Project Readme Team ishin

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"

1	Team Name: ishin
2	Team members names and netids: Ian Shin, ishin
3	Overall project attempted, with sub-projects: Overall Project: Simulating nondeterministic Turing machines (NTMs) with breadth-first search. **Sub-Projects**: Trace NTM behavior, handle acceptance/rejection of strings, and analyze nondeterminism
4	Overall success of the project: The project successfully implemented a robust NTM simulator capable of accurately processing .csv files and simulating non deterministic Turing machines. Key achievements include: 1. **Accurate NTM Simulation**: ○ Breadth-first exploration ensured all possible paths were considered, with the simulator correctly halting on acceptance or rejection. 2. **Testing and Validation**: ○ Tested with multiple .csv files, the simulator consistently produced correct results for both acceptance and rejection scenarios. ○ Metrics such as tree depth, configurations explored, and average nondeterminism aligned with expectations. 3. **Challenges Addressed**: ○ Resolved issues with malformed .csv files and infinite loops using clear debugging and depth limits. The project met all requirements, validated correctness, and showcased the simulator's ability to handle both simple and complex NTMs efficiently.
5	Approximately total time (in hours) to complete: 6 hours
6	Link to github repository: https://github.com/ianshin/ishin-project2-NTM
7	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.

File/folder Name	File Contents and Use
Code Files	
ntm_simulator_ishin.py	The ntm_simulator.py script is the main program for simulating NTMs. Its key components include: The NTM encapsulates all functionality for simulating NTMs. It has several key methods: ● The __init__(file_name) method initializes the NTM by loading its configuration, such as states and transitions, from a .csv file. ● The load_ntm(file_name) method parses the .csv file and populates internal data structures, including the transitions dictionary. ● The trace(input_string, max_depth=50) method simulates the NTM using breadth-first search. It tracks configurations, the depth of the configuration tree, and metrics such as the number of configurations explored and average nondeterminism. The method halts when an accepting or rejecting configuration is reached or when the maximum depth is exceeded. ● The print_accept_path(tree, config, depth) method prints the path from the initial configuration to an accepting configuration. ● The print_tree_summary(result, configs, depth, avg_nd) method outputs a summary of the simulation results, including whether the input string was accepted or rejected, the depth of the configuration tree, and the average nondeterminism.
Test Files (under testcases/)	

data_a_plus_ishin.csv	● **Contents**: ○ Defines an NTM that recognizes strings with one or more a's (a, aaa, aaaaa) ○ The NTM uses nondeterminism to decide which a in the string is the last one. ● **Use**: ○ Tests basic nondeterminism and the simulator's ability to handle repetitive patterns with simple branching.
data_abc_star_ishin.csv	**Contents**: ● Defines an NTM that accepts strings of the form abc* (e.g., abc, abcc, abccc, ab) ● Utilizes multiple transitions to account for varying numbers of c's. **Use**: ● Validates the simulator's ability to handle more complex patterns and deeper configuration trees.
data_equal_01s_ishin.csv	**Contents**: ● Defines an NTM that accepts strings with an equal number of 0's and 1's (0011, 0101, 000111, aaa). ● Requires tracking matches between 0's and 1's, often through nondeterministic exploration. **Use**: ● Tests the simulator's handling of balanced strings and its ability to navigate complex branching paths effectively.

Output Files (under outputs/)	
output_a_plus_ishin.txt	● **Contents**: ○ Records the simulation results for the data_a_plus_ishin.csv test case with input strings a, aaa, aaaaa ○ Includes details like acceptance/rejection, depth of the configuration tree, total configurations explored, and average nondeterminism. ● **Use**: ○ Demonstrates the simulator's ability to handle simple nondeterminism and correctly recognize strings with one or more a's.
output_abc_star_ishin.txt	● **Contents**: ○ Stores the results for the data_abc_star_ishin.csv test case with input strings abc, abcc, abccc, ab ○ Provides a summary of acceptance/rejection, path details, and simulation metrics. ● **Use**: ○ Validates the simulator's handling of more complex patterns and ensures proper branching exploration.
output_equal_01s_ishin.txt	**Contents**: ● Contains the simulation results for the data_equal_01s_ishin.csv test case with strings 0011, 0101, 000111, and aaa.

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>

- Includes whether the input is accepted or rejected, the depth of the tree, configurations explored, and nondeterminism metrics.

Use:

- Tests the simulator's ability to correctly identify strings with an equal number of 0's and 1's, showcasing its robustness with balanced patterns.

</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</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project): csv, argparse</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):
Core Functionality

The NTM simulator operates by loading transitions from .csv files and processing input strings based on the defined rules. The implementation uses breadth-first exploration to ensure that all possible configurations are considered until an acceptance or rejection state is reached.

Configuration Management: Configurations are managed using a tree structure, where each depth level represents all possible states the machine could be in after a specific number of transitions.

Transition Handling: Transitions are loaded from .csv files into a dictionary. The simulator evaluates all valid transitions for the current configuration, expanding the configuration tree as needed.

Halting Conditions: The simulation halts when an accepting configuration is reached, all paths lead to rejection, or the maximum depth (--max_depth) is exceeded.

Metrics Tracking: The simulator records the depth of the configuration tree, total configurations explored, and average nondeterminism for evaluation and debugging.</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.

Test Cases Used:</td></tr>
</table>

	1. data_a_plus_ishin.csv: ○ Accepts strings with one or more a's. ○ Used to test basic nondeterminism with simple branching logic. ○ Result: Verified the simulator's ability to explore paths and correctly handle repetitive patterns. 2. data_abc_star_ishin.csv: ○ Accepts strings in the form abc*. ○ Used to validate handling of deeper configuration trees and more complex branching. ○ Result: Demonstrated the simulator's capability to manage multiple transitions and halt on acceptance. 3. data_equal_01s_ishin.csv: ○ Accepts strings with equal numbers of 0's and 1's. ○ Used to test the simulator's robustness in handling balanced patterns. ○ Result: Confirmed that the simulator correctly matches symbols through nondeterministic exploration.
12	How you managed the code development **Incremental Approach**: ● The project was developed step-by-step, starting with the ability to parse .csv files and store transitions in a structured way. ● Once parsing was functional, the breadth-first search algorithm for simulating NTMs was implemented and tested iteratively. **Testing**: ● Began with simple test cases (data_a_plus_ishin.csv) to validate the basic functionality of the simulator. ● Gradually tested more complex NTMs (data_abc_star_ishin.csv, data_equal_01s_ishin.csv) to ensure robustness and handle deeper configuration trees)
13	Detailed discussion of results: **Breadth-First Exploration**: ● The use of breadth-first search ensured fair and complete exploration of all possible paths, avoiding infinite loops or premature termination. **Handling Complexity**: ● The simulator successfully handled NTMs with varying levels of complexity: ○ **Simple Patterns**: Tested with repetitive inputs (data_a_plus_ishin.csv) ○ **Branching Logic**: Validated with (data_abc_star_ishin.csv)

<table>
<tr><td></td><td>
Balanced Strings: Demonstrated robustness with data_equal_01s_ishin.csv.
Overall Performance:
The simulator met all functional requirements, provided detailed output for each test case, and demonstrated flexibility in supporting various NTMs.
</td></tr>
<tr><td>14</td><td>How team was organized

Did all parts individually.</td></tr>
<tr><td>15</td><td>What you might do differently if you did the project again

Maybe try a different option with a team member(s).</td></tr>
<tr><td>16</td><td>Any additional material:

N/A</td></tr>
<tr><td>17</td><td>Table of outputs

<table>
<tr><th>NTM File</th><th>Input String</th><th>Result</th><th>Depth</th><th>Configurations Explored</th><th>Avg Non-Det</th></tr>
<tr><td>a_plus</td><td>a</td><td>Accepted</td><td>2</td><td>4</td><td>1.33</td></tr>
<tr><td>a_plus</td><td>aaa</td><td>Accepted</td><td>4</td><td>8</td><td>1.14</td></tr>
<tr><td>a_plus</td><td>aaaaa</td><td>Accepted</td><td>6</td><td>12</td><td>1.09</td></tr>
<tr><td>abc_star</td><td>abc</td><td>Accepted</td><td>4</td><td>16</td><td>1.00</td></tr>
<tr><td>abc_star</td><td>abcc</td><td>Accepted</td><td>5</td><td>20</td><td>1.00</td></tr>
<tr><td>abc_star</td><td>abccc</td><td>Accepted</td><td>6</td><td>24</td><td>1.00</td></tr>
<tr><td>abc_star</td><td>ab</td><td>Rejected</td><td>0</td><td>1</td><td>1.00</td></tr>
<tr><td>equal_01s</td><td>aaa</td><td>Rejected</td><td>0</td><td>1</td><td>1.00</td></tr>
<tr><td>equal_01s</td><td>0011</td><td>Accepted</td><td>4</td><td>10</td><td>2.50</td></tr>
<tr><td>equal_01s</td><td>0101</td><td>Accepted</td><td>4</td><td>12</td><td>3.00</td></tr>
<tr><td>equal_01s</td><td>000111</td><td>Accepted</td><td>6</td><td>15</td><td>2.50</td></tr>
</table>
</td></tr>
</table>