# Project 2 Readme Team mfues

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: Maribella Fues</td></tr>
<tr><td>2</td><td>Team members names and netids: Maribella Fues, mfues</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: Tracing NTM Behavior</td></tr>
<tr><td>4</td><td>Overall success of the project: Successfully implemented a NTM Tracer to trace all possible paths the NTM might have taken and halt at the appropriate time.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 9</td></tr>
<tr><td>6</td><td>Link to github repository:<br>https://github.com/maribella-fues/theory_of_computing_NTM_Tracer</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_mfues.py</td><td>The NTM Tracing main code which implements and calls a class that will trace all possible paths on the given machine for the user-inputted string. It then outputs statistics for the trace and the result of evaluating the string.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>check_NTM_csv_files_mfues</td><td>Folder containing four different test machine files that were provided by Professor Kogge. They outline three different turing machines, one of which also has a deterministic version (labeled DTM in the name). These are used as input machine files to test traceTM_mfues.py.</td></tr>
<tr><td colspan="2">Output Files</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>output_traceTM_mfues.txt</td><td>Output text file produced by running the traceTM_mfues.py program on all four different test machine files and on different input strings. Includes a variety of result outputs to verify the program's correctness.</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:<br>Python with imported library of sys</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):<br>traceTM_mfues.py:<br>- Utilized a 3D list to implement the tree of configurations. The tree was a list of levels with indices corresponding to each level's depth in the tree. The levels were each lists consisting of individual 3-element lists that outlined each configuration.<br>- The transitions for the NTM were stored using a list of dictionaries with each transition represented by a dictionary. This made it simpler to access the different elements of each transition and be able to iterate through all the transitions.<br>- The tracer was implemented using a class. The class had attributes pertaining to the NTM that it was tracing as well as attributes for the configuration tree it was building. This was done in order to access and alter the machine's attributes in many different functions more simply, rather than having to return a large amount of information from each function.</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):<br>To run the code from the command line:<br>python ./traceTM_mfues.py [csv_file_name] [string] [-d | -t] [max limit number]<br>After the program name, you must first include the path and name of the machine csv file that you want the program to read the machine from and the string you want the machine to evaluate. The machine file must be correctly formatted for the program to run correctly. Then, you must indicate whether you want the termination flag that will prevent the machine from running too long to either be a max depth (-d) or a max number of transitions (-t) and follow this designation with what number you want to set the max to. All of these command line arguments are required to run the program.<br><br>traceTM_mfues.py code description:<br>The majority of the code is contained in the class runNTM. This allows easier access to all of the machine's and configuration tree's attributes in all of the functions. The class begins with the init() function that initializes an object of the class by using the function's arguments for the filename, string, and limit type and max number. The max depth and max total transition numbers are initially set</td></tr>
</table>

| | |
|---|---|
| | **to -1 and depending on the limit type that the user specifies, one of the numbers is changed to the max number they inputted. This is so that when checking the depth and total number of transitions against the max, only one can be used since neither number will ever equal -1. It also calls the read_input function on the file and sets attributes to be tracked for the configuration tree to their initial states.**<br>**The read_input() function uses the user-inputted file to extract all of the necessary information for the NTM, including the machine name, start state, accept state, reject state, and creating the list of transitions, adding each transition in the file to the list in the form of a dictionary. It assumes correct formatting for the machine input files.**<br>**The bfs() function is the main tracing function. It runs on the last list level of configurations in the current configuration tree. For each configuration, if the configuration is not already in the reject state, it iterates through each of the transitions in the transition lists and checks if the transition would apply to the configuration (by matching the current character under the tape head and the current state). If it does, it generates a new configuration depending on the direction that the transition wants the tape head to move and the new character it writes to the tape. This new configuration is then added to a new list of configurations to be added as a new level to the configuration tree and the total transition count is incremented. If the total transition count equals the max number of transitions, the function is ended by adding the new list of configurations to the configuration tree, incrementing the depth, setting the result to "Ran Too Long" and returning True to indicate that the machine should halt. If instead the new state transitioned to is the accept state, the function should also end in the same way but instead setting the result to "Accept." For each configuration, a boolean Transitioned is initially set to False and then if a transition matches it to generate a new configuration, this boolean is then set to True. This is used to check at the end if the configuration transitioned to any new configurations at all. If not, then a new configuration is generated by transitioning the current one to a reject state, which is also added to the new level of configurations and increments the total transition counter. After incrementing the transition counter, the max number of transitions is checked again. After iterating through all the configurations in the current last level, the new list of configurations is added to the configuration tree and the depth is incremented. If no new configurations were added, then the result is set to "Reject" and the function returns True to indicate that the machine should halt because all configurations are in the reject state. If the depth matches the max depth, then the result is set to "Ran Too Long" and True is returned as well. If this is not the case, then the function returns False to indicate that the trace should be run again on the next level because the machine has not halted yet.**<br>**The run() function simply loops the bfs() function until true is returned indicating that the machine has halted.**<br>**In the main() function outside of the runNTM class, it begins by extracting the filename, string, limit type and max number from the command line arguments. It then creates a runNTM object, using this data as arguments, and simply calls the run() function on the object. It then prints the result by accessing the name, string, depth, total number of transitions, and result attributes of the object. Depending on the result, it will print different data. Finally it writes the same output to a new file.** |

