INITIAL PROJECT REPORT

Dissertation submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(DATA SCIENCE WITH MACHINE LEARNING)

By

MD SAHIN ALI

Registration No: 12214686

Supervisor

Shubham Sharma

UID: 64339

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

October 2023

DECLARATION STATEMENT

I hereby declare that the work reported in the Assignment Project entitled "First Come First Serve Scheduling" in partial fulfilment of the requirementfor the award of Degree for Bachelor of Technology in Computer Science and Engineering – Data Science with Machine Learning at Lovely Professional University, Phagwara, Punjab is an authentic work carried out under supervision of my research supervisor Mr. Shubham Sharma. I have not submitted this workelsewhere for any degree or diploma.

MD SAHIN ALI

R. No: 12214686

Acknowledgement

I am deeply thankful to Mr. Shubham Sharma for their exceptional mentorship and support during the creation and submission of the Java project focused on First Come First Serve Scheduling. Shubham Sharma's insights and guidance played a pivotal role in the project's success, enhancing its quality and relevance. Their dedication to fostering learning and providing constructive feedback has been invaluable. This project has been an enriching experience, made possible by their expertise and encouragement. I am grateful for the opportunity to learn and grow under their mentorship, and I look forward to applying these skills in future endeavors. Thank you, Shubham Sharma, for your unwavering support.

TABLE OF CONTENTS

1. Introduction
2. Objectives And Scope of Project
3. Methodology
4. Application tool
5. Approach
6. Algorithm Implementation

INTRODUCTION

The "Shortest Job First (SJF) Scheduling Algorithm Implementation" project is a Java-based solution designed to efficiently manage a set of processes using the SJF scheduling algorithm. Processes are represented by unique Process IDs (pid), associated names, and their respective execution times. This project employs a Binary Search Tree (BST) based Priority Queue tailored for a **ProcessEntity** to prioritize and schedule processes. The primary objective is to optimize process execution order by favoring shorter execution times, ultimately minimizing waiting and turnaround times

OBJECTIVES AND SCOPE OF THE PROJECT

1. **Implementation of SJF Algorithm:** The project successfully implements the Shortest Job First (SJF) Scheduling Algorithm, a well-known scheduling technique that prioritizes processes with the shortest execution times. This algorithm helps minimize the waiting time and enhances the overall efficiency of process execution.
2. **Binary Search Tree Priority Queue:** A Binary Search Tree-based Priority Queue is developed specifically to handle **ProcessEntity** objects. The priority queue enables efficient insertion and retrieval of processes based on their execution times, a crucial aspect of the SJF algorithm.
3. **ProcessEntity Class:** A **ProcessEntity** class is introduced to encapsulate essential process attributes. This class includes the Process ID, Process Name, and Execution Time, simplifying the management of process data.
4. **User Interface:** A user-friendly interface is designed to facilitate user interaction with the scheduling system. Users can input and manage processes, visualize the scheduling order, and track the execution status. The user interface adds practicality to the project.
5. **Algorithm Efficiency:** The implemented SJF algorithm proves effective in reducing the average waiting and turnaround times for processes. It enhances system performance by optimizing the order in which processes are executed.
6. **Testing and Validation:** Rigorous testing is conducted to ensure the system's correctness and efficiency. Various process scenarios are examined to validate that the SJF algorithm effectively prioritizes processes with shorter execution times.

SCOPE:

The project's scope encompasses several key components:

- **Software Development:** The project encompasses the development of a software application in Java that implements the SJF Scheduling Algorithm. This software efficiently manages processes using a Binary Search Tree-based Priority Queue.
- **Binary Search Tree Priority Queue:** A specialized Priority Queue is created, designed to work seamlessly with **ProcessEntity** objects. This BST-based structure ensures efficient insertion and retrieval of processes, adhering to the SJF algorithm's requirements.
- **ProcessEntity Class:** The introduction of the **ProcessEntity** class streamlines the handling of process attributes, including Process ID, Process Name, and Execution Time. This class enhances data encapsulation and simplifies process management.
- **User Interface:** A user-friendly interface is provided to enhance user experience. Users can easily input, manage, and monitor the scheduling of processes. The interface serves as a practical tool for system administrators.
- **Algorithm Efficiency:** The SJF algorithm is central to the project's success. By efficiently prioritizing processes with shorter execution times, it significantly

reduces waiting times, improving overall system performance.

- **Testing and Validation:** The project undergoes rigorous testing to verify its correctness and efficiency. It is tested with a variety of process scenarios to ensure that the SJF algorithm operates effectively.

The "Shortest Job First Scheduling Algorithm Implementation" project not only successfully implements a vital scheduling algorithm but also enhances its efficiency through the use of a Binary Search Tree-based Priority Queue and a well-structured **ProcessEntity** class. It offers a practical solution for optimizing process execution and improving system performance.

