Batch: E-2	Roll No.: 16010123325
Experiment / assignment / tutorial	

TITLE: Implementation of LRU Page Replacement Algorithm.

AIM: The LRU algorithm replaces the least recently used that is the last accessed memory block from user.

Expected OUTCOME of Experiment: (Mention CO/CO's attained here)

Books/ Journals/ Websites referred:

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.
2. William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.

Pre Lab/ Prior Concepts:

It follows a simple logic, while replacing it will replace that page which has least recently used out of all.

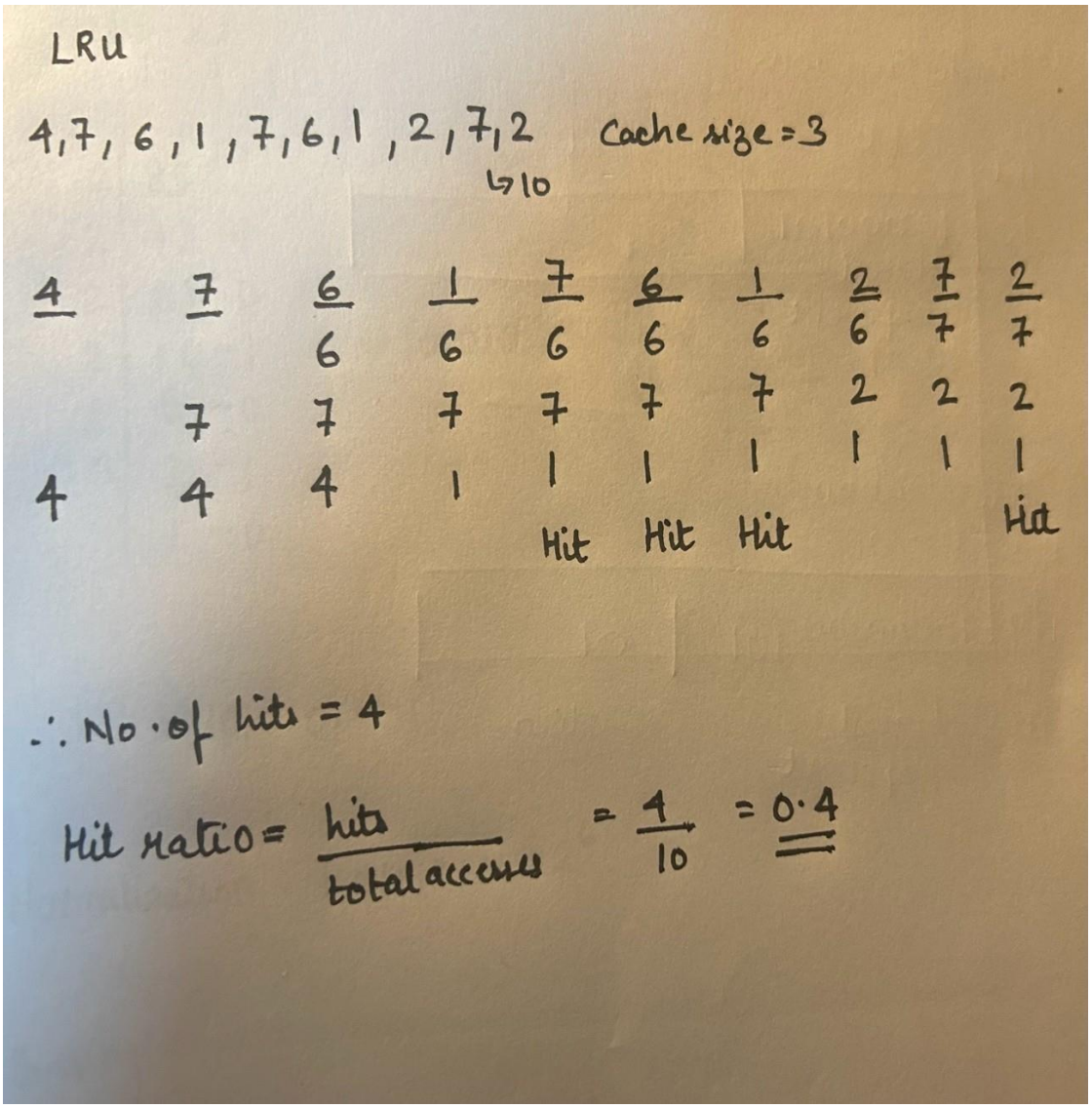
a) A hit is said to be occurred when a memory location requested is already in the cache.

b) When cache is not full, the number of blocks is added.
c) When cache is full, the block is replaced which is recently used

Algorithm:

1. Start
2. Get input as memory block to be added to cache
3. Consider an element of the array
4. If cache is not full, add element to the cache array
5. If cache is full, check if element is already present
6. If it is hit is incremented
7. If not, element is added to cache removing least recently used element
8. Repeat step 3 to 7 for remaining elements
9. Display the cache at very instance of step 8
10. Print hit ratio
11. End

