**Name : Rohit Mahadev Mane Roll No : CO313**

**Class : TE COMP**

# Assignment No:6

**Aim:**Write a java program to implement paging simulation using 1.Least Recently Used(LRU)

2.Optimal Algorithm

**PageReplacement:**

import java.util.Scanner;

public class PageReplacement { public static void main(String[] args) {

LRU lru=new LRU();

Optimal optimal=new Optimal(); int i;

do{

System.out.println("1:LRU/n2.Optimal"); Scanner sc=new Scanner(System.in); i=sc.nextInt();

switch(i){

case 1:lru.execute();

break; case 2:optimal.execute();

break;

default :System.out.println("Wrong Choice...");

}

}while(i<3);

}

}

## LRU:

import java.util.HashMap; import java.util.HashSet; import java.util.Iterator; import java.util.Scanner;

public class LRU { private Scanner sc;

public void execute()

{

sc = new Scanner(System.in); System.out.println("Enter Number of Pages:"); int numPages=sc.nextInt();

int pages[]=new int[numPages]; System.out.println("Enter Reference String: "); for(int i=0;i<numPages;i++)

{

pages[i]=sc.nextInt();

}

System.out.println("Enter Number of Frames"); int capacity=sc.nextInt();

//To represent set of current pages HashSet<Integer> frames=new HashSet<>(capacity);

//To store LRU Indexes of pages //<key=Page,value=index> HashMap<Integer,Integer> index=new HashMap<>();

int pageFaults=0; int hits=0;

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

{

if(frames.size()<capacity) //check if set can hold n=more pages

{

//IF page not present insert into set and increment pagefault if(!frames.contains(pages[i]))

{

frames.add(pages[i]);

index.put(pages[i],i); //push current page into queue pageFaults++;

/\*for(int j:index)

}

else

{

}

}

System.out.print(j+"\t");

System.out.println();\*/

hits++; index.put(pages[i],i);

else //set is full,need replacement

{

if(!frames.contains(pages[i])) //frame is not present present

{

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

Iterator<Integer> itr=frames.iterator(); while(itr.hasNext())

{

int temp=itr.next(); if(index.get(temp)<lru)

{

lru=index.get(temp); val=temp;

}

}

frames.remove(val); frames.add(pages[i]); pageFaults++; index.put(pages[i], i);

}

else //frame is present in set

{

hits++; index.put(pages[i],i);

}

}

frames.forEach(System.out::print); System.out.println();

}

System.out.println("Number of Page Faults : "+pageFaults);

System.out.println("Hits:\t"+hits);

System.out.println("hit ratio: "+((double)hits/(double)numPages));

}

}

**Optimal:**

import java.util.HashMap; import java.util.HashSet; import java.util.Iterator; import java.util.Scanner;

public class Optimal { private Scanner sc;

public void execute()

{

sc = new Scanner(System.in); System.out.println("Enter Number of Pages:"); int numPages=sc.nextInt();

int pages[]=new int[numPages]; System.out.println("Enter Reference String: "); for(int i=0;i<numPages;i++)

{

pages[i]=sc.nextInt();

}

System.out.println("Enter Number of Frames"); int capacity=sc.nextInt();

HashSet<Integer> frames=new HashSet<>();

HashMap<Integer, Integer> index=new HashMap<>(); int pagefaults=0;

int hits=0;

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

{

if(frames.size()<capacity)

{

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

{

pagefaults++; frames.add(pages[i]);

//finding next access of page

int farthest=nextIndex(pages, i); index.put(pages[i], farthest);

}

else

{

}

else

{

}

hits++;

index.put(pages[i], nextIndex(pages,i));

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

{

int farthest=-1; int val=0;

Iterator<Integer> itr=frames.iterator(); while(itr.hasNext())

{

int temp=itr.next();

if(index.get(temp)>farthest)

{

farthest=index.get(temp); val=temp;

}

}

}

else

{

frames.remove(val); index.remove(farthest); frames.add(pages[i]); pagefaults++;

int nextUse=nextIndex(pages, i); index.put(pages[i],nextUse);

hits++;

index.put(pages[i], nextIndex(pages, i));

}

}

frames.forEach(System.out::print); System.out.println();

}

System.out.println("Number of Page Faults : "+pagefaults); System.out.println("Hits:\t"+hits);

System.out.println("hit ratio: "+((double)hits/(double)numPages));

}

public static int nextIndex(int pages[],int curIndex)

{

int farthest=curIndex;

int currentPage=pages[curIndex]; int j=farthest; for(j=farthest+1;j<pages.length;j++)

{

if(pages[j]==currentPage)

{

farthest=j; return farthest;

}

}

return Integer.MAX\_VALUE;

}

}

# Output:

D:\SPOS\6>javac PageReplacement.java D:\SPOS\6>java PageReplacement 1:LRU/n2.Optimal

1

Enter Number of Pages: 12

Enter Reference String: 2

3

5

### 6

7

### 9

7

### 5

7

### 9

4

### 6

Enter Number of Frames 3

### 2

23

### 523

563

### 567

967

### 967

957

### 957

957

### 497

496

### Number of Page Faults : 9 Hits: 3

hit ratio: 0.25 1:LRU/n2.Optimal 2

### Enter Number of Pages: 11

Enter Reference String: 2

### 5

3

### 7

8

### 0

5

### 8

4

7

8

Enter Number of Frames 3

2

25

235

357

578

058

058

058

458

578

578

Number of Page Faults : 8 Hits: 3

hit ratio: 0.2727272727272727