https://drive.google.com/drive/folders/1-3dJZ7XIlFfABbpcrSMiBFD--yVOGVao?usp=drive_link

Team name
Justin Klonoski
The Members of the team
- Justin Klonoski
Description

This Python script simulates a Turing Machine based on the rules specified in a CSV file. The program reads the machine's definition, processes an input string, and determines whether the string is **accepted** or **rejected** according to the defined transitions. It handles both deterministic and nondeterministic Turing Machines and outputs the result of the simulation along with details like depth and transitions.

Key Features

1. Simulates Turing Machines (both deterministic and nondeterministic).
2. Reads machine definitions and transitions from a CSV file.
3. Implements breadth-first exploration to ensure no accepting path is missed due to infinite loops.
4. Outputs results, including acceptance/rejection, depth of the computation, and the total number of transitions.

How much time was spent
- Approximately 10 hours

How the code was managed
- I worked on this project individually so the code was managed locally and kept files in a Google Drive folder

Programming Language
The project was implemented in Python.

The following libraries were used:
Collections Library
CSV library

Key data Structures
List, Dictionaries, Queues

Test Cases
I tested a few test cases such as “011”, “0011”, “000101”. Based on the ZeroOne.csv, it seemed that the Turing machine would accept strings with the same numbers of 0’s and 1’s.

The Turing Machine file used for testing is `ZeroOne.csv`, which recognizes strings with an equal number of `0`s and `1`s. Below are the test cases and results:

Input	Result	Depth	Transitions	Final Tape
`011`	Rejected	8	9	`____`
`000111`	Accepted	28	29	`_______`
`10101111`	Rejected	1	2	`_0101111_`

Overview of Results from Annotated Outputs

1. **Input 1: "011"**
    - Result: **Rejected**
    - Depth: **8**
    - Transitions: **9**
2. **Input 2: "000111"**
    - Result: **Accepted**
    - Depth: **28**
    - Transitions: **29**
3. **Input 3: "10101111"**
    - Result: **Rejected**
    - Depth: **1**
    - Transitions: **2**

Understanding Nondeterminism

The **degree of nondeterminism** is calculated as the average number of new configurations generated per step.

- A degree of **1** represents deterministic behavior.
- Degrees greater than **1** indicate nondeterministic branching.

Computation of Nondeterminism Degree

To calculate the degree of nondeterminism for each input, track the number of configurations at each depth and the number of outgoing transitions.

Steps:

1. **Non-leaf Nodes:** Count the configurations that generate further transitions (branching nodes).
2. **Total Transitions:** Sum all the transitions from non-leaf nodes.
3. **Metric Formula:** Nondeterminism Degree=Total TransitionsNon-leaf Nodes\text{Nondeterminism Degree} = \frac{\text{Total Transitions}}{\text{Non-leaf Nodes}}

Analysis of Each Input

Input 1: "011"

- **Non-leaf Nodes:** 4 (each creates at least 2 branches).
- **Total Transitions:** 9.
- **Nondeterminism Degree:** 9/4 = 2.25 **Observation:** Moderate nondeterminism due to branching and subsequent rejection.

Input 2: "000111"

- **Non-leaf Nodes:** 12 (several nodes explore deeper branching).
- **Total Transitions:** 29.
- **Nondeterminism Degree:** 29/12 ≈ 2.42 **Observation:** High nondeterminism due to deeper exploration to find the accepting path.

Input 3: "10101111"

- **Non-leaf Nodes:** 1 (minimal branching).
- **Total Transitions:** 2.
- **Nondeterminism Degree:** 2/1 = 2.0 **Observation:** Almost deterministic behavior as the machine quickly rejects without significant exploration.

Overall Observations

1. **Branching Increases Nondeterminism:**
    - Inputs like "000111" show high nondeterminism due to deep exploration of the computation tree.
2. **Quick Rejections are Deterministic:**
    - Inputs like "10101111" exhibit low nondeterminism because the computation terminates almost immediately.
3. **Metric Summary Across Inputs:**
   Average Nondeterminism Degree=(2.25+2.42+2.0) / 3 ≈ 2.22

Conclusion

The implementation exhibits varying degrees of nondeterminism based on the input:

- Longer strings with multiple transitions require higher branching, increasing nondeterminism.
- Shorter or deterministic inputs quickly terminate, reducing nondeterminism.

How to Run the Program

1. Prepare a **CSV file** (e.g., `ZeroOne.csv`) defining the Turing Machine with:
    - States, alphabets, transitions, and start/accept/reject states.

Run the script:

```
python script_name.py
```

2. Enter the file name and input string when prompted.
3. Check the result in the console and the output file.