Project Report: Page Replacement Algorithm Simulator

1. Project Overview

This project is a Page Replacement Algorithm Simulator that visually demonstrates and analyzes various page replacement techniques used in operating systems. The primary goal is to help users understand and compare different algorithms such as FIFO (First-In-First-Out), LRU (Least Recently Used) and Optimal Page Replacement in terms of page faults and memory efficiency.

2. Module-Wise Breakdown

1. User Interface Module
    - Provides a GUI using Tkinter for user interaction.
    - Accepts user input for page reference strings and the number of frames.
    - Displays simulation results.
2. Algorithm Implementation Module
    - Implements FIFO, LRU, and Optimal page replacement algorithms.
    - Tracks hits, miss ratio and page faults.
    - Maintains a history of memory states for visualization.
3. Visualization Module
    - Uses Matplotlib to generate graphical representations of memory utilization.
    - Displays a page table and performance charts.

3. Functionalities

- Accepts user input for reference string and the number of frames.
- Allows the user to select an algorithm (FIFO, LRU, or Optimal).
- Computes page faults and hit/miss ratios.
- Displays memory states step-by-step.
- Provides graphical analysis of memory utilization.
- Offers a performance comparison of algorithms.

4. Technology Used

Programming Languages:

- Python

Libraries and Tools:

- Tkinter (GUI)

- Matplotlib (Visualization)
- NumPy (Data Handling)

Other Tools:

- GitHub for version control

4. Flow Diagram

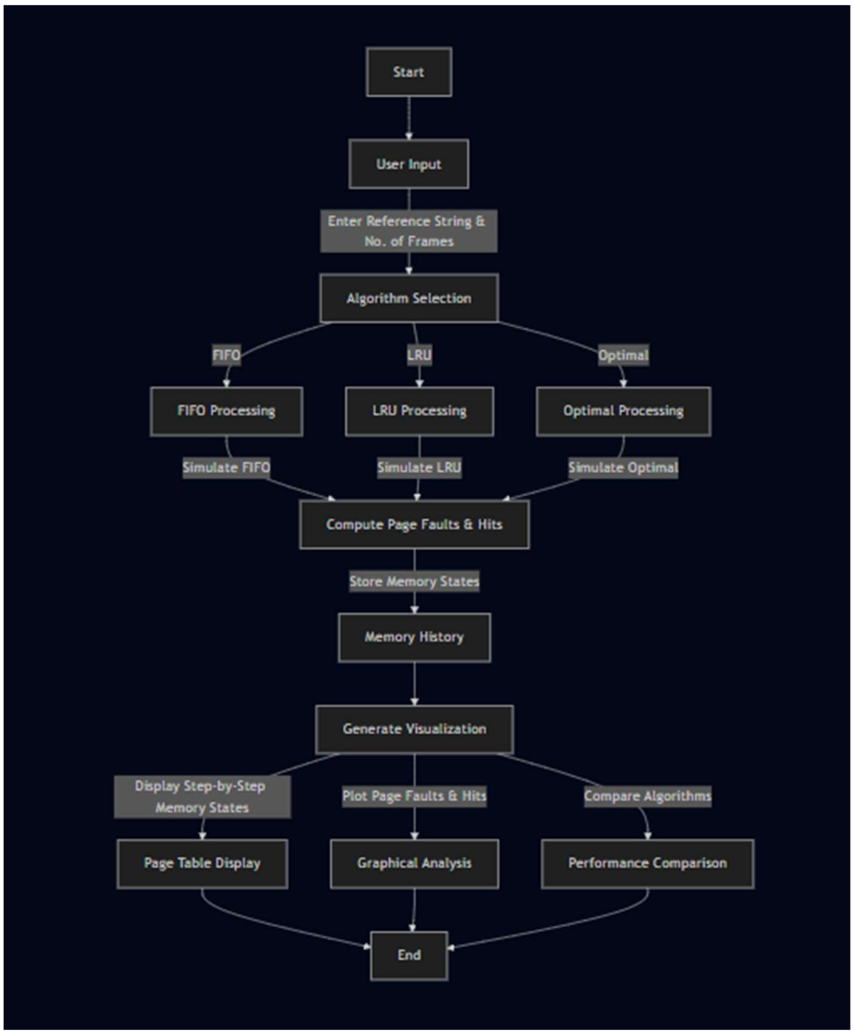

Figure 1: Page Replacement Algorithm Simulator Working Flowchart

1. **User Input Phase**

- The user provides a page reference string and several frames.
- Select a page replacement algorithm (FIFO, LRU, or Optimal).

