Project 2 Readme Team GC

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: GC</td></tr>
<tr><td>2</td><td>Team members names and netids: Giancarlos Reyes, GReyes2</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: NTM trace</td></tr>
<tr><td>4</td><td>Overall success of the project: Success !</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 6 hours…</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/GReyes87/Project2-TOC.git</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>./src/ntm_tracer.py</td><td>This file contains code for the functioning ntm tracer.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>./input/aplus.csv</td><td>Provided from forked repo</td></tr>
<tr><td>./input/composite.csv</td><td>Provided test file from googlesheet</td></tr>
<tr><td>./input/equal_01s.csv</td><td>Provided test file from googlesheet</td></tr>
<tr><td>./input/even_a.csv</td><td>Created this test even a.</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>./output/aplusfilescreenshot/apluscsvaaa.png</td><td>This is the screenshot</td></tr>
</table>
</td></tr>
</table>

		for the ‘aaa’ test on the aplus csv.
	./output/equal01outputscreenshots/equal01firstscreenshot.png	This is the screenshot of the first part of the output for the equal01 test, I had to take two screenshots since the output was long.
	./output/equal01outputscreenshots/equal01secondscreenshot.png	This is the screenshot of the second part of the output for the equal01 test, I had to take two screenshots since the output was long.
	./output/even_ascreenshot/even_ascreenshot.png	This is the screenshot of the output for the evena test.
	./output/screenshotforcompositefile/compositefirstscreenshot.png	This is the screenshot of the first part of the composite csv file test using input ‘111111’.
	./output/screenshotforcompositefile/compositesecondscreenshot.png	This is the screenshot of the second part of the composite csv file test using input ‘111111’ since it was long.
	./output/screenshotforcompositefile/compositethirdscreenshot.png	This is the screenshot of the third part of the composite csv file test using input ‘111111’ since it was long.
	./output/screenshotforcompositefile/compositefourthscreenshot.png	This is the screenshot of the fourth part of the composite csv file test using input ‘111111’ since it was long.
	Plots (as needed)	
	none	none
8	Programming languages used, and associated libraries: I used Python	

9	Key data structures (for each sub-project): The key data structures used are lists of lists to represent the breadth-first configuration tree, lists for individual configurations, and dictionaries to track parent relationships and transition mappings.
10	General operation of code (for each subproject): The code simulates a nondeterministic Turing machine by exploring all possible configurations level by level using a breadth-first search, stopping when an accept state is found or when all computation paths lead to rejection.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code: I tested my program using aplus.csv with input aaa because it is the basic example given in the assignment and should accept in only a few steps. This helped confirm that the breadth-first search, accept detection, and accepting-path tracing were all working as expected. I used composite.csv with input 111111 since it produces a much larger and more nondeterministic computation tree. This test showed that the code can correctly explore many branches, keep track of parent configurations, and still stop as soon as an accepting path is found. I also tested even_a.csv with input aaaa, which is mostly deterministic, to make sure the tracer behaves correctly when there is little or no branching and still reports the correct depth and transition count. Finally, I tested equal_01s.csv with input 0011 to check that the code handles nondeterminism properly and finds an accepting path only when the number of 0s and 1s is equal. I also ran rejecting inputs for these to verify that the program correctly reports rejection.
12	How you managed the code development: I built the code step by step, starting from the skeleton and getting a basic version running before adding more features. I tested each change using small machines like aplus to make sure things were working, then moved on to larger examples like composite to catch problems early. Whenever something didn't behave as expected, I fixed it right away and retested instead of changing everything at once.
13	Detailed discussion of results: Across all the test machines, the tracer behaved consistently with the expected theory of nondeterministic Turing machines. For simpler machines like aplus and even_a, the configuration tree stayed small, and the accepting path was easy to follow, which confirmed that the BFS logic and transition handling were working correctly. For more complex machines like composite and equal_01s, the tree grew much deeper and

	wider, but the program still tracked depth, total transitions, and the correct accepting path without crashing or looping incorrectly. These runs showed that the code can handle nondeterminism, implicit rejections, and long computation paths while still producing clear and structured output.
14	How team was organized : Worked by myself
15	What you might do differently if you did the project again: If I did the project again, I would simplify how transitions are stored and handled, and I would add a cleaner way to limit or shorten the output for large machines. I'd also test more often so fewer fixes were needed later.
16	Any additional material:n/a