LEAST RECENTLY USED (LRU) - CACHE REPLACEMENT ALGORITHM

EXPLANATION

The Least Recently Used (LRU) page replacement algorithm is a widely used memory management technique in operating systems that determines which page should be removed from memory when a page fault occurs. In modern computer systems, the main memory is limited, and when it is full, new pages must replace existing pages. LRU addresses this problem by selecting the page that has not been used for the longest period of time, based on the principle that pages used recently are more likely to be accessed again in the near future.

LRU operates by keeping track of the usage history of pages currently stored in memory. When a page is accessed, the algorithm updates the usage record to reflect its recent access. If the page is already present in memory, known as a page hit, it can be accessed immediately, and its usage status is updated. If the page is not in memory, resulting in a page fault, the system checks for free space. If memory is full, the least recently used page is replaced with the new page.

There are several implementation methods for LRU. One approach uses counters, where each page maintains a counter that records the time of its last access, and the page with the smallest counter is replaced during a page fault. Another approach uses a stack or linked list, where pages are arranged based on their access order. When a page is accessed, it is moved to the top of the stack, and the page at the bottom, representing the least recently used, is selected for replacement when needed.

The LRU algorithm has several advantages. It effectively reduces page faults by retaining frequently accessed pages and approximates the optimal page replacement strategy without requiring knowledge of future accesses. However, it also has some overhead due to the need for tracking page usage and additional storage for counters or stack management.

For example, consider a memory system with three page frames and a reference string 7, 0, 1, 2, 0, 3, 0, 4. The LRU algorithm would load pages into memory, replacing the least recently used page when necessary, resulting in a total of six page faults. This demonstrates how LRU prioritizes keeping recently used pages in memory, improving overall system performance.

Overall, LRU is widely adopted in operating systems and CPU cache management due to its balance of efficiency and simplicity. By ensuring that the least recently accessed pages are replaced first, LRU enhances memory utilization and reduces the average memory access time.

ALGORITHM

```
Step 1: Start.
Step 2: Read the cache size.
Step 3: Read the memory reference sequence.
Step 4: Initialize an empty cache and an empty list or queue to track usage order.
Step 5: Initialize hit counter = 0 and miss counter = 0.
Step 6: For each memory block in the reference sequence, do:
    6.1: If the block is already present in the cache (HIT):
        a. Increment hit counter.
        b. Update the usage order to mark this block as most recently used.
    6.2: Else (MISS):
        a. Increment miss counter.
        b. If cache has free space, insert the block.
        c. Else, remove the least recently used (LRU) block and insert the new block.
Step 7: Display the cache content after each memory access.
Step 8: After all references, calculate hit ratio:
            $hit\_ratio = hits / (hits + misses)$
Step 9: Display total hits, total misses, and hit ratio.
Step 10: Stop.
```

