Project 3 Readme Team spickfor

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: **spickfor**
2	Team members names and netids: Seth Pickford (**spickfor**)
3	Overall project attempted, with sub-projects: **Program 1: Tracing NTM Behavior**
4	Overall success of the project: **Successful, works as DTM and NTM**
5	Approximately total time (in hours) to complete: **15 hours**
6	Link to github repository: https://github.com/spickfor/TheoryProject2.git
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_spickfor.py **Test Files** • check_2x0_DTM_spickfor.csv • check_a_plus_DTM_spickfor.csv • check_a_plus_spickfor.csv • Check_abc_star_spickfor.csv • check_palindrome_DTM_spickfor.csv • check_palindrome_spickfor.csv **Output Files** • output_2x0_DTM_spickfor.txt • output_a_plus_DTM_spickfor.txt • output_a_plus_spickfor.txt • output_abc_star_spickfor.txt • output_palindrome_DTM_spickfor.txt • output_palindrome_spickfor.txt

	Plots (as needed)
8	Programming languages used, and associated libraries: **Used Python and utilized the CSV library**
9	Key data structures (for each sub-project): ● **The key data structures used are Tuples, Lists, and Strings. The Tuples represented configurations and transitions. The Lists were used to create the tree and were searched with Breadth First search. The strings were the actual data such as the tape contents, symbols, and states.** ● **The program parses the inputs from the input CSV file. Then it sets up the starting configuration with the input string on the tape. Then it explores all possible transitions from each configuration using a breadth-first search. It stops if the machine accepts, there are only rejects, or if it exceeds the limiting depth. The program then outputs the trace as a path of configurations, transitions, and nondeterminism.**
10	General operation of code (for each subproject) **Go into the program and run the "simulate_ntm()" function with inputs to it being the input file, input string, output file. It will then print the trace to the command line and write it to the specified file**

11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. **I used the test case that was provided in the directions for the project, however, the example seemed to be off so I had to make sure that what I was doing was correct. I chose this one for the simplicity. I also used other test files that I got from the google drive. I used these because I could assume them to be correct implementations of TMs and could therefore easily validate that my program worked based on its traces and based on whether it accepted or rejected. I knew that if I could put in strings I knew would be accepted by the TM and have them accepted and have string be rejected that should be rejected then the program worked. I also used both DTM and NTM machines so that I could validate that my Nondeterminism calculation was correct by looking at DTMs and making sure they had a nondeterminism of 1 and the NTMs had > 1**
12	How you managed the code development **I wrote all the code in python on my local machine and then pushed all at once to the repo**

13	Detailed discussion of results: **I determined that my program was correct in two ways. The first way was by inputting strings I knew would accept or reject and then looking at the traces and whether the program accepted or rejected to visually validate that the program was functioning as it should. The traces were helpful for making sure that the program moved through the TM properly. The second way I validated my correctness was by using DTM and NTMs. I could validate the correctness of my nondeterminism calculation by making sure that it gave a calculation of 1 for DTMs and >1 for NTMs. The test cases I used for validation came from the google drive, so I could assume correctness. Some of the test cases I used had a NTM and DTM that worked for the same language, which was extremely helpful for validation of the nondeterminism calculations because I could verify that the DTM was getting one and then the NTM was a number greater than 1. The traces output by the program stepped through each depth/level and so it was easy to see where the head was at what level, how the program moved through the tape, and what lead to accept or rejects. The nondeterminism varied across the NTMs, but was obviously 1 for all the DTMs. For the a+ NTM, it had a nondeterminism of 1.43, the a*b*c* had a nondeterminism of 1.57, and the palindrome had a nondeterminism of 1.25. So the most nondeterministic machine was the a*b*c*, which makes sense because the * adds a lot of nondeterminism because at each letter, it can choose to stay or to move on.**
14	How team was organized **I was a solo team so I did everything other than creating test cases**
15	What you might do differently if you did the project again **I would validate and double check the check files because I spent a long time trying to figure out why my outputs were strange, only to eventually figure out that it was how the person had designed the input file. I also would probably create my own test file for some language so I could validate myself. I also would start my logic writing with diagrams to help me visualize how TMs work, it would have saved time.**
16	Any additional material: **None**