Project 02 Readme Team tset

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: tset
2	Team members names and netids: Carlos Basurto, cbasurto
3	Overall project attempted, with sub-projects: BTracing Nondeterministic Turing Machine Behavior. The project can read Nondeterministic Turing Machines and test input against them, outputting every configuration at every transition step, determining whether any given string is accepted or rejected alongside printing the respective depth and degree of nondeterminism.
4	Overall success of the project: The project was entirely successful – the Nondeterministic Turing Machine tracing algorithm works, the configurations are appropriately calculated and accordingly processed and displayed, permitting the required analysis.
5	Approximately total time (in hours) to complete: Around 5 hours.
6	Link to github repository: https://github.com/cbasurtom/ntm
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	
traceTM_tset.py	Main deliverable. Input: NTMs (.csv files) NTM inputs (inputs_tset.txt files)

	Output: Text output (output_…_.txt files) After opening the appropriate input and output files, the program initializes a custom NTM class for each of the .csv files, following their stated parameters. For each, the program runs the NTMs to every input string in its input file using the .run() method, which simulates the behavior of the NTM utilizing the rules stated in its transition table at every possible configuration with an increasing depth. Every time it reaches an accept or reject state, the branch will be pruned (as the machine halts). The machine will keep running until all branches have been explored or the max depth is exceeded. Afterwards, the machine will print the configurations at every step (at each depth) and so long as one of the branches reached an accept state, it will accept the input string. While doing so, it will keep track of the depth and how many configurations there were per level, being able to calculate the depth and nondeterminism degree of each input string.
Test Files	
data_a_plus_tset.csv	Description of the NTM that accepts the language a^+: {w|w contains at least one a} Provided by the professor, slightly modified personally. The NTM is described as follows: name states input alphabet tape alphabet starting state accept state reject state transition table, where each transition – in the form current state, input character, next state, written character, direction – takes up a line
data_abc_star_tset.csv	Description of the NTM that accepts the language a*b*c* (there are zero or more sequential a’s, b’s, and c’s)

	Provided by the professor, slightly modified personally. The NTM is described in the same manner as data_a_plus_tset.csv
data_a_plus_inputs_tset.txt	Input strings for the a^+ NTM to test. One per line.
data_abc_star_inputs_tset.txt	Input strings for the a*b*c* NTM to test. One per line.
Output Files	
output_a_plus_tset.txt	Text output of traceNTM_tset.py corresponding to the a^+ NTM. First, it displays the NTM and its parameters Afterwards, it presents the information of each string in the following format: Running {name} on string {input}: {Configurations of the form “{left_of_head}, {state}, {right_of_head}”, separated with |, with each depth in their own line} String {input} (accepted/rejected) in {depth} transitions with a nondeterminism of: {nondeterminism}
output_abc_star_tset.txt	Text output of traceNTM_tset.py corresponding to the a*b*c* NTM, identical in format to output_a_plus_tset.txt

8	Programming languages used, and associated libraries: Python. Associated libraries: statistics, sys.
9	Key data structures (for each sub-project): For the entire project, the following data structure proved key: NTM class. The custom-made class utilized for reading inputs had methods developed to process the acceptance status, print configurations, depth, and calculate the nondeterminism degree.
10	General operation of code (for each subproject): Input files – The .csv files define the NTM parameters and their respective input .txt files contain a series of strings that the NTMs can process. These are directly passed to the

