PROJECT REPORT

Submitted by

AKANKSHA MISHRA (RA2211003011513)
KHUSHI PATWARI (RA2211003011529)
ADITYA ROY (RA2211003011539)

Topic: " Turing Machine in JAVA "

Under the Guidance of

DR. M. KARTHIKEYAN

Assistant Professor, Computing Technologies Department

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE ENGINEERING
with specialization in: CORE

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be **University** *u/s 3 of UGC Act, 1956)*

SCHOOL OF COMPUTING
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(May 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603203

BONAFIDE CERTIFICATE

Certified that this project report title **"Turing machine in Java"** is the Bonafide work done by **Akanksha Mishra (RA2211003011513); Khushi Patwari (RA2211003011529); Aditya Roy (RA2211003011539)** Who completed the project under my Supervision. Certified further, Get to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE
DR. M. KARTHIKEYAN
COA – Course Faculty
Assistant Professor
Department Of Computing Technology
SRMIST

SIGNATURE
DR. M. Pushpalatha
Professor & Head
Department Of Computing Technology
SRMIST

Index:

Team Members:

- ✓ Akanksha Mishra (RA2211003011513)
- ✓ Khushi Patwari (RA2211003011529)
- ✓ Aditya Roy (RA2211003011539)

Turing Machine

❖ Introduction:

The Turing machine, a cornerstone in the realm of theoretical computer science, has always symbolized the ultimate universality of computation and the deep-rooted principles underpinning today's computing systems. This project report serves as our gateway to unravel the essence of the Turing machine, while implementing it using the Java programming language.

This conceptual marvel, crafted by the genius mathematician and computer scientist Alan Turing, represents a theoretical model of computation that has laid the very foundation for modern computer science. Its elegance lies in its simplicity, yet it harbors profound computational abilities that testify to the beauty of theoretical constructs.

Our primary goal in this project is to navigate the intriguing world of Turing machines, to grasp their capabilities and their pivotal significance. We will inspect their fundamental elements – the tape, the read/write head, states, transition rules, and symbol alphabet. Moreover, we aim to showcase the Turing machine's universal nature, allowing it to replicate the functions of any other computational device.

This project also shines a light on the historical context, emphasizing the critical role that Turing machines played in addressing fundamental issues in mathematical logic. We delve into their profound contributions to the establishment of theoretical computer science, and we underscore their ongoing relevance in the study of computability, complexity theory, and the evaluation of programming languages and systems through the concept of Turing completeness.

As we embark on this expedition, we will construct a Turing machine in Java, offering tangible evidence of these theoretical concepts in action. Our intent is to bridge the gap between abstract notions and practical demonstrations, allowing us to witness firsthand the remarkable computational abilities of Turing machines.

This project report is a testament to our dedication to comprehending and celebrating the elegant intricacies of the Turing machine and its enduring impact on the field of computer science. It's an open invitation to join us on this journey into the realm of theory, where simplicity gives rise to complexity, and where the foundations of modern computation were initially laid.

❖ Abstract:

Let's dive into the fascinating world of the Turing machine, a creation of the brilliant mind of Alan Turing, a renowned mathematician and computer scientist from the 1930s. This abstract aims to simplify the concept of the Turing machine, emphasizing its importance in theoretical computer science and its significant role in shaping the evolution of modern computers.

The Turing machine serves as a theoretical blueprint for understanding computation. It's like a basic, yet highly effective, model of a computer system. Imagine it as a tape divided into cells, each capable of holding a symbol, along with a read/write head that can move left or right along the tape. This machine follows a set of rules and states, manipulating symbols on the tape as it reads and writes.

What truly sets the Turing machine apart is its universal nature. In other words, it can mimic the behavior of any other computer or computational device. Turing brilliantly showcased this in his 1936 paper "On Computable Numbers," using it to tackle the Entscheidungsproblem, a pivotal problem in mathematical logic. This discovery not only laid the groundwork for theoretical computer science but also played a vital role in the development of modern computers and programming.

Turing machines are indispensable tools in the realm of computability and complexity theory. They help us classify problems as either solvable or unsolvable and analyze the intricacies of algorithms. The concept of Turing completeness, which measures a system's ability to simulate a Turing machine, is a fundamental criterion in computer science for assessing the computational capabilities of programming languages and systems.

In summary, the Turing machine is a fundamental concept with a lasting impact on the world of computer science. Its simplicity and versatility make it a vital instrument for exploring the boundaries and possibilities of computation. The principles behind Turing machines continue to shape our understanding of algorithms, programming languages, and the theoretical foundations of modern computing systems.