2. **Algorithm Processing Phase**

- The chosen algorithm is executed, computing page faults, hits, and memory states.

3. **Visualization Phase**

   - Memory states are stored and visualized using step-by-step page table updates.
   - Graphical analysis is generated, showing page faults, hit/miss ratios, and algorithm comparisons.

6. Revision Tracking on GitHub

- Repository Name: Efficient-PRA-Simulator-Project
- GitHub Link: https://github.com/Raj-3435/Efficient-PRA-Simulator-Project

7. Conclusion and Future Scope

The project successfully simulates and visualizes FIFO, LRU, and Optimal page replacement algorithms offering an interactive approach to understanding these techniques. Future improvements could include:

- Adding additional algorithms such as Clock and LFU.
- Enhancing the UI with real-time animations.
- Providing exportable reports of simulation results.

8. References

1. Operating System Concepts by Abraham Silberschatz
2. Modern Operating Systems by Andrew S. Tanenbaum
3. GeeksforGeeks

Appendix

A. AI-Generated Project Elaboration/Breakdown Report

1. Project Overview

Goals:

- Develop a simulator to test and compare various page replacement algorithms efficiently.
- Provide real-time visualization of memory states and page faults.
- Enable users to analyse algorithm performance under different workloads.

Expected Outcomes:

- A functional **GUI-based simulator** for visualizing page replacement strategies.
- Detailed performance comparison metrics (e.g., page faults, execution time).

- Interactive features for users to customize inputs (e.g., page reference sequences, memory size).

Scope:

- Simulate FIFO, LRU, Optimal, LFU, and possibly AI-based algorithms.
- Provide graphical visualization of memory states.
- Compare algorithms based on various workload patterns.
- Focus on educational and research purposes, ensuring ease of understanding.

2. Module-Wise Breakdown

Module 1: GUI (Graphical User Interface)

- Purpose: Provide an interactive interface for users to input page reference sequences, choose algorithms, and visualize results.

Module 2: Core Algorithm Implementation

- Purpose: Implement and optimize various page replacement algorithms.

Module 3: Data Visualization & Analysis

- Purpose: Display statistics, generate comparative graphs, and highlight performance metrics.

3. Functionalities

Module 1: GUI

- Input Fields:
    - User-defined page reference sequences.
    - Selectable memory frame size.
    - Dropdown to choose algorithms.
- Buttons & Interactions:
    - Start Simulation to run algorithms.
    - Step-by-Step Execution mode for detailed understanding.
    - Reset & Export Results.
- Example Use Case:
    - User enters [7, 0, 1, 2, 0, 3, 4, 2, 3, 0, 3, 2], selects LRU, and views the simulation.

