Text Encoding and Compression System

Thomas QUARSHIE

Settor AMEDIKU

Mario ARYEH

Introduction

Compression involves the removal of redundant information to retain uncertain information. The type of compression to be implemented in this project is lossless compression, for text files with string data This allows the data compression process to be reversible to retrieve the original information. Data compression is possible because most real-world data contain large amount of statistical redundancy. This form of redundancy is referred to as information entropy redundancy. Data compression enables devices to transmit data with fewer bits. This saves time during transmission and storage space (e.g., file downloads).

Background

In the mid-1980s, PCs had storage devices (hard disks specifically) that were no bigger than 10MB; today, the smallest of hard disks are measured in multiple gigabytes. Even though hard drives are getting larger, the records we need to store (pictures, recordings, music, etc) appear to keep pace with that gradual growth which makes even the present large disks appear to be too small to store everything. (Sharma & Gupta, 2017). One procedure to utilize storage capacity more ideally is to compress the files. By exploiting overt repetitiveness such as double files, we might have the option to "truncate" the items in such a method for occupying less room, however, keep up with the capacity to recreate a full form of the first when required (Singh & Meenakshi, 2014).

There are two main types of compression: lossless and the irretrievable format; lossy (Gupta & Khanduja, 2017). There are two main implementations of lossless data compression. The most popular is the Lempel-Ziv implementation, which is mostly used to compress GIF files. A

difference between this and the Huffman implementation is that a hash table is not used to implement this since dictionaries are commonly used.

Encoding and decoding of Huffman codes using a greedy algorithm compresses sourced data as well as lossless transfer between the transmitter and receiver (Muthuchamy, 2018) .It uses the Binary Tree idea as its foundation for data compression (Huffman, 1952). This data compression approach compresses message bits by constructing binary trees, where inputs are presented as symbols with matching probabilities. In this case, each frequency measured is allocated as a node in a tree structure that represents data.

Methodology

High Level overview

- Read a file (text) of unknown length and store in string format.
- Output an encoded file of the processed text
- Output a decoded file of the encoded file

Pseudocode

The algorithms used are implemented in the code files. The pseudocode to implement this is as follows:

- Text in String format is read from a file.
- A hash map is then created , with the frequency of each string character as the value and the character itself as the key.
- We then establish a priority queue (reverse) and create a single-binary node from each element in the map.

- The queue is then traversed until it is empty.
- For every traversal, the two nodes with the highest “priority” are removed.
- An internal node is then created, with the total equal to the extracted nodes' frequency count.
- A new binary tree with its children removed and the internal node as its parent is created.
- The new internal node is then inserted back into the priority queue
- This is repeated until a single node is left
- A tree like structure is formed (called Huffman tree). Traversing the tree generates the Huffman codes for each symbol.

Data Structures Used

Array: The array is a linear data structure.

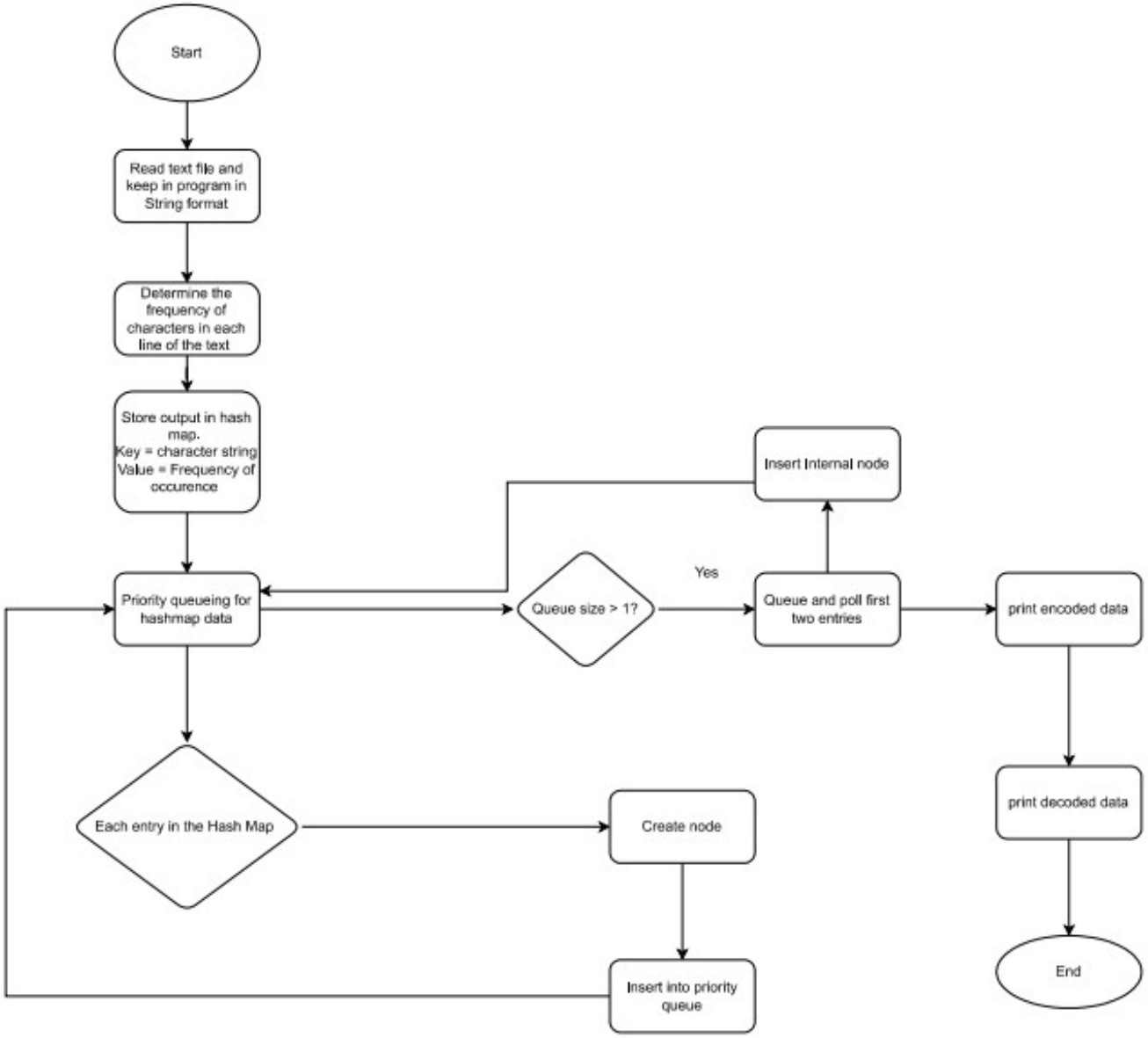
Priority Queues: This is an abstract data type of the linear data structure: queue.

