Project 2 Readme Team CD

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: CD</td></tr>
<tr><td>2</td><td>Team members names and netids - Andre Mayard, amayard</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: 4 hrs</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/amayard27/Project2-TOC</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</td><td>ntm_tracer_CD.py</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>input</td><td>ntm_n1n.csv</td></tr>
<tr><td></td><td>aplus.csv</td></tr>
<tr><td></td><td>composite.csv</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>.</td><td>output.txt</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td></td><td></td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries: python</td></tr>
</table>

9	Key data structures (for each sub-project): The program uses configurations stored as **lists** of four elements: the left tape, the current state, the right tape, and the parent index. These configurations are organized into a **tree** where each level is a list of configurations representing that BFS depth. Transitions are kept in a **dictionary** keyed by (state, symbol) and the values are **lists** of transition descriptions. A path is reconstructed by following parent indices level by level back to the root.
10	General operation of code (for each subproject) The program runs a BFS over all possible computation paths of the NTM. It starts from the initial configuration and expands level by level, checking each configuration for accept or reject status. If it is neither, it looks up valid transitions and generates new configurations with updated tape content and a parent pointer. The search continues until an accept state is found, all branches reject, or the maximum depth is reached. Once it ends, it reports acceptance or rejection and reconstructs the path if accepted.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I ran my code using various 0 and 1 combinations and determined that it returned the right answer each time - I edited and used the ntm_n1n.csv file to test this. I verified these answers by tracing through it myself, and I verified it was accurate. I also edited and used the composite and aplus csv's to test my code, with each accepting when expected. **The edits in these csv's were to fully implement the 0n1n (which was originally made to accept equal 0's and 1's) as well as putting a q0 transition and assumed start with $ in each.**
12	How you managed the code development I wrote my own pseudocode for both the print function and the actual tracing function, and then I implemented that with some hurdles to eventually it working. I did it all in one sitting
13	Detailed discussion of results: The results showed each time that the code was correct, indicating that there were no issues with the code. I therefore implemented an NTM tracer. The test aaa took a depth of 5, the test of 111111 on the composite took 28, and my own test of 000111 on the ZeroNOneN implementation took 52. These depths show how much branching and backtracking the machine must do. Unary strings resolve quickly, the composite test grows with the number of nondeterministic choices, and the ZeroNOneN case takes longest because the machine repeatedly scans and marks symbols while verifying equal counts, creating a larger search tree even for smaller input sizes.
14	How team was organized - I did all the work
15	What you might do differently if you did the project again - I wouldn't do anything differently besides starting earlier.
16	Any additional material: None