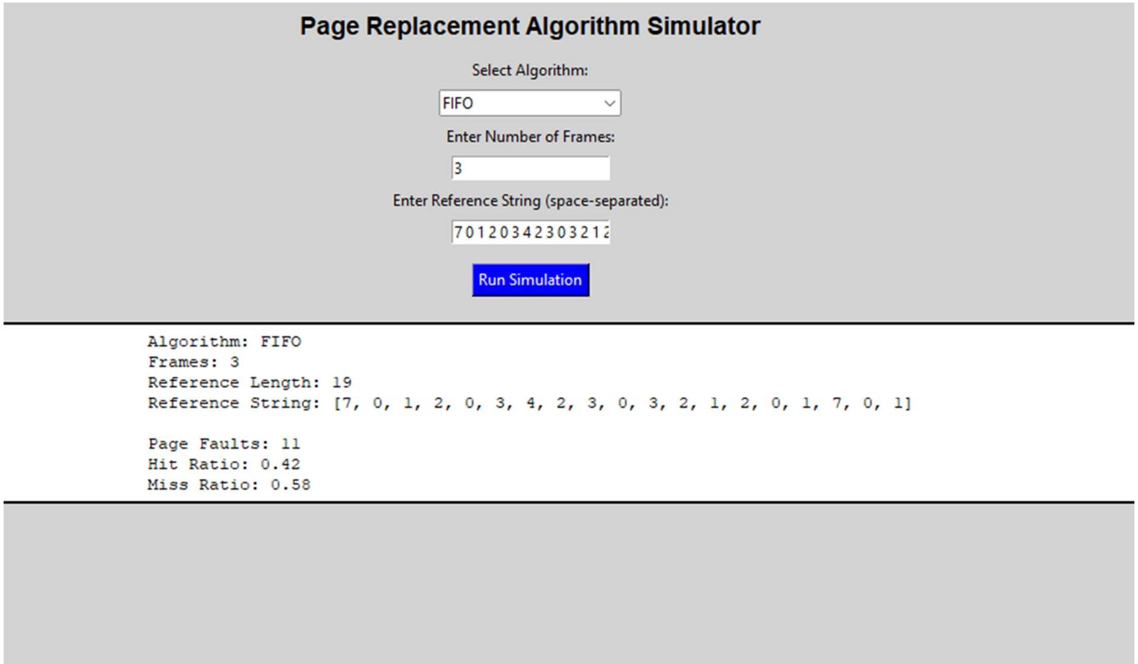

Figure 2: Page Replacement Algorithm Simulator Interface/GUI

Module 2: Core Algorithm Implementation

- Implement algorithms:
    - FIFO (First-In-First-Out)
    - LRU (Least Recently Used)
    - Optimal Algorithm
- Example Execution:
    - Given frame size = 3, sequence = [1, 3, 0, 3, 5]
    - FIFO Output:
        - 1 → Page Fault
        - 3 → Page Fault
        - 0 → Page Fault
        - 3 → Hit
        - 5 → Page Fault (1 removed)
- Optimization Features:
    - Adaptive Replacement Policies (ML-based tuning) for AI-driven enhancements.

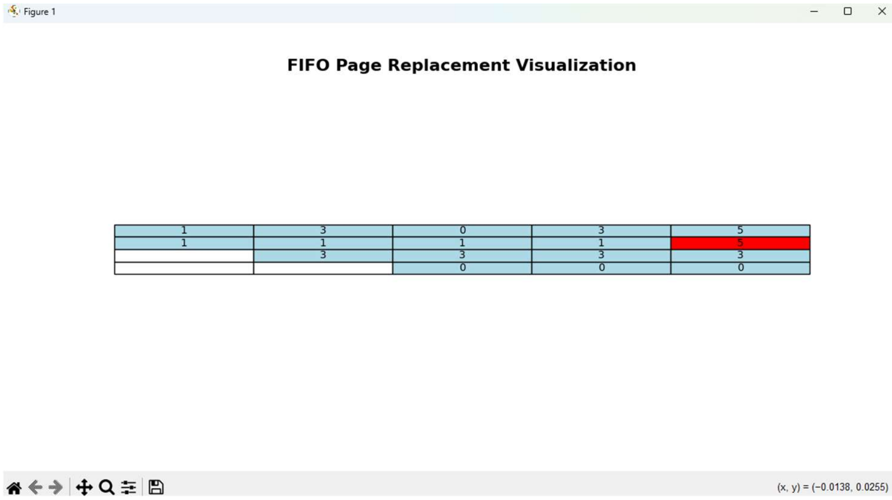

Figure 3: An example of FIFO Table Visualization

Module 3: Data Visualization & Analysis

- Graphical Visualization:
  - Stack representation for LRU.
  - Queue structure for FIFO.
- Comparison Metrics:
  - Page Fault Rate (%)
  - Execution Time
  - Memory Utilization
- Example Chart:
  - Bar Graph: Algorithm vs. Page Faults.
  - Line Graph: Page replacement efficiency over time.

