Project Readme Template

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: jkeller7
2	Team members names and netids: Jack Keller, jkeller7
3	Overall project attempted, with sub-projects: Tracing NTM Behavior
4	Overall success of the project: Complete Success!
5	Approximately total time (in hours) to complete: 10
6	Link to github repository: https://github.com/hokagejack/NTM_TRACE
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)

File/folder Name	File Contents and Use
Code Files	
traceTM_jkeller7.py	This code does the following: **Parses a Non-deterministic Turing Machine (NTM)**: reads an NTM configuration from a CSV file, extracting details like states, tape symbols, start/accept/reject states, and transitions. **Simulates NTM using Breadth-First Search (BFS)**: Uses BFS to explore the machine's possible configurations at each depth level, tracking transitions and determining if the input string is accepted or rejected. **Tracks Transitions and Nondeterminism**: counts the number of transitions simulated during the BFS search and calculates the degree of nondeterminism based on the number of branches and tree depth. **Backtracks to Find Accepting Path**: If accepted, backtracks through the tree of configurations to display the steps leading to the accepting state.
Test Files	
Folder: NTM_test_files	**data_2x0_DTM_jkeller7.csv**: Describes DTM for strings

Files: data_2x0_DTM_jkeller7.csv data_a_plus_DTM_jkeller7.csv data_a_plus_jkeller7.csv data_abPalindrome_DTM_jkeller7.csv data_abc_star_dtm_jkeller7.csv data_abc_star_jkeller7.csv data_equal_01s_DTM_jkeller7.csv data_equal_01s_jkeller7.csv	with twice the amount of 0’s as there are 1’s **data_a_plus_DTM_jkeller7**.csv**:** Describes DTM for strings made up of 1 or more a’s **data_a_plus_jkeller7.csv:** Describes NTM for strings made up of 1 or more a’s **data_abPalindrome_DTM_jkeller7.csv:** Describes DTM for strings with letters {a,b} that are palindromes **data_abc_star_DTM_jkeller7.csv:** Describes DTM for strings with a*b*c* **Data_abc_star_jkeller7.csv:** Describes NTM for strings with a*b*c* **data_equal_01s_DTM_jkeller7.csv:** Describes DTM for strings with an equal # of 0’s and 1’s **data_equal_01s_jkeller7.csv:** Describes NTM for strings with an equal # of 0’s and 1’s
Output Files	
output_01s_jkeller7.png output_2x0_jkeller7.png output_abc_jkeller7.png output_aplus_jkeller7.png output_palindrome0_jkeller7.png output_palindrome1_jkeller7.png	Each output is a screenshot consisting of the terminal output for different DTM/NTM tests. Each png contains all the specific information relevant to the corresponding test
Plots (as needed)	

<table>
<tr><td></td><td><table><tr><td>plots_jkeller7.png</td><td>All of the output information from above organized within an excel chart</td></tr></table></td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries: python, csv, collections</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):

Lists: storing data
Double Ended Queue: Used in BFS
Tree (list of lists): represents the tree structure where each level is a list of configurations at a particular step
Tuple: most of the written functions return their data in tuples</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):

Below lays out the general operation of how this code functions:

