Project 2 Readme Team cdeleon2

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: cdeleon2</td></tr>
<tr><td>2</td><td>Team members names and netids: Cesar De Leon - cdeleon2</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: Program 2: k-tape Turing Machine</td></tr>
<tr><td>4</td><td>Overall success of the project: In terms of meeting the objectives, I'd say the project was successful as I created a multi-tape Turing Machine simulator with dynamic test-case generation and pretty effective file management (output files are automatically stored in appropriate folders).</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 18</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/rasecde/k-tapeTM_cdeleon2</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 (src folder)</td></tr>
<tr><td>src/multiTapeTMSimulator.py</td><td>Implements the multi-tape Turing Machine simulator, which processes the input file, simulates the transitions based on defined rules, and generates an output file with the results.</td></tr>
<tr><td>src/runSimulator.py</td><td>Coordinates the workflow of the simulator by generating test cases, running the simulator, and saving input/output files to the appropriate folders</td></tr>
<tr><td>src/testCaseGenerator.py</td><td>Dynamically generates random Turing Machine input files, including tape configurations and transition rules, for testing the simulator.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td colspan="2">Test Files (Input Files Folder)</td></tr>
<tr><td>Input Files (folder)</td><td>Contains all the input files generated by testCaseGenerator.py, such as InputDefinition1.txt, which defines the initial state of the tapes and the machine's transition rules.</td></tr>
<tr><td colspan="2">Output Files (Output Files)</td></tr>
<tr><td>Output Files (Folder)</td><td>Stores the output files produced by the simulator, such as SimulationOutput1.txt, which logs the simulation steps, tape states, and the halting condition.</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:

Python
Libraries:
● Os
● Sys
● Subprocess
● glob</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):

multiTapeTMSimulator:
This script uses a few data structures for its functionality. In this case, the tapes are represented as lists of characters, including a separate list that tracks the position of the tape heads for each tape. The states of the Turing Machine are stored in a set for fast lookups and make sure that there are no dupes. The transition rules are stored as lists, where each rule contains the current state, input symbols, next state, replacement symbols, and head movements.

runSimulator:
This script uses strings and functions to manage the overall workflow. Strings are used to manage file paths, allowing for efficient handling of input and output files across different folders. The functions throughout allow for test case generation, simulation execution, and file renaming, while lists and glob patterns help identify and organize files within their appropriate input and output directories.

testCaseGenerator:
This script uses strings and lists to create randomized test cases. Strings are used to store the generated tape contents, machine names, and transition rules, while lists are used to manage collections of tapes and transition rules.</td></tr>
</table>

10	**General operation of code (for each subproject):** multiTapeTMSimulator: So this script is essentially the core of this project as it implements the logic of a multi-tape Turing Machine. It starts by processing the input file to extract the machine name, tape count, initial tape configurations, and transition rules. While it's simulating, it reads the current state and symbols on each tape, identifies a matching transition rule, updates the tapes and head positions, and moves to the next state. The simulation continues until either a final state is reached, the maximum number of steps is exceeded, or no valid transitions are found. Finally, the results of each step, including tape states and transitions, are written to an output file. runSimulator: script controls the overall workflow of the project by managing the generation of test cases, running the Turing Machine simulator, and organizing input and output files into their respective folders, sort of like a helper function. It begins by using testCaseGenerator.py to create an input file, then calls multiTapeTMSimulator.py to process the input and generate results. After the simulation completes, it renames and moves the input and output files to their appropriate folders, making sure everything is organized and nothing is being overwritten. testCaseGenerator: This script generates random inputs for testing. It creates random tape configurations and a set of transition rules that correspond to the machine's behavior. The generated inputs include a machine name, the number of tapes, initial tape contents, and transition rules. These inputs are then saved to a file that can be consumed by multiTapeTMSimulator.py for simulation.
11	**What test cases you used/added, why you used them, what did they tell you about the correctness of your code:** To evaluate the simulator, I used the testcase generator to create 250 random test cases per run, with a total of 10 input files with diffferent sets of test cases. These test cases included different tape counts, transition rules, and initial tape configurations to ensure efficient testing. The main goal of these tests was to validate the simulator's ability to handle diverse scenarios, including multi-tape operations, complex state transitions, and edge cases where no valid transitions exist or where blank (_) symbols dominate the tapes. By using randomized inputs, I could assess the generality/robustness of the simulator across wide range of possible Turing Machines. The outputs gave insights into the correctness of the simulator; for example, and for the most part, they confirmed that the machine followed transition rules, updated tapes and head movements as expected, and halted correctly when reaching final states or encountering invalid transitions. Additionally, these tests helped identify and resolve bugs related to file path handling and state tracking, ensuring the simulator worked consistently across all scenarios. But overall, the randomized and diversity of the test cases showed the flexibility and reliability of the implementation.
12	**How you managed the code development:** The development of the simulator was managed by breaking the project into smaller pieces. The project was split up into three main scripts: the test case generator, the simulator, and the coordinator. I began by designing a test case generator and focusing

	on creating randomized input files with. Once this was complete, I implemented the simulator, making sure it could process the generated input files and correctly execute Turing Machine operations based on state transitions. Finally, I developed the workflow script (run sim) to automate the process of generating inputs, running the simulator, and organizing input and output files into their appropriate folders. Additionally, I made sure to maintain a clear folder structure to separate the code, input files, and output files, which made debugging and testing a lot easier. By testing each component independently and together often, I made sure the code was reliable, efficient, and easy to extend.
13	**Detailed discussion of results:** The results of the simulator showed its ability to interpret input testcases, execute state transitions, and produce meaningful outputs that align with the provided rules (for the most part). In test cases where the machine reached a final state after a few steps, the results showed that the simulator did correctly identify and apply transitions based on the initial tape configurations and state. For example, in one simulation, the machine halted immediately at a final state (qf0), which was consistent with the transition rules defined in the input. In cases involving more complex transitions and multi-step simulations, the outputs validated the correctness of the head movements, symbol replacements, and state transitions across multiple tapes (most of the time). The simulator also handled edge cases, such as blank tape symbols (_) and (*), without errors. In total, I generated 2500 test cases across 10 input files, and the simulator demonstrated its scalability and efficiency, even with large input definitions. That said, these results displayed its ability to handle a wide range of custom test cases while showing the correctness of the simulator in most cases.
14	**How team was organized:** This was an individual project.
15	**What you might do differently if you did the project again:** If I were to redo this project, I would focus on optimizing the code for better performance. For example, while the current implementation processes large input files effectively, I would try to look at more efficient data structures, such as dictionaries for state and transition lookups, to reduce the time complexity of finding matching transitions. Additionally, I would try to implement a more comprehensive logging mechanism to track intermediate steps and debug issues more efficiently. I would also try to consider implementing a visual representation of the Turing Machine's operations to make the simulation more intuitive and easier to analyze (wouldn't really know how to do this though).
16	**Any additional material:** N/A

Machine (NTM CSV File Name)	**Input String**	**Kind of Result**	**Depth**	**# Configurations Explored**	**Average Non-Determinism (# Configurations / Depth)**
InputDefinition1.txt	cbbbcbaacccbbcc	Accept	1	1	1.0
InputDefinition2.txt	babcbaccbbcb	Accept	4	4	1.0
InputDefinition3.txt	abccabca	Accept	3	3	1.0
InputDefinition4.txt	aaaab	Accept	3	3	1.0
InputDefinition5.txt	bbcbcccbabc	Accept	0	0	0.0