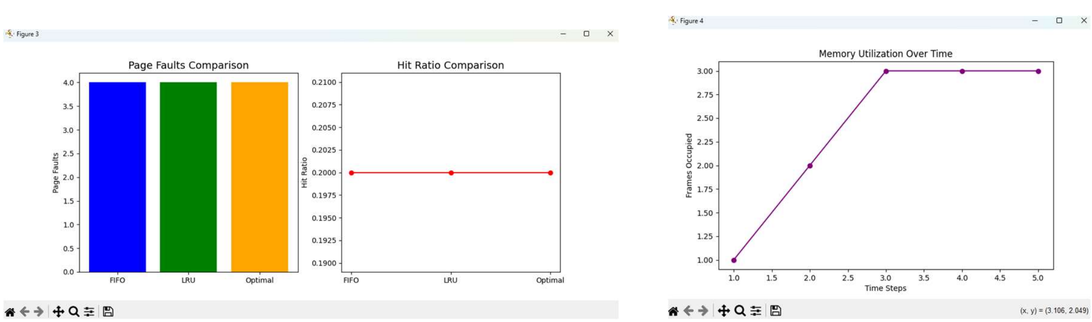

Figure 4: Showcasing Various Graphical Analysis based on certain criteria.

4. Technology Recommendations

Component	Technology/Tool
Programming Language	Python (simpler), Java (faster), or C++ (efficient)
GUI Development	Tkinter (Python), PyQt, or JavaFX
Algorithm Implementation	Python (NumPy, Pandas) or Java (Collections)
Data Visualization	Matplotlib, Seaborn, Plotly
Performance Analysis	Pandas, Scipy for statistical metrics
AI-Based Extensions	Reinforcement Learning (RL) with TensorFlow/PyTorch

5. Execution Plan

Step 1: Setup Development Environment

- Choose Python (Tkinter + Matplotlib) for easier GUI visualization.
- Install dependencies

```
pip install matplotlib pandas numpy seaborn
```

Step 2: Implement Page Replacement Algorithms

- Write functions for FIFO, LRU and Optimal in Python.
- Test each algorithm with small reference sequences.

Step 3: Build GUI

- Use Tkinter/PyQt to create input fields and buttons.
- Implement an output display area for memory state visualization.

Step 4: Integrate Data Visualization

- Implement heatmaps for memory states.
- Compare algorithms using graphs and bar charts.

Step 5: Optimize Performance

- Use efficient data structures (e.g., Deque for LRU).
- Optimize computation for large datasets.

Step 6: Testing & Debugging

- Test with different workloads (random, sequential, cyclic patterns).
- Measure execution time to ensure efficiency.

B. Problem Statement

18. Efficient Page Replacement Algorithm Simulator Description: Design a simulator that allows users to test and compare different page replacement algorithms (e.g., FIFO, LRU, Optimal). The simulator should provide visualizations and performance metrics to aid in understanding algorithm efficiency

