**Batch:** **C-2** **Roll No.:** **16010122267**

**Experiment / assignment / tutorial No. 6**

# 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

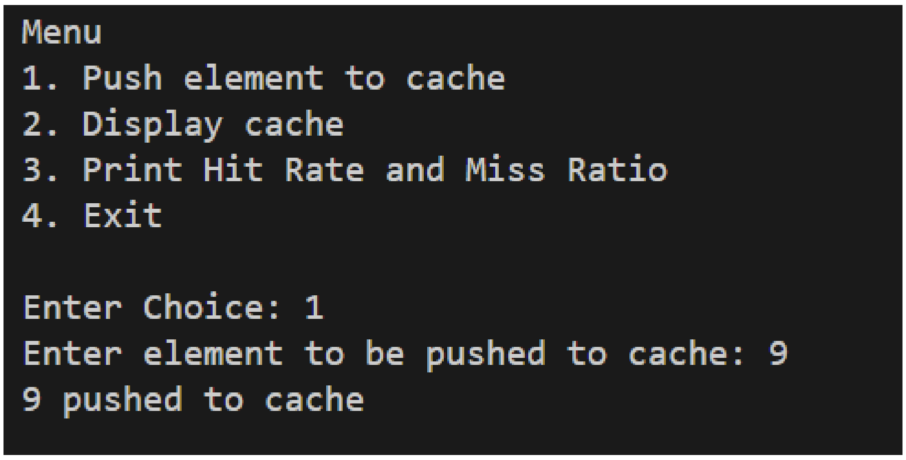
**Example:**

Code:

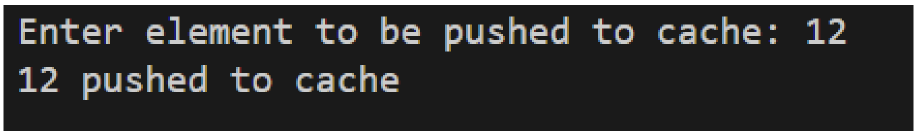
```
import java.util.*

public class LRUCache {
    public static boolean contains(List<int[]> arr, int n) {
        for (int[] element : arr) {
            if (element[0] == n) {
                return true;
            }
        }
        return false;
    }

    public static int getInRank(List<int[]> arr, int n) {
        for (int i = 0; i < arr.size(); i++) {
            if (arr.get(i)[1] == n) {
                return i;
            }
        }
        return -1;
    }

    public static int getInEle(List<int[]> arr, int n) {
        for (int i = 0; i < arr.size(); i++) {
            if (arr.get(i)[0] == n) {
                return i;
            }
        }
        return -1;
    }

    public static int push(List<int[]> arr, int n) {
        if (!contains(arr, n)) {
            int x = getInRank(arr, 3);
            arr.get(x)[0] = n;
            arr.get(x)[1] = 0;
            for (int i = 0; i < arr.size(); i++) {
                if (i != x) {
                    arr.get(i)[1]++;
                }
            }
            return 0;
        } else {
            int x = getInEle(arr, n);
            int l = arr.get(x)[1];
            arr.get(x)[1] = 0;
            for (int i = 0; i < arr.size(); i++) {
                if (i != x && arr.get(i)[1] <= l) {
                    arr.get(i)[1]++;
                }
            }
            return 1;
        }
    }

    public static void main(String[] args) {
        List<int[]> memory = new ArrayList<>();
        for (int i = 0; i < 4; i++) {
            memory.add(new int[]{-1, 3 - i});
        }
        int h = 0;
        int r = 0;
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nMenu");
            System.out.println("1. Push element to cache");
            System.out.println("2. Display cache");
            System.out.println("3. Print Hit Rate and Miss Ratio");
            System.out.println("4. Exit");
            System.out.print("\nEnter Choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter element to be pushed to 
cache: ");
                    int n = scanner.nextInt();
                    h += push(memory, n);
                    r++;
                    System.out.println(n + " pushed to cache");
                    break;
                case 2:
                    System.out.println("\nPF Age");
                    for (int[] element : memory) {
                        System.out.println(element[0] + " " +
element[1]);
                    }
                    break;
                case 3:
                    if (r == 0) {
                        System.out.println("\nEnter an element first");
                        break;
                    }
                    double hr = ((double) h / r) * 100;
                    double mr = 100 - hr;
                    System.out.println("\nNumber of Hits: " + h);
                    System.out.println("Number of References: " + r);
                    System.out.println("\nHit Rate: " + hr + "%");
                    System.out.println("Miss Ratio: " + mr + "%");
                    break;
                case 4:
                    System.out.println("Exiting Program \n");
                    scanner.close();
                    System.exit(0);
                default:
                    System.out.println("Invalid Choice");
                    break;
            }
        }
    }
}
```

