**FIFO**

import java.util.HashSet;

import java.util.LinkedList;

import java.util.Queue;

class Main

{

// Method to find page faults using FIFO

static int pageFaults(int pages[], int n, int capacity)

{

// To represent set of current pages. We use

// an unordered\_set so that we quickly check

// if a page is present in set or not

HashSet<Integer> s = new HashSet<>(capacity);

// To store the pages in FIFO manner

Queue<Integer> indexes = new LinkedList<>() ;

// Start from initial page

int page\_faults = 0;

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

{

// Check if the set can hold more pages

if (s.size() < capacity)

{

// Insert it into set if not present

// already which represents page fault

if (!s.contains(pages[i]))

{

s.add(pages[i]);

// increment page fault

page\_faults++;

// Push the current page into the queue

indexes.add(pages[i]);

}

}

// If the set is full then need to perform FIFO

// i.e. remove the first page of the queue from

// set and queue both and insert the current page

else

{

// Check if current page is not already

// present in the set

if (!s.contains(pages[i]))

{

//Pop the first page from the queue

int val = indexes.peek();

indexes.poll();

// Remove the indexes page

s.remove(val);

// insert the current page

s.add(pages[i]);

// push the current page into

// the queue

indexes.add(pages[i]);

// Increment page faults

page\_faults++;

}

}

}

return page\_faults;

}

// Driver method

public static void main(String args[])

{

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,

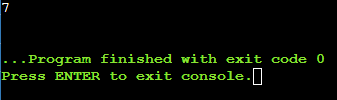
2, 3, 0, 3, 2};

int capacity = 4;

System.out.println(pageFaults(pages, pages.length, capacity));

}

}



**LRU**

import java.util.HashMap;

import java.util.HashSet;

import java.util.Iterator;

class Main

{

// Method to find page faults using indexes

static int pageFaults(int pages[], int n, int capacity)

{

// To represent set of current pages. We use

// an unordered\_set so that we quickly check

// if a page is present in set or not

HashSet<Integer> s = new HashSet<>(capacity);

// To store least recently used indexes

// of pages.

HashMap<Integer, Integer> indexes = new HashMap<>();

// Start from initial page

int page\_faults = 0;

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

{

// Check if the set can hold more pages

if (s.size() < capacity)

{

// Insert it into set if not present

// already which represents page fault

if (!s.contains(pages[i]))

{

s.add(pages[i]);

// increment page fault

page\_faults++;

}

// Store the recently used index of

// each page

indexes.put(pages[i], i);

}

// If the set is full then need to perform lru

// i.e. remove the least recently used page

// and insert the current page

else

{

// Check if current page is not already

// present in the set

if (!s.contains(pages[i]))

{

// Find the least recently used pages

// that is present in the set

int lru = Integer.MAX\_VALUE, val=Integer.MIN\_VALUE;

Iterator<Integer> itr = s.iterator();

while (itr.hasNext()) {

int temp = itr.next();

if (indexes.get(temp) < lru)

{

lru = indexes.get(temp);

val = temp;

}

}

// Remove the indexes page

s.remove(val);

//remove lru from hashmap

indexes.remove(val);

// insert the current page

s.add(pages[i]);

// Increment page faults

page\_faults++;

}

// Update the current page index

indexes.put(pages[i], i);

}

}

return page\_faults;

}

// Driver method

public static void main(String args[])

{

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

int capacity = 4;

System.out.println(pageFaults(pages, pages.length, capacity));

}

}

