**Complexity Report**

***Introduction***

This report investigates and compares the efficiency between two dictionary implementations:

1. A **linked list dictionary (Stage 1)**.
2. A **Patricia tree dictionary (Stage 2)**.

Both structures support lookups on the **EZI\_ADD** field (index 1 on the csv record), but their internal mechanisms are very different.

* The linked list preserves the order of records and searches linearly.
* The Patricia tree compresses common prefixes at the bit level, which hypothetically reduces redundant comparisons during lookups.

The goal is to evaluate their complexity in practice by running a series of queries under both **exact match** and **partial (mismatch/edit distance) match** conditions.

The metrics recorded are:

* **b**: number of bit comparisons
* **n**: number of node accesses
* **s**: number of string comparisons

I tested datasets of different sizes: **1, 2, 22, and 1067 records**. For each dataset, I ran queries and recorded the above metrics for both dictionary types.

***Note:***

* *Summary of Linked list dictionary output is stored in `dict1\_output` file*
* *Summary of Patricia tree dictionary output is stored in `dict2\_output` file*

**Results**

1. **Exact Match Results**

*The table below shows the range of values per query for exact matches. These are not cumulative totals across all queries; they represent how costs varied for different EZI\_ADD lookups on the same dataset. The same set of queries was used for both dictionaries to ensure a fair comparison.*

| **Dataset** | **Linked List (b / n / s)** | **Patricia Tree (b / n / s)** |
| --- | --- | --- |
| 1 | 272 / 1 / 1 | 272 / 1 / 1 |
| 2 | 276~286 / 2 / 2 | 272~280 / 2 / 1 |
| 22 | 406 ~ 5453 / 22 / 22 | 272~280 / 1~2 / 1 |
| 1067 | 6000~10200 / 1067 / 1067 | 272~315 / 6~24 / 1 |
|  |  |  |

**Analysis:**

* In the linked list, the program always traverses the full dataset to ensure duplicates are included and order preserved. That’s why **n** and **s** are always equal to the dataset size. The **variation** **in** **b** reflects the cumulative character-by-character checks across all entries, which depends on both query string length and dataset size. **As the dataset grows, b increases linearly.**
* In the Patricia tree, the costs stay much lower. Even for 1067 records, bit comparisons are around 272–315 and node accesses only 6–24. **This shows how** **compression of prefixes limits/reduce traversal, so per-query costs remain nearly constant with dataset size.**

1. **Partial Match Results**

*(The partial match case arises when no exact match is found. In the linked list, the program still scans the entire dataset. In the Patricia tree, once a mismatch occurs, the algorithm collects descendants at the mismatch point and applies edit distance to identify the closest candidate(s).)*

*The following table shows the* ***range of values per query*** *for partial match lookups. Just like with exact matches, these numbers are* ***per query****, not cumulative across all queries. The same set of queries was used for both dictionaries to ensure a fair comparison.*

| **Dataset** | **Linked List (b / n / s)** | **Patricia Tree (b / n / s)** |
| --- | --- | --- |
| 1 | 140~147/ 1 /1 | 140~147 / 1 / 1 |
| 2 | 40~48/ 2 / 2 | 34~42/ 2 / 1 |
| 22 | 184~1333 / 2 / 2 | 50~66 / 2~3 / 1 |
| 1067 | 7300 ~ 10000 / 1067 / 1067 | 20 ~296 / 11 ~ 19 / 1 |

**Analysis:**

* These values represent the per-query costs of partial matches.
* For the linked list, the algorithm still traverses the entire dataset for each query to ensure duplicates are captured and order is preserved. This explains why n and s always equal the dataset size. The **variation in b** comes from the number of per-character comparisons required, which **grows consistently with dataset size** — showing the expected **linear scaling**.
* For the Patricia tree, partial matches remain **very efficient**. Even at 1067 records, bit comparisons only ranged between **20 and 296**, with node accesses between 11 and 19. This is a **tiny cost compared to the linked list**, showing how mismatch handling and descendant collection prune the search space significantly.

***Note on the 2-record dataset:***

* *The 2 dataset results look smaller than the 1 dataset because the query used there was a single letter (like “P”). This naturally leads to fewer bit comparisons.*
* *Therefore, the absolute values for dataset size 2 are not directly meaningful — what matters more is the side-by-side comparison between the two structures. And in that comparison, the Patricia tree consistently requires fewer bit and string operations than the linked list.*

**Comparison with Theory**

**From theory:**

* **Linked List Dictionary**
  + Lookup in a linked list requires scanning through all nodes, since duplicates may exist and the dataset must be fully traversed to guarantee correctness.
  + This gives a worst-case and average-case complexity of **O(n)** per query, where *n* is the number of records.
  + In terms of counters:
    - **Node (n) and string (s) accesses** should scale linearly with dataset size, as every node is touched for each query.
    - **Bit comparisons (b)** should also grow linearly, since more strings are compared as the list grows.
* **Patricia Tree Dictionary**
  + Lookup follows a path determined by the bits of the query key. Each branching decision reduces the search space, so the complexity is **O(L)**, where *L* is the number of bits in the key.
  + This makes Patricia tree performance independent of dataset size and instead tied to key length (plus any descendant checks in mismatch cases).
  + In terms of counters:
    - **Bit comparisons (b)** should be bounded by the key length (including '\0'), typically under a few hundred in our datasets.
    - **Node accesses (n)** scale with the height of the tree, which grows slowly (logarithmic in practice) compared to the dataset size.
    - **String comparisons (s)** only occur during edit-distance checks in mismatch scenarios, and remain small relative to linked list lookups.

**How results align with theory:**

* For the **linked list**, the results show exactly what theory predicts:
  + n and s always equal the dataset size (e.g., 1067 nodes/strings for dataset 1067).
  + b grows proportionally with dataset size, ranging into the thousands for 1067 records, showing clear **linear growth**.
* For the **Patricia tree**, the results match the expected **key-length bound**:
  + b values stayed roughly between 20 and 315 even for dataset 1067, confirming independence from dataset size.
  + n remained very low (single to low double digits), consistent with shallow tree height.
  + s stayed close to 1 except during mismatch checks, which is consistent with only comparing candidate descendants.

Overall, the experimental data strongly supports the theoretical expectations:

* Linked lists scale linearly with dataset size, making them simple but inefficient for large datasets.
* Patricia trees achieve near-constant-time lookups tied to key length, not dataset size, which explains their efficiency for both exact and partial matches.

**Discussion**

The experiments make the trade-offs very clear:

* With small datasets, both structures perform similarly.
* As dataset size increases, the **linked list becomes increasingly costly**, with bit comparisons ballooning into the millions.
* The **Patricia tree maintains consistently low costs**, thanks to compressed paths and branching only where necessary. Even for partial matches, the cost is only slightly above exact lookups.

One interesting point is that string comparison counts (s) remain tied to dataset size in both cases. This reflects the fact that each record still needs validation, but the Patricia tree shifts the bulk of the work to efficient bit-level prefix checks.

Overall, the Patricia tree provides a **massive improvement in scalability**, demonstrating why prefix trees are preferable for large datasets with frequent lookups.

**Conclusion**

* The **linked list dictionary** scales linearly and quickly become inefficient for large datasets.
* The **Patricia tree dictionary** scales efficiently, with costs that remain nearly constant per query.
* For both exact and partial matches, Patricia trees significantly outperform linked lists, especially as the dataset grows.

These results match theoretical expectations: linear behaviour for linked lists, and compressed, near-logarithmic performance for Patricia trees.