Output:

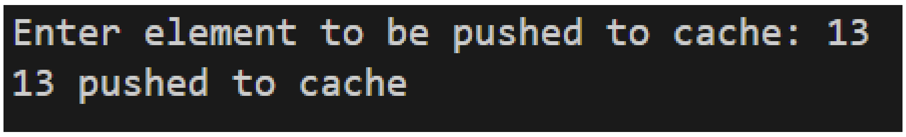
- **Adding elements 9 ,12, 13, 7 to the cache:**



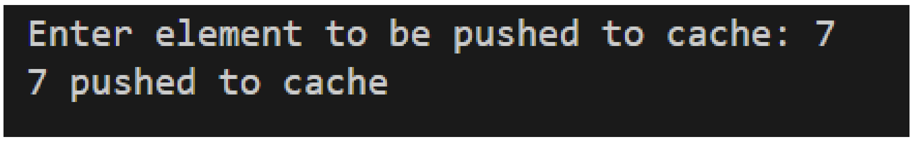
```
Menu
1. Push element to cache
2. Display cache
3. Print Hit Rate and Miss Ratio
4. Exit

Enter Choice: 1
Enter element to be pushed to cache: 9
9 pushed to cache
```



```
Enter element to be pushed to cache: 12
12 pushed to cache
```

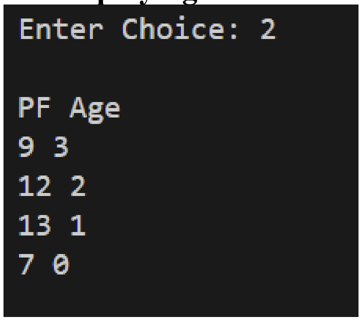


```
Enter element to be pushed to cache: 13
13 pushed to cache
```



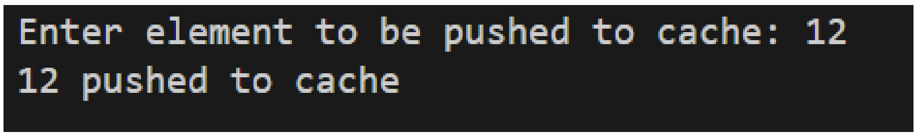
```
Enter element to be pushed to cache: 7
7 pushed to cache
```

- **Displaying Cache:**

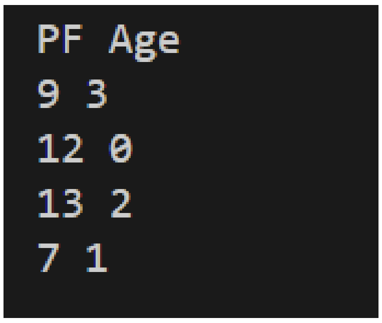


```
Enter Choice: 2

PF Age
9 3
12 2
13 1
7 0
```

- **Adding page 12, as it already exists in the cache, age the page 12 becomes 0 and the ages of rest of the pages increase by 1. (Hit: 1)**

