TECHNO INTERNATIONAL
BATANAGAR

# Term Paper On Optimal Page Replacement Algorithm

| | |
|---|---|
| Name | : Soumyadip Mondal |
| Roll No | : 33200118011 |
| Sem | : 5th (3rd Year) |
| Dept. | : C.S.E. |
| Subject | : Operating System |
| Sub. Code | : PCC – CS – 502 |

# Page Replacement Algorithms in Operating Systems:

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 new page comes in. The Optimal Page Replacement Algorithm The best possible page replacement algorithm is easy to describe but impossible to implement. It goes like this. At the moment that a page fault occurs, some set of pages is in memory. One of these pages will be referenced on the very next instruction (the page containing that instruction). Other pages may not be referenced until 10, 100, or perhaps 1000 instructions later. Each page can be labeled with the number of instructions that will be executed before that page is first referenced. The optimal page algorithm simply says that the page with the highest label should be removed. If one page will not be used for 8 million instructions and another page will not be used for 6 million instructions, removing the former pushes the page fault that will fetch it back as far into the future as possible.

The theoretically optimal page replacement algorithm (also known as OPT, clairvoyant replacement algorithm, or Bélády's optimal page replacement policy) is an algorithm that works as follows: when a page needs to be swapped in, the operating system swaps out the page whose next use will occur farthest in the future. For example, a page that is not going to be used for the next 6 seconds will be swapped out over a page that is going to be used within the next 0.4 seconds.

This algorithm cannot be implemented in a general purpose operating system because it is impossible to compute reliably how long it will be before a page is going to be used, except when all software that will run on a system is either known beforehand and is amenable to static analysis of its memory reference patterns, or only a class of applications allowing run-time analysis. Despite this limitation, algorithms exist that can offer near-optimal performance  the operating system keeps track of all pages referenced by the program, and it uses those data to decide which pages to swap in and out on subsequent runs. This algorithm can offer near-optimal performance, but not on the first run of a program, and only if the program's memory reference pattern is relatively consistent each time it runs.

## 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 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.

Each operating system uses different page replacement algorithms. To select the particular algorithm, the algorithm with lowest page fault rate is considered.

1.Optimal page replacement algorithm

2.Not recently used page replacement

3.First-In, First-Out page replacement

4.Second chance page replacement

5.Clock page replacement

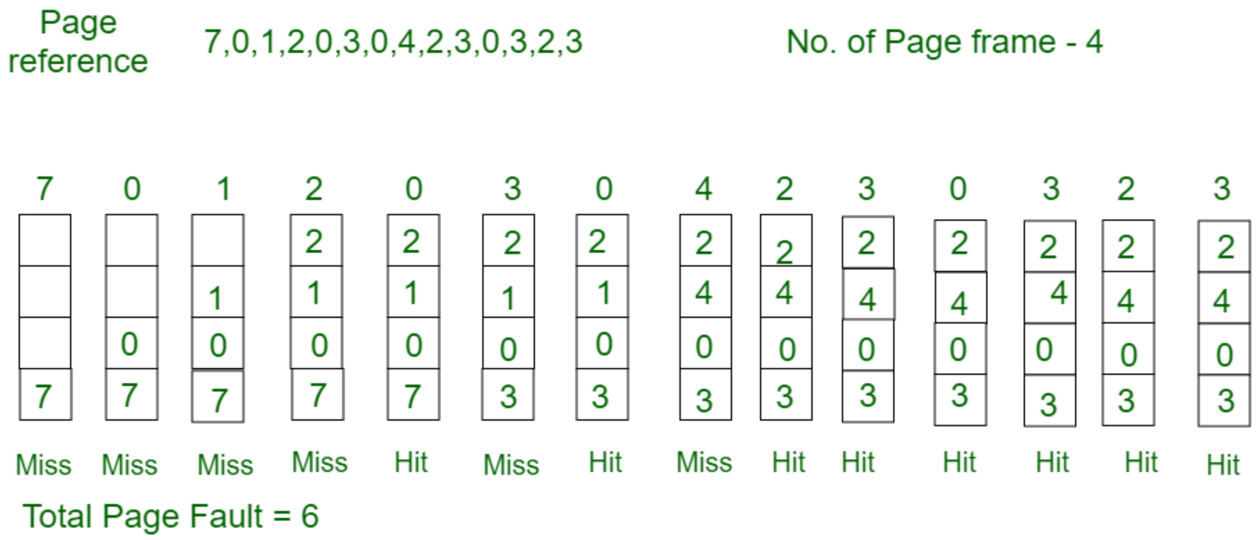
6.Least recently used page replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

## Example:

## Problem-01:

**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.**



Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|  |  | 1 | 1 | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 7 | 7 | 7 | 7 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Miss | Miss | Miss | Miss | Hit | Miss | Hit | Miss | Hit | Hit | Hit | Hit | Hit | Hit |

Total Page Fault = 6

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.

