Simulating Cache Memory Mapping and Replacement Algorithms using Git Collaboration

24MCAT103-Digital Fundamentals and Computer Architecture

&

24MCAT107-Advanced Software Engineering

Name: Anandhu Pradeep

Roll No.: 14

Objective:

To integrate theoretical knowledge of cache memory organization and replacement algorithms (from COA) with practical software version control and team collaboration skills (from SE - Git).

Scenario:

You are part of a development team tasked with creating a Cache Simulation Tool that models the behavior of different cache mapping functions (Direct, Associative, Set Associative) and cache replacement algorithms (FIFO, LRU, LFU).

Your team will collaboratively develop, test, and document the project using Git for version control. Each member must contribute through branches, commits, mergers, and documentation.

Part A: 24MCAT103-Digital Fundamentals and Computer Architecture Component (Simulation Design)

Introduction:

This project implements a Cache Simulation Tool to demonstrate cache mapping and replacement techniques. The section I mainly focuses on the Least Frequently Used (LFU) cache replacement algorithm.

LFU Algorithm Explanation:

LFU replaces the cache block with the lowest access frequency when the cache is full. Each cache block maintains a frequency counter which is incremented on every access.

Algorithm / Pseudocode

Initialize cache and frequency table.

For each memory reference:

- If block is found → Hit, increment frequency.
- Else → Miss.
- If cache is full → remove least frequently used block.
- Insert new block with frequency = 1.

Display cache state after each reference.

Design and implement a program (in C / Java / Python) to simulate:

- Here in used python programming languages to implement LFU simulations

Program code:

```
def lfu(pages, frames):
    memory = []
    freq = {}
    time = {}
    page_faults = 0
    t = 0

    for page in pages:
        t += 1
        print(f"\nRequest page: {page}")

        if page in memory:
            freq[page] += 1
            print("-> Hit")
        else:
            page_faults += 1
            print("-> Fault")

            if len(memory) < frames:
                memory.append(page)
```

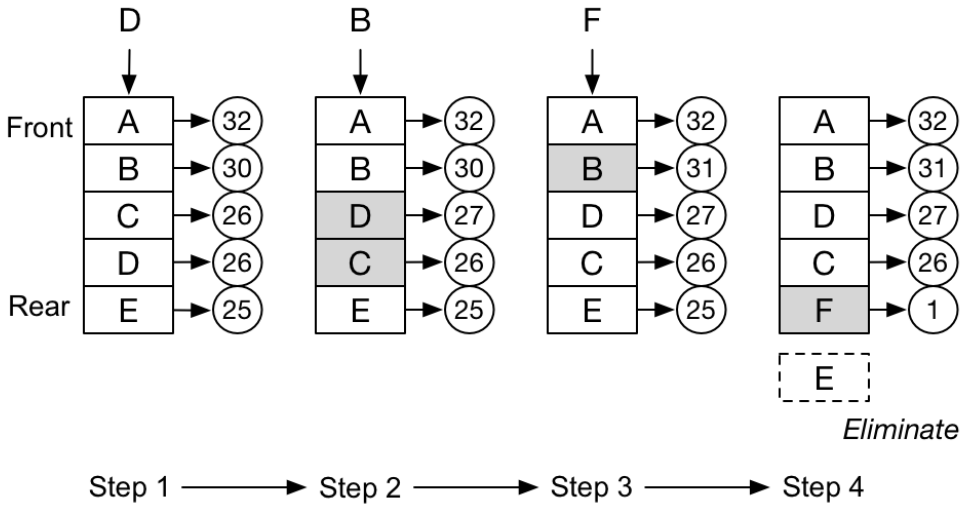
```
            else:
                victim = memory[0]
                for p in memory:
                    if freq[p] < freq[victim]:
                        victim = p
                    elif freq[p] == freq[victim]:
                        if time[p] < time[victim]:
                            victim = p
                print(f"   Replacing page: {victim}")
                memory[memory.index(victim)] = page
                del freq[victim]
                del time[victim]
            freq[page] = freq.get(page, 0) + 1
            time[page] = t

        print(f"   Memory: {memory}")
        print(f"   Freq:   {freq}")

    print("\nTotal Page Faults:", page_faults)
# Example usage
pages = [1, 2, 3, 2, 4, 1, 5, 2, 1, 2, 3, 4]
frames = 3

lfu(pages, frames)
```

Output:

```
Request page: 1
-> Fault
   Memory: [1]
   Freq:   {1: 1}

Request page: 2
-> Fault
   Memory: [1, 2]
   Freq:   {1: 1, 2: 1}

Request page: 3
-> Fault
   Memory: [1, 2, 3]
   Freq:   {1: 1, 2: 1, 3: 1}

Request page: 2
-> Hit
   Memory: [1, 2, 3]
   Freq:   {1: 1, 2: 2, 3: 1}

Request page: 4
-> Fault
   Replacing page: 1
   Memory: [4, 2, 3]
   Freq:   {2: 2, 3: 1, 4: 1}

Request page: 1
-> Fault
   Replacing page: 3
   Memory: [4, 2, 1]
   Freq:   {2: 2, 4: 1, 1: 1}

Request page: 5
-> Fault
   Replacing page: 4
   Memory: [5, 2, 1]
   Freq:   {2: 2, 1: 1, 5: 1}
```

