```java
package Assign_1;

/** This class is an implementation of the main class
 for this program called Cipher.
 * Micah Rose-Mighty
 * 6498935
 * 2020-09-24
 * Created using IntelliJ
 */


import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.text.MessageFormat;
import java.util.*;


public class Cipher {

    public static void main(String[] args) throws FileNotFoundException {

        File fileText = new File("assn1in.txt");

        Scanner s = new Scanner(fileText);

        int n = s.nextInt();
        int m = s.nextInt();
        String cipherText = s.next();
        assert cipherText.length() == n;
        String knownWord = s.next();
        assert knownWord.length() == m;
        int k = s.nextInt();
        int i = s.nextInt();

        Collection<Permutation>  acceptedPermutations = new ArrayList<>(); // This is a Collection of accepted Permutations that match the known word
        String wordSegment = cipherText.substring(0,m);
        System.out.println("n = " + n + " m = "+ m + " Ciphertext = " + cipherText + " Known word = "+ knownWord + " k = " + k +" i = " + i + " Word Segment
```

```
37  = "+ wordSegment);
38          s.close();
39
40          for (Permutation permutation:Permutation.
   generatePermutations(m)){ // Applies all the m-length
    permutations to the word segment
41              String permuted = permutation.apply(
   wordSegment);
42              System.out.println(MessageFormat.format(
   "{0} {1}", permutation, permuted));
43
44              if(knownWord.equals(permuted)){ //Adds
   all permutations that match the known word to the
   acceptedPermutations ArrayList
45                  acceptedPermutations.add(permutation
   );
46              }
47
48          }
49
50          int counter  = 1;
51          for (Permutation permutation :
   acceptedPermutations) { // This counts all accepted
   permutations to re-encrypt the k'th potential
   plaintext
52              String decrypted = permutation.apply(
   cipherText);
53              System.out.println(decrypted);
54
55              if (counter == k) {
56                  Reencipher reencipher = new
   Reencipher(decrypted, i);
57                  System.out.println(reencipher.
   encipher());
58              }
59              counter++;
60          }
61
62          }}
63
64
65
```

```java
package Assign_1;

/** This class is an implementation of the Circular Linked List used to Reencipher a given word
 * Micah Rose-Mighty
 * 6498935
 * 2020-09-24
 * Created using IntelliJ
 */


public class Reencipher {

    private ReencipherNode head = null;
    private ReencipherNode tail = null;
    private final int step;
    private final int n;

    public Reencipher(String s, int step) { // This a Constructor that accepts the word to encipher and the corresponding enciphering parameter
        this.step = step;
        this.n = s.length();

        for (char c : s.toCharArray()) {
            add(c);
        }
    }

    public String encipher(){ //This is a method to actually generate the enciphered word
        ReencipherNode current = head;

        char [] chars = new char[n];
        int counter = 0;
        while(true) {
            for (int i = 0; i<step-2; i++){
                current = current.next;
            }

            if (current == current.next){
                chars[counter] = current.c;
                break;

```

```
41              }
42
43              chars[counter] = current.next.c;
44              counter++;
45
46              current.next = current.next.next;
47              current = current.next;
48          }
49          return new String(chars);
50      }
51
52
53      private void add(char c){ // This is a private
    method that adds a character c correctly to the
    circular linked list
54
55          ReencipherNode newNode = new ReencipherNode(c
    );
56
57          if(head == null){
58              head = newNode;
59              tail = head;
60          }
61          else if(head == tail) {
62              head.next = newNode;
63              newNode.next = head;
64              tail = newNode;
65          }
66          else {
67              tail.next = newNode;
68              newNode.next = head;
69              tail = newNode;
70          }
71      }
72
73
74
75  }
76
```

```java
package Assign_1;

/** This class is an implementation of the Permutation Instance
 * Micah Rose-Mighty
 * 6498935
 * 2020-09-24
 * Created using IntelliJ
 */


import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.stream.Collectors;

public class Permutation {

    private final int n;
    private final int[] values;


    public static Collection<Permutation> generatePermutations(int n) { //This is a Static method that creates a Collection that contains all the permutations of the given length
        int [] elements = new int[n];
        for (int i = 0; i<n; i++) {
            elements[i] = i;
        }
        Collection<Permutation> result = new ArrayList<>();
        for (int[] values : permute(n,elements)) {
            result.add(new Permutation(values));
        }
        return result;
    }


    private static Collection<int[]> permute(int l, int[] values) { //This is a Recursive Method for creating all the permutations.
        Collection<int[]> result = new ArrayList<>();

```

```
38          if(l == 1) {
39              result.add(values.clone());
40          }
41          else {
42              for (int i: values) {
43                  int[] trunc = new int[values.length-1
    ];
44                  int counter = 0;
45                  for (int value : values) {
46                      if (value == i){
47                          continue;
48                      }
49                      trunc[counter] = value;
50                      counter++;
51                  }
52                  Collection<int[]> shorters = permute(
    l-1,trunc);
53                  for (int[] shorter : shorters) {
54                      int[] newValues = new int[shorter
    .length+1];
55                      System.arraycopy(shorter,0,
    newValues,1, shorter.length);
56                      newValues[0] = i;
57                      result.add(newValues);
58                  }
59              }
60          }
61          return result;
62      }
63      private Permutation(int[] values){ // This is a
    Private Constructor created for use in the
    generatePermutations method
64          this.n = values.length;
65          this.values = values.clone();
66      }
67      public String apply(String s) { // This is a
    method for permuting the characters of the given
    string with the correct permutation pattern
68          if (s.length() % n != 0){
69              throw new IllegalArgumentException("The
    length of this string must be a multiple of the
    permutation size since we can't have a partial
    permutation.");
70          }
```

```
71          int parts = s.length() / n;
72          char[] chars = s.toCharArray();
73          char[] charsPermuted = new char[s.length()];
74          for (int i = 0; i<parts; i++) {
75              for (int j = 0; j<n; j++){
76                  charsPermuted[i*n + j] = chars[i*n
    + values[j]];
77              }
78          }
79          return new String(charsPermuted);
80      }
81      @Override
82      public String toString(){ // This is an
   overridden toString Method used to create the proper
    representation of the permutation
83          return "[" + Arrays.stream(values).mapToObj(
   i -> Integer.valueOf(i+1).toString()).collect(
   Collectors.joining("")) + "]";
84
85      }
86 }
87
```

```java
package Assign_1;

/** This class is an implementation of the circular
  linked list node containing a single character and
  next node pointer
 * Micah Rose-Mighty
 * 6498935
 * 2020-09-24
 * Created using IntelliJ
 */

public class ReencipherNode {

    char c;
    ReencipherNode next;

    ReencipherNode(char c) {
        this.c = c;
        this.next = null;
    }

}

```