C. Solution/Code

```python
import tkinter as tk
from tkinter import ttk, messagebox
import numpy as np
import matplotlib.pyplot as plt

# ------------------ Page Replacement Algorithms ------------------

def fifo_page_replacement(pages, frames):
    frame_list = []
    history = []
    fault_positions = []
    hits, misses = 0, 0
    fifo_index = 0  # Tracks which index to replace

    for i, page in enumerate(pages):
        if page in frame_list:
            hits += 1
        else:
            misses += 1
            if len(frame_list) < frames:
                frame_list.append(page)
            else:
                replaced_index = fifo_index  # Get index of oldest page
                frame_list[replaced_index] = page  # Replace at the same position
                fault_positions.append((i, page, replaced_index))  # Store step, new page, replaced index
                fifo_index = (fifo_index + 1) % frames  # Move FIFO pointer

        history.append(frame_list.copy())

    return history, hits, misses, fault_positions


def lru_page_replacement(pages, frames):
    frame_list = []  # Stores pages in memory
    history = []  # Stores frame state history
    fault_positions = []  # Stores replaced page details
    indexes = {}  # Stores last used index of each page

    hits, misses = 0, 0

    for i, page in enumerate(pages):
        if page in frame_list:
            hits += 1
            indexes[page] = i  # Update last used index
        else:
            misses += 1
            if len(frame_list) < frames:
                frame_list.append(page)  # Add page if space available
            else:
                # Find the least recently used page (smallest index in 'indexes')
                lru_page = min(indexes, key=indexes.get)  # Page with the lowest index
                replaced_index = frame_list.index(lru_page)  # Find its position in the frame
```

```
                frame_list[replaced_index] = page  # Replace with new page
                fault_positions.append((i, page, replaced_index))  # Store fault info
                indexes.pop(lru_page)  # Remove old page from usage tracker

            indexes[page] = i  # Update last used index

        history.append(frame_list.copy())  # Store frame state

    return history, hits, misses, fault_positions



def optimal_page_replacement(pages, frames):
    frame_list = []  # Stores current pages in memory
    history = []  # Stores state of frames at each step
    fault_positions = []  # Stores (step, new page, replaced index)
    hits, misses = 0, 0

    for i, page in enumerate(pages):
        if page in frame_list:
            hits += 1  # Page hit
        else:
            misses += 1  # Page fault

            if len(frame_list) < frames:
                frame_list.append(page)  # Fill empty frames first
            else:
                # Dictionary to store the next occurrence index of each page in the frame
                future_use = {frame: float('inf') for frame in frame_list}

                for frame in frame_list:
                    if frame in pages[i+1:]:  # Check if frame appears in future
                        future_use[frame] = pages[i+1:].index(frame) + i + 1  # Absolute index

                # Find the page that is used farthest in the future
                page_to_replace = max(future_use, key=future_use.get)
                replaced_index = frame_list.index(page_to_replace)

                # Replace the page
                frame_list[replaced_index] = page
                fault_positions.append((i, page, replaced_index))  # Store replacement step

        history.append(frame_list.copy())

    return history, hits, misses, fault_positions


# ------------------ Visualization ------------------

def visualize_page_replacement(algorithm, pages, frames, history, fault_positions, hits, misses):
    fig, ax = plt.subplots(figsize=(12, 6))
    ax.set_title(f'{algorithm} Page Replacement Visualization', fontsize=16, fontweight='bold')

    num_steps = len(pages)
    num_frames = frames

    # Create table data
    table_data = [[''] * num_steps for _ in range(num_frames)]
    cell_colors = [['white'] * num_steps for _ in range(num_frames)]

    # Store replacement steps for proper visualization
    replaced_positions = {pos[0]: (pos[1], pos[2]) for pos in fault_positions}  # (step: (new_page, position))

    for col, page in enumerate(pages):
        frame_state = history[col]  # Get frame content at current step

