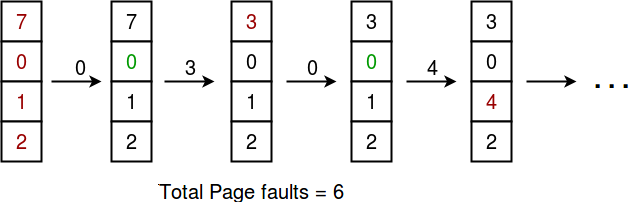
**Lab no 11**

**Simulate Page Replacement Algorithm**

***Objectives:***

* What is page replacement in operating system?
* Implementation of FIFO, LRU and optimal page replacement.

In operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.  
In **L**east **R**ecently **U**sed (LRU) algorithm is a Greedy algorithm where the page to be replaced is least recently used. The idea is based on locality of reference, the least recently used page is not likely   
Let say the page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 . Initially we have 4 page slots empty.   
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.



**Given memory capacity (as number of pages it can hold) and a string representing pages to be referred, write a function to find number of page faults.**

**Algorithm:**

Let **capacity** be the number of pages that

memory can hold. Let **set** be the current

set of pages in memory.

1- Start traversing the pages.

i) **If set holds less pages than capacity.**

a) Insert page into the set one by one until

the size of **set** reaches **capacity** or all

page requests are processed.

b) Simultaneously maintain the recent occurred

index of each page in a map called **indexes**.

c) Increment page fault

ii) **Else**

**If** current page is present in **set**, do nothing.

**Else**

a) Find the page in the set that was least

recently used. We find it using index array.

We basically need to replace the page with

minimum index.

b) Replace the found page with current page.

c) Increment page faults.

d) Update index of current page.

2. Return page faults.

**Implementations:**

# Python3 implementation of FIFO page

# replacement in Operating Systems.

from queue import Queue

# Function to find page faults using FIFO

def pageFaults(incomingStream, n, frames):

print("Incoming \t pages")

# Using Hashset to quickly check if a given

# incoming stream item in set or not

s = set()

# Queue created to store pages in FIFO manner

# since set will not store order or entry

# we will use queue to note order of entry of incoming page

queue = Queue()

page\_faults = 0

for i in range(n):

# if set has lesser item than frames

# i.e. set can hold more items

if len(s) < frames:

# If incoming item is not present, add to set

if incomingStream[i] not in s:

s.add(incomingStream[i])

# increment page fault

page\_faults += 1

# Push the incoming page into the queue

queue.put(incomingStream[i])

# If the set is full then we need to do page replacement

# in FIFO manner that is remove first item from both

# set and queue then insert incoming page

else:

# If incoming item is not present

if incomingStream[i] not in s:

# remove the first page from the queue

val = queue.queue[0]

queue.get()

# Remove from set

s.remove(val)

# insert incoming page to set

s.add(incomingStream[i])

# push incoming page to queue

queue.put(incomingStream[i])

# Increment page faults

page\_faults += 1

print(incomingStream[i], end="\t\t")

for q\_item in queue.queue:

print(q\_item, end="\t")

print()

return page\_faults

# Driver code

incomingStream = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1]

n = len(incomingStream)

frames = 6

page\_faults = pageFaults(incomingStream, n, frames)

hits = n - page\_faults

print("\nPage Faults: " + str(page\_faults))

print("Hit: " + str(hits))

**Task:**

* Create code for LRU and Optimal

# Python3 program for page replacement algorithm

# Driver code

capacity = 4

processList = [ 7, 0, 1, 2, 0, 3, 0,

                4, 2, 3, 0, 3, 2]

# List of current pages in Main Memory

s = []

pageFaults = 0

# pageHits = 0

for i in processList:

    # If i is not present in currentPages list

    if i not in s:

        # Check if the list can hold equal pages

        if(len(s) == capacity):

            s.remove(s[0])

            s.append(i)

        else:

            s.append(i)

        # Increment Page faults

        pageFaults +=1

    # If page is already there in

    # currentPages i.e in Main

    else:

        # Remove previous index of current page

        s.remove(i)

        # Now append it, at last index

        s.append(i)

print("{}".format(pageFaults))

Ouput:

