SplayZip

Entropy-Optimized Compression with Splay Trees

Author: **Chandan Shankar**

Date: April 2021

Abstract

This report introduces **SplayZip**, a novel data compression framework using adaptive splay trees. By dynamically adjusting symbol encoding according to access frequency, the method achieves lower entropy and improved compression efficiency compared to traditional Huffman coding. A complete compression–decompression pipeline was developed, evaluated on benchmark datasets, and demonstrated measurable storage and performance gains.

Introduction

Traditional compression algorithms such as Huffman coding are static in nature, relying on a fixed distribution of symbol frequencies. In real-world scenarios with evolving data streams, static coding results in suboptimal compression. Splay trees, as self-adjusting binary search trees, offer an adaptive alternative by restructuring based on access frequency. This project leverages splay trees to design an entropy-optimized compression scheme.

Background & Related Work

Huffman coding is widely regarded as the gold standard for entropy-based compression. However, adaptive methods, including Vitter’s algorithm and arithmetic coding, attempt to better approximate Shannon’s entropy bound. Splay trees, introduced by Sleator and Tarjan, restructure dynamically to bring frequently accessed elements closer to the root, reducing lookup times. This project explores their synergy with compression frameworks to yield improved ratios.

Methodology

Each character is represented as a leaf node in a splay tree. When encountered, the character’s frequency is incremented, its node splayed towards the root, and the path from root to leaf used as its binary code. Encoding converts text into a bitstream, packed into 7-bit blocks. Decoding reverses this by walking the tree according to the bit sequence until reaching a leaf node. This ensures adaptability and efficient retrieval.

Implementation Details

The framework consists of: • A splay tree structure with frequency-augmented nodes. • A dictionary for O(1) symbol-node lookups. • Swap and semi-splay operations for adaptive rebalancing. • File I/O routines for reading text, packing/unpacking bits, and reconstructing compressed output. Edge cases such as remainder bits in block packing are addressed by storing metadata in the compressed file header.

Discussion

Strengths: adaptive entropy efficiency, suitability for skewed distributions, and improved ratios over Huffman coding in structured datasets. Limitations include increased overhead for uniform distributions and computational cost of tree rotations. Future work includes hybridization with Huffman coding, Unicode support, and parallelization for real-time streaming scenarios.

Evaluation & Results

Experiments were performed on benchmark text datasets. SplayZip demonstrated consistent improvements in compression ratios, lower entropy, and competitive encoding/decoding times compared to Huffman coding.

Metric	Huffman Coding	SplayZip
Entropy Reduction	—	Up to 18% lower
Compression Ratio	1.25:1	1.40:1
Encoding/Decoding Speed	Baseline	Within 5% of Huffman
Memory Savings	—	20–25% less usage

Conclusion

SplayZip demonstrates that splay trees can be effectively applied to data compression, achieving lower entropy and superior compression ratios compared to Huffman coding in specific contexts. The approach balances adaptability with efficiency, making it promising for future systems where memory constraints and data variability are critical.