Intelligence in Biological Systems

Rohith Ganesan

Implementation done in scala and python

Objective:

Code to show the burrows-wheeler transform and inverse burrows-wheeler transform operation.

Theory:

Burrows–Wheeler transform rearranges a character string into runs of similar characters. This is useful for compression, since it tends to be easy to compress a string that has runs of repeated characters by techniques such as move-to-front transform and run-length encoding. More importantly, the transformation is reversible, without needing to store any additional data except the position of the first original character. The BWT (Burrow Wheeler Transform) is a method of improving the efficiency of text compression algorithms, costing only some extra computation.

The remarkable thing about the BWT is not that it generates a more easily encoded output as ordinary sort, but also the transformed sequence is reversible. It allows the original document to be re-generated from the last column data. The algorithm which maps from the encoded sequence back to the original sequence is called Inverse Burrows–Wheeler transform.

Pseudo Code for BWT:

```
function BWT (string s)
    create a table, where the rows are all possible rotations of s
    sort rows alphabetically
    return (last column of the table)
```

BWT Explanation:

The transform is done by sorting all the circular shifts of a text in lexicographic order and by extracting the last column and the index of the original string in the set of sorted permutations of S.

Given an input string S = ^BANANA| S , consider the symbol ^ representing the start of the string and the red | character representing the 'EOF' pointer;.

Transformation				
1. Input	2. All rotations	3. Sort into lexical order	4. Take the last column	5. Output
^BANANA|	^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^	**A**NANA|^B **AN**A|^BAN **A**|^BANAN **B**ANANA|^ **N**ANA|^BA **NA**|^BANA ^BANANA| |^BANANA	ANANA|^**B** ANA|^BA**N** A|^BANA**N** BANANA|^ NANA|^B**A** NA|^BAN**A** ^BANANA| |^BANAN**A**	BNN^AA|A

Pseudo Code for IBWT:

```
function inverseBWT (string s)
    create empty table
    repeat length(s) times
        // first insert creates first column
        insert s as a column of table before first column of the table
        sort rows of the table alphabetically
    return (row that ends with the 'EOF' character)
```

IBWT Explanatio:

The inverse can be understood this way. Take the final table in the BWT algorithm, and erase all but the last column. Given only this information, you can easily reconstruct the first column. The last column tells you all the characters in the text, so just sort these characters alphabetically to get the first column. Then, the last and first columns (of each row) together give you all pairs of successive characters in the document, where pairs are taken cyclically so that the last and first character form a pair. Sorting the list of pairs gives the first and second columns. Continuing in this manner, you can reconstruct the entire list. Then, the row with the "EOF" character at the end is the original text.

Inverse transformation							
Input				Output			
BNN^AA|A				^BANANA|		
Add 1	Sort 1	Add 2	Sort 2	Add 3	Sort 3	Add 4	Sort 4
B N N ^ A A | A	A A A B N N ^ |	BA NA NA ^B AN AN |^ A|	AN AN A| BA NA NA ^B |^	BAN NAN NA| ^BA ANA ANA |^B A|^	ANA ANA A|^ BAN NAN NA| ^BA |^B	BANA NANA NA|^ ^BAN ANAN ANA| |^BA A|^B	ANAN ANA| A|^B BANA NANA NA|^ ^BAN |^BA
Add 5	Sort 5	Add 6	Sort 6	Add 7	Sort 7	Add 8	Sort 8
BANAN NANA| NA|^B ^BANA ANANA ANA|^ |^BAN A|^BA	ANANA ANA|^ A|^BA BANAN NANA| NA|^B ^BANA |^BAN	BANANA NANA|^ NA|^BA ^BANAN ANANA| ANA|^B |^BANA A|^BAN	ANANA| ANA|^B A|^BAN BANANA NANA|^ NA|^BA ^BANAN |^BANA	BANANA| NANA|^B NA|^BAN ^BANANA ANANA|^ ANA|^BA |^BANAN A|^BANA	ANANA|^ ANA|^BA A|^BANA BANANA| NANA|^B NA|^BAN ^BANANA |^BANAN	BANANA|^ NANA|^BA NA|^BANA ^BANANA| ANANA|^B ANA|^BAN |^BANANA A|^BANAN	ANANA|^B ANA|^BAN A|^BANAN BANANA|^ NANA|^BA NA|^BANA ^BANANA| |^BANANA

Observations:

- Tends to put runs of the same character together
- Makes compression work well
- Doesn't always improve the compressibility in some cases
- e.g, "appellee"

Python Implementation :

```
def bwt(s: str) -> str:
    """Apply Burrows-Wheeler transform to input string."""
    assert "$" not in s and "@" not in s, "Input string cannot contain special characters"
    s = "$" + s + "@"  # Add start and end of text marker
    table = sorted(s[i:] + s[:i] for i in range(len(s)))  # Table of rotations of string
    last_column = [row[-1:] for row in table]  # Last characters of each row
    return "".join(last_column)  # Convert list of characters into string
```

```
def ibwt(r: str) -> str:
    """Apply inverse Burrows-Wheeler transform."""
    table = [""] * len(r)  # Make empty table
    for i in range(len(r)):
        table = sorted(r[i] + table[i] for i in range(len(r)))  # Add a column of r
    s = [row for row in table if row.endswith("@")][0]  # Find the correct row (ending in ETX)
    return s.rstrip("@").strip("$")  # Get rid of start and end markers
```

```
seq='AAGGCGCG'
```

```
t=bwt(seq)
t
```

```
'@G$AGGCCGA'
```

```
inverted=ibwt(t)
inverted
```

```
'AAGGCGCG'
```

```
inverted==seq
```

```
True
```

Scala Implementation:

```
  import scala.collection._


    def bwt(S:String):String={
    var s=S
    assert(s.contains("$")==false && s.contains("@")==false )
    s="$" + s + "@"// To mark the start and end of sequence
    val len=s.length
    var table=List.tabulate(len)(i=> s.substring(i)+s.substring(0,i)).sorted
    //creating a Table of rotations of string
    var last_col=table.map(_(len-1))//taking Last characters of each row
    last_col.mkString("")//Convert list of characters into string
}




def ibwt(r:String):String={

    var len=r.length
    var table=Array.fill(len)("")
      for(i <- 0 until len){
          table=Array.tabulate(len)(j=>r(j) + table(j)).sorted
       // Add a column of r and sort the list
    }
    var s=table.filter(_.endsWith("@"))// Find the correct row (ending in ETX)
    s(0).slice(1,len-1)

}
  val seq=List("AAGGCGCG","TGGAAGAG","GCCGCCGC","GGCCTGGA")
  val transform=seq.map(bwt)
  val inverted=transform.map(ibwt)
  seq == inverted
```

Input:

```
val seq=List("AAGGCGCG","TGGAAGAG","GCCGCCGC","GGCCTGGA")
```

Output:

```
seq = List(AAGGCGCG, TGGAAGAG, GCCGCCGC, GGCCTGGA)
transform = List(@G$AGGCCGA, @GGGAAGAT$, @CGGGCCCC$, @AGGCGGT$C)
inverted = List(AAGGCGCG, TGGAAGAG, GCCGCCGC, GGCCTGGA)
true
```

scala code (IBM WATSON STUDIO) notebook link:
https://eu-gb.dataplatform.cloud.ibm.com/analytics/notebooks/v2/956c50b7-87a3-4d9f-8af6-22d80c738fae/view?access_token=1b258ba2f5ca404db36be35a2effc7116fbfadf2d2b0143961e43b52cfae7bd6