Ruben Rehal

Project 3 : Hierarchical Text Classification

Design Document

Class Design

Node Class:

The Node class represents an individual unit in the trie. Each Node encapsulates a label, a terminal flag indicating if the node is a final classification, and an array of 15 pointers to child nodes. The terminal flag distinguishes terminal classifications from internal ones, ensuring that classifications cannot terminate at intermediate levels unless explicitly defined as terminal. The child pointers enable dynamic growth of the trie, constrained by the maximum of 15 child nodes per parent.

Key Design Decisions:

- Encapsulation: All members are private to ensure that external classes cannot directly modify node attributes, which maintains integrity. The Trie class handles all node interactions to centralize logic and simplify the interface.
- Dynamic Memory Allocation: Child nodes are dynamically allocated when needed, optimizing memory use by creating nodes only for valid classifications.
- Destructor Implementation: The Node destructor recursively deletes all child nodes, ensuring no memory leaks when the trie is cleared or destroyed.

Trie Class:

The Trie class is the primary structure for managing classifications. It organizes the nodes into a hierarchy and provides public methods for inserting, deleting, classifying, and printing classifications. The Trie also manages memory through recursive cleanup functions and maintains metadata such as the total number of terminal classifications [1].

Key Design Decisions:

- Encapsulation:
    - Private members include the root node, helper functions (e.g., place, clear), and metadata (e.g., itemcount), ensuring external code interacts with the trie only through defined public methods.
    - Public methods include operations such as insert, classify, and erase, providing a controlled interface for user interactions.
- Error Handling with Exceptions: Input validation ensures all classification strings contain only lowercase letters, catching invalid inputs with a custom illegal_exception class.
- Efficient Traversal: The fixed array of 15 child pointers per node simplifies navigation and ensures predictable memory allocation, adhering to the problem's constraints.

Function Design

insert (std::string classification) : The insert function adds a classification to the trie by invoking the private place helper function. First, it validates the input string to ensure it contains no uppercase

characters, throwing an illegal_exception for invalid inputs. Then, place splits the classification string into comma-separated labels and traverses the trie level by level. At each level, it checks if the label exists among the children. If the label is found, traversal continues to the next level. If the label does not exist, a new node is created in the first available child slot. Finally, the terminal flag is set for the last node, indicating a complete classification[1].

classify(std::string input) : The classify function traverses the trie to refine a text input into a specific classification. Starting at the root, it collects the labels of all child nodes at the current level. These labels are passed to an external classifier function, which selects the next node to explore. The process repeats at each level until either a terminal node is reached or the classifier cannot refine further.

erase(std::string classification) : The erase function removes a classification path from the trie. It begins by traversing the trie based on the provided classification string. Once the terminal node is located, it is deleted, and the parent pointer is updated to nullptr. If the parent node becomes a leaf and is not terminal, it too is removed recursively. This cleanup ensures no orphaned nodes remain in the trie.

print( ) : The print function outputs all classifications in the trie. Using a recursive helper function, it traverses the trie from the root, constructing classification paths by concatenating node labels. When a terminal node is reached, the constructed path is added to the result string. The function ensures all classifications are printed in the specified comma-separated and underscore-separated format.

Run Time

insert : The place function has a time complexity of $O(n)$, where n is the number of labels in the classification string. This is because the function processes each label sequentially, performing a fixed amount of work (checking up to 15 children) at each level. The worst case occurs when the classification string is long and requires traversing or creating nodes at every level [1].

classify : The runtime of classify is $O(N)$, where N is the total number of classifications in the trie. In the worst case, the function must explore every node to build the list of child labels at each level. However, with a well-structured trie and classifier, the function typically terminates much earlier, depending on the depth of the classification hierarchy.

erase : The erase function runs in $O(n)$, where n is the number of labels in the classification string. Traversal occurs once to locate the terminal node, and deletion propagates back up the hierarchy only for nodes that become leaves. The function performs a fixed amount of work at each level, making the runtime proportional to the depth of the classification.