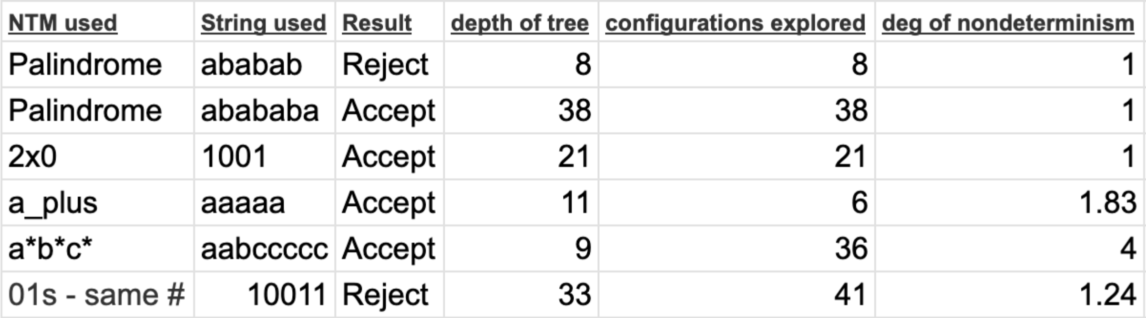
1. <b>Parse the NTM Configuration</b>: gathers component info from csv like name, start state, accept state, reject state, alphabet, etc.
2. <b>Simulate the NTM with BFS</b>: uses BFS to explore the NTM's possible configurations. Starting from the initial state and input string, the algorithm simulates all possible transitions the machine can make at each step. It stores these configurations in a tree structure where each level represents a set of configurations at that particular depth.
3. <b>Transition Logic</b>: checks for transitions based on the current state and tape symbol under the machine’s head. Depending on the direction (left or right), it updates the tape and moves the head accordingly, generating new configurations. If an accepting state is reached, the simulation stops and returns success.
4. <b>Tracks Nondeterminism</b>: counts the total number of transitions and branches (possible paths) at each level, allowing it to compute the "degree of nondeterminism" (how many possible branches exist relative to the depth).
5. <b>Backtracks the Accepting Path</b>: the code backtracks through the state tree to find and display the sequence of configurations that led to the acceptance.
6. <b>Displays Results</b>: shows information like the machine's name, the initial string, the depth of the state tree, the total transitions simulated, the degree of nondeterminism, and the accepting path (if found).</td></tr>
<tr><td>11</td><td>What test cases you used/added, why you used them, what did they tell you about the correctness of your code.

I used all the test cases provided by the instructor in the announcements section of canvas. With each test case, I spent time testing through 5 strings that would accept and 5 strings that would reject to make sure my code functioned effectively. They told me that my code was accurate, and when I was in the debug phase, helped me parse out my errors and isolate mistakes.</td></tr>
</table>

12 How you managed the code development;

To manage the code development, I started by breaking down the task into smaller chunks. First, I simply spent an extensive amount of time reading and understanding the directions content. Then, I focused on parsing the NTM configuration and ensuring I understood the machine's structure. Next, I implemented the BFS algorithm to simulate the NTM's execution step-by-step, making sure to keep track of all my transitions and handle the nondeterministic branches properly. I kept things organized by using lists and deques to manage the configurations and transitions efficiently. I also added various amounts of debug prints to check the process as I went along. Finally, I implemented the backtracking part once I was confident the BFS was working, ensuring that the path to acceptance was correctly traced.

13 Detailed discussion of results:



NTM used	String used	Result	depth of tree	configurations explored	deg of nondeterminism
Palindrome	ababab	Reject	8	8	1
Palindrome	abababa	Accept	38	38	1
2x0	1001	Accept	21	21	1
a_plus	aaaaa	Accept	11	6	1.83
a*b*c*	aabccccc	Accept	9	36	4
01s - same #	10011	Reject	33	41	1.24

My results taught me a lot about the complexity of BFS, as I was able to write a piece of code that accurately parsed through both NTMs and DTMs without any error. Before I created my backtracking function, I was simply printing out the entirety of the tree, with every possible configuration, not just the accepted path. This mistake actually taught me a lot about conceptualizing what it means to be deterministic vs nondeterministic, as well as how the "oracle" or "crystal ball" might operate in real life. Seeing all the possible configurations / paths that were explored before the function was able to output a correct solution taught me about how important the basics of understanding BFS are before a problem like this is expanded to deal with thousands or even more data points. While my code is relatively inefficient, (n^2 time complexity) I did not have any problems with waiting for an output. It was interesting also to see the process of how the code gradually parsed through the string with large tree depths. Understanding that the degree of nondeterminism is the ratio of total configurations explored to the depth reached was also a new learning point for me.

14 How team was organized:

I was the only group member.

15 What you might do differently if you did the project again: If I were to do this project again, I would implement the backtracking within my ntm_bfs function.

16 Any additional material:

None