Trie-Based Typeahead System Design

Problem:

We need a real-time, prefix-based typeahead system that returns the top 10 search suggestions for a given prefix. The data must be stored in-memory using a highly efficient structure and support 5 billion search queries per day.

Data Structure: Trie

Each node stores:

- children: dict of character -> TrieNode
- is_end: boolean indicating end of a word
- frequency: int (how often the word occurs)
- top_k: list of top suggestions for that prefix (min-heap or sorted list)

Time to Traverse a Subtree:

- Prefix traversal: O(k), where k is length of prefix
- Fetch top suggestions: O(1), since stored at each node

Top Suggestions per Node:

Yes. Store a top-k list (size 10) at each node. Use a min-heap or sorted list. Update this list on insertions and frequency changes.

Building the Trie:

1. Offline: Parse search logs, insert each term with its frequency.
2. Parallelize using prefix shards.
3. For each insertion, update top_k at every node in the path.

Updating the Trie for 5B Searches/Day:

- Buffer updates with Kafka or a message queue.
- Aggregate counts with stream processors (e.g., Flink).
- Update the Trie asynchronously or periodically.
- Use approximate counters like Count-Min Sketch for performance.

Updating Frequencies:

- Aggregate counts periodically.
- Traverse affected prefix path and update frequency + top_k at each node.
- Run in background to not affect latency.

Removing a Term:

- Traverse Trie recursively to find and delete term.
- Remove the term from top_k lists of affected ancestor nodes.
- Delete empty child nodes to clean up.

Storing the Trie:

- Use Protobuf or FlatBuffers for compact serialization.
- Use DAWG for minimizing memory if updates are rare.
- Store snapshots + write-ahead logs (WAL) for durability.

Count-Min Sketch:

- Probabilistic data structure for frequency approximation.
- Uses d hash functions over w-sized arrays.
- On insert: increment counters at hash positions.
- On query: return min of the d counter values.
- Fast, low memory, suitable for high-throughput systems.
- Overestimates, but never underestimates.