**Assignment #1**

|  |
| --- |
| Lama alshaiban |

**Question 1 (Page Replacement Algorithms):**

public class Pagereplacement {

/\*\*

\* @param args the command line arguments

\*/

private static int nextReplaceIndex = 0; private static int leastpagefult=Integer.MAX\_VALUE; private static String BestAlgorithm; public static void main(String[] args) {

Scanner scanner = new Scanner(System.in); System.out.print("Enter the number of frames: "); int numFrames = scanner.nextInt();

int[] pages = new int[20];

System.out.println("Enter the 20 page string : "); for (int i = 0; i < 20; i++) { pages[i] = scanner.nextInt();

}

System.out.println("\nFIFO Algorithm:");

Algorithm(pages, numFrames, "FIFO");

System.out.println("\n LRU Algorithm:");

Algorithm(pages, numFrames, "LRU");

System.out.println("\nOptimal Algorithm:");

Algorithm(pages, numFrames, "Optimal");

System.out.println("\nTHE BEST AlGORITHM "+ BestAlgorithm);

}

public static void Algorithm(int[] pages, int numFrames, String algorithm) { ArrayList<Integer> frames = new ArrayList<>(); int pageFaultsWithReplacement = 0; int pageFaultsWithoutReplacement = 0; int TotalPageFaults = 0;

for (int i = 0; i < pages.length; i++) { int page = pages[i];

if (!frames.contains(page)) {

if (frames.size() < numFrames) { frames.add(page);

pageFaultsWithoutReplacement++;

System.out.println("Page fault without replacement: " + frames);

} else {

if (algorithm.equals("FIFO")) {

frames.set(nextReplaceIndex, page);

nextReplaceIndex = (nextReplaceIndex + 1) % numFrames;

} else if (algorithm.equals("LRU")) {

int replaceIndex = LRUReplaceIndex(frames, pages, i); frames.set(replaceIndex, page);

} else if (algorithm.equals("Optimal")) {

int replaceIndex = OptimalReplaceIndex(pages, frames, i); frames.set(replaceIndex, page);

}

pageFaultsWithReplacement++;

System.out.println("Page fault with replacement: " + frames);

}

TotalPageFaults++;

} else {

System.out.println("No page fault: " + frames);

}

}

System.out.println("\nResults for " + algorithm + ":"); System.out.println("Page faults without replacement: " + pageFaultsWithoutReplacement);

System.out.println("Page faults with replacement: " + pageFaultsWithReplacement); System.out.println("Total page faults: " + TotalPageFaults); if(TotalPageFaults<leastpagefult) { BestAlgorithm= algorithm; leastpagefult=TotalPageFaults;

}

}

public static int OptimalReplaceIndex(int[] pages, ArrayList<Integer> frames, int currentIndex) { int farthestIndex = -1; int replaceIndex = 0;

for (int j = 0; j < frames.size(); j++) { int page = frames.get(j);

int nextUse = Nextchange(pages, currentIndex + 1, page);

if (nextUse > farthestIndex) { farthestIndex = nextUse; replaceIndex = j;

}

}

return replaceIndex;

}

public static int Nextchange(int[] pages, int startIndex, int page) { for (int i = startIndex; i < pages.length; i++) { if (pages[i] == page) { return i;

}

}

return pages.length + 1;

}

public static int LRUReplaceIndex(ArrayList<Integer> frames, int[] pages, int currentIndex) { int leastRecentIndex = Integer.MAX\_VALUE; int replaceIndex = 0;

for (int j = 0; j < frames.size(); j++) { int page = frames.get(j);

int lastupdate= lastupdate(pages, currentIndex - 1, page);

if (lastupdate < leastRecentIndex) { leastRecentIndex = lastupdate; replaceIndex = j;

}

}

return replaceIndex;

}

public static int lastupdate(int[] pages, int endIndex, int page) { for (int i = endIndex; i >= 0; i--) { if (pages[i] == page) { return i;

}

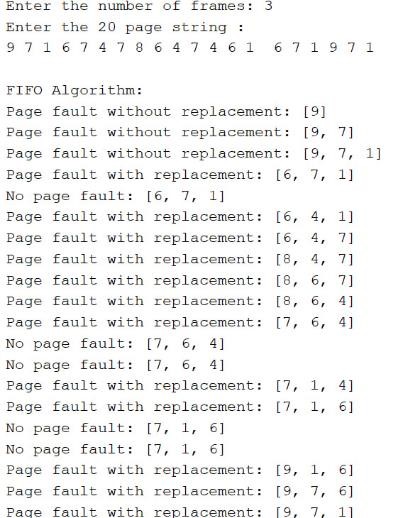
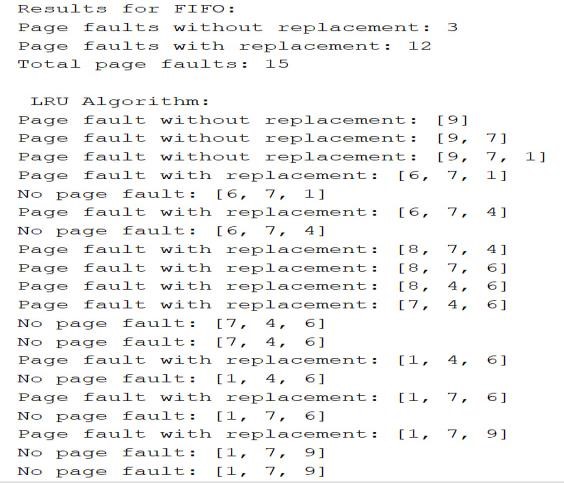
}

return -1;