	process file. Process file, traceTM_tset.py – The main code of the project. Within, it declares the attributes and methods of the NTM class which is used to read from the input files and run the necessary Output files – These are the direct textual output of the process file. Each NTM has its own output file which lists the parameters of the machine, and for each string that is run it displays the configurations at every step of increasing depth, its final result status, depth, and nondeterminism degree. Analysis file, plots_ntm_table_analysis_tset.pdf – Table of handwritten, high-end analysis of output files. It excludes the step-by-step depiction of all possible configurations and instead gives a succinct summary of the quantified depth, configurations, and nondeterminism for each input for each machine.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. a^+ NTM – The simple structure and limited number of states with a very obvious way to tell whether a string should be accepted or rejected made this machine an ideal starting unit with which to repeatedly test the correctness of my code. I could manually sketch in paper what I expected my results to be, so I could easily debug at every stage. This was an ability that proved to be quite essential, as a matter of fact. a*b*c* NTM – However nice simplicity was, simply because my code worked in the a^+ NTM it did not mean it could work for all NTMs, so a second NTM to test against was necessary. The a*b*c* NTM proved much more interesting than the other options due to its three valid characters, which would result in many more possible transitions. Testing my algorithm against this more complex system allowed me to spot some particular edge cases in regards to the head movements that I needed to consider which allowed me to enhance the logic of my configuration depictions.
12	How you managed the code development: The first step after retrieving the .csv files provided by the professor was to code a way to read the NTM. Since this would result in an object with specified attributes and running would be a function of it, I decided using a Python class object would be the most effective way of going around it. After initializing a simple class that contained and clearly printed all of the parameters listed in the .csv file, I then wrote the read_ntm() function to initialize the object directly from the input file. Afterwards, and the hardest part, was developing the .run() method to simulate the behavior of the NTM. This .run() method went through many changes in its logic but it ultimately returned a boolean, an integer, and a float. Finally, after testing it many times with a variety of different inputs, I coded a way for it to generate output files and for it to run all of the inputs and machines in a single run.
13	Detailed discussion of results:

	Output files – The a^+ output file clearly demonstrates the simple structure of the NTM, with at any given point being capable of only holding up to three concurrent possibilities. As a result, nondeterminism was maximized with longer strings that appear valid (have a large amount of initial a's) as its degree approaches, yet will never reach, 3. With the current maximum depth of 10, the highest possible nondeterminism is 2.7. This is heavily contrasted by the a*b*c* NTM which has a much more complex structure which exhibits the capacity of being able to hold up to six concurrent possibilities, with the highest nondeterminism of 4.5 with long, valid inputs which combine all three possible characters. The text depiction of the configurations allowed to perceive the branching paths visually and follow the calculation of the machine. Analysis file, plots_ntm_table_analysis_tset.pdf – The differences between the two NTMs are made even more evident in the analysis table, which allows for a quick summary of all input results. Even though both machines had the same depth limit and faced strings of similar lengths, the second NTM had a much higher number of all criteria, especially noticeable in the configurations. This intuitively makes sense, as since it goes deeper and considers more possibilities at every step, the number of configurations will multiply. This table can be very useful to understand the nondeterministic complexity of any NTM.
14	How team was organized: I did not have other members in my team and, given the current scope of the project, I do not believe more would have been necessary. However, if I were to attempt to tackle a more generalized form of this program, perhaps building an application with a more interactive user interface where the user can create, modify and save NTMs and dynamically provide input and observe its process would have been pretty neat. I am also certain that the code as-is can be made more efficient by someone that knows Python syntax better and some of my band-aid solutions may tear on weird edge cases with other NTMs.
15	What you might do differently if you did the project again: For the current scope of the project, I am not certain whether I would modify much of my behavior other than perhaps forcing myself to write out the behavior in all possible scenarios rather than assuming general functions. I ran into some issues when it came to the heads of tapes attempting to point on the edges of the tape as well as my accept and reject states ending their configurations a bit too prematurely. Planning out the pseudocode in these edge cases instead of assuming that a general approach would work across the table would have saved me quite a bit of debugging, but such is the nature of coding – if you pretend you know exactly what you're about to code, you'll only be disappointed.
16	Any additional material: Note that in the output results when a string is either accepted or rejected the program states that it was reached in a particular amount of transitions. This is in regard to the branch taken to reach the outcome or the deepest branch explored, the depth, not the total number of configurations explored (this number can be found in the analysis table under Configurations).

	Any input that is longer than the maximum depth is automatically rejected, under the assumption that it may be a loop, even if it is not. As mentioned, the NTMs were provided by the professor though slightly personally modified.