**Experiment No:10**

**Problem Statement: Write a program to calculate the number of page faults for a reference string for the following page replacement algorithms:  
a) FIFO b) LRU c) Optimal**

**NAME: Omkar Sunil Khanvilkar ROLLNO: 7**

**CLASS: IT-B BATCH: 2**

**DATE OF PERFORMANCE: 10/10/2024**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Code:

package Programs.OS;  
  
import java.util.\*;  
  
public class PageReplacement {  
 // Function to calculate page faults using FIFO algorithm  
 public static int fifo(int[] pages, int capacity) {  
 Set<Integer> memory = new HashSet<>();  
 Queue<Integer> fifoQueue = new LinkedList<>();  
 int pageFaults = 0;  
  
 for (int page : pages) {  
 if (!memory.contains(page)) {  
 if (memory.size() == capacity) {  
 int oldestPage = fifoQueue.poll();  
 memory.remove(oldestPage);  
 }  
 memory.add(page);  
 fifoQueue.add(page);  
 pageFaults++;  
 }  
 }  
 return pageFaults;  
 }  
  
 // Function to calculate page faults using LRU algorithm  
 public static int lru(int[] pages, int capacity) {  
 Set<Integer> memory = new HashSet<>();  
 Map<Integer, Integer> lruMap = new HashMap<>();  
 int pageFaults = 0, time = 0;  
  
 for (int page : pages) {  
 if (!memory.contains(page)) {  
 if (memory.size() == capacity) {  
 int lruPage = Collections.*min*(lruMap.entrySet(), Map.Entry.*comparingByValue*()).getKey();  
 memory.remove(lruPage);  
 lruMap.remove(lruPage);  
 }  
 memory.add(page);  
 pageFaults++;  
 }  
 lruMap.put(page, time++);  
 }  
 return pageFaults;  
 }  
  
 // Function to calculate page faults using Optimal algorithm  
 public static int optimal(int[] pages, int capacity) {  
 Set<Integer> memory = new HashSet<>();  
 int pageFaults = 0;  
  
 for (int i = 0; i < pages.length; i++) {  
 int page = pages[i];  
 if (!memory.contains(page)) {  
 if (memory.size() == capacity) {  
 int farthest = i, farthestPage = -1;  
  
 for (int memPage : memory) {  
 int nextUse = Integer.MAX\_VALUE;  
 for (int j = i + 1; j < pages.length; j++) {  
 if (pages[j] == memPage) {  
 nextUse = j;  
 break;  
 }  
 }  
 if (nextUse > farthest) {  
 farthest = nextUse;  
 farthestPage = memPage;  
 }  
 }  
 memory.remove(farthestPage);  
 }  
 memory.add(page);  
 pageFaults++;  
 }  
 }  
 return pageFaults;  
 }  
  
 // Main function to run the program  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
  
 // Getting user input for the number of pages and reference string  
 System.out.println("Enter the number of pages in the reference string:");  
 int n = scanner.nextInt();  
 int[] pages = new int[n];  
  
 System.out.println("Enter the reference string (space-separated page numbers):");  
 for (int i = 0; i < n; i++) {  
 pages[i] = scanner.nextInt();  
 }  
  
 // Getting user input for the capacity of page frames  
 System.out.println("Enter the page frame capacity:");  
 int capacity = scanner.nextInt();  
  
 // Calculating and printing page faults for each algorithm  
 System.out.println("FIFO Page Faults: " + *fifo*(pages, capacity));  
 System.out.println("LRU Page Faults: " + *lru*(pages, capacity));  
 System.out.println("Optimal Page Faults: " + *optimal*(pages, capacity));  
  
 scanner.close();  
 }  
}

# Output:

