**PAGE REPLACEMENT**

package PageReplacement;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Iterator;

import java.util.ArrayList;

public class PageReplacement

{

static int pageFrames=0;

//Least Recently Used(LRU) Page Replacement Algorithm

static int lru(int referenceString[])

{

//This array list will contain all the pages that are currently in memory

 ArrayList<Integer> pages = new ArrayList<Integer>(pageFrames);

 // This hashmap will store least recently used indexes of the pages

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

 // Start from initial page

 int page\_faults = 0, n = referenceString.length, curPage;

 for (int i=0; i<n; i++)

 {

 curPage = referenceString[i];

 // Check if the set can hold more pages

 if (pages.size() < pageFrames)

 {

 // Insert it into set if not already present

 // This represents a page fault

 if (!pages.contains(curPage))

 {

 pages.add(curPage);

 // increment page fault count

 page\_faults++;

 displayPageFrames(pages, page\_faults);

 }

 // Store the recently used index of each page

 indexes.put(curPage, i);

 }

 // If the set is full then need to select a page to be replaced

 // The page that is selected for replacement is the least recently used page

 else

 {

 // Check if current page is not already present in the set

 if (!pages.contains(curPage))

 {

 // The page having the lowest value of associated index will be the least recently used page

 int lru = Integer.MAX\_VALUE, pageToBeReplaced =0;

 int temp;

 for(int j = 0; j < pages.size(); j++){

 temp = pages.get(j);

 if (indexes.get(temp) < lru)

 {

 lru = indexes.get(temp);

 pageToBeReplaced = j;

 }

 }

 indexes.remove(pages.get(pageToBeReplaced));

 pages.set(pageToBeReplaced, curPage);

 // Increment page fault count

 page\_faults++;

 displayPageFrames(pages, page\_faults);

 }

//Update the current page index

indexes.put(curPage, i);

}

}

return page\_faults;

}

//Optimal Page Replacement Algorithm

static int optimal(int referenceString[])

{

//This array list will contain all the pages that are currently in memory

ArrayList<Integer> pages = new ArrayList<Integer>(pageFrames);

// This hashmap will store least recently used indexes of the pages

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

// Start from initial page

int page\_faults = 0, curPage, n = referenceString.length;

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

{

curPage = referenceString[i];

// Check if the set can hold more pages

if (pages.size() < pageFrames)

{

// Insert it into set if not already present

// This represents a page fault

if (!pages.contains(curPage))

{

pages.add(curPage);

// increment page fault count

page\_faults++;

displayPageFrames(pages, page\_faults);

}

// Store the future index of the page

indexes.put(curPage, findNextIndex(curPage,i, referenceString));

}

// If the set is full then need to select a page to be replaced

// The page that is selected for replacement is the one that will not be used for the longest period of time

else

{

// Check if current page is not already present in the set

if (!pages.contains(curPage))

{

// Find a page that is referenced farthest in the future

//This is implemented by finding a page that has greatest index value

int optimal = Integer.MIN\_VALUE, pageToBeReplaced =0;;

int temp;

for(int j = 0; j < pages.size(); j++){

temp = pages.get(j);

if (indexes.get(temp) > optimal)

{

optimal = indexes.get(temp);

pageToBeReplaced = j;

}

}

indexes.remove(pages.get(pageToBeReplaced));

pages.set(pageToBeReplaced, curPage);

// Increment page faults

page\_faults++;

displayPageFrames(pages, page\_faults);

}

// Update the current page index

indexes.put(curPage, findNextIndex(curPage,i, referenceString));

}

}

return page\_faults;

}

static int findNextIndex(int curPage, int curIndex, int pages[]){

//Starting at the current index find the index of future use of the page

int i;

for(i= curIndex+1; i < pages.length; i++){

if(pages[i] == curPage){

break;

}

}

return i;

}

static void displayPageFrames(ArrayList<Integer> pages, int page\_faults){

System.out.print("At PageFault- " + page\_faults + " :: Pages- ");

for(int i = 0; i < pages.size(); i++) {

System.out.print(" " + pages.get(i));

}

System.out.print("\n");

}

//Driver method

public static void main(String args[])

{

System.out.println("Name :Ketan Devraj ");

System.out.println("Roll No.: TACO-22122 ");

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

pageFrames = 3;

int pageFaults;

System.out.println("--- Implementing Least Recently Used Page Replacement Algorithm --- --");

pageFaults = lru(pages);

System.out.println("Number of page faults = " + pageFaults);

System.out.print("\n");

System.out.println("--- Implementing Optimal Page Replacement Algorithm -----");

pageFaults = optimal(pages);

System.out.println("Number of page faults = " + pageFaults);

}

}

**OUTPUT**

