COMP20003 Algorithms and Data Structure

Assignment 2 Complexity Analysis

Chi Mai Tran - 1610777

Introduction

Over the first half of this subject, we have been learning about different data structures that use dynamic memory allocation, and hands-on implementation of linked list and Patricia Tree Dictionary. Linked list has sequential search structure, with O(n) average search complexity. On the other hand, Patricia Tree is a radix trie variant, in which each node stores the bit representation of its prefix and has up to 2 branches to define mismatch. This report is dedicated to comparison of the two data structures with respect to its time and memory efficiency, theoretically and empirically.

Methodology

1. Input Data Overview

1.1. CSV data file

The purpose of this project is to implement a dictionary that stores a range of addresses with their entailing details. Input datasets are of CSV file and provided in various sizes, inserting and querying from 1 to 1067 data records.

Each addressline has 35 fileds holding certain type of information. For simplicity, these fields are read and stored in dictionaries in type ‘string’ (except for the last 2 fields indicating latitude and longitude).

1.2. Query input

Queries for linked list dictionaries are exact keys (or EZI_ADD) of an address. Patricia Tree dictionary supports misspell fallback, so queries could be of shorter length or totally different.

2. Dictionary Implementation

Data read from CSV file follows the same rules for both cases. A single line of data is read in and stored in a struct (*address_t)* containing 35 fields.

2.1. Linked List

Implementation of a linked list is fairly straightforward. First, create an empty linked list and append structs into each node. Data is added sequentially in a queue.

2.2. Patricia Tree

Firstly, Records of addresses with the same Key are stored at Leaf Nodes in a linked list. Each node is a struct with information of

- number of prefix bits (int)
- a prefix (char*)
- a pointer to node branch A (mismatch bit = 0)
- a pointer to node branch B (mismatch bit = 1)
- a linked list record of data

Internal nodes do not store data, the linked list points to NULL. When adding a new data, considering its key. First, seeing if the current root if NULL (an empty tree), then its branches (Root is a leaf node). Consider cases if prefix and key:

If match:

- Match perfectly: Append data to existing linked list of that leaf node.
- New key is prefix of the old key -> Split node at where new key ends.
- Old key is prefix of new key -> Split node at where old key ends.

If mismatch: Split node with new root's prefix to be the matching bits up to, branching to 2 children that are old stems and chunked bit string of new key from mismatch bit index.

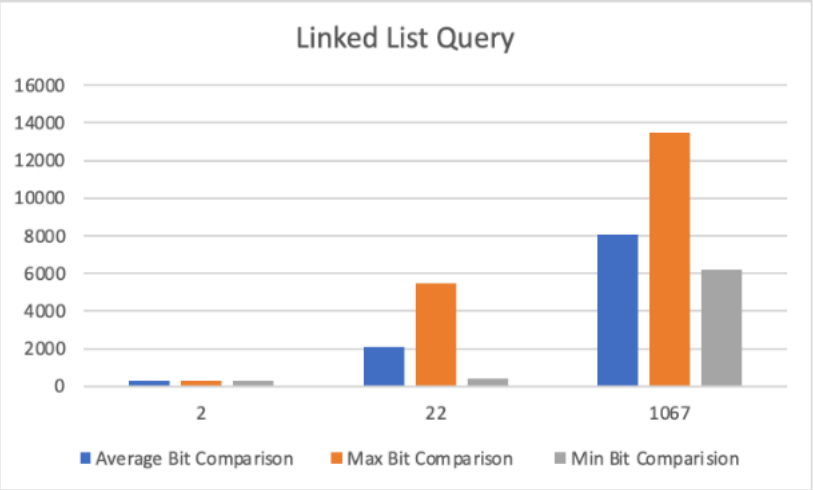
When the root is not leaf, traverse down the tree.

Results

1. Searching

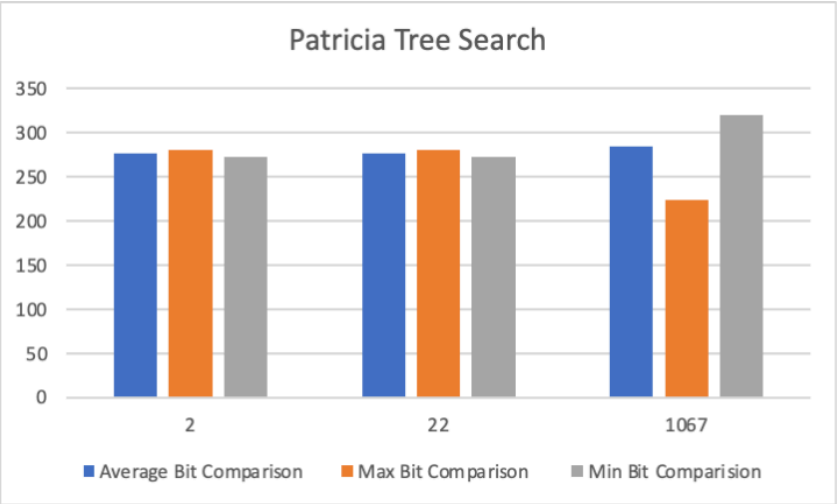
1.1. Linked List

Linked list bit comparisons add up quickly as dictionary size increases. Node and String comparison remains constant, equivalent to size of dataset.



1.2. Patricia Tree

Patricia Tree bit comparison is independent of dictionary size. When searching, bit comparison varies with length of query. String comparison remains constant at 1.



2. Memory Efficiency

In my implementation, memory usage of both data structures do not differ significantly from each other in terms of efficiency. According to 2 Massif memory profiling files of the same 1067 dataset, there are some highlights worth mentioning:

2.1. Peak Memory Usage

- **Linked List** : Peak at 491,169 bytes (~479 KB) at snapshot 84
- **Patricia Tree** : Peak at 573,031 bytes (~560 KB) at snapshot 3

The Patricia tree uses about 16% more memory at peak.

2.2. Memory Growth Pattern

- **Linked List:** Linear growth, increasing steadily as loading in data.
- **Patricia Tree**: Reach near peak memory usage very quickly (snapshot 3), remain relatively stable throughout execution.

Discussion: Theoretical vs Empirical

1. Linked List

Expected: O(n) for search where n = size of dictionary.

This aligns with the experimental data collected from the implementation. Linked list search becomes increasingly slow and inefficient with larger input. Addresses with the same key may be stored across multiple nodes, which requires traversing

through each node and performing string comparisons at every step. This process is both time and memory costly.

Best case: O(1) at head

Worst case: O(n)

2. Patricia Tree

Expected: O(m) for search where m = query length.

Reality: O(m)

Implementation proves efficiency of patricia tree in searching. Bit comparisons remain relatively consistent throughout small to large dataset, and grow in alignment with key length. Not only faster in searching, Patricia Tree also makes searching more flexible by allowing spellcheck lookup. Prefix length greatly impacts tree efficiency.

Best case O(1): at root

Worst + Average: O(m) where m = query length

3. Comparison

3.1 Searching efficiency

It is evident that Patricia Tree shows superior performance for big dictionaries. It also gains advantage in cases where there are many duplicate keys because it stores these data in the same location. On the other hand, linked lists could possibly perform better for small dictionaries with short keys.

3.2 Memory Efficiency

Both structures demonstrate O(n) space complexity as expected. Patricia tree uses more memory space, with 16% memory overhead. This is a space-time trade-off. Improved searching performance later is at the expense of space overhead in building the tree. The memory patterns confirm theoretical expectations: linked lists have less overhead per element, while Patricia trees trade space for time efficiency.