Project Tracing NTM Behavior Readme Team ack

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: ack</td></tr>
<tr><td>2</td><td>Team members names and netids: Anna Kelley akelley5</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: Successful</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 24</td></tr>
<tr><td>6</td><td>Link to github repository:
https://github.com/akelley04/akelley-TOC-fall-2024-proj2/blob/main/traceTM_ack</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_ack.py</td><td>main code for trace NTM</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>check-a_plus-ack.csv</td><td>transitions for a^+</td></tr>
<tr><td>check-abc_star-ack.csv</td><td>transition for a*b*c*</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>output_a_plus_ack.txt</td><td>outputs for different strings going through machine a^+</td></tr>
<tr><td>output_abc_star_ack.txt</td><td>outputs for different strings going through machine a*b*c*</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:</td></tr>
</table>

	a. language: Python b. libraries: csv, time, defaultdict
9	Key data structures (for each sub-project): a. My main data structures included a list of lists of lists for **configurations**, a list of lists for **transitions**, and a dictionary for **transitions_dict**. **configurations**: this is the final list printed at the end of the program. The outer list is all configurations. Next inside, each index is a level of the configuration tree. And within that, the index is each configuration. **transitions**: this is a list that holds the lists of the information read in from the csv. Each index is a list that represents [name of state the machine might be in, input character from Σ, name of state the machine might go into, output character from Γ that replaces the current character the machine is looking at, whether the machine will move left (L) or right (R) next]. **transitions_dict:** this is a dictionary with the keys as states the machine might be in and the values being all the transitions that can happen from that state.
10	General operation of code (for each subproject): a. This program traces all possible paths taken of a given NTM based off of a given string. It stops when either one of the possible paths ends at an accept state, or all possible paths end at a reject state. To do this, the code uses a BFS method to create a tree of all paths to lead to a reject or accept state. b. **read_csv:** For this particular project, there also had to be the added step of reading in a csv of a certain format. This is the first step of the program. Our function receives an existing cvs as well as an input string. We then collect the name of the machine, the start state, the accept state, the reject state, and all listed transitions. c. **transdict**: receives transitions from csv file. Turns it into a dictionary with the key as states that show up the machine might be in. Values are the transitions associated with that state. **d. bfs**: the main function of this code which creates the trace. begins configurations with the start state and input string with nothing have been read in yet. We then search for an accepted state with the flag **notfound**. Within there is a for loop which goes through our configurations at each level (**level** is associated with the level of the tree). Inside, we loop through our **transitions_dict[curr_state]** this means we are looking at the transitions of the specific state we are currently in. If the character we are reading in matches an input character for a transition, we then check if we must go left or right based on the transition instructions. We call on **go_right** or **go_left** accordingly. If there is no transition found for the current state we are in that matches the current tape's character, we call **go_right** with a special flag to signify this transition must go to a reject state. Whether or not the an input character is found which matches the character being read in, we append the new configuration to our level of configurations. After going through all of those possibilities, we remove duplicate states from the list with special steps due to problems of mutability and tuples. We then check through each of the configurations for the accept state to see if we are able to stop. If an accept state is found, we break our while loop. If not we continue creating more levels for our graph. It should be noted also that the level is crucial to insure the index does not go out of range of our length of configurations. This is what will trigger the situation where we stop from all paths leading to reject.

	e. **go_left**: First we must check to see if the string is blank because if so, we will add a '_' to perform changes on the string. Going left means removing the last character of the left string and adding it to the beginning of the right string. We return a list of the new left string, state we will move into, and a new right string. **f.** **go_right**: Similar to **go_left**, this function must check to see if the string is long enough to perform changes on it. Our new right is the same as the old without its head. We must then see if our flag for whether or not this will lead to a rejected state is triggered. If we must send to a reject state, we shift normally but with no rewrite of the character and then return a list with the new strings and a reject state. If it does not lead to a reject state, the new left string is the old left string with the addition of the rewritten character to its end. We return a list of the new strings and the new state as give in our transition list. **g.** **output**: Prints the requested output from document TM_Project_2024-v2.pdf. Prints the machine name, initial string, depth of tree of configurations, and the total number transitions simulated. Depending on whether the string is accepted or denied the machine will print "String accepted in" or "String denied in" however many levels to get to either the max depth or accepted state.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. a. **check-a_plus-ack.csv**: this helped to correct not accepting an empty string and fine-tuning how to deal with that situation. This also had many opportunities to check nondeterminism and see how the transitions expand and die. b. **check-abc_star-ack.csv**: this helped to fix going to a reject state. let's say if you see a c and then a b, the machine should reject very quickly. Also tested a machine able to accept the empty string.
12	How you managed the code development: a. I began by writing code to work to read in a csv and the information necessary from the file itself. I began with **bfs** but had to make many adjustments and then made **transdict** to help with my confusion in **bfs**. The functions to go left and right were simple to begin with but grew as I came across more errors. After many print statements and trying different types of strings, I found more and more errors. It was very much a trial and error for what showed up and based on the outputs. Key corrections were figuring out how to remove duplicate configurations and to control the level exceeding the length of the configurations.
13	Detailed discussion of results: a. Each line printed represents a level of the configurations tree. The original results were tested based on the example given in TM_Project_2024-v2.pdf. The results were also checked using hand-drawn state diagrams and visualizing the transitions. The results were satisfiable to determine whether or not the machine accepted or reject as well as the configurations of the tree to this decision.
14	How team was organized: a. The team began on the code first and once complete filled out all the necessary documents.

15	What you might do differently if you did the project again: a. I would definitely refine the output and make it look nicer. Although not required for the project, I would like the have a print option to show the exact path of configurations to the accepted state.
16	Any additional material: N/A