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"

<table>
<tr><td>1</td><td>Team Name: Lonely_Garcia</td></tr>
<tr><td>2</td><td>Team members names and netids:
Name: Rene Alzina
NetID: ralzina</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects:
Attempted the NTM simulation with the NTM Trace</td></tr>
<tr><td>4</td><td>Overall success of the project:
Fully successful, all cases work well with the NTM and the trace of the accepting path or the longest rejected path works well.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete:
4 hours</td></tr>
<tr><td>6</td><td>Link to github repository:
https://github.com/ralzina/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/ntm_tracer.py</td><td>Implements the tracing of NTM tree and handles the reconstruction of the accepting path or the longest rejected path</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>Input/*</td><td>These files have the different cases tested with the NTM: 0^n1^n, composite numbers, and a+</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>Stdout</td><td>The program prints everything into stdout so there are no specific output files</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>N/A</td><td>N/A</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:
Python
Didn’t add any aditional library than was provided in the original github</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):
Use a tree with list of lists to perform breadth first expansion of each depth level in the NTM tree for each possibility. This tree allows us to find when the first correct answer is reached as well as the reconstruction of the accepting path or longest rejected path.</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject)
Use a loop that runs until you accept the string, reject the string, or reach the max depth of the tree.

In each loop iteration, start of by getting the last list that was added to the tree, which is the list of all states that will be expanded. At each state, get the possible transitions based on the character on the tape, and add each possible transition to the new list for the next level of the tree. If you ever reach an accept state or a reject state (no more valid transitions were found), stop and send the tree to the function that will reconstruct the accept path or the longest reject path and print it. If you didn’t reach an accept or reject state, continue with the loop.

The function that reconstructs the accept path or the longest reject path takes the tree as an argument and the final node which is either the accept state or the node that had no valid transitions or reached a reject state. From there, it basically becomes a DFA that traverses the longest path. First, I reconstruct the path to know which state comes after each. Then, I start again with the input string, and I go through the string, but with the reconstructed path I now know what the next state is I need to go to even if there are multiple transitions for the same character in the tape. As I reconstruct this path, I print the current configuration of the state and the tape.</td></tr>
<tr><td>11</td><td>What test cases you used/added, why you used them, what did they tell you about the correctness of your code.
I added 3 test cases. I added a+, 0^n1^n, and composite

I used these test cases because they were recommended in the project description file, and since these test cases are all different and work with my same implementation, it means that my NTM simulation is adequate and produces correct results.</td></tr>
</table>

12	How you managed the code development I first looked at the provided GitHub repository to understand what each function was meant to do and understand the behavior of the code. Then, I realized I needed to add the functionality for ntm_tracer.py. From there, I implemented the requested function that uses a tree as a list of lists and at each level gets valid transitions from the current state and tape character and then evaluates where to go next while managing the nondeterminism as well. After that, I continued to the reconstruction of the path, where I realized it would be much easier if I knew at each level, what was my previous state that brought me to the current level. So, I went back to add this extra functionality to the past function so that when I had to reconstruct the path, it would be much easier.
13	Detailed discussion of results: The results were correct because the machine clearly demonstrates when a string is accepted or rejected since it also shows the final correct accepted path or the longest rejected path. I know they were correct because not only do my results show when a string is accepted or rejected, it shows a trace of each state to understand why. The results were very impressive because it proves how NTMs can solve any problem that a modern computer can solve, and by implementing it in code, I have gained a much better understanding of how NTMs work.
14	How team was organized I was the only one on my team, so I took care of developing the entire code.
15	What you might do differently if you did the project again If I could do this project differently, I would validate that the test cases are correct before testing them with my code so that I don't think that my code is the problem. I spent some time trying to debug my code when the test case was what was wrong.
16	Any additional material: I created tests for composite numbers and for the language 0^n1^n. These can be found in the input/ folder