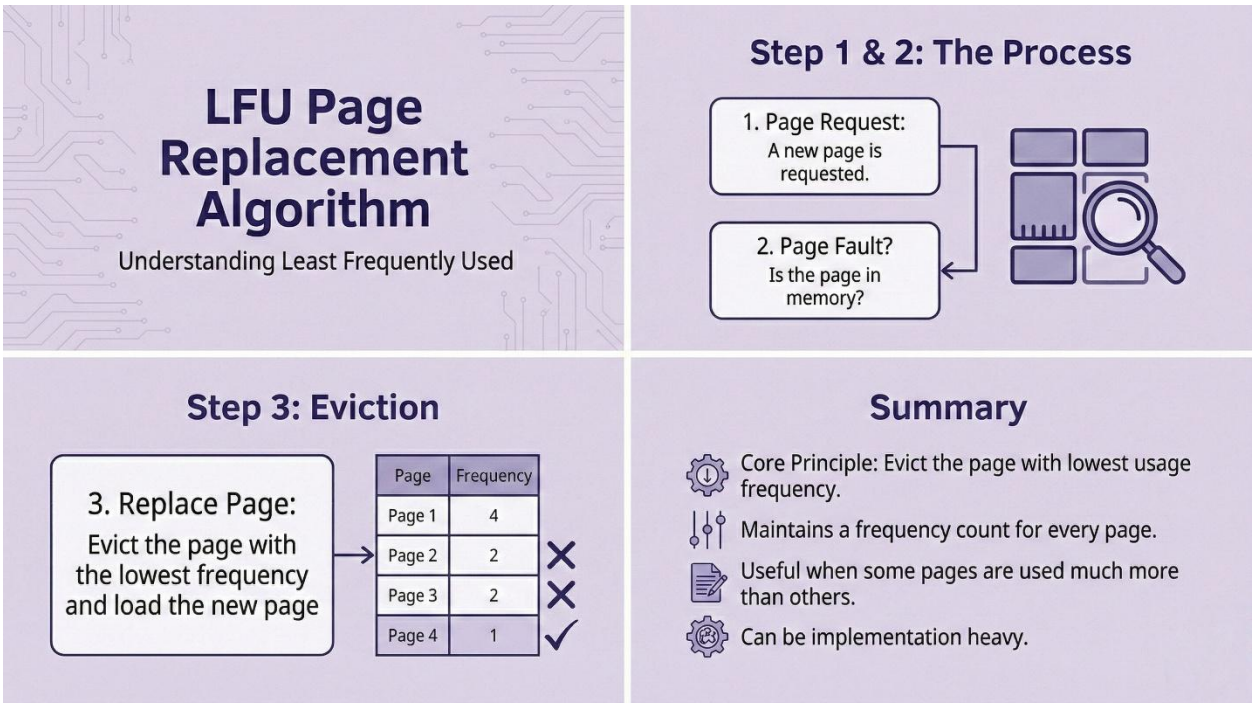
```
Request page: 2
-> Hit
   Memory: [5, 2, 1]
   Freq:   {2: 3, 1: 1, 5: 1}

Request page: 1
-> Hit
   Memory: [5, 2, 1]
   Freq:   {2: 3, 1: 2, 5: 1}

Request page: 2
-> Hit
   Memory: [5, 2, 1]
   Freq:   {2: 4, 1: 2, 5: 1}

Request page: 3
-> Fault
   Replacing page: 5
   Memory: [3, 2, 1]
   Freq:   {2: 4, 1: 2, 3: 1}

Request page: 4
-> Fault
   Replacing page: 3
   Memory: [4, 2, 1]
   Freq:   {2: 4, 1: 2, 4: 1}

Total Page Faults: 8
```

Flow diagram / explanation:



- When a page is accessed again → its **frequency increases**
- When a new page comes and cache is full → the page with **least frequency** is removed
- If frequencies are same → the **oldest page** is removed (FIFO rule)

Sample workflow created using Canva:



Part B – 24MCAT107-Advanced Software Engineering (Git Collaboration)

Repository Setup:

A GitHub repository named CacheSim_Project_Team_03 was created to manage collaborative development of the Cache Simulation Tool.

Branching Strategy:

Each team member worked on an individual branch. Mine, the LFU algorithm was implemented in the Anandhu (LFU) branch. Members committed changes frequently with meaningful messages.

Merging and Conflict Resolution:

The team merged all branches into the main branch after pulling the latest changes. Merge conflicts were resolved collaboratively to ensure code correctness.

Evidence Included

- Screenshot of git log
- Screenshot of git branch
- Screenshot of git merge
- Screenshot of git commits
- Repository contribution graph

Part C – Reflection (Individual Submission)

Through this project, I gained a clear understanding of how cache memory concepts can be implemented using software logic. Cache mechanisms such as replacement algorithms do not exist only in hardware; they can be simulated using data structures like arrays, counters, and conditional logic. For example, in algorithms like LFU (Least Frequently Used), frequency counters were used to track how often data items were accessed, and decisions were made programmatically to replace the least-used item. This helped me understand how theoretical concepts like cache hits, cache misses, and replacement policies work in real systems.

Git played an important role in collaborative coding and version management. By using Git and GitHub, our team members were able to work on separate branches without affecting each other's code. Git allowed us to track changes, revert mistakes, and merge code efficiently. This reduced conflicts and ensured that everyone's contributions were properly managed. It also taught me the importance of writing meaningful commit messages and following a structured workflow while working in a team.

During the project, we faced challenges such as merge conflicts, logical errors in implementation, and understanding how to divide tasks effectively among team members. Initially, handling Git branches was confusing, but regular practice improved my confidence. Overall, this project enhanced my technical skills, problem-solving ability, and teamwork, and gave me practical exposure to both system-level concepts and collaborative software development.