Project Readme Template

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: bbaron2</td></tr>
<tr><td>2</td><td>Team members names and netids
Brendan Baron
bbaron2</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects:

NTM Tracing
Tracing NTM Behavior</td></tr>
<tr><td>4</td><td>Overall success of the project:
High. I felt like I learned a lot about non-determinism in Turing machines and Breadth-First Search. I also think I did a good job testing the correctness and that the results show this.</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/bbaron26/Theory2</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>main_bbaron2.py
parser_bbaron2.py
__init__bbaron2.py
runner_bbaron2.py
simulator_bbaron2.py</td><td>Code for Tracing the NTM Behavior using a BFS</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>abc_star_DTM_bbaron2.csv
abc_star_bbaron2.csv
aplus_DTM_bbaron2.csv
aplus_bbaron2.csv</td><td>Languages to test input strings with to make sure that the code correctly accepts/rejects</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>equal_01s_DTM_bbaron2.csv
equal_01s_bbaron2.csv</td><td></td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>data_bbaron2.txt</td><td>Contains the table below of results from test cases. It includes machines, input, results, depth, # of configurations, and average non-determinism
Note: the code just prints this out but I copied it and put it in a .txt file to make it easy to access</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:
python
csv, collections.deque, os</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):
Queue: Used to manage configurations for BFS
Dictionary: Stored the transitions for fast lookup based on the state and symbol
List: Used to represent the tape, path taken, and multiple configurations
Tuple: Used to store individual transitions and configurations</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject)
First, it parses .csv files to extract the machine’s states, transitions, and rules. Using Breadth-First Search, the simulator would explore all of the possible configurations, storing the tape contents, current state, and head position for each path. The deterministic machines would follow one transition per state-symbol pair while the non-deterministic machines would explore multiple transitions simultaneously. The simulator stops when an accept state, reject state, or maximum depth is reached. The maximum depth is included to prevent the code from running for too long. The code would output the result, depth of the search, total configurations explored, and average non-determinism.</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.
I wanted to test multiple different languages to validate the correctness. In each language, I wanted to test multiple strings. I wanted to make sure I tested strings that should be accepted and strings that should be rejected. I also wanted to test edge cases such as the empty string. I wanted to keep the input strings short to not make it take a long time to run. I even added a maximum depth (that can easily be changed) to avoid code running for too long. I made abc_star.csv (a*b*c*). I tested abcc, which was correctly accepted, and cab, which was correctly rejected. I made equal_01s, which I tested with the empty string, which was correctly accepted, and 010, which was correctly rejected. I made aplus, which correctly accepted strings such as aaa, and correctly rejected strings such as abc. I also made a deterministic counterpart for each</td></tr>
</table>

	of the above to ensure that the deterministic and nondeterministic aligned. These tests confirmed the correctness of my code.
12	How you managed the code development: I managed the code development by breaking down the project into different parts, each of them serving a different purpose. I started by figuring out what I needed to parse and making the csv files to test. I first wrote parser.py to handle the .csv input files. Next, simulator.py implemented the Breadth-First Search (BFS) for the Turing Machines. The code in runner.py connected the components and main.py, allowing me to test various inputs. Throughout, I would test the code. I would make sure to check edge cases to make sure it works and deal with test cases that I knew the correct result of to make sure that my code was correct.
13	Detailed discussion of results: As discussed above, the results indicated that the machines performed correctly. It accepted strings that were in the language and rejected strings that were not in the language. The depths and configurations explored also seemed to match. From my results, the non-deterministic machines tended to show a higher non-determinism than the deterministic counterparts. This is due to their nature that led them to check a higher number of configurations. As the input strings got more complicated, this number would go much higher.
14	How team was organized I was the leader and I had to obey myself
15	What you might do differently if you did the project again If I were to do it again, I would try to develop a testing framework earlier on in the process. This would allow me to validate the components of the code earlier, making debugging easier. Something that I think would be really cool to see is a visual representation of the path the non-deterministic Turing machine takes. If I had more time to work on it, I think making a visual to see this, probably a graph, would improve the project. I would also look into optimizing the space and time complexity.
16	Any additional material:

Table:

Machine	Input	Results	Depth	# Configs	Avg Non-Det
aplus.csv	abba	Rejected	2	3	1.50
abc_star.csv	abcc	Accepted	5	21	4.20
abc_star_DTM.csv	abcc	Accepted	5	6	1.20
aplus_DTM.csv	aaa	Accepted	4	5	1.25

aplus.csv	aaa	Accepted	4	6	2.00
aplus_DTM.csv	a	Accepted	2	3	1.50
aplus_DTM.csv	abc	Rejected	2	2	1.00
equal_01s.csv		Accepted	1	2	2.00
abc_star.csv	cab	Rejected	2	3	1.50
abc_star_DTM.csv	ab	Accepted	3	4	1.33
equal_01s_DTM.csv	010	Rejected	4	4	1.00