❖ Objective:

Our goal is to dive into the essential concepts and principles behind the Turing machine, a creation of the brilliant mathematician and computer scientist, Alan Turing. Our aim is to illuminate the importance of the Turing machine in the realm of theoretical computer science and its lasting impact on the evolution of modern computers. Specifically, we want to:

1. Get a grasp of the fundamental components and operations of the Turing machine, including its tape, read/write head, states, transition rules, and symbol alphabet.

2. Explore the remarkable universal capability of the Turing machine, which allows it to mimic the functions of any other computational device.

3. Delve into the historical context and the pivotal role the Turing machine played in addressing core issues in mathematical logic and its contribution to the foundation of theoretical computer science.

4. Investigate the practical uses and consequences of Turing machines in the examination of computability, complexity theory, and the evaluation of programming languages and systems using the concept of Turing completeness.

5. Acknowledge the enduring significance of the Turing machine as a crucial tool for comprehending computation, algorithms, and the theoretical foundations of modern computing systems.

In summary, our aim is to gain a comprehensive understanding of the Turing machine and its importance in the world of computer science and computation.

❖ **Code:**

```
// Java Program to Illustrate Construction of Turing Machine

// Importing package
// package turing_machine;

// Importing required classes
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;

// Class 1
// Helper class
class Transition {

    char read;
    char write;
    char shift;
    int nextState;

    // Constructor
    // This divides string into specific symbols
    // and next state's number
    Transition(String s)
    {

        read = s.charAt(0);
        write = s.charAt(2);
        shift = s.charAt(4);

        int l = s.length();
        String substr = s.substring(6, l);
        nextState = Integer.parseInt(substr);
    }
}

// Class 2
// Helper class
class State {

    // List of transitions for a state by
```

```
    // creating ArrayList object of Transaction type
    ArrayList<Transition> trs;

    State(ArrayList<Transition> ts) { trs = ts; }
}

// Class 3
// Helper class
class Machine {

    // Scanner object to read input
    Scanner fs;
    // Number of states to be read
    int stateCount;
    // Initialized to start state, and then to keep track
    // of current state in automaton
    int currState;
    // To halt the machine when reached, must not contain
    // any transitions
    int finalState;
    // Blank symbol defined for the machine in the input
    // file
    char blankSym;

    // TAPE is a member of machine
    StringBuffer Tape = new StringBuffer();

    // List of states
    ArrayList<State> states = new ArrayList<>();

    // Method 1
    void buildMachine(Scanner f)
    {
        this.fs = f;

        // Printing the title in the first line of input
        // file
        System.out.println("\n\t" + readString());

        // Reading the string of input symbols (space
        // separated)
        String s = readString();
        System.out.println("Input symbols: " + s);

        // Reading string of other tape symbols defined in
```

```
        // transitions
        s += " " + readString();

        // Reading the blank symbol from the file
        blankSym = readChar();
        System.out.println("Blank symbol: " + blankSym);

        s += " " + blankSym;
        System.out.println("Tape symbols: " + s);

        // Number of states to be defined, say N
        stateCount = readInt();
        System.out.println("\nNumber of States: "
                           + stateCount);

        // Current state variable (currState) is initialized
        // to start-state
        currState = readInt();
        System.out.println("Start State: " + currState);

        // addState() method is called N number of times
        for (int i = 0; i < stateCount; i++)
            addState(i);
    }

    // Method 2
    void addState(int ind)
    {
        // number of transitions is read for a state and
        // stored in trCount
        int trCount = readInt();

        // state with 0 transitions is assigned to be final
        // state for the machine to halt
        if (trCount == 0)
            finalState = ind;
        ArrayList<Transition> trs = new ArrayList<>();

        for (int i = 0; i < trCount; i++) {

            // Each transition object is created and
            // appended to list
            // of transitions
            String s = readString();
            Transition tr = new Transition(s);
```

```
            trs.add(tr);
        }

        // new state object is created by passing list of
        // transitions with the constructor
        State st = new State(trs);
        states.add(st);
    }

    // Method 3
    // To read input from file object "fs" and return it
    String readString()
    {

        String s = fs.next();
        // To ignore lines starting from '//'
        while (s.startsWith("//") || s.isEmpty())
            s = fs.next();
        return s;
    }

    // Method 4
    // To read input from file object as string and
    // return the first character
    char readChar()
    {

        String s = fs.next();
        while (s.startsWith("//") || s.isEmpty())
            s = fs.next();
        return s.charAt(0);
    }

    // Method 5
    // To read input from file object and
    // return it's integer form
    int readInt()
    {
        String s = fs.next();
        while (s.startsWith("//") || s.isEmpty())
            s = fs.next();
        return Integer.parseInt(s);
    }

    // Method 6
```

```
    // To perform transitions on the tape starting from
    // currState
    void runTuring(int index) throws InterruptedException
    {

        while (currState != finalState) {

            // Calling makeTrans() to perform transition and
            // returning the index pointed by the R/W head
            index = makeTrans(index);
            if (index == -1)
                throw new InterruptedException(
                    "ERROR: Transition Not Found! Machine HALTED.");

            // Tape instance printed after each transition
            printTape(index);
        }
    }

    int makeTrans(int index) throws InterruptedException
    {
        if (Tape.charAt(index) == '$')
            throw new InterruptedException(
                "ERROR: Head left the Tape boundary! Machine HALTED.");

        State st = states.get(currState);

        // to traverse across the list of transitions to
        // match tape symbol with read symbol
        for (Transition tr : st.trs) {
            if (tr.read == Tape.charAt(index)) {
                // to write the write-symbol onto the tape
                Tape.replace(index, index + 1,
                             String.valueOf(tr.write));
                currState = tr.nextState;

                switch (tr.shift) {
                case 'R':
                    return index + 1; // shift right on tape
                case 'L':
                    return index - 1; // shift left on tape
                default:
                    return -1; // unknown shift symbol
                }
            }
```

```
        }
        return -1; // transition not found
    }

    void printTape(int index)
    {
        int interval = 500; // in milliseconds
        System.out.println("Tape: " + Tape);
        for (int i = 0; i < index; i++)
            System.out.print(" "); // to align

        // to print the R/W head of machine pointing to
        // particular tape index along with current state
        // index
        System.out.println("     ^q" + currState + "\n");
        try {
            // to print new instance of tape with a
            // particular interval
            Thread.sleep(interval);
        }
        catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}

// Class 4
// Helper class
class FileScanner {
    Scanner scan = new Scanner(System.in);
    Scanner fileScan;
    String inputstr;

    FileScanner() throws FileNotFoundException
    {
        // to read the input from .txt file
        System.out.print("Enter file path: ");
        String path = scan.nextLine();
        fileScan = new Scanner(new File(path));
        fileScan.useDelimiter("\n");
    }

    String buildTape(String str, char blank)
    {
        // str is the input string to be added to the tape
```

```
        // tape defined to begin and end with '$' symbol to
        // avoid indefinite transitions

        String s = "$"; // begin
        for (int i = 0; i < 5; i++)
            s += blank; // adding few blank symbols
        s = s.concat(str); // adding the input string
        for (int i = 0; i < 30; i++)
            s += blank; // adding few more blanks
        s += '$'; // end
        // this concatenated string forms a Tape and is
        // returned
        return s;
    }

    void setTape(Machine m)
    {
        // read input string from console
        System.out.print("\nEnter input string: ");
        inputstr = scan.nextLine();

        // pass string as parameter to buildTape() method
        m.Tape = new StringBuffer(
            buildTape(inputstr, m.blankSym));

        // 6 == initial index of tape that is pointed by R/W
        // head
        m.printTape(6);
    }
}

// Class 5
// Main class
public class TuringMain {

    // Main driver method
    public static void main(String[] args)
    {

        // Display message for better readability
        System.out.println(
            "\n\tTRANSDUCER TURING MACHINE BUILDER\n");

        // Creating new object of Machine class
        Machine m = new Machine();
```

