Project 02 Readme Team erhee

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: erhee</td></tr>
<tr><td>2</td><td>Team members names and netids: Elizabeth Rhee; erhee (just me)</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: Project 1: Trace TM behavior</td></tr>
<tr><td>4</td><td>Overall success of the project: somewhat working- the code ran through and reached an end. Sometimes the end state was correct; however, the full output, especially the depth and paths taken, was off.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 20 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/ejr88/nd-cse-30151-project02-erhee</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><td>File/folder Name</td><td>File Contents and Use</td></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>traceTM_erhee.py</td><td>Code to implement the NTM tracing</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>a_plus.csv
abc_star.csv</td><td>Csv files given by the professor to test the traceTM code to see if it outputs whether an inputted string is accepted or rejected.</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>output1.txt
output2.txt
output3.txt</td><td>These are output text files in the form of tables to show the results of running the code</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
</table>
</td></tr>
</table>

	output1_erhee.png output2_erhee.png output3_erhee.png — These are pictures of the tables
8	Programming languages used, and associated libraries: Python in google colabs. The os, csv, collections libraries were used.
9	Key data structures (for each sub-project): Lists and list of lists of a triple. Dictionaries were also used to keep track of transitions. Tuples were present as well
10	General operation of code (for each subproject): The code starts in main where it takes in a csv and gets the name, the start state, the accept states, the reject states, and the transitions from it. Once that part is done, the code creates an NTM object using these values, and it then asks for string input from the user and a max depth as well. Next, it calls the bfs_ntm_trace function. The trace function will take the input and create a deque from it so it can pop the leftmost value and perform breadth first search. This will run while the queue is not empty and the max depth has not been exceeded. It pops the first value on the queue, and checks if the state is in an accept state, from which it will break out of the while loop. If not, then it is a reject and it will continue in the loop, potentially moving to a transition. The transition code will attempt to move the head in the specified direction (right or left) and provide a new tape. After the while loop, the code will return the accept, the # of transitions, the path, and the height it went to. This will then be printed out as well as put into text files.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I did not add any new test cases, but I used multiple of the csv files that were provided by the professor to test the code. I used these files to see if the code did what should have been done based on the string input and the contents of this file. I would also test with different strings to see how it would check out. I also used the professor's project file to see if my code matched some of the examples in it.
12	How you managed the code development: I began by reviewing how Turing Machines work, and specifically how NTM's work. I figured out what I needed, especially when it came to the objects of NTM and tape. I decided to create a class for them to make it easier. Instead of making a main, I just did not indent the main code. To debug, I used multiple print and break statements, especially in main, so I could check singular values.
13	Detailed discussion of results: The program was only somewhat working, unfortunately, so these results are most likely incorrect and deviate from what it was intended to do and show. For some of the examples, such as the aplus, I believe I got the correct end state, and almost the correct transitions. However, some paths were missed and depth was off. The time complexity of an NTM is the maximum number of steps it takes to reach a halting point for a string of length n. The time complexity of BFS is O(V + E); however, sometimes the program will find an accept first and stop before it goes through the entire tree, which shortens

	the time. And because nondeterminism allows the program to explore multiple pathways simultaneously rather than going one at a time, so this will speed up the process somewhat compared to a deterministic turing machine. The average non determinism was not too high because my depth was messed up, so many of the runs were similar time complexity.
14	How team was organized Team was organized by me. I did all of the coding and project
15	What you might do differently if you did the project again I would spend more time understanding all cases. I would also start earlier.
16	Any additional material: I apologize- I thought the table of the 6 things was what you wanted displayed, not a table here in the discussion. I did not have enough time to add that here by the time I realized.