Max Feinberg
Mr. Lefebvre
Data Structures and Algorithms
7 May 2021

An Implementation of Burrows Wheeler Data Compression and Pattern Searching

Introduction

I implemented the Burrows-Wheeler Data Compression Algorithm (inspiration found on https://web.stanford.edu/class/archive/cs/cs166/cs166.1166/handouts/090%20Final%20Project%20Topics.pdf). The Burrows Wheeler Data Compression is a lossless algorithm which means that it allows the original data from the user to be perfectly reconstructed from the compressed data. The three steps for the BWDCA include: completing the Burrows Wheeler Transform, performing move-to-front encoding, and then performing Huffman compression. The Burrows Wheeler Transform rearranges a string into runs of similar characters. It takes an input, and constructs a suffix array based on each possible suffix of the string. It then sorts these suffixes and uses the last character from each of them to generate a new, usually more easily compressed string because more characters repeat more often. The notable aspect of the BWT, however, is that it is completely reversible without any external information, so that the original message can be retrieved from the returned string. Move-to-front encoding then compresses the string further. It does this by repeatedly reading a character from the input message, printing the position in a given alphabet in which that character appears, and moving that character to the front of the sequence. Finally, Huffman compression returns the final compressed string. For an extensive explanation of Huffman Encoding, reference this. To decompress this text, the program simply runs the encoding processes in reverse. It starts by decoding the Huffman Compression, then it moves characters to front in reverse, and finally it inverts the Burrows Wheeler Transform. I wanted to expand upon my knowledge of data compression from the last project and this seemed like a perfect algorithm with real-world applications to do that. Bzip2 is a free file compression program that uses the BWDCA exactly as I've described it above. Besides lossless data compression, the Burrows-Wheeler transform itself (not including MTF encoding or Huffman compression), has been used in bioinformatics. The Burrows Wheeler Transform has a "last-first" mapping property, which means that the jth occurrence of a symbol in the BWT corresponds to its jth occurrence in its suffix array. For example, the 1st "a" in a Burrows Wheeler Transform will also be the first "a" in its given suffix array. An FM-Index is a helper data structure that can be used to efficiently find the number of occurrences of a pattern within the compressed text, as well as locate the position of each occurrence. It keeps track of how many of each symbol have been seen in the BWT prior to its ith symbol, and at the end it determines the BWT index belonging to the first of each symbol in its suffix array. Using an FM-Index, you can determine all occurrences of a string in the BWT. It searches characters in reverse order to narrow down the range of possibilities for an occurrence of the string, and then returns all occurrences. This same logic has been applied on a more massive scale to genome sequence alignment. A human genome contains 3 billion base pairs and instead of storing all 3 billion, storing just the Burrows Wheeler Transform of the string significantly reduces the memory needed to complete the alignment searches. Researchers can then use the FM-index searching method to locate identical strings within the BWT. Several programs like Bowtie and BWA were developed with the Burrows Wheeler transform with the goal of decreasing the memory required

for alignment. The BWT has also been implemented in image compression and data compression, however, much less commonly than its use in genome alignment sequencing. One downside of BWT compression is that most often, there is often no improvement in compression size compared to Huffman Encoding alone until a certain length of string is used.

Analysis

Please see separate PDF.

References

https://web.stanford.edu/class/cs262/presentations/lecture4.pdf

- Gave a good overview of BWT with genome sequencing and helped understand last-first property

http://www.cs.jhu.edu/~langmea/resources/bwt_fm.pdf

- Helped me understand the theoretical side of BWT in a more step by step fashion.

https://www.cs.princeton.edu/courses/archive/spring21/cos226/assignments/burrows/specification.php

- Instructions as to what order BWT data compression should occur in, and helped me structure my work accordingly

http://csbio.unc.edu/mcmillan/Comp555S21/Lecture11.pdf

- Helped extensively with understanding pattern searching with and without BWT and inverting the BWT. Also gave a good understanding of the last-first mapping problem and a somewhat helpful understanding of the FM Index.

https://www.geeksforgeeks.org/burrows-wheeler-data-transform-algorithm/

- Good overview of BWT and steps to construct it

https://web.stanford.edu/class/archive/cs/cs166/cs166.1166/handouts/090%20Final%20Project%20Topics.pdf

- Inspiration for project

https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform

- Examples and testing data for BWTs and gave some examples of applications in the world (bioinformatics, image compression, data compression)

https://docs.python.org/3/howto/sorting.html

- Helped me understand the lambda function for sorting

https://en.wikipedia.org/wiki/FM-index

- Really broke down the FM Index in a more comprehensible way and helped me understand the last-first mapping problem

https://wiki.python.org/moin/TimeComplexity

- Helped with function complexity