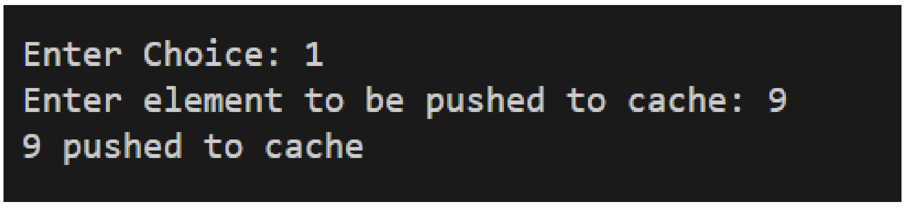


```
Enter element to be pushed to cache: 12
12 pushed to cache
```

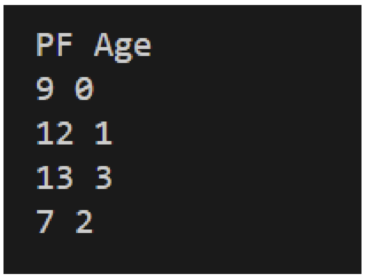


```
PF Age
9 3
12 0
13 2
7 1
```

- **Adding page 9, as it already exists in the cache, age the page 9 becomes 0 and the ages of rest of the pages increase by 1. (Hit: 2)**

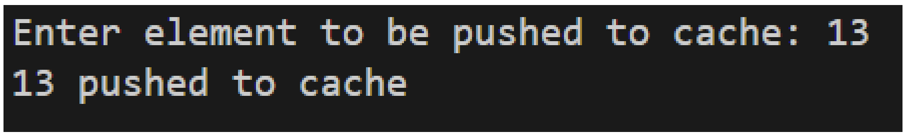


```
Enter Choice: 1
Enter element to be pushed to cache: 9
9 pushed to cache
```

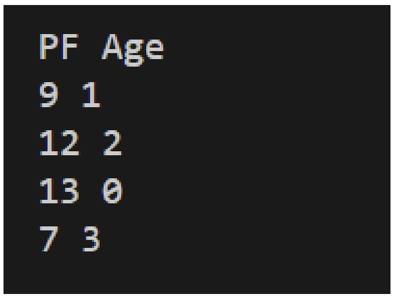


```
PF Age
9 0
12 1
13 3
7 2
```

- **Adding page 13, as it already exists in the cache, age the page 13 becomes 0 and the ages of rest of the pages increase by 1. (Hit: 3)**

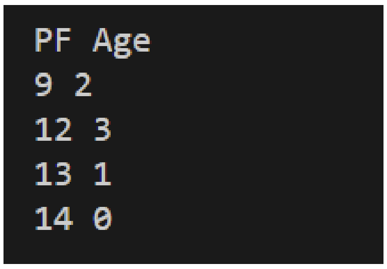


```
Enter element to be pushed to cache: 13
13 pushed to cache
```



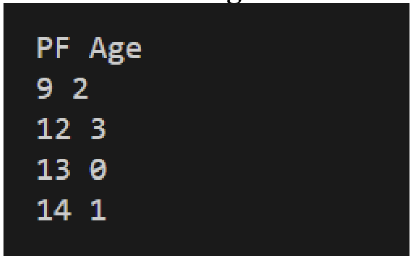
```
PF Age
9 1
12 2
13 0
7 3
```

- **Adding new element 14 to the cache:**



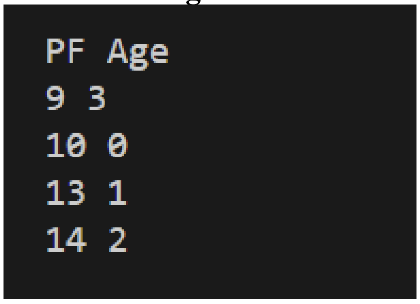
```
PF Age
9 2
12 3
13 1
14 0
```

- **Adding page 13, as it already exists in the cache, age the page 13 becomes 0 and the ages of rest of the pages increase by 1. (Hit: 4)**



