Project Proposal

1 Project Background

In modern data systems, the escalating speed of data generation makes the compression of data more significant [1]. This technological approach proves especially valuable in diminishing the data size of expansive datasets, facilitating data transfer between devices, and enhancing the efficiency of archival processes for legacy data.

The primary metric in data compression is the compression ratio [2], a key indicator that predominantly hinges on the selected compression algorithm and the characteristics of the data. The compression ratio represents the extent to which the original data size is reduced after compression. Choosing the most suitable algorithm is crucial, as the effectiveness of compression is tied to the inherent characteristics of the data being processed.

One crucial characteristic influencing data compression is the degree to which the data is sorted [3]. For instance, time data, such as a sequence of timestamped events, tends to be mostly sorted. On the other hand, an example of unordered data could be random sensor readings, where no inherent order or pattern exists. By systematically exploring the performance of different algorithms across sorted, the study aims to provide insights into the optimization of compression strategies based on the sortedness of the data.

2 Project Content

- Collecting a bunch of compression algorithms and their implementations, including huffman encoding[4], arithmetic encoding[5], run-length encoding[6], Lempel-Ziv[7], Frequent Value Compression[8], BDI compression[9]. Learning the mechanism of these compression algorithms
- Building a framework that can convert a list of integers to the input format of each compression algorithm, and can execute on different data.
- Using a data generator to generate integers with different sortedness. Benchmarking collected algorithms to get the compression ratio, figure out the most suitable algorithm, and figure out the reason theoretically.
- Trying to improve existing algorithms on certain sortedness. Or trying to build an adaptive framework, which can detect the sortedness of the input data and automatically choose the optimal compression algorithm.

3 Timeline For The Project

Checkpoint	Estimated Finish Date
Collecting Algorithms	3/3
Learning Compression Theories	3/10
Building framework	3/22
Testing Algorithms	4/01
Analyzing Experiment Results	4/10
Trying improvement	4/22

4 Reference

[1] Roth, M.A. and Van Horn, S.J., 1993. Database compression. *ACM sigmod record*, *22*(3), pp.31-39.

[2] Wikipedia Contributors (2019). *Data compression*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Data_compression.

[3] Ben-Moshe, S., Kanza, Y., Fischer, E., Matsliah, A., Fischer, M. and Staelin, C., 2011, March. Detecting and exploiting near-sortedness for efficient relational query evaluation. In *Proceedings of the 14th International Conference on Database Theory* (pp. 256-267).

[4] Wikipedia Contributors (2019). *Huffman coding*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Huffman_coding.

[5] Wikipedia. (2024). *Arithmetic coding*. [online] Available at: https://en.wikipedia.org/wiki/Arithmetic_coding [Accessed 23 Feb. 2024].

[6] Wikipedia. (2022). *Run-length encoding*. [online] Available at: https://en.wikipedia.org/wiki/Run-length_encoding.

[7] Wikipedia. (2021). *LZ77 and LZ78*. [online] Available at: https://en.wikipedia.org/wiki/LZ77_and_LZ78.

[8] Yang, J., Zhang, Y. and Gupta, R., 2000, December. Frequent value compression in data caches. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture* (pp. 258-265).

[9] Pekhimenko, G., Seshadri, V., Mutlu, O., Gibbons, P.B., Kozuch, M.A. and Mowry, T.C., 2012, September. Base-delta-immediate compression: Practical data compression for on-chip caches. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques* (pp. 377-388).