12/TOC_project02_djohan
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). Add more rows as necessary.

File/folder Name	File Contents and Use
Code Files	
traceTM_djohan.py	Python script that reads in a turing machine csv file, a maximum depth, and a series of strings and determine whether the string can reach the accept state within the maximum depth and all possible transitions that it can tape for a given machine. Once the tree of configurations has been computed, the result is written to an output file as well as vital information such as all transitions, depth, nonleaves, and nondeterminism
Test Files	
a+TM.csv	Turing machine csv for language a+
a^nb^bTM.csv	Turing machine csv for language a^nb^n
abc_starTM.csv	Turing machine csv for language a*b*c*
Output Files	
a+TM_output.txt	Output of turing machine configurations path for input strings for the a+ language
a^nb^bTM_output.txt	Output of turing machine configurations path for input strings for the a^nb^n language
abc_starTM_output.txt	Output of turing machine configurations path for input

	strings for the a*b*c* language
Plots (as needed)	
Same as output files	The output files provide a txt file that contains the name of the machine, the strings, whether the string is accepted, rejected, or times out. It also shows transitions at each configuration level, The total number of transitions, non leaves, and the average nondeterminism.

8	Programming languages used, and associated libraries: The entire project was implemented in python with the csv, sys, collections, and math libraries.
9	Key data structures (for each sub-project): The main data structures in the project are the csv files which contain the information on the turing machines tested. When the program reads in the turing machine, the information is stored in a dictionary to be used for later. In the actual tracing part of the code, the main data structure is a 2d-list tree which reads in all the configurations and their possible transitions by using a BFS approach. Once all the data has been acquired, the results are written to each turing machines respective output files
10	General operation of code (for each subproject): The turing machines must be written in this format: <name>, <states>,<string alphabet>,<tape alphabet>,<start>,<accept>,<reject>,<transitions> ● For attributes with multiple elements, separate with spaces. ● Transitions are written like this: <current_state>/<input_symbol>/<next_state>/<tape_character>/<L or R> python3 traceTM_djohan.py <Turing Machine csv> <Max Transitions> <string0> <string2> ... <stringx>
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I used three different turing machines to test the robustness of my code. The a+ turing machine was provided in the project outline, so I used it as my first test to see how my code would work for a tape that only moves to the right. I also tried a similar simple turing machine that accepted the language a*b*c* which also worked. Finally, I tried implementing a turing machine that goes both directions, which is the a^nb^n TM to make sure everything worked as intended. When I ran it, it followed the process that a regular turing machine would go through for multiple different inputs and correctly accepted/rejected them, so it was a clear sign that my code was doing it correctly.

12	How you managed the code development: I started by coming up with a format to write the turing machines csv, and after that I wanted to see if I could correctly classify and organize the different elements of the turing machine. Once that was finished, I began testing how I would transition between the different states. I got my code to work for the a+ and a*b*c* turing machines, but when I got to the a^nb^n one, I had a couple set backs trying to implement constant forward and backwards movements of the tape, but once that was figured out, I tested my code to make sure everything worked properly, and after that, I finally worked on formatting the output the way I wanted.
13	Detailed discussion of results: Overall, I am very satisfied with the results of the project. All of my code worked the way I intended; and I'm fairly impressed with how well the code is able to read in a variety of turing machines correctly. This project gave me a better understanding of how turing machines work and was a good practice of using BFS to find all configurations along the tree structure. The outputs meet all the requirements specified by the outline. All transitions are shown between each level, and the appropriate statistics such as transitions, depth, non-leaf nodes, and nondeterminism are provided.
14	How team was organized: Individual, so none
15	What you might do differently if you did the project again: The main thing I would've improved would be improving code efficiency. There were likely a couple things I could've changed in my code to make it run faster, and the formatting could've used some more work, but given the time crunch, I decided that those changes weren't crucial, but if I were doing this project again, I'd like to challenge myself to see how I could improve the overall work.
16	Any additional material: N/A