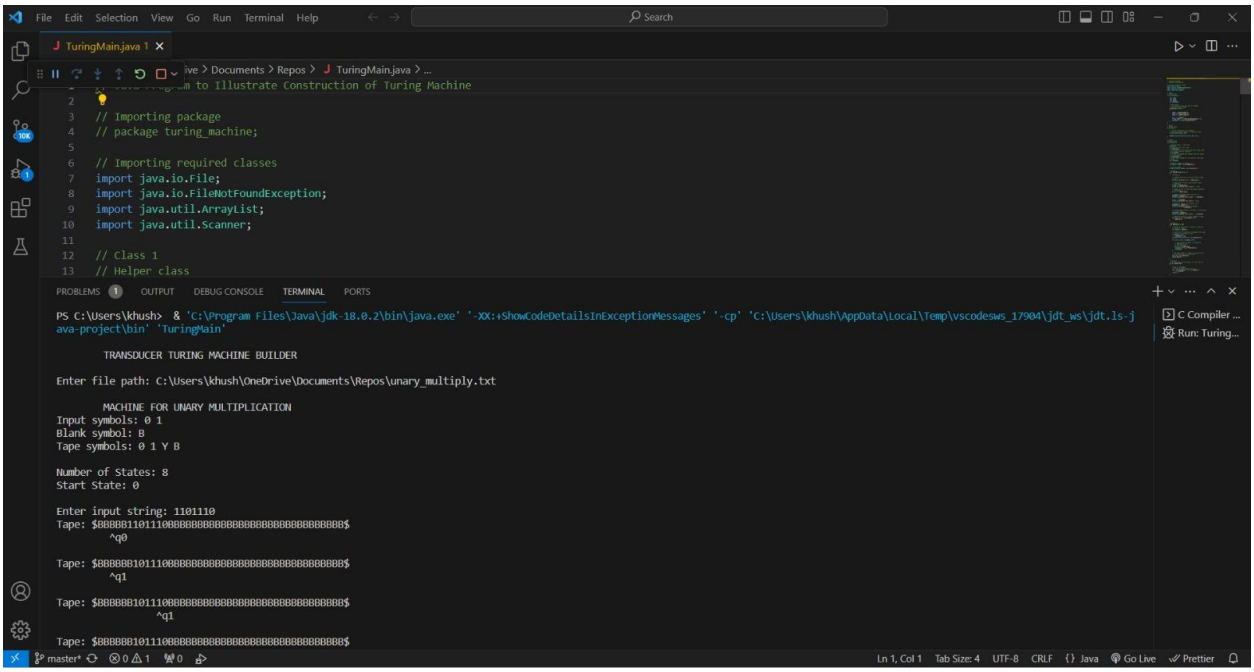
```
        // Try block to check for exceptions
        try {
            FileScanner fileScanner = new FileScanner();

            // constructing the machine using details from
            // Scanner object that reads the file
            m.buildMachine(fileScanner.fileScan);
            fileScanner.setTape(
                m); // setting tape for the machine
            m.runTuring(
                6); // to start execution of Turing Machine
        }
        catch (FileNotFoundException
            | InterruptedException e) {
            System.out.println(e);
            System.exit(0);
        }
    }
}
```

