Project 2 Readme Team cdevine5

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: cdevine5</td></tr>
<tr><td>2</td><td>Team members names and netids: Charlie Devine, netID: cdevine5</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: tracing NTM behavior given various input strings</td></tr>
<tr><td>4</td><td>Overall success of the project: The project implementation was pretty successful. I was able to simulate and analyze NTMs effectively using various test cases. The simulator handled input strings correctly, produced configuration trees, and provided insightful metrics such as the average nondeterminism and depth of computation.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 6</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/charlesdevine/project2_theory_cdevine5</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><th>File/folder Name</th><th>File Contents and Use</th></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>NTMtrace_cdevine5.py
testingfile_cdevine5.py</td><td>NTMtrace incrementally explores NTM configurations, optimizing transitions for efficient non deterministic simulation.

Testing file tests a nondeterministic Turing machine with input strings, tracks its behavior, and calculates</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>metrics like nondeterminism and tree depth.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>data_a_plus_cdevine5.csv</td><td>File defines the behavior of a NTM for the language a+, which accepts strings with one or more 'a's</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>output_resultsfile_cdevine5.txt</td><td>Outputs the following data: which machine, what input string, whether it was accepted or rejected, depth, number of configurations explored, average non-determinism (# configurations / depth).</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: The code is written in Python using standard libraries like csv, typing, and collections.defaultdict.</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project): This used lists, strings, tuples, and dictionaries.</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):

The NTM trace script implements a NTM simulator that can process arbitrary input strings. When initialized, the simulator loads machine specifications from a CSV file, including the machine's transitions, start state, accept state, and reject state. The core simulate method explores all possible computational paths for a given input by generating a configuration tree. At each step, the simulator can create multiple configurations based on the possible transitions from the current state and input symbol. It tracks these configurations across depth levels, allowing for non-deterministic behavior where the machine can simultaneously explore different computational paths. The simulation terminates when it either reaches an accept state, all paths are rejected, or the maximum number of steps is reached. The script captures detailed information</td></tr>
</table>

	about the computation, including the configuration tree, number of transitions, and whether the input was accepted or rejected. The provided code analyzes the behavior of a NTM by simulating its operation on a list of test input strings. The analyze_ntm function initializes the NTMSimulator with the machine definition specified in a CSV file. For each input string, it runs the simulation up to a maximum number of steps, generating a tree of configurations representing the machine's states and tape contents at each point. It writes detailed information to an output text file, including the input string, the configuration tree at each depth, the depth of explored transitions, and the total number of configurations the machine explored. The analysis also evaluates nondeterminism, measuring it as the average number of configurations per depth level in the tree. Based on the nondeterminism value, it categorizes the machine's computation as deterministic, slightly nondeterministic, moderately nondeterministic, or highly nondeterministic, providing a comment for each scenario. This analysis allows insight into the machine's nondeterministic behavior and efficiency in exploring different computational paths.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I used a diverse set of test input strings with varying lengths and characters to demonstrate the Turing machine's behavior across accepted, rejected, and nondeterministic paths. The test cases included short strings like "" (empty string) and "a" to verify basic acceptance and rejection conditions. Medium-length strings such as "aa" and "aaa" allowed me to observe transitions and nondeterministic branching in more typical scenarios. Additionally, longer strings like "aaaaa" and "aaaaaaaa" were useful in testing the scalability and depth of the machine's exploration, ensuring that all possible paths were accurately simulated. I also included invalid strings like "ab" and "b" to confirm that the machine correctly rejects inputs that do not match the language of "a-plus." By combining different string lengths and character combinations, I showcased the machine’s correctness in exploring accepted and rejected paths and highlighted its nondeterministic behavior, ensuring complete coverage of its computational operations.
12	How you managed the code development I managed code development within VScode. I was able to successfully and efficiently manage all of my development within one folder and two scripts and stored all of my input and output files within the same folder in VScode.
13	Detailed discussion of results: The results from analyzing the NTM showed that the simulator effectively handled a variety of test strings, accurately determining accepted and rejected inputs. The test cases included different string lengths and character combinations, which highlighted the machine's ability to explore multiple paths simultaneously due to its nondeterministic nature. As input complexity increased, the configuration tree depth and the number of

	states expanded, as well as the degree of nondeterminism, reflecting the machine's branching transitions. These results confirmed the simulator's correctness and robustness in exploring all possible transitions while maintaining performance across various input scenarios.
14	How team was organized I worked by myself for this project.
15	What you might do differently if you did the project again If I were to do this project again, I would take more diverse test cases to see how that would impact the performance of the algorithm I developed. Working in a team may have helped streamline the process more efficiently, using a different set of eyes and minds to look over the code and see if any improvements could be made.
16	Any additional material: