Project x Readme Team phoebe

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: phoebe
2	Team members names and netids: Phoebe Huang, chuang26
3	Overall project attempted, with sub-projects: Tracing NTM behavior
4	Overall success of the project: successful in my opinion
5	Approximately total time (in hours) to complete: 6-7 hours
6	Link to github repository: https://github.com/chuang26-hue/traceTM_phoebe
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	
traceNTM_phoebe.py	This file is the main code file that traces and simulates a NTM by using BFS
Test Files	
input.txt	Includes the name of the file describing the machine, the input strings to test, a ”termination” flag that will stop execution under some circumstance such as if the depth of the configuration tree exceeds a limit
NTM.csv	Defines the NTM, with the header lines being Line 1: Name of machine 5 • Line 2: List of state names for Q • Line 3: List of characters from Σ • Line 4: List of characters from Γ • Line 5: The start state • Line 6: Accept state • Line 7: Reject state

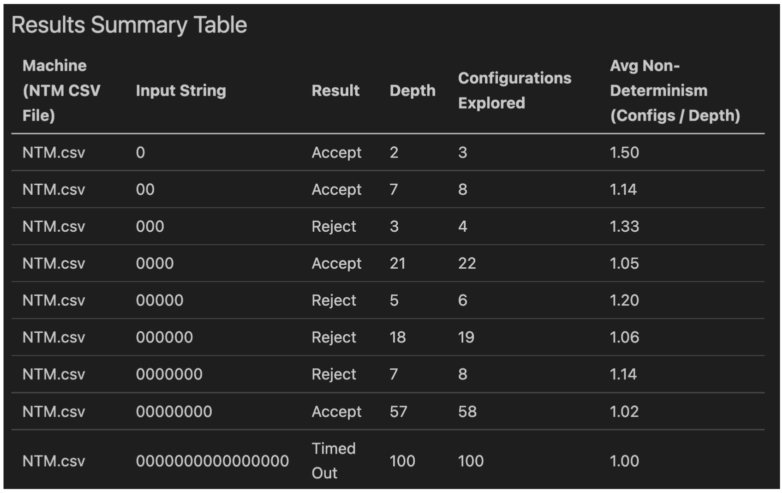
	The rest are transition lines

Output Files

output_phoebe.txt	The output echoes the name of the machine, the initial string, the depth of the tree of configurations, and the total number of transitions simulated. 1. If the simulation halts because of reaching an accept configuration, print out the following: a. ”String accepted in ” and then the number of transitions from the start to the accept on just the accepting path (the depth of the tree). b. Starting at the starting configuration, print out each configuration in the format: left of head string, state, head character and right of string. 2. If the simulation halts because of all paths lead to reject, ” String rejected in ”, followed the number of steps from the start to the last reject 3. If the step limit is exceeded, print out something like ”Execution stopped after” the max step limit.

Tables (as needed)

Included in README.md on GitHub



Results Summary Table

Machine (NTM CSV File)	Input String	Result	Depth	Configurations Explored	Avg Non-Determinism (Configs / Depth)
NTM.csv	0	Accept	2	3	1.50
NTM.csv	00	Accept	7	8	1.14
NTM.csv	000	Reject	3	4	1.33
NTM.csv	0000	Accept	21	22	1.05
NTM.csv	00000	Reject	5	6	1.20
NTM.csv	000000	Reject	18	19	1.06
NTM.csv	0000000	Reject	7	8	1.14
NTM.csv	00000000	Accept	57	58	1.02
NTM.csv	0000000000000000	Timed Out	100	100	1.00

8	Programming languages used, and associated libraries: Python Libraries used: os, csv
9	Key data structures (for each sub-project): arrays (lists of lists), tree
10	General operation of code (for each subproject):

