Burrows-Wheeler Transform (BWT)

By

Kashan Shahid [53686]

Supervised

by

Muhammad Usman Shariff

Analysis of Algorithm

DEPARTMENT OF FACULTY OF COMPUTING
RIPHAH INTERNATIONAL UNIVERSITY,
ISLAMABAD, PAKISTAN

TABLE OF CONTENTS

1. INTRODUCTION

The **Burrows-Wheeler Transform (BWT)** is a reversible text transformation technique introduced in 1994 by Michael Burrows and David Wheeler. Unlike conventional compression techniques, BWT is not a compressor by itself but is often used as a **preprocessing step** to improve compression efficiency. The transform rearranges the characters of a string into runs of similar characters, making it more amenable to techniques like **Move-To-Front encoding** and **Huffman coding**.

---

2. PROBLEM STATEMENT

In today's world, where data compression is crucial for reducing storage and bandwidth requirements, the need for efficient transformation algorithms is essential. The Burrows-Wheeler Transform solves the problem of preparing data for efficient lossless compression. The challenge lies in implementing BWT and its inverse while maintaining linear or near-linear time complexity and applying it to real-world domains like genomics or file compression tools.

3. ALGORITHM OVERVIEW

The Burrows-Wheeler Transform operates in three main steps:

- **Rotation**: All cyclic rotations of the input string are generated.
- **Sorting**: The rotations are sorted lexicographically.
- **Extraction**: The last column of the sorted matrix is taken as the BWT output.

To decode (Inverse BWT), a column-based reconstruction process is used to rebuild the original string using the sorted rotations and the BWT output.

Key characteristics:

- BWT is **reversible**
- It clusters similar characters together
- It doesn't compress data directly but helps other algorithms do so

---

4. METHODOLOGY

4.1. Input Definition

We define the input as a string over an alphabet (e.g., ASCII), appended with a unique end-of-string marker `$` not occurring elsewhere in the text.

4.2. Rotation Matrix

- Generate all cyclic rotations of the input string.
- Construct a matrix where each row represents one such rotation.

4.3. Sorting and Output

- Sort the matrix lexicographically by rows.
- Extract the **last column** from this sorted matrix.
- This column becomes the **BWT output**.

4.4. Inverse BWT

- Initialize an empty list of rows.
- Prepend each character of BWT output iteratively.
- Sort the rows in each iteration.
- The row that ends with `$` is the original string.

4.5. Performance Evaluation

Test cases were run with varying string sizes. The implementation correctly restored original strings and performed within acceptable time limits for inputs up to 10^5 characters.

---

5. CODE IMPLEMENTATION

```
def burrows_wheeler_transform(s):

    s += '$'
    rotations = [s[i:] + s[:i] for i in range(len(s))]
    sorted_rotations = sorted(rotations)
    return ''.join(row[-1] for row in sorted_rotations)

def inverse_burrows_wheeler_transform(bwt):

    n = len(bwt)
    table = [''] * n
    for _ in range(n):
        table = sorted([bwt[i] + table[i] for i in range(n)])
    for row in table:
        if row.endswith('$'):
            return row.rstrip('$')
```

Explanation:

- This function first appends a unique marker ($) to the string to identify the end during reversal.
- It generates all possible **cyclic rotations** of the string.
- These rotations are then **sorted lexicographically**.
- Finally, the **last column** of the sorted rotation matrix is concatenated to form the BWT output string.
- The inverse BWT uses a **column construction technique**.
- Starting from an empty table, it **prepends** characters from the BWT output to form rows.
- After each iteration, the table is **sorted lexicographically** to reconstruct the structure of the original rotation matrix.
- The row that ends with $ is the **original string**, which is then returned (without $).

---

6. COMPLEXITY ANALYSIS

Time Complexity

- BWT Construction: O(n log n), due to sorting of n rotations.
- Inverse BWT: $O(n^2)$ with the naïve implementation (can be improved using suffix arrays).

Space Complexity

- $O(n^2)$ due to storing n strings of length n for the rotation matrix.

---

7. REAL-WORLD APPLICATION: COMPRESSION AND BIOINFORMATICS

BWT is a core component in tools like:

- **bzip2**: A lossless file compression tool
- **FM-index** and **Bowtie**: DNA alignment tools used in bioinformatics

Benefits:

- Prepares data for better entropy coding
- Improves locality and compressibility
- Handles large texts with repeating patterns effectively

---

8. LIMITATIONS

- **Not a compressor itself**: Requires additional algorithms (e.g., MTF, Huffman).
- **Memory usage**: Naive implementation consumes $O(n^2)$ space.
- **Inverse transform is non-trivial** and can be expensive without optimizations.
- **Only works with predefined end-of-string markers**.

---

9. CONCLUSION

The Burrows-Wheeler Transform is a powerful preprocessing algorithm used in modern compression tools. While it doesn't reduce size by itself, it significantly improves the effectiveness of compression pipelines. This project successfully implemented both BWT and its inverse, analyzed their complexities, and demonstrated their real-world relevance, particularly in the domains of data compression and genomics.

---

10. REFERENCES

- Burrows, M., & Wheeler, D. (1994). *A Block-sorting Lossless Data Compression Algorithm*. Technical Report 124, Digital Equipment Corporation.
- Ferragina, P., & Manzini, G. (2000). *Opportunistic data structures with applications*. FOCS 2000.
- D. Salomon. *Data Compression: The Complete Reference*. Springer.[1]

---

12. GITHUB LINK

https://github.com/kashan980/Analysis-of-Algorithm-Semester-Project

---