Binary Trees: This is a type of a non-linear data structure: tree.

Hash Map: This is a non-linear data structure that implements the set abstract data type to map keys to values.

Descriptive Flowchart

The program execution sequence is described by this flowchart below:


Start
Read text file and keep in program in String format
Determine the frequency of characters in each line of the text
Store output in hash map.
Key = character string
Value = Frequency of occurence
Priority queueing for hashmap data
Each entry in the Hash Map
Create node
Insert into priority queue
Queue size > 1?
Yes
Insert Internal node
Queue and poll first two entries
print encoded data
print decoded data
End

Results

We expect that given the source or where the file can be located, the file will be compressed into a smaller size than its original to occupy a small section of the computer memory. We also expect that the compressed file will contain a copy of the files from the original folder yet smaller in size. There may be issues with the unknown length of the file. The limits to which this algorithm is efficient must be observed.

Conclusion

Research made from the CS106B website informs us that the Huffman trees is the best technique to compress and decompress files due to its lossless image compression (Huffman, 1991). Alternative encoding techniques such as byte pair encoding can be implemented to encode files in compressed folders more efficiently(Shibata et al., 1999). In this project we used the data structures Arrays Priority Queues, Binary trees and HashMap. The HashMap was used to store key-value pairs for each character entered in a file and its frequency. The binary tree will have the sum of the frequencies as the root and the child nodes having the individual frequencies of the texts or strings to be converted. The array acted as a placeholder for split strings of encoded or decoded data. The priority queue was used in the implementation of the Huffman tree.

References

Brar, R. S., & Singh, B. (2013). A survey on different compression techniques and bit reduction algorithm for compression of text data. *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE) Volume*, *3*.

Girardot, M., & Sundaresan, N. (2000). Millau: an encoding format for efficient representation and exchange of XML over the Web. *Computer Networks*, *33*(1-6), 747–765. https://doi.org/10.1016/s1389-1286(00)00051-7

Gupta, A., Bansal, A., & Khanduja, V. (2017). Modern lossless compression techniques: Review, comparison and analysis. *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. https://doi.org/10.1109/icecct.2017.8117850

Huffman, D. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, *40*(9), 1098–1101. https://doi.org/10.1109/jrproc.1952.273898

Huffman, D. (1991). *Background on huffman coding*. CS106B. Retrieved 2022, from https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1224/resources/huffman.html

Muthuchamy, K. (2018). A study on various data compression types and techniques. *International Journal of Research and Analytical Reviews*, *5*.

Sharma, N., & Gupta, S. K. (2017). A Huffman Algorithm Optimal Tree Approach for Data Compression and Decompression. *International Journal of Engineering Applied Sciences and Technology*, *2*(3), 38–44. https://www.ijeast.com/papers/38-44,Tesma203,IJEAST.pdf

Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., & Arikawa, S. (1999). Byte pair encoding: a text compression scheme that accelerates pattern matching.

In *CiteSeerX*. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.46.4046&rep=rep1&type=pdf

Singh, A., Tech, M., & Meenakshi Garg, E. (2014). Research Paper on Text Data Compression Algorithm using Hybrid Approach. *International Journal of Computer Science and Mobile Computing*, *3*(12), 1–10.