Project 2 Readme Team jrelling

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: jrelling</td></tr>
<tr><td>2</td><td>Team members names and netids: Jack Rellinger (jrelling)</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: traceTM program to trace the execution of nondeterministic (and deterministic) turing machines</td></tr>
<tr><td>4</td><td>Overall success of the project: The project was very successful, I learned a lot about how a deterministic machine like my computer can execute a nondeterministic program using a breadth-first approach at each level of execution for turing machines.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: approx. 8 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/jackrell/toc-project02-jrelling-traceTM</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_jrelling.py</td><td>This contains all of the functional code for parsing the command line, reading in TM csv files, and running the machine on the specified input string.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>check_a_plus.csv, check_a_plus_DTM.csv
data_abc_star.csv, data_abc_star_DTM.csv
data_equal_01s.csv, data_equal_01s_DTM.csv</td><td>These files were provided by another student, and they were the machines I used in all of my testing.</td></tr>
<tr><td colspan="2">Output Files</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>output_a_plus.txt, output_a_plus_DTM.txt
output_abc_star.txt, output_abc_star_DTM.txt
output_equal_01s.txt, output_equal_01s_DTM.txt
output_exceeds_max_depth.txt</td><td>These are the printed outputs (>> into the files from the command line). There is one for each machine file containing multiple runs, as well as one that shows my machine exceeded a set maximum depth of 100 (which can be changed from the command line)</td></tr>
<tr><td colspan="2">Discussion</td></tr>
<tr><td>readme_project02_jrelling.pdf
discussion_jrelling.pdf</td><td>The readme pdf is this document, the discussion pdf discusses the output and how my code modeled nondeterminism.</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:
The project was written completely in Python, using the csv and sys libraries for handling the turing machine input files and handling command line arguments respectively.</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):
The most important data structure used was the list of list of lists, which functioned as a tree to store every configuration as I moved down levels in the tree. I added to this structure another layer by also storing the parent configuration for each node. This made it easy to backtrack and trace the execution for accepting paths. Other important data structures I used were my config data structure, which was simply a list that had the left of the head string, the state, and the right of the head. This structure made it easy to access the parts of the tape that I had to manipulate in each step.</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):
The traceTM_jrelling.py code generally works by parsing command line arguments to find the machine file, the input string, and optionally the max depth. It then reads the machine file to find the machine name, start state, accept state, and all of the transitions. Then my main function calls the function run_tm, which contains the logic for running the TM. This function contains a while loop that indefinitely (until accept, reject or max depth is reached) runs through each level of the tree structure, which is initialized with the starting configuration at depth 0. This loop has a for loop that goes through each configuration at that level, and for each configuration searches for any transitions out of it. If a level has no transitions for any configuration, the string is rejected. If a level has transitions, they are carried out accordingly and added to the next level to be searched. If a configuration at a level contains the accept state, it results in a halt as well, and the config_print function is called to backtrack through the tree and</td></tr>
</table>

	print out the configurations searched on the way to the accept. No matter the result, the number of transitions and the depth of the tree are used to calculate the average nondeterminism, printing out these three values and ending execution.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. The test cases I used were those provided by the instructor in an announcement on 12/2/24 which were originally published by another student. These were NTMs and their corresponding DTMs, which I verified accepted the strings they were intended to.These helped me evaluate the correctness of my code by allowing me first to check that I was reading in the machine csv files correctly, then they allowed me to test their operation against different input strings. They were simple enough machines that I could tell which strings should or should not be accepted, so using these test cases I could evaluate if my algorithm correctly identified transitions and took the ones that were available, moving towards an accept or reject or looping. In my code, you can see where I would print each level in debugging, which was my primary way of modeling my problems.
12	How you managed the code development: I managed the code development through breaking up the problem into steps such as: reading in csv, creating the configuration and tree model, a loop to go through each configuration in the tree, a loop to go through each transition in the list of transitions, and finally checking for halting conditions and printing. By thinking of it this way, I could solve each of these problems one at a time and combine to my finished result. My main method of debugging was printing the tree out after every step and completed run, which allowed me to visualize more effectively.
13	Detailed discussion of results: The results serve to primarily highlight the differences between DTMs and NTMs in their efficiency and computational behavior. Because the DTMs follow a single path, there was largely less computations, transitions, and more consistency. However, it is worth noting the strengths of the NTMs which lie in the perceived flexibility and adaptability, because my code allowed me to explore multiple paths concurrently. This draw comes at the cost of more transitions and computation cost. This was especially evident in the abc_star tests, notably for the input aaabbbccc, where this complex and longer input led to a great deal more nondeterminism (4.4), the result of more transitions made and evaluated. Both this machine and the deterministic machine version of this problem were accepted in 10 steps, demonstrating that for these smaller-scale problems deterministic machines are more efficient, but for more pattern based and unpredictable machines NTMs can be attractive. However, it is important to note that all nondeterministically modeled problems can be computed with a deterministic machine, so ultimately for problems of the scale I was testing, DTMs were more effective.
14	How team was organized: I worked alone, organized through individual effort.
15	What you might do differently if you did the project again: I think I would start by creating pseudocode for a more robust breadth first search algorithm next time. I think my tree parsing skills aren't as good as they used to be, which led to many headaches as I was designing my algorithm to run the TMs. I would also be more deliberate in my handling of reject configurations – my approach worked

	but I feel as though it could have been clearer, and actually shifted the machine to a “reject” state, not just recognizing that it was rejected.
16	Any additional material: N/A