❖ Unary_Multiply:

MACHINE FOR UNARY MULTIPLICATION
//input symbols
0 1
//tape symbols - other than inputs
Y
//blank symbol
B
//number of states
8
//start state
0
//transitions
//format:
//number of states, and then for each state
//read,write,shift,next state symbols
//shift: R-right, L-lef
//q0

```
2
1 B R 1
0 B R 6
//q1
2
0 0 R 2
1 1 R 1
//q2
2
0 0 L 5
1 Y R 3
//q3
3
0 0 R 3
1 1 R 3
B 1 L 4
//q4
3
0 0 L 4
1 1 L 4
Y Y R 2
//q5
4
0 0 L 5
1 1 L 5
Y 1 L 5
B B R 0
//q6
2
1 B R 6
0 B R 7
//q7
0
```

❖ Out Put:



❖ Explanation & Conclusion:

The provided Java code represents a basic implementation of a Turing machine, a theoretical model of computation. Let's break down the key components and functionality of this code:

1. Classes and Structure:

- The code is structured into multiple classes for better organization.
- There are helper classes for `Transition`, `State`, and `Machine`, each responsible for specific aspects of the Turing machine.

2. File Input and Parsing:

- The code reads input from an external file, which contains the machine's configuration.
- It processes and parses this input to set up the Turing machine, including defining states, transitions, input symbols, tape symbols, and the blank symbol.

3. Tape Representation:

- The machine's tape is represented as a `StringBuffer`, with a read/write head that moves left or right on the tape.

4. Transition Execution:

- The `makeTrans` method executes transitions based on the current state and the symbol read from the tape.
- It updates the tape, current state, and the position of the read/write head accordingly.

5. Tape Output and Visualization:
- The code provides a simple visualization of the tape, showing the current state and the position of the read/write head.
- It uses a sleep interval to create a visual effect.

6. Main Method:
- The `main` method serves as the entry point for the program.
- It constructs the Turing machine, reads input from the user (input string), and initiates the Turing machine's execution.

7. Error Handling:
- The code includes basic error handling to catch exceptions, such as file not found and interrupted execution.

8. User Interaction:
- The user is prompted to provide a file path for the machine's configuration and an input string.

In conclusion, this code provides a simple and interactive way to demonstrate the operation of a Turing machine. It reads a machine's configuration from a file, allows users to input a string for processing, and visually shows the machine's operation on the tape. It's a useful educational tool for understanding the fundamentals of Turing machines and their computational capabilities. However, it's important to note that this code represents a basic and minimalistic Turing machine simulator, and actual Turing machines can be much more complex and powerful.

❖ GitHub Link:

https://github.com/adi4231/Turing-Machine-for-unary-Multiplication-in-JAVA.git