| | |
|---|---|
| 11 | What test cases you used/added, why you used them, what did they tell you about the correctness of your code.<br>**I used multiple test machine csv files to test the performance of the NTM tracer. These included check_a_plus_mfues.csv, check_abc_star_mfues.csv, check_equal_01s_DTM_mfues.csv, and check_equal_01s_mfues.csv, all of which are stored in the folder check_NTM_csv_files_mfues in the github repository. Using these test files, I ran the code on multiple test strings. First, I used strings that I knew would either reject or accept and verified that the result was correct. Next, I inputted large strings with lower limits for the depth or number of transitions to verify that the program effectively terminates when it reaches the termination limit. Finally, I ran the code multiple times to have it print out the resulting tree for the string that I inputted. I then created the same tree by hand using the machine description and checked the program's output with my own to make sure that it accurately traced all possible paths. I found it useful to use three different types of machines to verify the program's correctness on multiple inputs, especially since some machines were larger and more nondeterministic than others. I also included the deterministic machine version of the equal 01s TM so that I could verify that if my program is run on a deterministic machine, it would still be correct and would have an average nondeterminism of 1, or rather only 1 configuration per level on the tree. I verified that this was the case when the program was run on a deterministic machine, which further verified the correctness of my program.** |
| 12 | How you managed the code development<br>**I first focused on developing the breadth first search program in the code, as I thought this would be the most complex and important part of the program. Next, I created the run() program to repeatedly call the bfs() function and actually trace the entire NTM. After successfully implementing the trace part of the program, I created the class around it in order to initialize the attributes of the machine that it uses. I then implemented the function read_input() and the parts of main() that read in the input from the machine file and the command line, and focused on sending that information where it needed to be used. Finally, I focused on formatting the output to match the requirements of the project, rather than the test output that I had been using to verify the correctness of my code as I was developing it.** |
| 13 | Detailed discussion of results:<br>**The program successfully traces all execution paths for a nondeterministic turing machine until it either reaches an accept state and halts, all the paths end in a reject state and it halts, or it reaches the maximum limit for the depth or number of transitions and halts for running too long (preventing it from getting stuck in an infinite loop). The program also correctly tracked the depth and total number of transitions, and from this, calculated the average nondeterminism. By doing so, it was easy to see trends for the nondeterministic machines. As expected, the machine with the highest number of nondeterministic transitions (transitions that run on the same current state and tape character), which was the check_abc_star_mfues.csv machine file, had the highest average nondeterminism. Conversely, the deterministic machine, check_equal_01s_DTM_mfues.csv, only had an average nondeterminism of 1 because, for a deterministic machine, there is only one possible transition for** |

| | |
|---|---|
| | **each state and input combination. By comparing these different results and verifying the correctness of the output of the program in the file output_traceNTM_mfues.txt, I confirmed that the program traceTM_mfues.py accurately traces all possible paths the NTM might have taken and halts when it is supposed to.** |
| 14 | How team was organized<br>**N/A, I did all the tasks.** |
| 15 | What you might do differently if you did the project again:<br>**If I could do the project again, I would develop more complex machine files in order to test the program more thoroughly and have it produce more useful results since such a program is needed most when the NTM is too complex to trace by hand.** |
| 16 | Any additional material: |