}

}

The Output:



A screenshot of a computer error

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Question 2 (Disk Scheduling Algorithms):**

public class Diskscheduling {

/\*\*

\* @param args the command line arguments

\*/

private static int least=Integer.MAX\_VALUE; private static String BestAlgorithm; public static void main(String[] args) { Scanner scanner = new Scanner(System.in);

int[] requestQueue = new int[15];

System.out.println("Enter the 15 pending requests: ");

for (int i = 0; i < 15; i++) { requestQueue[i] = scanner.nextInt();

}

System.out.print("Enter the Total number of cylinders: "); int totalCylinders = scanner.nextInt();

System.out.print("Enter the current head position: "); int headPosition = scanner.nextInt();

System.out.print("Enter the head direction SCAN (left/right) ");

String headDirection = scanner.next().toLowerCase();

System.out.println("\nRunning FCFS Algorithm:");

FCFS(requestQueue, headPosition); System.out.println("\nRunning SSTF Algorithm:");

SSTF(requestQueue, headPosition);

System.out.println("\nRunning SCAN Algorithm:");

SCAN(requestQueue, headPosition, headDirection, totalCylinders);

System.out.println("\nTHE BASTA AlGORITHM "+BestAlgorithm);

}

public static void FCFS(int[] queue, int headPosition) { int totalSeekDistance = 0; int currentPosition = headPosition;

System.out.print("Head movement sequence: " + currentPosition);

for (int request : queue) {

if (request != headPosition) {

totalSeekDistance += Math.abs(request - currentPosition); currentPosition = request;

System.out.print(" -> " + currentPosition);

}

}

System.out.println("\nTotal seek distance: " + totalSeekDistance); if(totalSeekDistance<least) { BestAlgorithm="FCFS" ; least=totalSeekDistance;

}

}

public static void SSTF(int[] queue, int headPosition) { int totalSeekDistance = 0; int currentPosition = headPosition;

ArrayList<Integer> requests = new ArrayList<>(); for (int request : queue) { if (request != currentPosition) { requests.add(request);

}

}

System.out.print("Head movement sequence: " + currentPosition); while (!requests.isEmpty()) {

int closestRequest = ClosestRequest(requests, currentPosition); totalSeekDistance += Math.abs(closestRequest - currentPosition); currentPosition = closestRequest;

requests.remove(Integer.valueOf(closestRequest));

System.out.print(" -> " + currentPosition);

}

System.out.println("\nTotal seek distance: " + totalSeekDistance); if(totalSeekDistance<least) { BestAlgorithm="SSTF" ; least=totalSeekDistance;

}

}

public static int ClosestRequest(ArrayList<Integer> requests, int currentPosition) { int minDistance = Integer.MAX\_VALUE; int closestRequest = -1;

for (int request : requests) {

int distance = Math.abs(request - currentPosition); if (distance < minDistance) { minDistance = distance; closestRequest = request;

}

}

return closestRequest;

}

public static void SCAN(int[] queue, int headPosition, String headDirection, int totalCylinders) {

int totalSeekDistance = 0; int currentPosition = headPosition;

ArrayList<Integer> leftRequests = new ArrayList<>();

ArrayList<Integer> rightRequests = new ArrayList<>();

for (int request : queue) { if (request < currentPosition) { leftRequests.add(request); } else if (request > currentPosition) { rightRequests.add(request);

}

}

Collections.sort(leftRequests);

Collections.sort(rightRequests);

System.out.print("Head movement sequence: " + currentPosition);

if (headDirection.equals("left")) {

for (int i = leftRequests.size() - 1; i >= 0; i--) { int request = leftRequests.get(i);

totalSeekDistance += Math.abs(currentPosition - request); currentPosition = request;

System.out.print(" -> " + currentPosition);

}

totalSeekDistance += currentPosition; currentPosition = 0;

System.out.print(" -> " + currentPosition);

for (int request : rightRequests) {

totalSeekDistance += Math.abs(currentPosition - request); currentPosition = request;

System.out.print(" -> " + currentPosition);

}

} else {

for (int request : rightRequests) {

totalSeekDistance += Math.abs(currentPosition - request); currentPosition = request;

System.out.print(" -> " + currentPosition);

}

totalSeekDistance += Math.abs(currentPosition - (totalCylinders - 1)); currentPosition = totalCylinders - 1;

System.out.print(" -> " + currentPosition);

for (int i = leftRequests.size() - 1; i >= 0; i--) { int request = leftRequests.get(i);

totalSeekDistance += Math.abs(currentPosition - request); currentPosition = request;

System.out.print(" -> " + currentPosition);

}

}

System.out.println("\nTotal seek distance: " + totalSeekDistance); if(totalSeekDistance<least) { BestAlgorithm="SCAN" ; least=totalSeekDistance;

}

}

}

The Output:

