import java.util.\*;

public class Practice

{

    public static void fifo(int[] refs, int memorysize)

    {

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

        int hits=0;

        for(int ref : refs)

        {

            if(!memory.contains(ref))

            {

                if(memory.size() == memorysize)

                {

                memory.poll();

                }

                memory.add(ref);

            }

            else{

                hits++;

            }

            System.out.println(ref+"  "+memory);

        }

        int misses = refs.length - hits;

        System.out.println("Page Faults: "+misses+"\tPage Hits: "+hits);

    }

    public static void lru(int[] refs, int memorysize)

    {

        List<Integer> memory = new ArrayList<>();

        int hits=0;

        for(int ref : refs)

        {

            if(memory.contains(ref))

            {

                hits++;

                memory.remove((Integer) ref);

                memory.add(ref);

            }

            else{

                if(memory.size() == memorysize)

                {

                    memory.remove(0);

                }

                memory.add(ref);

            }

            System.out.println(ref+"  "+memory);

        }

        int misses = refs.length - hits;

        System.out.println("Page Faults: "+misses+"\tPage Hits: "+hits);

    }

    public static void optimal(int[] refs, int memorysize)

    {

        List<Integer> memory = new ArrayList<>();

        int hits=0;

        for(int i=0; i<refs.length; i++)

        {

            int ref = refs[i];

            if(memory.contains(ref))

            {

                hits++;

            }

            else

            {

            if(memory.size() == memorysize)

            {

                int maxGap = -1;

                int pageToReplace = 0;

                for(int page : memory)

                {

                    int gap = findNextOccurence(refs, i, page);

                    if(gap > maxGap)

                    {

                        maxGap = gap;

                        pageToReplace = page;

                    }

                }

                memory.remove((Integer) pageToReplace);

            }

            memory.add(ref);

            }

            System.out.println(ref+" "+memory);

        }

        int misses = refs.length - hits;

        System.out.println("Page Faults: "+misses+"\tPage Hits: "+hits);

    }

    private static int findNextOccurence(int[] refs, int start, int page)

    {

        for(int i = start+1; i<refs.length; i++)

        {

            if(refs[i] == page)

            {

                return i;

            }

        }

        return Integer.MAX\_VALUE;

    }

    public static void main(String args[])

    {

        Scanner sc = new Scanner(System.in);

        int n;

        int f;

        System.out.println("Enter the number of members present in refernce string: ");

        n = sc.nextInt();

        System.out.println("Enter the no. of frames: ");

        f = sc.nextInt();

        System.out.println("Enter values:  ");

        int[] refs = new int[n];

        for(int i=0; i<n; i++)

        {

            refs[i] = sc.nextInt();

        }

        int choice;

        do

        {

            System.out.println("\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*");

            System.out.println("1. FIFO");

            System.out.println("2. LRU");

            System.out.println("3. Optimal");

            System.out.println("Enter the your choice: ");

            choice = sc.nextInt();

            switch(choice)

            {

                case 1:

                fifo(refs, f);

                break;

                case 2:

                lru(refs, f);

                break;

                case 3:

                optimal(refs, f);

                break;

                case 4:

                break;

                default:

                System.out.println("Invalid Argument");

                break;

            }

        }while(choice != 4);

    }

}

/\*

FIFO:

1) 6 7 8 9 6 7 1 6 7 8 9 1

Page faults: 9          Page Hits: 3

2)  6 1 1 2 0 3 4 6 0 2 1 2 1 2 0 3 2 1 2 0

Page Faults: 12         Page Hits: 8

LRU:

1) 4 7 6 1 7 6 1 2 7 2

Page Faults: 6          Page Hits: 4

2) 6 1 1 2 0 3 4 6 0 2 1 2 1 2 0 3 2 1 2 0

Page Faults: 13         Page Hits: 7

Optimal:

1) 6 1 1 2 0 3 4 6 0 2 1 2 1 2 0 3 2 1 2 0

Page Faults: 10 Page Hits: 10

2) 6 7 8 9 6 7 1 6 7 8 9 1 7 9 6

Page Faults: 8  Page Hits: 7

\*/