Algorithm Implementation

```
import java.util.*;

/**
 * The Process class represents a process with a unique
ID (pid), a name, and an execution time.
 */
class Process {
    int pid;
    String name;
    int executionTime;

    public Process(int pid, String name, int
executionTime) {
        this.pid = pid;
        this.name = name;
        this.executionTime = executionTime;
    }
}

/**
 * The BSTNode class represents a node in a Binary Search
Tree (BST) used for priority queue.
 */
class BSTNode {
    Process process;
    BSTNode left;
    BSTNode right;

    public BSTNode(Process process) {
        this.process = process;
        this.left = null;
        this.right = null;
    }
}

/**
 * The BSTPriorityQueue class implements a priority queue
using a Binary Search Tree (BST).
 */
class BSTPriorityQueue {
    BSTNode root;

    public BSTPriorityQueue() {
        root = null;
    }

    /**
     * Insert a process into the priority queue based on
```

```
its execution time.
     */
    public void insert(Process process) {
        root = _insert(root, process);
    }

    private BSTNode _insert(BSTNode node, Process
process) {
        if (node == null) {
            return new BSTNode(process);
        }
        if (process.executionTime <
node.process.executionTime) {
            node.left = _insert(node.left, process);
        } else {
            node.right = _insert(node.right, process);
        }
        return node;
    }

    /**
     * Retrieve the process with the shortest execution
time from the priority queue.
     */
    public Process getProcessWithShortestExecutionTime()
{
        return
_getProcessWithShortestExecutionTime(root);
    }

    private Process
_getProcessWithShortestExecutionTime(BSTNode node) {
        if (node == null) {
            return null;
        }
        if (node.left == null) {
            return node.process;
        }
        return
_getProcessWithShortestExecutionTime(node.left);
    }

    /**
     * Remove a process from the priority queue.
     */
    public void remove(Process process) {
        root = _remove(root, process);
    }

    private BSTNode _remove(BSTNode node, Process
process) {
        if (node == null) {
```

```
            return null;
        }
        if (process.executionTime <
node.process.executionTime) {
            node.left = _remove(node.left, process);
        } else if (process.executionTime >
node.process.executionTime) {
            node.right = _remove(node.right, process);
        } else {
            if (node.left == null) {
                return node.right;
            } else if (node.right == null) {
                return node.left;
            }
            BSTNode minNode = getMin(node.right);
            node.process = minNode.process;
            node.right = _remove(node.right,
minNode.process);
        }
        return node;
    }

    private BSTNode getMin(BSTNode node) {
        while (node.left != null) {
            node = node.left;
        }
        return node;
    }
}

/**
 * The SJFScheduling class implements the Shortest Job
First (SJF) scheduling algorithm using a BST-based
Priority Queue.
 */
public class SJFScheduling {

    /**
     * Calculate the average waiting time for a list of
processes using the SJF scheduling algorithm.
     */
    public static double shortestJobFirst(List<Process>
processes) {
        BSTPriorityQueue priorityQueue = new
BSTPriorityQueue();
        int totalWaitingTime = 0;
        int currentTime = 0;

        for (Process process : processes) {
            priorityQueue.insert(process);
        }
```

```
        while (priorityQueue.root != null) {
            Process shortestProcess =
priorityQueue.getProcessWithShortestExecutionTime();
            int waitingTime = Math.max(currentTime,
shortestProcess.executionTime);
            totalWaitingTime += waitingTime;
            currentTime += shortestProcess.executionTime;
            priorityQueue.remove(shortestProcess);

            System.out.println("Process " +
shortestProcess.name + " completed.");
        }

        return (double) totalWaitingTime /
processes.size();
    }

    public static void main(String[] args) {
        List<Process> processes = new ArrayList<>();
        processes.add(new Process(1, "Process A", 6));
        processes.add(new Process(2, "Process B", 8));
        processes.add(new Process(3, "Process C", 7));
        processes.add(new Process(4, "Process D", 3));

        double averageWaitingTime =
shortestJobFirst(processes);
        System.out.println("Average Waiting Time: " +
averageWaitingTime);
    }
```

Output

Run | Debug | Stop | Share | Save | { } Beautify | Language Java

SJFScheduling.java

```
            priorityQueue.insert(process);
        }

        while (priorityQueue.root != null) {
            Process shortestProcess = priorityQueue.getProcessWithShortestExecutionTime();
            int waitingTime = Math.max(currentTime, shortestProcess.executionTime);
            totalWaitingTime += waitingTime;
            currentTime += shortestProcess.executionTime;
            priorityQueue.remove(shortestProcess);

            System.out.println("Process " + shortestProcess.name + " completed.");
        }

        return (double) totalWaitingTime / processes.size();
    }

    public static void main(String[] args) {
        List<Process> processes = new ArrayList<>();
        processes.add(new Process(1, "Process A", 6));
        processes.add(new Process(2, "Process B", 8));
        processes.add(new Process(3, "Process C", 7));
        processes.add(new Process(4, "Process D", 3));

        double averageWaitingTime = shortestJobFirst(processes);
        System.out.println("Average Waiting Time: " + averageWaitingTime);
    }
}

```

input

```
Process Process D completed.
Process Process A completed.
Process Process C completed.
Process Process B completed.
Average Waiting Time: 8.5


...Program finished with exit code 0
```

GitHub Repository link

https://github.com/DEvIL2770/FIRST-COME-FIRST-SERVE-SCHEDULING