Assignment 2 Stage 3: Complexity Analysis Report

Introduction

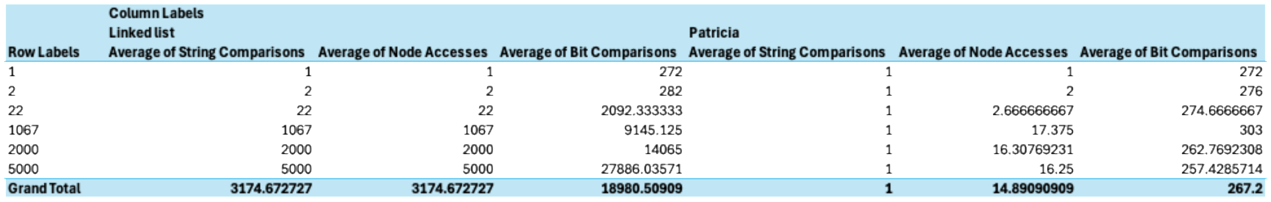
This report will analyse the complexity of two dictionary implementations; one achieved using the data structure Linked list in Stage 1 and the other using Patricia tree in Stage 2. Both dictionaries have the functionality to store address records from the Vicmap Address dataset, and support exact lookup of the data records by the key EZI_ADD by taking input queries from stdin. Additionally, the Patricia tree dictionary has a spellchecking logic that can return a closest match when an exact match cannot be found.

In theory, since linked list stores each record linearly, searching for a matching key would require traversing the entire list, meaning the time complexity for all operations for linked list is O(n). On the other hand, Patricia tree stores records by creating a prefix tree for the key and splitting at the bit level, so the complexity will depend on the bit comparisons and length of key instead of the number of records. For exact matches, the tree will take O(k) for bit comparisons, where k is the number of bits compared from the key, and O(log(n)) for node accesses. For closest match cases, the complexity is O(k+h), where h is the size of possible candidates in the mismatch subtree. This l is expected to decrease as input prefix is longer and more specific. This report will compare the expectations to the actual experiment results and statistical measurements to discuss the performance analysis for both dictionaries. For larger dataset, it is expected that the list's complexity grows linearly with the dataset, whereas the tree would outperform the list with growth depending on key lengths.

Linked list and Patricia tree Experiments

For examining the complexity and evaluating the efficiency of the two dictionaries, the programs output the counts of bit, node, and string comparisons to support statistical measurements and results. The additional dataset files were retrieved and modified from the full Vicmap Address dataset. I made additional files of dataset with sizes of 2000 and 5000 records, and duplicate records. The experiment tests datasets with various sizes, key lengths, and duplicate records both from the provided dataset and additional dataset files. I have created stdin input files key search inputs including exact matches, non-existing keys, and prefixes that are short and long. For each stdout file generated, the comparison counts are recorded in the CSV file containing the statistical results. Below is a data comparison between list dictionary and tree dictionary of their average growth for bit, node, and string counts across different dataset sizes.

Exact search cost vs dataset size for list and tree dictionaries



	Column Labels					
	Linked list			Patricia		
Row Labels	Average of String Comparisons	Average of Node Accesses	Average of Bit Comparisons	Average of String Comparisons	Average of Node Accesses	Average of Bit Comparisons
1	1	1	272	1	1	272
2	2	2	282	1	2	276
22	22	22	2092.333333	1	2.666666667	274.6666667
1067	1067	1067	9145.125	1	17.375	303
2000	2000	2000	14065	1	16.30769231	262.7692308
5000	5000	5000	27886.03571	1	16.25	257.4285714
Grand Total	**3174.672727**	**3174.672727**	**18980.50909**	**1**	**14.89090909**	**267.2**

Table 1: Average string, node, and bit comparisons for dataset sizes (1, 2, …, 5000) values compared between Linked list and Patricia tree

