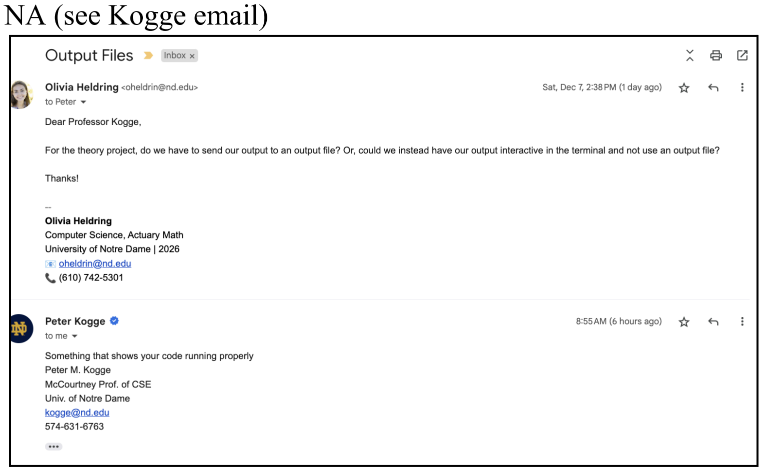
Project Readme Team Oheldrin

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:** oheldrin
2	**Team members names and netids:** Olivia Heldring (oheldrin)
3	**Overall project attempted, with sub-projects:** NTM (project #1)
4	**Overall success of the project:** Success! I was able to write successful code that processes input files correctly.
5	**Approximately total time (in hours) to complete:** 16
6	**Link to github repository:** https://github.com/oliviaheldring/NTM_Project
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_oheldrin.py	Main code
Test Files	
abc_star.csv aplus.csv	Input files (test code)
Output Files	
NA (see Kogge email)  Output Files Inbox Olivia Heldring <oheldrin@nd.edu> Sat, Dec 7, 2:38 PM (1 day ago) to Peter Dear Professor Kogge, For the theory project, do we have to send our output to an output file? Or, could we instead have our output interactive in the terminal and not use an output file? Thanks! -- **Olivia Heldring** Computer Science, Actuary Math University of Notre Dame | 2026 oheldrin@nd.edu (610) 742-5301 Peter Kogge 8:55 AM (6 hours ago) to me Something that shows your code running properly Peter M. Kogge McCourtney Prof. of CSE Univ. of Notre Dame kogge@nd.edu 574-631-6763	

Plots (as needed)	
NA (Tables are apart of my interactive output in the terminal)	

8 **Programming languages used, and associated libraries:** Python (csv, os, sys, deque, tabulate)

9 **Key data structures (for each sub-project):**

Dictionaries: Stores state transition rules, maps current states and input symbols to possible transitions.

Lists: Represented the left and right sections of the Turing machine's tape.

Deque: Used in BFS for change in tape configurations.

Tuples: Next state, symbol to write, movement direction, etc.

Strings: Processed user inputs and kept tape's configuration.

10 **General operation of code (for each subproject)**

The program starts by reading a Turing machine specification file. This file contains details like states, alphabets, start and accept states, and transitions. Then, the code prints out a table to consolidate the input file into an organized and readable format. If the user doesn't pass a valid turing machine file to the command line, there is a usage error to help them correct the input.

My Turingmachine object parses the file, and builds a dictionary of transition rules.

The Tape class operates like the Turing machine's tape. In other words, it tracks the head's position and the state of the left and right sections.

When a string is provided from the user, the code initializes the tape with the input string and sets the machine to its start state.

My program uses a BFS to look at all possible transitions starting from the initial configuration. If the state is an accept state, the process stops, and spits out a table containing information about key metrics. Observe:

```
Give string: abc
+---------------+-------------------+---------+-----------------+--------------------+----------------------+
| User String   | Accepted Status   |   Depth |   # Transitions |   Configs Explored |   Avg Nondeterminism |
+===============+===================+=========+=================+====================+======================+
| abc           | Accepted          |       4 |               5 |                 10 |                  1.3 |
+---------------+-------------------+---------+-----------------+--------------------+----------------------+

[[('', 'q0', 'abc')]]
[['a', 'q0', 'bc'], ['a', 'q1', 'bc'], ['a', 'q2', 'bc'], ['a', 'q3', 'bc']]
[['ab', 'q1', 'c'], ['ab', 'q2', 'c'], ['ab', 'q3', 'c'], ['ab', 'q1', 'c'], ['ab', 'q2', 'c'], ['ab', 'q3', 'c']]
[['abc', 'q2', '_'], ['abc', 'q3', '_']]
[['abc_', 'qacc', '_']]
--------------------------------------
```

If the state is a reject state, it is skipped, and the search continues.

