Project TM Readme Team nefario

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: team nefario
2	Brandon Hulse (bhulse)
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: 5-6 hours
6	Link to github repository: https://github.com/bhulse72/TOC-TM-proj-dr_nefario
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	
TM_trace_nefario.py	the program that traces the inputted Turing Machine and input string
Test Files	
testFolder_nefario	folder of Turing machines used for testing the program
Output Files	
outputFolder_nefario	contains the results of the test turing machines and test input strings
Plots (as needed)	
Project-Analytics_nefario .xlsx	contains a table of runs to demonstrate Turing

	<table><tr><td></td><td>machine traces (data)</td></tr></table>
8	Programming languages used, and associated libraries: python (csv, sys, collections)
9	Key data structures (for each sub-project): trees, lists
10	General operation of code (for each subproject) loads configuration from a CSV file and simulates the machine's operation using a breadth-first search approach. The NTM configuration includes states, input and tape alphabets, transitions, and special states (start, accept, and reject). The simulation explores all possible configurations (branches) simultaneously, maintaining a tree of configurations at each depth level. It halts when an accept state is reached, when all configurations reach the reject state, or when predefined limits on depth or transitions are exceeded. The program tracks and prints the path to acceptance, if any, including the sequence of configurations leading to the accept state. The user specifies the input string, the machine's file, and optional limits for depth and transitions when running the simulation.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. For a*b*c* machine, I used test cases of aaa, abc, and abcabc to demonstrate the correctness of the trace. aaa would be accepted, abc would be accepted, but not abcabc. for the a+ machine, I used test cases of aaa and bcbcb. aaa would be accepted but not bcbcb.
12	How you managed the code development since it was a solo project, I just did it at my discretion using git for iterative development
13	Detailed discussion of results: The results of the NTM simulation project provide a fascinating insight into how nondeterministic Turing machines behave across a variety of input strings and configurations. For each test case, the machine traces through its configuration tree, calculating key metrics such as the maximum depth reached, the total number of configurations explored, and the average nondeterminism at each step. One consistent observation is that as the input string length increases, the average degree of nondeterminism also climbs, reflecting the exponential growth of possible paths the machine must explore. This phenomenon is particularly evident in cases like the abc_star machine processing the string aaabbcccc, where the average nondeterminism reached a striking 6.6 compared to much lower values for shorter strings. This increase highlights the inherent tradeoff in simulating nondeterministic behavior: while short strings result in compact and efficient configuration trees, longer strings drastically expand the search space due to the machine branching into multiple possible configurations at each step. Tracing a nondeterministic Turing machine reveals the intricate mechanics of how it explores multiple potential paths simultaneously to determine whether a string is accepted or rejected. For instance, in testing the abc_star machine with the string aaa, the machine quickly reached an accepting state with a relatively shallow tree depth of 4 and an average nondeterminism of 5.75. However,

	even for strings that are eventually rejected, such as bcbcb under the a_plus machine, the machine's exploration process illustrates how it exhaustively checks all potential paths, ultimately ensuring that no accepting configuration exists. The depth and configuration count for these cases were notably smaller, reflecting the deterministic-like behavior when the input fails early. This ability to rigorously explore and eliminate possibilities is one of the strengths of nondeterministic computation, even as it requires significant resources for larger input strings. One of the limitations of this project is tied to the computational feasibility of tracing an NTM for arbitrarily long strings. Since the machine relies on a breadth-first expansion of the configuration tree, the exponential growth in possible paths makes it impractical to process very large strings or machines with high branching factors. This issue becomes even more apparent when examining the number of configurations explored for strings like aaabbcccc, where 66 configurations were generated before reaching an accepting state. While smaller machines like a_plus can simulate simpler patterns with fewer resources, the limitations of memory and processing power inevitably cap the size and complexity of input that can be effectively handled.
14	How team was organized solo project; did it myself
15	What you might do differently if you did the project again I would change it so that my program took arguments for the command line. It was a bit annoying typing input strings and file names into the promt over and over; should have just made the input on the command line so I could do a bash script or something
16	Any additional material: no