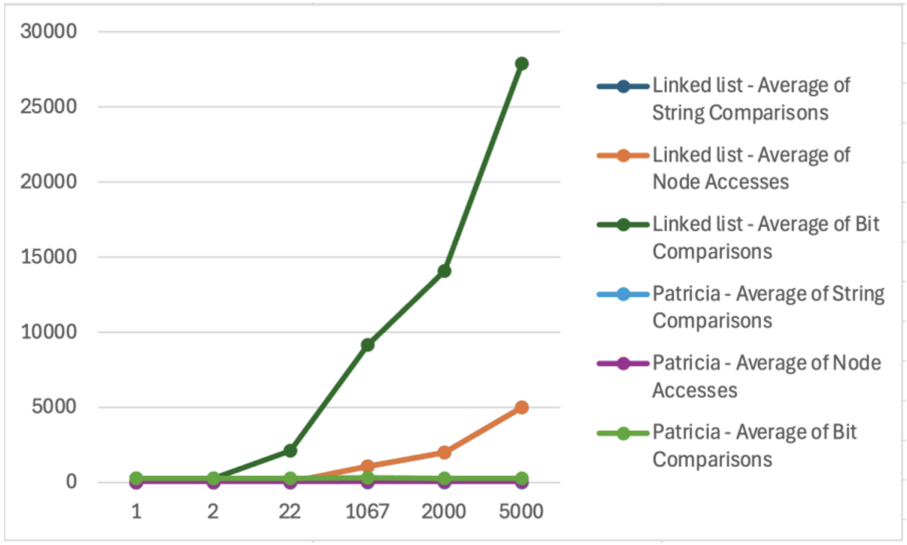
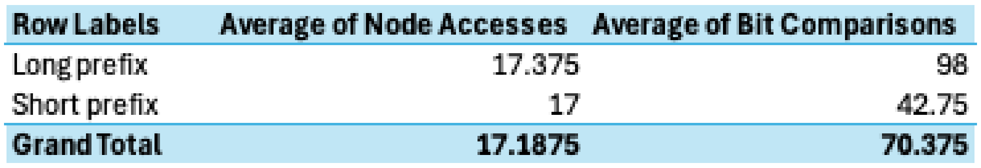


Figure 1: Line chart depicting the growth of average string, node, and bit comparisons for Linked list and Patricia tree

From Figure 1, for searching exact matches of different dataset sizes, the linked list dictionary shows a linear increase in string comparisons and node accesses as dataset grows from 1 to 5000 records. This growth matches the expected O(n) complexity where it rises linearly with increasing dataset sizes. Contrastingly, the tree dictionary has a flat trend for average bit comparisons, which is consistent with the O(k) behaviour where the growth depends on the key length instead of dataset size. For average node accesses, the growth is the slowest, remaining around 16 as dataset size grows to 5000. This matches the expectation of O(log(n)) complexity.

Average bit comparison grows the most rapidly for both dictionaries, indicating that bit operations growth is more sensitive to the growth in dataset size.

Effect of prefix length on Patricia Tree search with dataset size 1067.



Row Labels	Average of Node Accesses	Average of Bit Comparisons
Long prefix	17.375	98
Short prefix	17	42.75
Grand Total	**17.1875**	**70.375**

Table 2: Average node accesses and bit comparisons for long and short input key for tree dictionary lookup