	To prevent infinite loops, I implemented a step limit of 100. The process stops if no result is found after 100 steps. If the machine rejects the string, I print the same table as above but with the appropriate information instead. The program loops, allowing users to keep testing strings interactively.
11	**What test cases you used/added, why you used them, what did they tell you about the correctness of your code.**

1. **Empty String as Input**
   I tested an empty string to verify that the machine correctly initializes the tape with only blank symbols.

```
Give string:
+---------------+-------------------+---------+-----------------+--------------------+----------------------+
| User String   | Accepted Status   |   Depth |   # Transitions |   Configs Explored |   Avg Nondeterminism |
+===============+===================+=========+=================+====================+======================+
|               | Rejected          |       0 |               1 |                  1 |                    0 |
+---------------+-------------------+---------+-----------------+--------------------+----------------------+

[[('', 'q1', ' ')]]
```

2. **String with Valid Transitions**
   Of course, I tested various strings that matched the turing machine rules, confirming that the machine processes transitions correctly.

```
Give string: aaa
+---------------+-------------------+---------+-----------------+--------------------+----------------------+
| User String   | Accepted Status   |   Depth |   # Transitions |   Configs Explored |   Avg Nondeterminism |
+===============+===================+=========+=================+====================+======================+
| aaa           | Accepted          |       4 |               5 |                  7 |                    1 |
+---------------+-------------------+---------+-----------------+--------------------+----------------------+

[[('', 'q1', 'aaa')]]
[['a', 'q1', 'aa'], ['a', 'q2', 'aa']]
[['aa', 'q1', 'a'], ['aa', 'q2', 'a']]
[['aaa', 'q1', '_'], ['aaa', 'q2', '_']]
[['aa', 'q3', 'a_']]
--------------------------------------
```

3. **String Leading to a Reject State**
   I tested strings that should reject to validate that the machine correctly halts on invalid paths

```
Give string: ababa
+---------------+-------------------+---------+-----------------+--------------------+----------------------+
| User String   | Accepted Status   |   Depth |   # Transitions |   Configs Explored |   Avg Nondeterminism |
+===============+===================+=========+=================+====================+======================+
| ababa         | Rejected          |       1 |               2 |                  3 |                 0.67 |
+---------------+-------------------+---------+-----------------+--------------------+----------------------+

[[('', 'q1', 'ababa')]]
[['a', 'q1', 'baba'], ['a', 'q2', 'baba']]
```

4. **Non-Alphabet Characters in Input**
   I tested invalid strings with symbols not in the input alphabet. This checks if the machine rejects such inputs gracefully.

```
Give string: 86gf4
+---------------+-------------------+---------+-----------------+---------------------+-----------------------+
| User String   | Accepted Status   |   Depth |   # Transitions |    Configs Explored |   Avg Nondeterminism |
+===============+===================+=========+=================+=====================+=======================+
| 86gf4         | Rejected          |       0 |               1 |                   1 |                     0 |
+---------------+-------------------+---------+-----------------+---------------------+-----------------------+

[[('', 'q1', '86gf4')]]
___________________________________
```

5. **Nondeterminism**
   I created inputs with multiple valid transitions at several steps. This validated that the BFS worked.

```
Give string: aaaa
+---------------+-------------------+---------+-----------------+---------------------+-----------------------+
| User String   | Accepted Status   |   Depth |   # Transitions |    Configs Explored |   Avg Nondeterminism |
+===============+===================+=========+=================+=====================+=======================+
| aaaa          | Accepted          |       5 |               6 |                   9 |                     1 |
+---------------+-------------------+---------+-----------------+---------------------+-----------------------+

[[('', 'q1', 'aaaa')]]
[['a', 'q1', 'aaa'], ['a', 'q2', 'aaa']]
[['aa', 'q1', 'aa'], ['aa', 'q2', 'aa']]
[['aaa', 'q1', 'a'], ['aaa', 'q2', 'a']]
[['aaaa', 'q1', '_'], ['aaaa', 'q2', '_']]
[['aaa', 'q3', 'a_']]
___________________________________
```