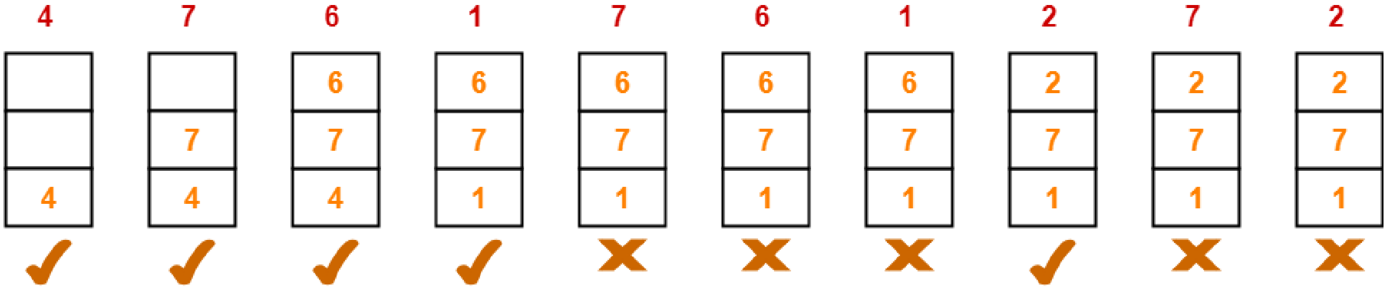
## Problem-02:

**A system uses 3 page frames for storing process pages in main memory. It uses the Optimal page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-**

**4 , 7, 6, 1, 7, 6, 1, 2, 7, 2**

**Also calculate the hit ratio and miss ratio.**

Total number of references = 10



| 4 | 7 | 6 | 1 | 7 | 6 | 1 | 2 | 7 | 2 |
|---|---|---|---|---|---|---|---|---|---|
|  |  | 6 | 6 | 6 | 6 | 6 | 2 | 2 | 2 |
|  | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |

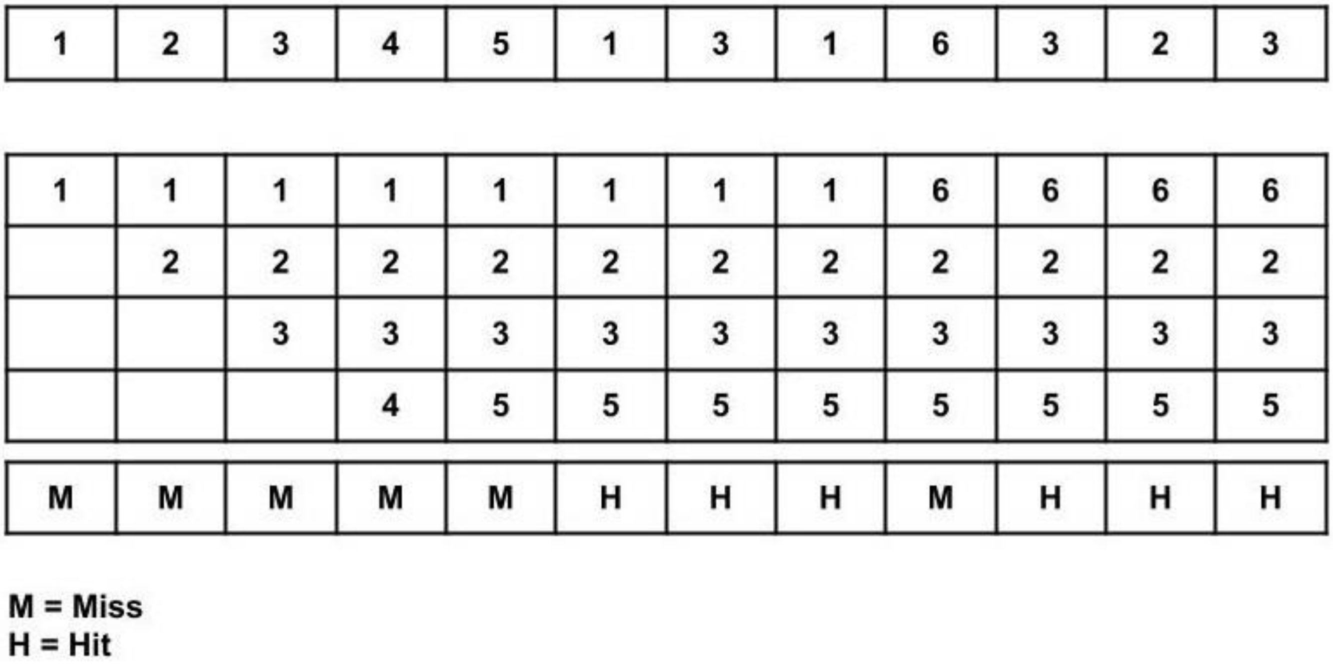
From here,

Total number of page faults occurred = 5

In the similar manner as above-

- Hit ratio = 0.5 or 50%
- Miss ratio = 0.5 or 50%.

## Problem-03:



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 1 | 3 | 1 | 6 | 3 | 2 | 3 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 |
| | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| | | | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| M | M | M | M | M | H | H | H | M | H | H | H |

M = Miss
H = Hit

Total Page Fault = 6

Initially, all 4 slots are empty, so when 1, 2, 3, 4 came they are allocated to the empty slots in order of their arrival. This is page fault as 1, 2, 3, 4 are not available in memory.

When 5 comes, it is not available in memory so page fault occurs and it replaces 4 which is going to be used farthest in the future among 1, 2, 3, 4.

When 1,3,1 comes, they are available in the memory, i.e., Page Hit, so no replacement occurs.

When 6 comes, it is not available in memory so page fault occurs and it replaces 1.

When 3, 2, 3 comes, it is available in the memory, i.e., Page Hit, so no replacement occurs.

Page Fault ratio = 6/12.