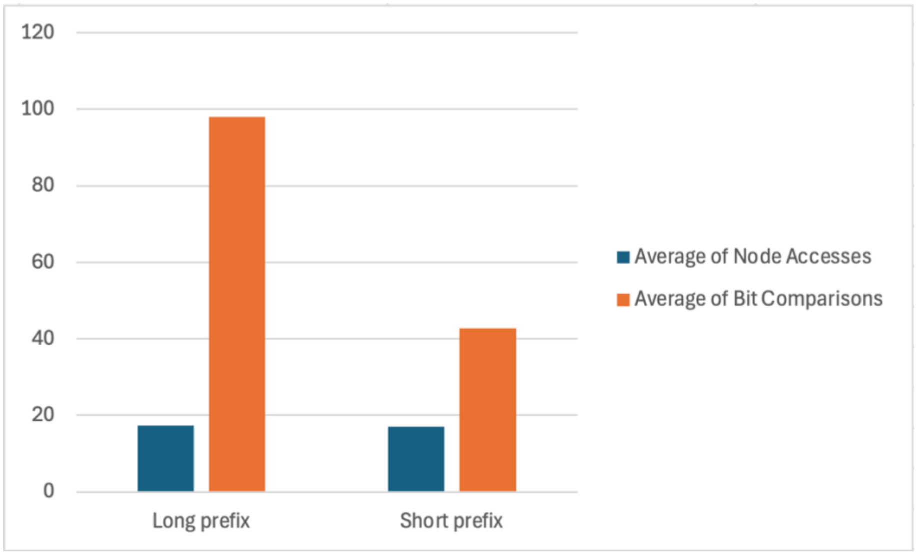


Figure 2: Column graph to compare average node accesses and bit comparisons for long and short input key for tree dictionary lookup

From Figure 2, longer prefix search key for closest search in tree dictionary causes more bit comparisons than short prefix since more bits are compared along the search path, which is expected from the complexity O(k). This is to confirm that the cost for tree dictionary is dominated by prefix key length locally and not dataset size. However, average node accesses are similar for both long and short prefix, differing only by around 0.3. This indicates that for this dataset, the candidate keys to choose from are small, so the need to search further is little. This does not fully support the expected effect of longer search key reducing the search complexity.

Discussion

For Linked list dictionary implementation, it was expected that the search process has O(n) complexity since the search can be carried throughout the whole list of linearly stored records. From the results, we can see matching behaviour as increasing the dataset size from 1 to 5000, and the average comparisons growing from 1 to 5000. This means that the complexity grows with dataset record size, explaining why node accesses and string comprisons count shows dramatic growth in Figure 1. For Patricia tree dictionary implementation, the theoretical analysis predicts that the node access would be O(log(n)), and bit comparison is O(k). The decreased complexity results from that it depends on the length of key instead of the number of records. Since the tree diverges nodes through keeping common prefixes, the searching traversal is directed by the prefix , making sure that similar keys stay close within subtrees and node accesses remains small. Looking as the results, the growth rates are indeed much slower than linked list, with string comparisons staying constant, node accesses growing to around 16, and bit comparisons increasing to around 250 when dataset rises to 5000. Additionally, for non-exact match case for tree dictionary, we verify O(k) complexity for bit comparisons since as key length increases, so does bit comparisons from Figure 2. The expectation of node accesses decreasing when key length increases is not strongly supported here, since the values for short and long prefixes are close to each other. This indicated that the mismatch node often is caught deeper down the tree when candidate keys substree is typically small. This may due to some biased choice of input search key or the structure of the dataset is not random enough. In more thoughtful cases of dataset and search key inputs, longer and more specific search keys should reduce the candidate subtree size and decrease node accesses more notably.

Considering different characteristics of the dataset, there are tradeoffs to consider when comparing the efficiency of Linked list and Patricia tree dictionaries. When dataset is small, searching behaviours are similar for list and tree dictionaries. As dataset size grows, the tree dictionary exhibits more advantages by slowing down the cost, especially when keys are long. When dataset contains duplication, list is less efficient when the tree stores duplicates at the same place. While the tree provides faster lookups, there is also space-time tradeoff to consider. Linked list is simpler to implement, requiring less memory with fewer pointers while the tree requires additional memory to support branching and duplicates.

A limitation for the experiments is that the dataset and search input examples are small and potentially biased. More thoughtful examples are needed to accurately analyse all aspects of the performances for both dictionaries. Also, there is a nuance that the comparison between the two structures based on the three counters may be imbalanced. This report's comparison is only based on the trend of growths, since the two data structures have different implementation methods for storing the records. It is noticeable that string comparison stays constant for tree, since only one key is reached with the terminate byte. So for tree dictionary, only bit comparisons and node accesses are counted and valid for statistical analysis. Meanwhile, for list, string comparison grows with node accesses, indicating that this parameter is not comparable.

To conclude, the linked list demonstrates linear growth in string comparisons with dataset size, while the Patricia tree's complexity is independent of dataset size and dominated by length and branching. The Patricia tree therefore delivers the expected advantage at the price of extra constructing time and memory, a trade-off that is favourable when query search is required often or scalability is emphasised.