**Burrows Wheeler**

**Problem Statement**

Burrows Wheeler outcompresses gzip and PKZIP, is relatively easy to implement, and is not protected by any patents. It forms the basis of the Unix compression utility bzip2. The Burrows–Wheeler data compression algorithm consists of three algorithmic components, which are applied in succession:

* **Burrows–Wheeler transform:** Given a typical English text file, transform it into a text file in which sequences of the same character occur near each other many times.
* **Move-to-front encoding:** Given a text file in which sequences of the same character occur near each other many times, convert it into a text file in which certain characters appear much more frequently than others.
* **Huffman compression:** Given a text file in which certain characters appear much more frequently than others, compress it by encoding frequently occurring characters with short codewords and infrequently occurring characters with long codewords.

**This program expects us to**

* Encode a using MoveToFront method and decode the encoded string
* Circular Suffix Array helps us in reducing the timing.
* Implement Burrows-Wheeler Transform on the given string and obtain the original string by using Inverse Burrows-Wheeler Transform method.

**Related Concepts**

* Programming Language - Java
* Hash Tables
* Suffix Arrays
* Circular Suffix Arrays

The Score for the Project is 90 / 100.

**API**

**MoveToFront :**

**public class MoveToFront {**

// apply move-to-front encoding, reading from standard input and writing to standard output

**Time Complexity : proportional nR**

**Space Complexity : proportional to n + R**

**public static void encode()**

// apply move-to-front decoding, reading from standard input and writing to standard output

**Time Complexity : proportional to n + R**

**Space Complexity : proportional to n + R**

**public static void decode()**

// if args[0] is "-", apply move-to-front encoding

// if args[0] is "+", apply move-to-front decoding

**public static void main(String[] args)**

**}**

**CircularSuffixArray :**

**public class CircularSuffixArray {**

// circular suffix array of s

**Time Complexity : proportional to n^2**

**Space Complexity : proportional to n + R**

**public CircularSuffixArray(String s)**

// length of s

**Time Complexity : O(1)**

**Space Complexity : O(1)**

**public int length()**

// returns index of ith sorted suffix

**Time Complexity : O(1)**

**Space Complexity : O(1)**

**public int index(int i)**

// unit testing (required)

**public static void main(String[] args)**

**}**

**BurrowsWheeler :**

**public class BurrowsWheeler {**

// apply Burrows-Wheeler transform,

// reading from standard input and writing to standard output

**Time Complexity : proportional to n + R**

**Space Complexity : proportional to n + R**

**public static void transform()**

// apply Burrows-Wheeler inverse transform,

// reading from standard input and writing to standard output

**Time Complexity : proportional to n + R**

**Space Complexity : proportional to n + R**

**public static void inverseTransform()**

// if args[0] is "-", apply Burrows-Wheeler transform

// if args[0] is "+", apply Burrows-Wheeler inverse transform

**Time Complexity :**

**Space Complexity :**

**public static void main(String[] args)**

**}**