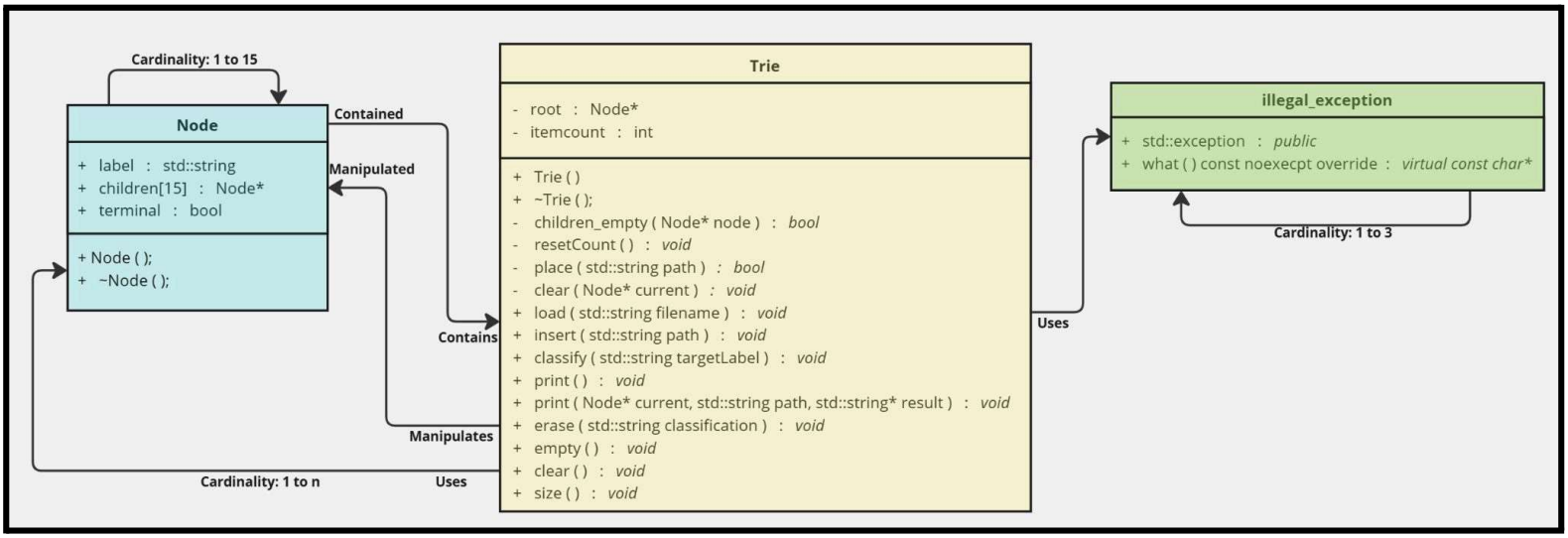
print() : The print function has a time complexity of $O(N)$, where N is the number of terminal classifications. Each terminal node is visited exactly once to construct its path, and intermediate nodes are processed during traversal. The recursion ensures no node is revisited, making the function efficient and scalable.

empty() : The empty function operates in $O(1)$ time. Both conditions—checking if root is nullptr and evaluating itemcount—are constant-time operations since they involve accessing a single memory location. This ensures that the function quickly determines whether the trie is empty.

clear() : The clear function has a runtime of $O(N)$, where N is the total number of nodes in the trie. Each node is visited exactly once for deletion, and its child pointers are checked before releasing memory. This makes the operation proportional to the size of the trie, ensuring that all resources are deallocated.

size() : The size function operates in $O(1)$ time. It directly retrieves the value of itemcount, which is a pre-computed metadata field. No traversal or additional computation is required, making this function highly efficient.

UML Diagram



References

[1] ChatGPT, paraphrasing tool. OpenAI [Online]. https://chatgpt.com/ (accessed November 17, 2024).