Turing Machine and Multi-Tape Turing Machine Simulators

This repository presents two Python programs designed for the purpose of simulating Turing Machines:

1. `traceTM-aniceto.py` - This is a simulator for single-tape Turing Machines.
2. `ktape-aniceto.py` (Extra Credit) - This is a simulator created for multi-tape Turing Machines.

Table of Contents

Team Information

- Team Name: Aniceto Team
- Team Members:
    - Gustavo Aniceto (Email: gpradofe@nd.edu)

For this project, the code was collaboratively developed using GitHub for version control. We adhered to a structured development approach, leveraging Git branches for distinct features and bug fixes. This methodology enabled us to work independently on separate sections of the codebase, ensuring that our modifications did not conflict with each other. We seamlessly integrated our contributions through Git pull requests.

The project's developmental lifecycle, spanning from the initial code setup to final testing and documentation, required approximately 6 hours.

Project Overview

The project encompasses two Python programs dedicated to the simulation of Turing Machines:

1. `traceTM-aniceto.py`: This program acts as a simulator for single-tape Turing Machines.
2. `ktape-aniceto.py` (Extra Credit): This program serves as a simulator for multi-tape Turing Machines.

These simulators read and execute Turing machine configurations specified within input files.

File Structure

- `traceTM-aniceto.py`: The single-tape Turing Machine simulator.
- `ktape-aniceto.py` (Extra Credit): The multi-tape Turing Machine simulator.
- `input_files/`: A directory containing input files for Turing Machines.
- `sample_outputs/`: A directory containing sample output files.

Turing Machine Simulator (`traceTM-aniceto.py`)

Documentation for `traceTM`

Overview

The `traceTM` code offers a simulator for single-tape Turing Machines, comprising two classes: `NewTuringMachine` and `TuringMachineSimulator`.

`NewTuringMachine` Class

- This class represents a single-tape Turing Machine.
- Constructor: `__init__(self, input_file)`
  - It initializes the machine with configurations retrieved from the specified input file.
- Method: `parse_input(self, input_file)`
  - This method parses the input file to configure the Turing machine's settings.
- Method: `decompose_transition(self)`
  - This method restructures transition rules for more efficient access.

`TuringMachineSimulator` Class

- This class simulates the operation of a Turing machine.
- Constructor: `__init__(self, turing_machine)`
  - It initializes the simulator with a specific Turing machine.
- Methods:
  - `compute_tree(self, input_string, states_map)`
    - This method simulates the Turing machine on an input string and constructs a computation tree.
  - `compute_path_len(self, computation_tree)`
    - This method traces the path from the accept state (if reached) back to the start state.
  - `set_output(self, computation_tree)`
    - This method configures and returns the output based on the computation tree.
  - `compute_transitions(self, computation_tree)`
    - This method calculates and returns the total number of transitions.

Main Function: `main()`

- This function executes the Turing machine simulation based on command-line arguments.

Test Cases for `traceTM` (Not Extra Credit)

The Turing machine simulator has been rigorously tested with various input strings to ensure accuracy. A test file named `contains111.csv`, created by Gustavo Aniceto, has been uploaded to the collaborative

environment on Google Drive. This file contains comprehensive explanations of the test cases and their expected outcomes. Presented below are some of the test cases alongside their results:

Test Case 1: Turing Machine 'contains010'

- Input String: "000"
- Total Transitions Traced: 7
- Result: String rejected in 3 steps

Test Case 2: Turing Machine 'contains010'

- Input String: "010"
- Total Transitions Traced: 5
- Result: String accepted in 3 steps

Test Case 3: Turing Machine 'contains010'

- Input String: "1111"
- Total Transitions Traced: 9
- Result: String rejected in 4 steps

Test Case 4: Turing Machine 'contains010'

- Input String: "01011010"
- Total Transitions Traced: 5
- Result: String accepted in 3 steps

Test Case 5: Turing Machine 'contains111'

- Input String: "000111000"
- Total Transitions Traced: 11
- Result: String accepted in 6 steps

Test Case 6: Turing Machine 'contains111'

- Input String: "111"
- Total Transitions Traced: 5
- Result: String accepted in 3 steps

Test Case 7: Turing Machine 'contains111'

- Input String: "000"
- Total Transitions Traced: 7
- Result: String rejected in 3 steps

Test Case 8: Turing Machine 'contains111'

- Input String: "0101110"
- Total Transitions Traced: 11

- Result: String accepted in 6 steps

Multi-Tape Turing Machine Simulator (`ktape-aniceto.py`) (Extra Credit)

Documentation for `ktape`

Overview

The `ktape` (Extra Credit) code introduces a simulator tailored for multi-tape Turing Machines. It incorporates a class named `KTapeTuringMachine` designed for the simulation of multi-tape machines.

`KTapeTuringMachine` Class

- This class represents a multi-tape Turing Machine.
- Constructor: `__init__(self, num_tapes, transitions, initial_state, name)`
  - It initializes the machine with the number of tapes, transitions, initial state, and a name.
- Methods:
  - `move_head(self, tape_index, direction)`
    - This method moves the tape head in a specified direction.
  - `read_tape(self, tape_index)`
    - This method reads the symbol at the current tape head position.
  - `write_tape(self, tape_index, symbol)`
    - This method writes a symbol at the current tape head position.
  - `get_tape_heads_symbols(self)`
    - This method retrieves the symbols at all tape head positions.
  - `execute_transition(self)`
    - This method executes a transition based on the current state and symbols at tape head positions.
  - `simulate(self)`
    - This method runs the simulation and displays the steps.

Test Cases for `ktape` (Extra Credit)

- Three test cases have been provided to demonstrate the functionality of the multi-tape Turing Machine:

Test Case 1: Incrementing "101" (Expected Result: "110")

- Input: "101"
- Result: Simulation complete, final tape content: "110"

Test Case 2: Incrementing "111" (Expected Result: "1000")

- Input: "111"
- Result: Simulation complete, final tape content: "1000"

Test Case 3: Incrementing "1010" (Expected Result: "1011")

- Input: "1010"
- Result: Simulation complete, final tape content: "1011"

Testing and Examples

Both simulators have undergone thorough testing with various input strings to ensure their correctness. The repository includes input files and sample output files for verification. For illustrative examples, please refer to the sample_outputs/ directory.

Team Members

- Gustavo Aniceto (Email: gpradofe@nd.edu)