Project Readme Template

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: Yohannes "The Bulldozer" Mariam
2	Team members names and netids Yohannes Mariam (ymariam)
3	Overall project attempted, with sub-projects: ntm_tracer
4	Overall success of the project: Sucessful
5	Approximately total time (in hours) to complete: 8 hours
6	Link to github repository: https://github.com/ymariam1/Project1-TOC
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	
Ntm_tracer.py	This file contains the function to run the NTM_tracer function and perform a BFS trace of the NTM aswell as print out the tree.
Test Files	
Aplus.csv Composite.csv Find_101.csv Union_anbn_ancn.csv Palindrome.csv	These csv files were used to test the ntm tracer. Aplus, composite, and palindrome were created by other students. I created Union_anbn_ancn.csv and find_101.csv to test the machine.
Output Files	

Aplus_output Composite_output Find_101_output Palindrome_output Union_anbn_ancn_output	These files print the result of running the NTM_tracer class on the above inputs. Prints traced tree, and whether or not string is accepted.
Plots (as needed)	
N/A	

8	Programming languages used, and associated libraries: Python sys Csv Argparse
9	Key data structures (for each sub-project): Config (list): Stored the different states of the NTM machine. Has syntax [left stack, current state, right stack] Tree (list of lists): List of configs, each config was grouped into specific lists which were separated by depth Transitions (List): List of transitions the machine can take in a given state. Loops through transitions to explore all possible branches to find an accept state.
10	General operation of code (for each subproject) NTM_tracer: The tracer class uses the get_transitions function to find every possible transition that the machine can make given the current state and head of the tape. In doing this the machine can simulate nondeterminism by going through and simulating every possible path that the machine can take. When evaluating every configuration the machine checks if it is currently in an accept/reject state (where it will stop the machine and print success or kill that branch respectively). Then it checks if there are any available transitions (if not treats as a rejection). Then we generate every possible child configuration and explore them. Print_trace_path: In order to print the tree I modified the given function to take the final path, total nonleaf nodes, and total transitions seen in the bfs generated by the search tree in NTM_tracer. We print the configuration at each level and on separate lines.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I started by testing against the aplus.csv to first check the correctness of my code. I chose this as a start because it was a pretty small NTM that would demonstrate

	nondeterminism. Once I got correct results from aplus.csv I started to test on palindrome.csv which was a test NTM shared by another student. This was a little more complicated and opened me up to edge cases that I had not considered before. Lastly I built union_anbn_ancn.csv and find_101.csv. When these followed the expected results for the inputs I knew that my code was in good shape.
12	How you managed the code development: Since this was a solo project, the whole development cycle was on my shoulders. I started by outlining the algorithm in pseudocode, and walking through what the program would need to do each step of the aplus.csv NTM on paper. When I had an intimate understanding of the algorithm I began put this into code. With this first pass working for aplus.csv I moved to more complex NTM and began adjusting to handle edge cases. A key step to debug was to print things like the config or the level, or the different sides of the tape just to make sure everything flowed logically and as desired.
13	Detailed discussion of results: Overall, my results were good. Code runs as expected and produces correct outputs based on the provided turing machine. Ntm_tracer.py is able to run through every possible transition and find an accepting path (if one exists) using a BFS. Valid inputs are either accepted, rejected, or reach the max depth, and invalid inputs are correctly rejected. Finally, I calculated the degree of nondeterminism by dividing the total number of transitions to by the number of nonleaf transitions.
14	How team was organized: N/A
15	What you might do differently if you did the project again: I if I had the chance to do this project again I would build and utilize more helper functions. The while loop in run is very very long, and things like applying transitions could be put into a helper function in order to clean up the code.
16	Any additional material: N/A