**Experiment no – 07**

**Aim**:

Write a program that implements the FIFO and LRU page-replacement algorithms. First, generate a random page reference string where page numbers range from 0 to 9. Apply the random page-reference string to each algorithm, and record the number of page faults incurred by each algorithm. Implement the replacement algorithms so that the number of page frames can vary as well. Assume that demand paging is used. Design and implement two classes—LRU and FIFO—that extend Replacement Algorithm. Each of these classes will implement the insert () method, one class using the LRU page-replacement algorithm and the other using the FIFO algorithm. Test your algorithm with suitable Java programs.

**Theory:**

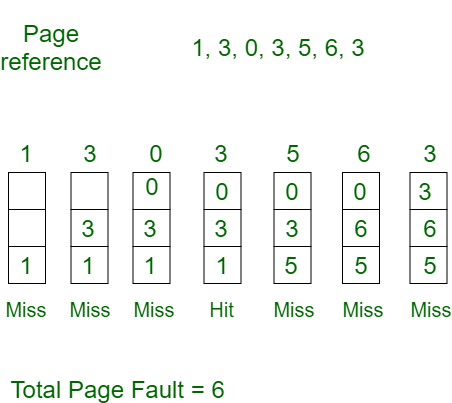
In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

**Page Fault –** A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

**Page Replacement Algorithms :**

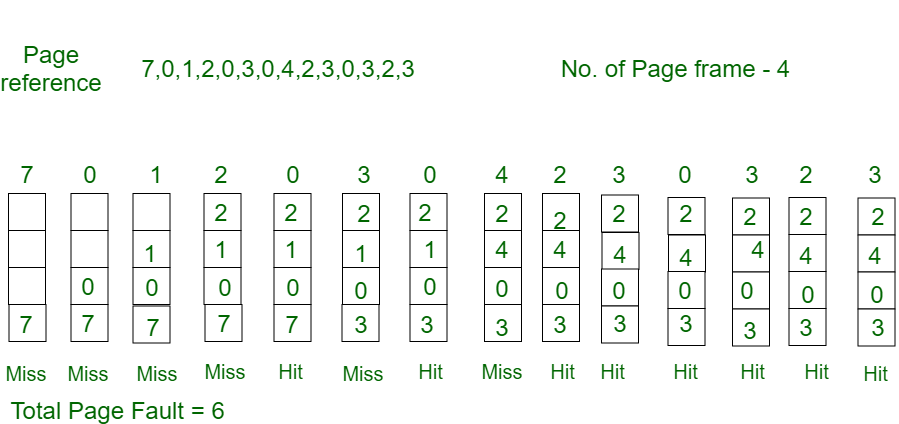
**1. First In First Out (FIFO) –**   
This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.   
**Example-1**Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames.Find the number of page faults.



Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**   
when 3 comes, it is already in memory so —> **0 Page Faults.**   
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**   
6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**   
Finally, when 3 come it is not available so it replaces 0 **1 page fault**

[**Belady’s anomaly**](https://www.geeksforgeeks.org/operating-system-beladys-anomaly/)**–** Belady’s anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm.  For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults.

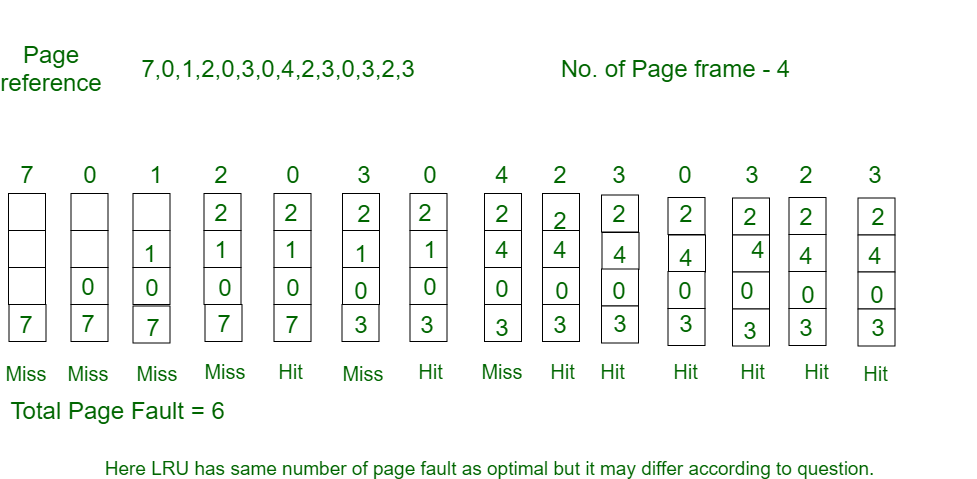
**2. Optimal Page replacement –**   
In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.   
**Example-2:**Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.



Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>**4 Page faults**   
0 is already there so —> **0 Page fault.**   
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**   
0 is already there so —>**0 Page fault.**.   
4 will takes place of 1 —>**1 Page Fault.**

Now for the further page reference string —>**0 Page fault** because they are already available in the memory.   
Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

**3. Least Recently Used –**   
In this algorithm, page will be replaced which is least recently used.   
**Example-3**Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.



Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>**4 Page faults**   
0 is already their so —> **0 Page fault.**   
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**   
0 is already in memory so —>**0 Page fault**.   
4 will takes place of 1 —>**1 Page Fault**   
Now for the further page reference string —>**0 Page fault** because they are already available in the memory.

