import java.io.BufferedInputStream;

import java.util.\*;

public class page\_replacement {

private final int n;//内储页框

private final int m;//访问次数

private int F;//没能直接找到的次数,(F/m)为缺页率

private List<Integer> list = null;//访问地址走向

private Map<Integer, Integer> map = null;

public page\_replacement() {

F = 0;

map = new HashMap<Integer, Integer>();//存储每一个内储页框所存的内容

Scanner cin = new Scanner(new BufferedInputStream(System.in));

System.out.println("请输入用户访问页地址走向");

list = new ArrayList<Integer>();

String s = cin.nextLine();

String[] s1 = s.split(" ");

m = s1.length;

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

list.add(Integer.valueOf(s1[i]));

System.out.println("请输入内储叶框数量");

n = cin.nextInt();

menu();

switch (cin.nextInt()) {

case 1:

OPT();

break;

case 2:

FIFO();

break;

case 3:

LRU();

break;

case 4:

NRU();

break;

}

cin.close();

}

public void menu() {

System.out.println("\*\*\*\* 选择最优置换算法请按1 \*\*\*\*\*\*\*\*\*\*");

System.out.println("\*\*\*\* 选择先进先出置换算法请按2 \*\*\*\*\*\*\*");

System.out.println("\*\*\*\* 选择最近最远未使用置换算法请按3 \*\*\*");

System.out.println("\*\*\*\* 选择最近未使用置换算法请按4 \*\*\*\*\*");

}

public void OPT() {//最佳置换算法

int j;

for (int i = 0; i < m; i++) {

int k = list.get(i);//待处理元素

if (!map.containsValue(k)) {

F++;//不能直接找到次数加1

if (map.size() < n) {//如果没有装满

int temp = map.size();

map.put(temp, k);

} else {//如果装满了

int index = 0;//把哪个位置的淘汰出去

int min = 0;//初始最长长度

for (int t = 0; t < n; t++) {

for (j = i + 1; j < m; j++) {//看后面哪一个出现的最晚

if (list.get(j).equals(map.get(t))) {//第一次找到

if (j - i > min) {

index = t;//更新值

min = j - i;

}

break;

}

}

if (j == m) {//如果到最后

index = t;

min = j - i;

}

}

map.remove(index);

map.put(index, k);//修改表内元素

}

}

System.out.println(map);

}

System.out.println("缺页次数:" + F);

}

public void FIFO() {//先进先出置换算法

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

for (int i = 0; i < m; i++) {

int k = list.get(i);//待处理元素

if (!map.containsValue(k)) {

F++;//不能直接找到次数加1

if (map.size() < n) {//如果没有装满

int temp = map.size();

map.put(temp, k);

q.offer(temp);

} else {

int temp = q.poll();//排除的元素位置

map.remove(temp);

map.put(temp, k);

q.offer(temp);

}

}

System.out.println(map);

}

System.out.println("缺页次数:" + F);

}

public void LRU() {//最近最远未使用置换算法

List<Integer> linkedlist = new LinkedList<Integer>();

int start = 0;

for (int i = 0; i < m; i++) {

int k = list.get(i);//待处理元素

if (!map.containsKey(k)) {

F++;//不能直接找到次数加1

if (map.size() < n) {//如果没有装满

int temp = map.size();

map.put(k, temp);

linkedlist.add(k);//添加位置

} else {

int temp = linkedlist.get(0);

int c = map.get(temp);//位置

map.remove(temp);

map.put(k, c);

linkedlist.remove(0);

linkedlist.add(k);

}

} else//如果包含这个值，把这个值拿走并在后面压入

{

//查找存在位置

linkedlist.remove((Integer) k);

linkedlist.add(k);

}

System.out.println(map);

}

System.out.println("缺页次数:" + F);

}

public void NRU() {//最近最远未使用置换算法

List<Integer> linkedlist = new LinkedList<Integer>();

int start = 0;

for (int i = 0; i < m; i++) {

int k = list.get(i);//待处理元素

if (!map.containsKey(k)) {

F++;//不能直接找到次数加1

if (map.size() < n) {//如果没有装满

int temp = map.size();

map.put(k, temp);

linkedlist.add(k);//添加位置

} else {

int temp = linkedlist.get(0);

int c = map.get(temp);//位置

map.remove(temp);

map.put(k, c);

linkedlist.remove(0);

linkedlist.add(k);

}

} else//如果包含这个值，把这个值拿走并在后面压入

{

//查找存在位置

linkedlist.remove((Integer) k);

linkedlist.add(k);

}

System.out.println(map);

}

System.out.println("缺页次数:" + F);

}

public static void main(String[] args) {

page\_replacement p = new page\_replacement();

}

}