```

```python
        for row in range(num_frames):
            if row < len(frame_state):
                table_data[row][col] = str(frame_state[row])

                # Highlight replaced pages in red
                if col in replaced_positions and row == replaced_positions[col][1]:
                    cell_colors[row][col] = 'red'
                else:
                    cell_colors[row][col] = 'lightblue'

    table_data = np.array(table_data)

    ax.axis('tight')
    ax.axis('off')
    table = ax.table(cellText=table_data, cellLoc='center', loc='center',
                     cellColours=cell_colors, colLabels=[str(p) for p in pages],
                     colColours=['lightblue'] * num_steps)

    plt.show()


# ------------------ Enhanced Visualization ------------------
def visualize_page_replacement(algorithm, pages, frames, history, fault_positions, hits, misses):
    fig, ax = plt.subplots(figsize=(12, 6))
    ax.set_title(f'{algorithm} Page Replacement Visualization', fontsize=16, fontweight='bold')
    num_steps = len(pages)
    num_frames = frames

    # Table Data Setup
    table_data = [[''] * num_steps for _ in range(num_frames)]
    cell_colors = [['white'] * num_steps for _ in range(num_frames)]
    replaced_positions = {pos[0]: (pos[1], pos[2]) for pos in fault_positions}

    for col, page in enumerate(pages):
        frame_state = history[col]
        for row in range(num_frames):
            if row < len(frame_state):
                table_data[row][col] = str(frame_state[row])
                if col in replaced_positions and row == replaced_positions[col][1]:
                    cell_colors[row][col] = 'red'  # Highlight replaced pages
                else:
                    cell_colors[row][col] = 'lightblue'

    table_data = np.array(table_data)
    ax.axis('tight')
    ax.axis('off')
    ax.table(cellText=table_data, cellLoc='center', loc='center',
             cellColours=cell_colors, colLabels=[str(p) for p in pages],
             colColours=['lightblue'] * num_steps)
    plt.show()

# ------------------ Performance Visualization ------------------
def visualize_performance(algorithms, page_faults, hit_ratios):
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    # Bar Chart for Page Faults
    axes[0].bar(algorithms, page_faults, color=['blue', 'green', 'orange'])
    axes[0].set_title("Page Faults Comparison", fontsize=14)
    axes[0].set_ylabel("Page Faults")

    # Line Chart for Hit Ratios
    axes[1].plot(algorithms, hit_ratios, marker='o', linestyle='-', color='red')
    axes[1].set_title("Hit Ratio Comparison", fontsize=14)
    axes[1].set_ylabel("Hit Ratio")

    plt.show()

def visualize_memory_utilization(page_refs, frames):
```

```python
    utilization = [min(i + 1, frames) for i in range(len(page_refs))]
    plt.figure(figsize=(8, 5))
    plt.plot(range(1, len(page_refs) + 1), utilization, marker='o', linestyle='-', color='purple')
    plt.xlabel("Time Steps")
    plt.ylabel("Frames Occupied")
    plt.title("Memory Utilization Over Time")
    plt.show()

# ------------------ Main GUI and Execution ------------------
def run_simulation():
    try:
        page_refs = list(map(int, entry_pages.get().split()))
        num_frames = int(entry_frames.get())
        selected_algo = algo_var.get()

        if num_frames <= 0 or not page_refs:
            messagebox.showerror("Error", "Invalid Input!")
            return

        algorithms = ["FIFO", "LRU", "Optimal"]
        results = {}

        for algo in algorithms:
            if algo == "FIFO":
                history, hits, misses, fault_positions = fifo_page_replacement(page_refs, num_frames)
            elif algo == "LRU":
                history, hits, misses, fault_positions = lru_page_replacement(page_refs, num_frames)
            elif algo == "Optimal":
                history, hits, misses, fault_positions = optimal_page_replacement(page_refs, num_frames)
            results[algo] = (misses, hits / (hits + misses))

            if algo == selected_algo:
                visualize_page_replacement(algo, page_refs, num_frames, history, fault_positions, hits, misses)

                # Computation Outline
                result_text.set(
                    f"Algorithm: {selected_algo}\n"
                    f"Frames: {num_frames}\n"
                    f"Reference Length: {len(page_refs)}\n"
                    f"Reference String: {page_refs}\n\n"
                    f"Page Faults: {misses}\n"
                    f"Hit Ratio: {hits / (hits + misses):.2f}\n"
                    f"Miss Ratio: {misses / (hits + misses):.2f}"
                )

        visualize_performance(algorithms, [results[a][0] for a in algorithms], [results[a][1] for a in algorithms])
        visualize_memory_utilization(page_refs, num_frames)

    except ValueError:
        messagebox.showerror("Error", "Invalid input! Enter space-separated integers.")

     # ------------------ GUI Implementation ------------------

root = tk.Tk()
root.title("Page Replacement Algorithm Simulator")
root.geometry("500x400")
root.configure(bg="lightgray")

tk.Label(root, text="Page Replacement Algorithm Simulator", font=("Arial", 14, "bold"), bg="lightgray").pack(pady=10)

tk.Label(root, text="Select Algorithm:", bg="lightgray").pack()
algo_var = tk.StringVar(value="FIFO")
```

```python
algo_menu = ttk.Combobox(root, textvariable=algo_var, values=["FIFO", "LRU", "Optimal"], state="readonly")
algo_menu.pack(pady=5)

tk.Label(root, text="Enter Number of Frames:", bg="lightgray").pack()
entry_frames = tk.Entry(root)
entry_frames.pack(pady=5)

tk.Label(root, text="Enter Reference String (space-separated):", bg="lightgray").pack()
entry_pages = tk.Entry(root)
entry_pages.pack(pady=5)

tk.Button(root, text="Run Simulation", command=run_simulation, bg="blue", fg="white").pack(pady=10)

result_text = tk.StringVar()
result_label = tk.Label(root, textvariable=result_text, justify="left", bg="white", font=("Courier", 10), relief="solid", padx=10, pady=5)
result_label.pack(pady=10, fill="both")

root.mainloop()  # Ensure the GUI loop starts

```

--END---