**Code:**

FIFO.java

package prac7;

import java.io.\*;

public class FIFO {

public static void main(String[] args) throws IOException {

{

System.out.println("Yash Prajapati - 022 ");

int n = 0;

int fifo[] = new int[5];

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter no of inputs : ");

n = Integer.parseInt(br.readLine());

int inp[] = new int[5];

System.out.println("Enter the inputs : ");

for (int i = 0; i < 5; i++) {

inp[i] = Integer.parseInt(br.readLine());

}

for (int i = 0; i < 5; i++) {

fifo[i] = -1;

}

int hit = 0;

int fault = 0;

int j = 0;

boolean check;

for (int i = 0; i < 5; i++) {

check = false;

for (int k = 0; k < 5; k++) {

if (fifo[k] == inp[i]) {

check = true;

hit = hit + 1;

}

}

if (check == false) {

fifo[j] = inp[i];

j++;

if (j >= 4) {

j = 0;

}

fault = fault + 1;

}

}

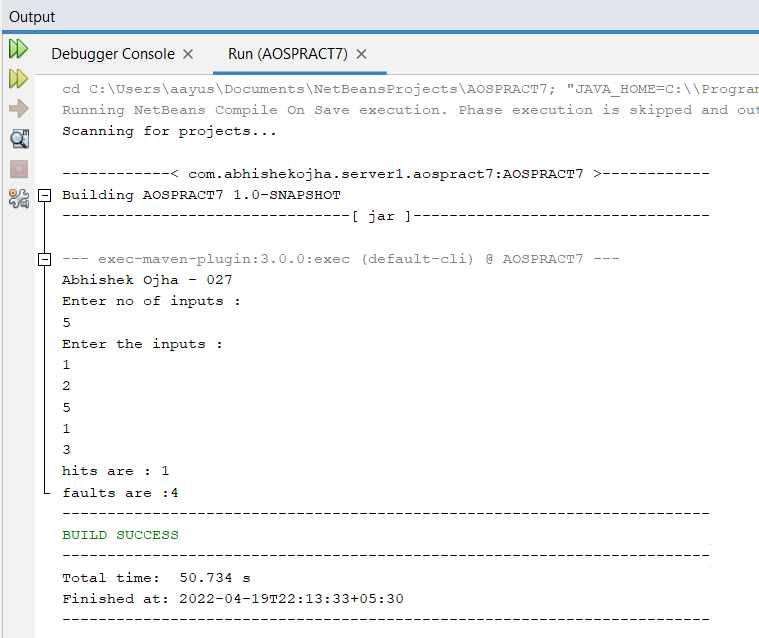
System.out.println("hits are : " + hit + "\nfaults are :" + fault);

}

}

}

**Output:**



LRU.java

package prac7;

import java.io.\*;

public class LRU {

public static int sort(int c[]) {

int max = -1;

int temp = -1;

for (int k = 0; k < 3; k++) {

if (c[k] > max) {

max = c[k];

temp = k;

}

}

return temp;

}

public static void main(String[] args) throws IOException {

System.out.println("Yash Prajapati - 022 ");

int z, m = 0, hit = 0, fault = 0;

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("enter the size of the array");

int n = Integer.parseInt(br.readLine());

int a[] = new int[n];

int flag[] = new int[n];

System.out.println("enter the elements :");

for (int i = 0; i < n; i++) {

a[i] = Integer.parseInt(br.readLine());

flag[i] = 0;

}

int b[] = new int[3];

int c[] = new int[3];

for (int i = 0; i < 3; i++) {

b[i] = -1;

c[i] = 0;

}

for (int i = 0; i < n; i++) {

z = a[i];

for (int j = 0; j < 3; j++) {

if (z == b[j]) {

flag[i] = 1;

hit = hit + 1;

break;

}

}

if (flag[i] == 0 && b[2] == -1) {

for (int j = 0; j < 3; j++) {

if (b[j] == -1) {

b[j] = z;

fault = fault + 1;

flag[i] = 1;

break;

}

}

}

if (flag[i] == 0) {

m = sort(c);

b[m] = z;

fault = fault + 1;

flag[i] = 1;

for (int k = 0; k < 3; k++) {

c[k] = c[k] + 1;

}

c[m] = 0;

}

}

System.out.println(" no of hits : " + hit);

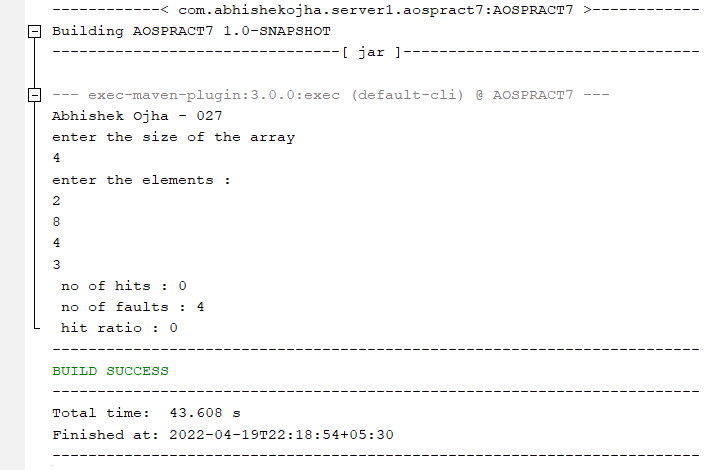
System.out.println(" no of faults : " + fault);

System.out.println(" hit ratio : " + (hit \* 100) / (hit + fault));

}

}

**Output:**



Conclusion:

Successfully demonstrated page replacement algorithm FIFO and LRU.