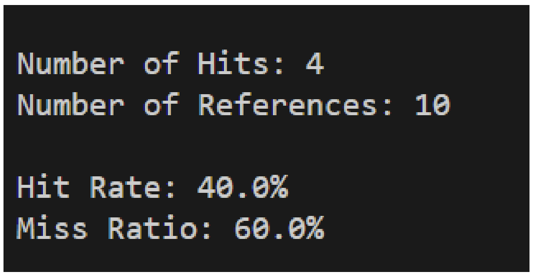
```
PF Age
9 2
12 3
13 0
14 1
```

- **Adding new element 10 to the cache:**



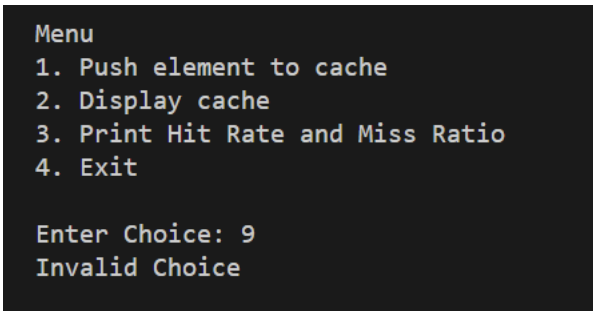
```
PF Age
9 3
10 0
13 1
14 2
```

- **Hit rate and Miss Ratio:**



```
Number of Hits: 4
Number of References: 10

Hit Rate: 40.0%
Miss Ratio: 60.0%
```

- **Invalid choice:**



```
Menu
1. Push element to cache
2. Display cache
3. Print Hit Rate and Miss Ratio
4. Exit

Enter Choice: 9
Invalid Choice
```

## <u>Post Lab Descriptive Questions</u>

**1. Define hit rate and miss ratio?**

**Ans:**

Hit rate and miss ratio are terms commonly used in the context of cache memory systems, especially in computer architecture and computer systems. They help evaluate the effectiveness of a cache in reducing memory access latency and improving overall system performance.

**Hit Rate:**

- The hit rate, also known as the cache hit rate, is a metric that measures the percentage of memory accesses that are successfully serviced by the cache without having to access the main memory or a lower-level cache.
- It is usually expressed as a percentage and can be calculated using the following formula:
Hit Rate (%) = (Number of Cache Hits / Total Number of Memory Accesses) * 100
- A high hit rate indicates that the cache is doing a good job of storing frequently used data and reducing the number of accesses to slower memory levels (such as RAM or disk).

**Miss Ratio:**

- The miss ratio, also known as the cache miss ratio or cache miss rate, is a metric that measures the percentage of memory accesses that result in cache misses. In other words, it represents the proportion of memory accesses that cannot be serviced by the cache and require accessing a slower memory level.
- Like the hit rate, the miss ratio is also expressed as a percentage and can be calculated using the following formula:
Miss Ratio (%) = (Number of Cache Misses / Total Number of Memory Accesses) * 100
- A lower miss ratio is desirable because it indicates that the cache is effective in storing the most frequently used data and minimizing the need to access slower memory.

In summary, hit rate and miss ratio are essential performance metrics for evaluating the efficiency of a cache memory system. A high hit rate and a low miss ratio are indicative of a well-designed cache that effectively reduces memory access latency and improves system performance.

2. **What is the need for virtual memory**?

Ans:

1. **Expanded Address Space:** Virtual memory allows programs to access more memory than physically available.
2. **Isolation and Protection:** It separates processes to prevent interference and data corruption.
3. **Efficient Multitasking:** Enables seamless switching between multiple running programs.
4. **Memory Management:** Simplifies allocation and deallocation of memory by the

operating system.

5. Resource Overcommitment: Lets the OS allocate more memory than physically available.
6. **Optimized Physical Memory Use:** Improves performance by storing frequently used data in RAM.
7. **Simplified Program Development:** Programmers can develop without worrying about physical memory constraints.
8. **Dynamic Memory Allocation:** Allows programs to request and release memory as needed.
9. **Enhanced Reliability:** Isolates faulty programs, reducing the risk of system-wide crashes.

**Conclusion:** Through this experiment, we implemented LRU Page Replacement Algorithm using Java, and also displayed its real-life application. LRU proves its effectiveness in reducing page faults by giving preference to frequently accessed pages, thereby improving the overall performance of the system.

**Date:** ______________