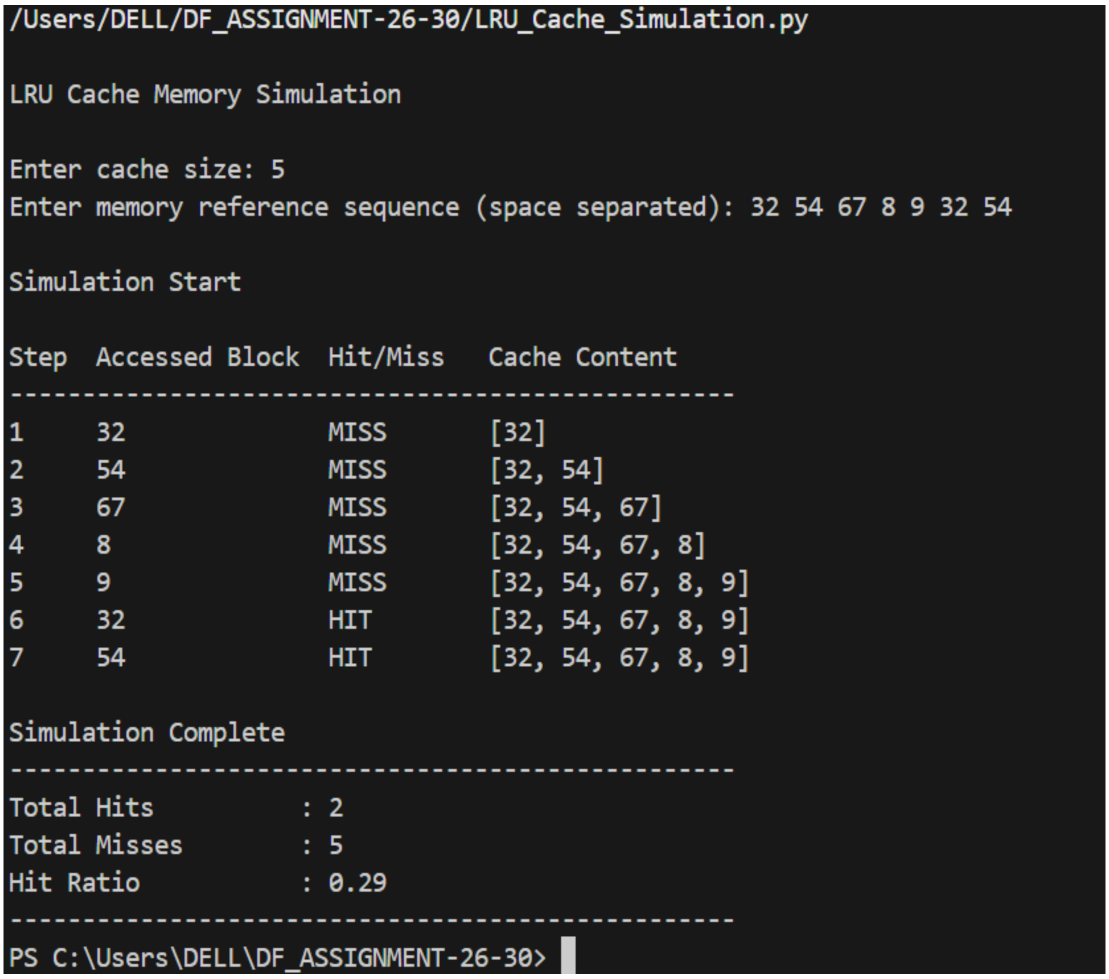
SAMPLE INPUT – OUTPUT RESULTS

Cache Size:4

Reference String: a b c a d c a e c

Step	**Memory Access**	**Hit / Miss**	**Action Taken**	**Cache Status**
1	a	Miss	Cache empty → Insert a	[a]
2	b	Miss	Insert b	[a, b]
3	c	Miss	Insert c	[a, b, c]
4	a	**Hit**	Block already in cache → update recent use	[b, c, a]
5	d	Miss	Insert d	[b, c, a, d]
6	c	**Hit**	Block already in cache → update recent use	[b, a, d, c]
7	a	**Hit**	Block already in cache → update recent use	[b, d, c, a]
8	e	Miss	Cache full → replace least recently used block (b)	[d, c, a, e]
9	c	**Hit**	Block already in cache → update recent use	[d, a, e, c]

PROGRAM OUTPUT



```
/Users/DELL/DF_ASSIGNMENT-26-30/LRU_Cache_Simulation.py

LRU Cache Memory Simulation

Enter cache size: 5
Enter memory reference sequence (space separated): 32 54 67 8 9 32 54

Simulation Start

Step  Accessed Block  Hit/Miss   Cache Content
--------------------------------------------------
1     32              MISS       [32]
2     54              MISS       [32, 54]
3     67              MISS       [32, 54, 67]
4     8               MISS       [32, 54, 67, 8]
5     9               MISS       [32, 54, 67, 8, 9]
6     32              HIT        [32, 54, 67, 8, 9]
7     54              HIT        [32, 54, 67, 8, 9]

Simulation Complete
--------------------------------------------------
Total Hits         : 2
Total Misses       : 5
Hit Ratio          : 0.29
--------------------------------------------------
PS C:\Users\DELL\DF_ASSIGNMENT-26-30>
```