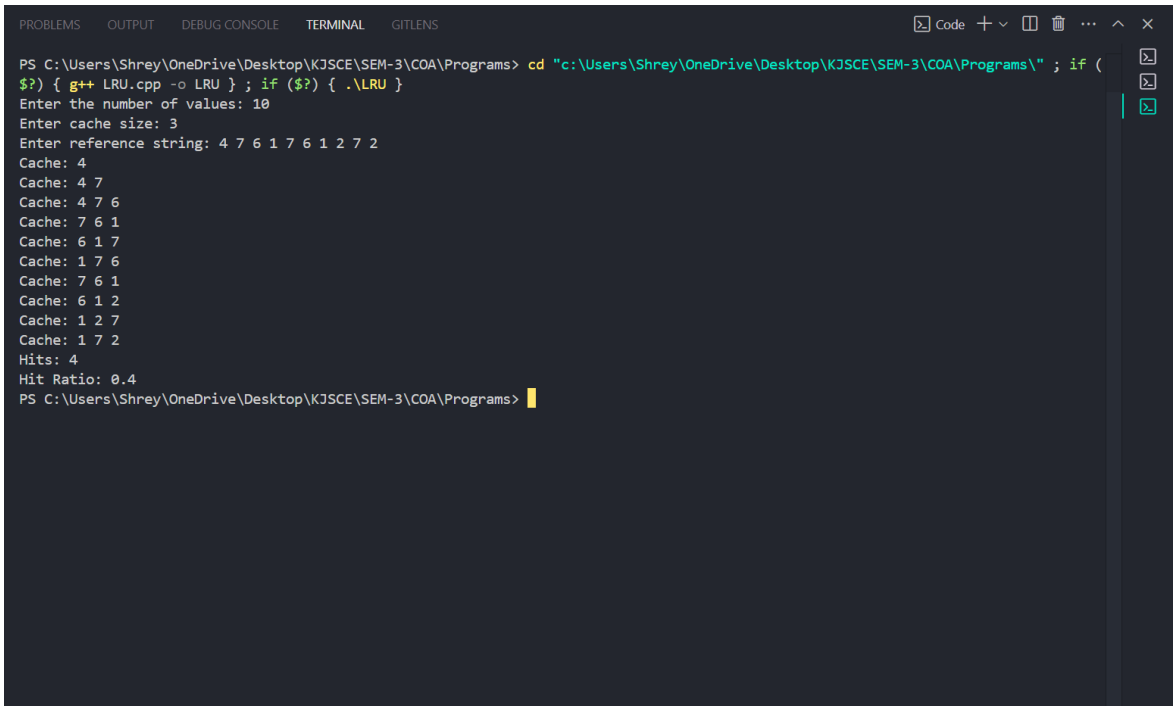
Example:


LRU
4, 7, 6, 1, 7, 6, 1, 2, 7, 2
Cache size = 3
Hit
Hit
Hit
Hit
∴ No. of hits = 4
Hit ratio = hits / total accesses = 4/10 = 0.4

Code-

```
#include<bits/stdc++.h>
using namespace std;

int main() {
    int n, cacheSize;
    cout << "Enter the number of values: ";
    cin >> n;

    cout << "Enter cache size: ";
    cin >> cacheSize;

    vector<int> pages(n);
    cout << "Enter reference string: ";
    for (int i = 0; i < n; i++) {
        cin >> pages[i];
    }

    deque<int> cache;
    int hit = 0;

    for (int page : pages) {
        if (find(cache.begin(), cache.end(), page) != cache.end()) {
            hit++;
            cache.erase(remove(cache.begin(), cache.end(), page),
cache.end());
        } else {
            if (cache.size() == cacheSize) {
                cache.pop_front();
            }
        }
        cache.push_back(page);
        cout << "Cache: ";
        for (int p : cache) {
            cout << p << " ";
        }
        cout << endl;
    }

    double hitRatio = (double)hit / n;
    cout << "Hits: " << hit << endl;
    cout << "Hit Ratio: " << hitRatio << endl;

    return 0;
}
```

Output-


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
GITLENS
Code
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\COA\Programs> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\COA\Programs\" ; if ($?) { g++ LRU.cpp -o LRU } ; if ($?) { .\LRU }
Enter the number of values: 10
Enter cache size: 3
Enter reference string: 4 7 6 1 7 6 1 2 7 2
Cache: 4
Cache: 4 7
Cache: 4 7 6
Cache: 7 6 1
Cache: 6 1 7
Cache: 1 7 6
Cache: 7 6 1
Cache: 6 1 2
Cache: 1 2 7
Cache: 1 7 2
Hits: 4
Hit Ratio: 0.4
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\COA\Programs>

Post Lab Descriptive Questions

1. Define hit rate and miss ratio?

Hit Rate

Definition: The hit rate is the percentage of memory accesses that successfully find the requested data in the cache. It is an indicator of cache efficiency.

Hit Rate = Number of Hits / Total Memory Accesses ×100 %

Miss Ratio

Definition: The miss ratio is the percentage of memory accesses that do not find the requested data in the cache, requiring retrieval from slower memory sources. It reflects the cache's shortcomings in serving data.

Miss Ratio= Number of Misses / Total Memory Accesses ×100 %

2. What is the need for virtual memory?

- **Extended Memory**: It allows the system to use more memory than physically available by using disk space as an extension of RAM.
- **Isolation**: Each process has its own virtual address space, enhancing security and preventing interference between processes.
- **Simplified Management**: It abstracts physical memory, allowing efficient memory allocation and management through techniques like paging.
- **Optimized Usage**: It keeps frequently accessed pages in RAM while less-used ones can be stored on disk, improving performance.
- **Multitasking**: Virtual memory enables multiple processes to run concurrently, allowing users to use several applications simultaneously.

Conclusion

In this experiment, we implemented the **Least Recently Used (LRU) caching algorithm** in Java to evaluate the efficiency of a cache system.

Date: _____