	Tracing NTM: 1. Input Specification: a. The NTM specification is provided in a .csv file containing states, transitions, and acceptance conditions. b. An input.txt file specifies the input strings to simulate, along with parameters like maximum depth and step limits. 2. Parsing the NTM: a. The code reads the .csv file to construct the NTM's state transition table and configurations. 3. Simulation Process: a. For each input string, the NTM begins at the start state and attempts to process the string according to its transition rules. b. Non-deterministic branching is handled by exploring multiple paths simultaneously. 4. State Transition Tracing: a. The code logs each state transition and keeps track of paths to acceptance or rejection. b. If the maximum depth or step limit is reached without an outcome, the process times out. 5. Output Generation: a. The results for each input string include: i. Acceptance Status: Whether the string is accepted, rejected, or times out. ii. Path Details: Step-by-step state transitions leading to the result. 6. Error Handling: a. The code checks for missing files, malformed inputs, or invalid NTM configurations, providing clear error messages for debugging. 7. Results Compilation: a. All outputs are written to output.txt for further analysis, including detailed paths and summary metrics.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. Used 0, 00, 000, ..., 00000000, and 0000000000000000 (16 0s). ● These test cases represent a range of input lengths, including valid strings with lengths that are powers of 2 (e.g., 0, 00, 0000, 00000000) and invalid strings where the length is not a power of 2 (e.g., 000, 00000, 0000000). ● Shorter inputs like 0, 00, and 000 were used to verify basic functionality and correctness of the state transitions in simple cases. ● Longer inputs like 00000000 and 0000000000000000 were used to stress-test the code for deeper exploration and ensure it handles large inputs efficiently, correctly identifying acceptance, rejection, or timeout. Observations: ● For valid inputs, the program successfully traced paths to acceptance, demonstrating that the NTM correctly simulates transition rules and identifies strings in the language.

	● For invalid inputs, the program correctly rejected the strings, confirming proper handling of cases where no valid paths lead to an accepting state. ● For the longest input (0000000000000000), the program timed out as expected, confirming that the maximum depth limit is correctly enforced to prevent infinite exploration. ● The results showed that path logging and metrics (e.g., depth, configurations explored) were accurately calculated, providing insights into the NTM's behavior and branching. These test cases validated the program’s correctness, robustness, and efficiency while ensuring it handles edge cases like timeouts and large input sizes gracefully.
12	How you managed the code development: **Planning**: Began by breaking down the requirements into smaller, manageable tasks, such as drawing the NTM out, drawing the configurations, parsing the NTM, simulating transitions, and logging results. **Incremental Development**: Built the program incrementally, starting with basic parsing and state transition handling, followed by adding non-deterministic branching and depth/step limits. **Testing**: Continuously tested the code after implementing each major feature using predefined test cases to ensure correctness before proceeding to the next step. **Debugging**: Addressed issues like infinite loops or incorrect state transitions by adding error handling, assertions, and debug statements to trace the program’s behavior.
13	Detailed discussion of results: **Correctness**: The program correctly identified whether input strings were accepted, rejected, or timed out. The paths to acceptance or rejection were logged accurately, verifying that the NTM transitions were implemented correctly. **Metrics**: Depth and configuration metrics provided useful insights into the complexity of the NTM’s behavior, including the average non-determinism (>1). **Performance**: The program handled short inputs efficiently and longer inputs within reasonable time and space constraints, demonstrating scalability up to the defined depth limit. **Edge Cases**: The results for invalid strings and timeout cases confirmed that the program could gracefully handle scenarios where no valid path to acceptance existed or when the exploration exceeded the limits.
14	How team was organized: it was a 1-person team so I completed everything myself
15	What you might do differently if you did the project again: I will add more features to make the code more robust such as: **Visualization**: Add a feature to visualize state transitions and branching paths graphically to better understand the NTM's behavior. **Comprehensive Testing**: Expand the test cases to include edge cases like empty strings or inputs with unexpected symbols **Additional Metrics**: Include more detailed metrics, such as average branching factor per state or time complexity analysis for each input.
16	Any additional material: N/A