# Project NTM readme_gflesher_gflesher

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: gflesher</td></tr>
<tr><td>2</td><td>Team members names and netids: Grant Flesher - gflesher</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: I attempted the NTM project, where i coded a NTM that could evaluate if an inputted string would be accepted or rejected</td></tr>
<tr><td>4</td><td>Overall success of the project: I completed the program! My program was able to come up with combinations for varying outputs!</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/grantflesher/NTM</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>NTM_gflesher.py</td><td>This file is my project code. It takes a csv file as input from one of the test files, and a string from the user. From this it runs a NTM, to see if the string would be accepted or rejected on the NTM inputted.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>test_files (foulder)</td><td>The csv files include inputs that showcase varying NTMs. They show various cases to show how various machines work with the</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>project</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>output_files (foulder)</td><td>These are the output files for each of the test files. Each test file was run 5 times, with a different input each time. This would showcase the test files accepting and rejecting varying inputs.</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td></td><td></td></tr>
<tr><td colspan="2">Analytics File</td></tr>
<tr><td>NTM Project Analytics_gflesher.pdf</td><td>This file showcases the inputs and outputs of varying machines! Additionally, each machines nondeterminism and depth for each run is provided aswell</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries: Python (csv)</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project): The Key Data structures I used were a dictionary, a queue and lists to store all the data. I used a dictionary to store all the parsed data from the csv files inputted. I used a queue to use breadth first search to optimize my search through possible outcomes. I used lists to maintain tree configurations to be able to retrace the path for the outcome.</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):<br><br>The program intakes CSV data from a file containing information about a NTM. This csv file is parsed and the data is put into a dictionary. From there, the user inputs an input string, and a max length, which is used in the run_NTM function, which simulates the behavior of a NTM. The NTM will follow possible outcomes until and accept state is reached, all possible outcomes fail, or it reaches the depth limit (in case of never ending loops). Once a correct path is found, the code will find the path that was taken to get to the accepted state in the trace_path function. From there the path taken will be outputted.</td></tr>
<tr><td>11</td><td>What test cases you used/added, why you used them, what did they tell you about the</td></tr>
</table>

| | |
|---|---|
| | correctness of your code.<br><br>I added 5 tests for each NTM, to make sure that the machines would work with varying inputs. I used inputs that went over the depth limit, and I used inputs that were wrong, to see if the machine would catch when a faulty input was given. By doing such, I was able to make sure that my code would work correctly, even on edge cases. |
| 12 | How you managed the code development:  I started by handling the input. I worked on getting all the csv data into a dictionary, so the information about the NTA would be easier to use in the other functions. From there, I worked on the run_NTM function, that utilized breadth first search to find the optimal accept state, with the shortest path. Then I worked trace_path, which would go backwards and find the path taken to reach the accept state. By putting this all together, I was able to complete the code file. |
| 13 | Detailed discussion of results: From looking over my results, I could see how varying NTAs would operate and how their tree depth and number of configurations explored varied. Some NTAs has significant variance when it came to their average nondeterminism, but others had a stagnant 1. I think this is a result of how the machine was set up. If there was opportunity for the tree to grow, it would;however, certain NTAs could work without ever having differing paths, making the average nondeterminism 1. |
| 14 | How team was organized: solo. |
| 15 | What you might do differently if you did the project again: Not accidentally delete my code file by pushing to main and deleting it, having to code major portions of it again because I messed up trying to restore it. |
| 16 | Any additional material:<br>This helps to this day: https://www3.nd.edu/~pbui/teaching/cse.20312.fa23/ |