Project 3 Readme Team WSULLI22

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 Name"

<table>
<tr><td>1</td><td>Team Name: wsulli22</td></tr>
<tr><td>2</td><td>Team members names and netids: wsulli22</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: Program 1: Tracing NTM Behavior</td></tr>
<tr><td>4</td><td>Overall success of the project: Completed full/accurately (...i think :))</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 7 Hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/wsulli22/toc-project-3</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>tracetm_wsulli22.py</td><td>Main program for tracing NTM behavior</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>check-abc_star-wsulli22.csv</td><td>Sample test input from shared "NTM test files" folder</td></tr>
<tr><td>check-equal_01s-wsulli22.csv</td><td>Sample test input from shared "NTM test files" folder</td></tr>
<tr><td colspan="2">Output Files/Screenshots</td></tr>
<tr><td>output_abc_star_wsulli22.jpg</td><td>Output screenshots for 4 test cases successful ran</td></tr>
<tr><td>output_equal_01s_wsulli22.jpg</td><td>Output screenshots for 4 test cases successful ran</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td colspan="2">Refer to the table in"(13) Detailed Discussion of Results" section below</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:
Python with csv and sys libraries</td></tr>
</table>

9	Key data structures (for each sub-project): [only 1 project, so no sub-projects] Dicts, lists, csv, data, command-line arguments, class-based structures (e.g. the data class), measurements (max_depth, transition count, ect)
10	General operation of code (for each subproject) [only 1 project, so no sub-projects] My program is a NTS tracer that runs in a single Python file. It takes a CSV file from the command line containing the machine's states, alphabets, and transitions, then prompts the user for an input string. The program uses a "CurrentConfiguration" data class to track the machine's current state and a "TuringMachine" class to handle the machine's operations. It takes in inputs through a breadth-first search that explores all possible paths, including handling non-deterministic transitions. At the end, the program either reaches an accept state, a reject state, or a maximum depth that is hard-coded to 200 steps. Finally, it outputs details about the current "run" of the program, including the machine's name, the initial input string, the depth of the tree, the total number of transitions, and the path followed if the input was accepted.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I used 2 test input files ("abc_star.csv" and "equal_01.csv") that were provided in the shared NTM test files Google drive folder. I ran each with 4 different inputs - 2 that had known accepts and 2 with known rejects) with the goal of checking the accuracy of the main "tracetm_wsulli22.py" file. By referencing similar NTM trace examples in the project instructions and related problems in the textbook, I was able to confirm that the outputs of my program matched the expected outcomes. This suggests that the code is correctly tracing the NTM.
12	How you managed the code development: I started by understanding the requirements for simulating a NTS machine by reviewing the project specifications and consulting online resources about how to implement a turning machine in Python. I then designed my approach by first creating the core data structures that would handle the machine's state and operations. After that, I built the program piece by piece, first starting with the CSV parsing, then implementing the deterministic TM functionality, and finally modified it to be able to handle the nondeterministic transitions. As I went, I tested the program using the provided examples in the shared drive to check that the program was reaching the correct accept/reject cases. I organized the code into a one Python file with clear sections for imports, class definitions, and the main. I handled version control locally and focused on making the code readable and easy to follow with comments.

13 Detailed discussion of results:

Below is a table containing 8 trial runs across 2 different test files. The data referenced in the table below is pulled from the my “output_wsulli22.pdf” file and features the NTM used, the input string used, the result (accepted/rejected), the depth of tree, and number of configurations explored for each of the test cases (as requested on the Canvas “Project 2 Evaluation” assignment).

*abc = “a*b*c*” NTM”(check-abc_star-wsulli22.csv)*
equal = “equal number of 0s and 1s” NTM (check-equal_01-wsulli22.csv)

NTM	INPUT	RESULT	DEPTH	# EXPLORED	AVG NONDETERMINISM
equal	01	ACCEPTED	5	9	1.80
equal	1100	ACCEPTED	15	32	2.13
equal	011	REJECTED	5	10	2.00
equal	100	REJECTED	5	10	2.00
abc	abc	ACCEPTED	4	34	8.50
abc	aabbcc	ACCEPTED	7	94	13.43
abc	bac	REJECTED	1	6	6.00
abc	abcb	REJECTED	3	32	10.67
average nondeterminism = # configurations / depth					

Commentary:

Two NTM machines were tested with my tracing program. For each machine, four tests were performed. Two of those tests correctly led to accept states, and two correctly led to reject states for each machine. For all inputs, a result state was reached (no timeout).

As defined in the project instructions document, “the degree of no-ndeterminism [is defined] as the average number of new configurations that come from an average configuration.” Using this definition, the “equal number of 0s and 1s” NTM had a greater degree of nondeterminism, as shown by this NTM having an average “average degree of nondeterminism” of 9.65, compared to 1.98 for the “a*b*c*” NTM. This indicates a higher degree of branching for the inputs used in the “equal number of 0s and 1s” NTM. Also, since all tested machines are nondeterministic, they all have average nondeterminism values greater than 1, where 1 would correspond to a completely deterministic computation.

For the “equal number of 0s and 1s” NTM, the average nondeterminism is steady around 9.65. For the reject cases of the “a*b*c*” NTM inputs (“bac” and “abcb”), the behavior does not follow the expected pattern, and they are correctly rejected. The average

	nondeterminism is high for these cases, particularly for “abcb” (10.67), which may be due to the complexity of identifying violations in the pattern. Additionally, the number of configurations explored does not seem to correlate with the average nondeterminism. Similarly, the average nondeterminism, tree depth, number of configurations explored, nor the average nondeterminism values consistently correlate with whether an input is accepted or rejected.
14	How team was organized: N/A - Worked Alone
15	What you might do differently if you did the project again: I would ask TAs for guidance earlier in the project process, spend more time planning out my approach before actually coding, and better handle edge cases to make my program testing/debugging process more efficient.
16	Any additional material: N/A