6. **Long Input Strings**
   I tested really long input strings. Particularly ones that would work all the way up to the last char (for a+ a example would be aaaaaaaaaaaaaaaaaa7).

```
Give string: aaaaaaaaaaaaaaaaaaa7
+----------------------+-------------------+---------+-----------------+---------------------+-----------------------+
| User String          | Accepted Status   |   Depth |   # Transitions |    Configs Explored |   Avg Nondeterminism |
+======================+===================+=========+=================+=====================+=======================+
| aaaaaaaaaaaaaaaaaaa7 | Rejected          |      19 |              20 |                  39 |                  0.97 |
+----------------------+-------------------+---------+-----------------+---------------------+-----------------------+

[[('', 'q1', 'aaaaaaaaaaaaaaaaaaa7')]]
[['a', 'q1', 'aaaaaaaaaaaaaaaaaa7'], ['a', 'q2', 'aaaaaaaaaaaaaaaaaa7']]
[['aa', 'q1', 'aaaaaaaaaaaaaaaaa7'], ['aa', 'q2', 'aaaaaaaaaaaaaaaaa7']]
[['aaa', 'q1', 'aaaaaaaaaaaaaaaa7'], ['aaa', 'q2', 'aaaaaaaaaaaaaaaa7']]
[['aaaa', 'q1', 'aaaaaaaaaaaaaaa7'], ['aaaa', 'q2', 'aaaaaaaaaaaaaaa7']]
[['aaaaa', 'q1', 'aaaaaaaaaaaaaa7'], ['aaaaa', 'q2', 'aaaaaaaaaaaaaa7']]
[['aaaaaa', 'q1', 'aaaaaaaaaaaaa7'], ['aaaaaa', 'q2', 'aaaaaaaaaaaaa7']]
[['aaaaaaa', 'q1', 'aaaaaaaaaaaa7'], ['aaaaaaa', 'q2', 'aaaaaaaaaaaa7']]
[['aaaaaaaa', 'q1', 'aaaaaaaaaaa7'], ['aaaaaaaa', 'q2', 'aaaaaaaaaaa7']]
[['aaaaaaaaa', 'q1', 'aaaaaaaaaa7'], ['aaaaaaaaa', 'q2', 'aaaaaaaaaa7']]
[['aaaaaaaaaa', 'q1', 'aaaaaaaaa7'], ['aaaaaaaaaa', 'q2', 'aaaaaaaaa7']]
[['aaaaaaaaaaa', 'q1', 'aaaaaaaa7'], ['aaaaaaaaaaa', 'q2', 'aaaaaaaa7']]
[['aaaaaaaaaaaa', 'q1', 'aaaaaaa7'], ['aaaaaaaaaaaa', 'q2', 'aaaaaaa7']]
[['aaaaaaaaaaaaa', 'q1', 'aaaaaa7'], ['aaaaaaaaaaaaa', 'q2', 'aaaaaa7']]
[['aaaaaaaaaaaaaa', 'q1', 'aaaaa7'], ['aaaaaaaaaaaaaa', 'q2', 'aaaaa7']]
[['aaaaaaaaaaaaaaa', 'q1', 'aaaa7'], ['aaaaaaaaaaaaaaa', 'q2', 'aaaa7']]
[['aaaaaaaaaaaaaaaa', 'q1', 'aaa7'], ['aaaaaaaaaaaaaaaa', 'q2', 'aaa7']]
[['aaaaaaaaaaaaaaaaa', 'q1', 'aa7'], ['aaaaaaaaaaaaaaaaa', 'q2', 'aa7']]
[['aaaaaaaaaaaaaaaaaa', 'q1', 'a7'], ['aaaaaaaaaaaaaaaaaa', 'q2', 'a7']]
[['aaaaaaaaaaaaaaaaaaa', 'q1', '7'], ['aaaaaaaaaaaaaaaaaaa', 'q2', '7']]
___________________________________
```

By doing all these tests, I felt confident that my functions were working properly.

12	**How you managed the code development** I started by sketching out what the Turing machine would need—states, transitions, and then treating each issue independently. First, I worked on the main function and processing the input file. Once I could load the machine rules, I worked on breaking things into manageable pieces, like figuring out how to move the tape head or interpret a transition. I tested every time I added something new. Then, I worked on the BFS algorithm which taught me a lot about traversing through the machine systematically.
13	**Detailed discussion of results:** The program simulates a NTM. It reads input files and interprets states, transitions, and tape data. The functions explore all possible state paths using BFS. The BFS avoids infinite loops by enforcing a 100-step limit, which ensures timely feedback. Accept and reject states are handled properly, and the program outputs clear results, including the number of transitions and the depth of exploration. Tape configurations update dynamically, showing accurate head movements and symbol rewrites. Tested with multiple inputs, the program consistently identified whether a string was accepted or rejected. Overall, it demonstrates the computational power of Turing machines while being robust and user-friendly.
14	**How team was organized:** I worked alone, and did everything myself.
15	**What you might do differently if you did the project again:** Start earlier!!!! I was really stressed over the weekend finishing it up so I would start earlier. I’m alright with the code itself. I just would have allotted more time to complete it.

16 **Any additional material: ReadMe Table**

Turing Machine a+

Input string	Result	Depth	Configs explored	Non-determinism
a	accepted	2	3	1.5
aa	accepted	3	5	1.67
aaa	accepted	4	7	1.75
aaaa	accepted	5	9	1.8
aaaaa	accepted	6	11	1.83
ab	rejected	1	3	3
6	rejected	0	1	0
aabc	rejected	2	5	2.5
8aa	rejected	0	1	0
aaaal	rejected	4	9	2.25

Turing Machine a*b*c*

Input string	Result	Depth	Configs explored	Non-determinism
abc	accepted	4	10	2.5
aabbcc	accepted	7	19	2.71
bbcc	accepted	5	11	2.2
ccc	accepted	4	7	1.75
aaabcccc	accepted	9	24	2.67
cba	rejected	1	3	3
abab	rejected	2	8	4
aabbcq	rejected	5	17	3.4
aabbcccabc	rejected	7	21	3
abbbcac	rejected	5	16	3.2