Project 2 Readme Team tfriedma

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"

<table>
<tr><td>1</td><td>Team Name: tfriedma</td></tr>
<tr><td>2</td><td>Team members names and netids: Ty Friedman - tfriedma</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: Program 1: Tracing NTM Behavior</td></tr>
<tr><td>4</td><td>Overall success of the project: Very successful</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 7 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/tyfriedman/NTM-tracing</td></tr>
<tr><td>7</td><td>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.
<table>
<tr><th>File/folder Name</th><th>File Contents and Use</th></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>traceTM_tfriedma.py</td><td>This is the file in which I do all of the actual work. It is broken into a main and three functions. Main is where you specify the machine to use, the max depth that the program will go to before stopping, whether you want to see the state transitions or not, and where the trace is called. Main is where you set everything that you need to before running the file.
The functions are what do the work of the program. The trace_tm function is what does the breadth-first search for an accept state. The get_transitions function returns possible next transitions based on current state and input. The parse_tm_file function is what reads in the file that specifies how the machine acts.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>input_a_plus_tfriedma.csv</td><td>This is an input file for the non-deterministic</td></tr>
</table>
</td></tr>
</table>

	Turing machine that accepts the language a+
input_a_plus_DTM_tfriedma.csv	This is an input file for the deterministic Turing machine that accepts the language a+
input_equal_01s_tfriedma.csv	This is an input file for the non-deterministic Turing machine that accepts the language that is an equal number of 1s and 0s
input_equal_01s_DTM_tfriedma.csv	This is an input file for the deterministic Turing machine that accepts the language that is an equal number of 1s and 0s
input_abc_star_tfriedma.csv	This is an input file for the non-deterministic Turing machine that accepts the language a*b*c*
input_abc_star_DTM_tfriedma.csv	This is an input file for the deterministic Turing machine that accepts the language a*b*c*
Output Files	
output_a_plus_tfriedma.txt	This is the output from four different input strings ran on the machine that accepts the language a+
output_equal_01s_tfriedma.txt	This is the output from four different input strings ran on the machine that accepts the language that is an equal number of 1s and 0s
output_abc_star_tfriedma.txt	This is the output from four different input strings ran on the machine that accepts the language a*b*c*
Plots (as needed)	
There are no plots for this project, but Dr. Kogge specified specific things to be discussed. These things are discussed below in point 13 - detailed discussion of results.	

8	Programming languages used, and associated libraries: language - python libraries - csv, deque
9	Key data structures (for each sub-project): Only one project - NTM Tracing - the machine is a dictionary - the search is implemented with a queue

	- the states are 4-tuples that contain the left tape, the state, the right tape, and the path of states as a list of states (tuples) - the input strings are strings
10	**General operation of code (for each subproject):** NTM Tracing: This code starts by specifying the inputs (machine, string, whether to print states or not, and the max depth) in main. Main calls the function that parses the machine, which returns a machine. The machine data structure is a dictionary that contains things like the start state, the name, the alphabet, and the transition functions most importantly. The transition functions are also a dictionary. The key is the current state and the input character, and the values are a list of states that the machine could transition to. If there are multiple states in the list, then the machine is non-deterministic. After the machine input file is parsed, the trace_tm function is called. This will initialize a queue that holds all of the states that need to be evaluated, and the start state of the machine is initially the only thing on it. Each state is a 3-tuple that contains the left side of the tape, the current state, and the right side of the tape. The first character on the right side of the tape is the character that is currently being evaluated. Then, while the queue is not empty, we start to evaluate states by calling the get_transitions function. This returns all of the next possible states, which we evaluate one by one. As we do this, we are pushing all of the transitions that are possible from these states onto the end of the queue. If the state that is being evaluated is the accept state, then we end evaluations and print the metrics. Additionally, as we are evaluating states on the queue, we are keeping track of the paths that each state has taken, so that we have the option to print out the full path at the end. Finally, if we evaluate states to the point that the queue becomes empty (because there are no valid transitions), then we have reached a reject state and the input string is not in the language. This also prints out metrics before returning.
11	**What test cases you used/added, why you used them, what did they tell you about the correctness of your code.** I added all of the test cases given by the class except for the palindrome example. This is because I found flaws with the machine when I drew out a state diagram for it. I included these files because I was able to take the time to verify their correctness by drawing out the state diagrams for them and checking them with example input strings. Additionally, there were also DTMs that were provided, with which I could double-check the inputs. I was then able to verify the correctness of my code by checking to see if I entered all of the same states that I did when I drew out the machines.
12	**How you managed the code development** I did all of my code using VS Code and used Git to hold and share my code. All code, inputs, outputs, and this readme should be available in the Git repo. Because I was working alone, I didn't need to create multiple branches and I was able to work on the master branch the entire time.
13	Detailed discussion of results: **How correctness was verified:** - I verified correctness in multiple ways. The first way was to check many different inputs vs. deterministic and non-deterministic machines to ensure that the output matched. You will notice that I have included three versions of machines that are

	DTMs instead of NTMs in my input files. These were used to ensure that the correct output was decided depending on the input string. The second thing that was done to verify output was that I added the ability to print out all of the states that the machine goes through to get to the accept state. This allowed me to draw out the machine and then verify that the code implementation of the machine was following the same sequence of states.
14	How team was organized: This was a one-person team, so I had control over the entire project, including verifying that my outputs were correct and writing all of the code.
15	What you might do differently if you did the project again If I were to do this project over again, I would specify exactly how I would be dealing with states, because that is what tripped me up when writing the code. I didn't clearly understand how I would be representing states in my code and later that led to a lot of confusion. I cleaned it up, but it took extra time, so I should have just taken the time to write everything out before coding.
16	Any additional material: N/A