Project 3 Design Document
Kelly Kim
ECE 250
4 November 2024
This is the design document for ECE250 Project 3 Hierarchical Text Classification

<u>**Header Files:**</u>
Used four header files (3 created, 1 given) in total for this project keeps things organized and reduces complexity in the implementation for the Trie data structure.

trie.h: Contains all commands that the program reads
trie_node.h: This represents the individual nodes in the trie. Contains variables like className, children, and terminality.
illlegal.h: Defines the custom illegal_exception class. It handles invalid input scenarios.

<u>**Public:**</u>

• Trie() • ~Trie()	• TrieNode() • ~TrieNode()

Both constructor and destructor are <u>public,</u> allowing objects of the class to be created and destroyed outside of class. The constructor allows external code to create instances with specific parameters. The destructor cleans up the resources when they are done using.

• void load (const std::string& filename) • void insert (const std::string& classification) • void classify (const std::string& input) • void erase (const std::string& classification)	• void print() • void empty() • void clear() • bool exit();

These functions are public because they provide the main ways to interact with the Trie. It allows the user to load data, insert or erase classifications, classify input, et cetera. By keeping these functions public, it is easier to use and ensures the Trie remains functional and user-friendly while keeping internal details hidden.

load function will have a void return type, always printing out “success”.
insert/erase function will have a void return type, printing out “success”, “failure” or “illegal argument” depending on their conditions.
classify function will have a void return type, printing out the classifications or “illegal argument”
print function will have a void return type, printing out list of classifications or “trie is empty” depending on its condition.
empty function will have a void return type, printing out “empty 0” or “empty 1” depending on the trie condition.
size function will have a void return type, printing out the size of the trie.
clear function will have a void return type, always printing out “success”.
exit function will be used to control when the program stops. Returning true for the program to exit, returning false for the program to keep going.

- bool exitsign = true;
- bool terminal
- std::string className
- std::vector<TrieNode*> children

exitsign variable changes to false when the exit function is called to exit the program.
terminal variable will be used to indicate if the node is a terminal node or not.
className variable will be used to store the classification label.
children will be a vector of pointers to the child nodes of the current node.

Helper Functions

● void clean (TrieNode* node) ● void printPrint (TrieNode* node, std::string currentPath, std::string& result)	● void invalid (const std::string& input)

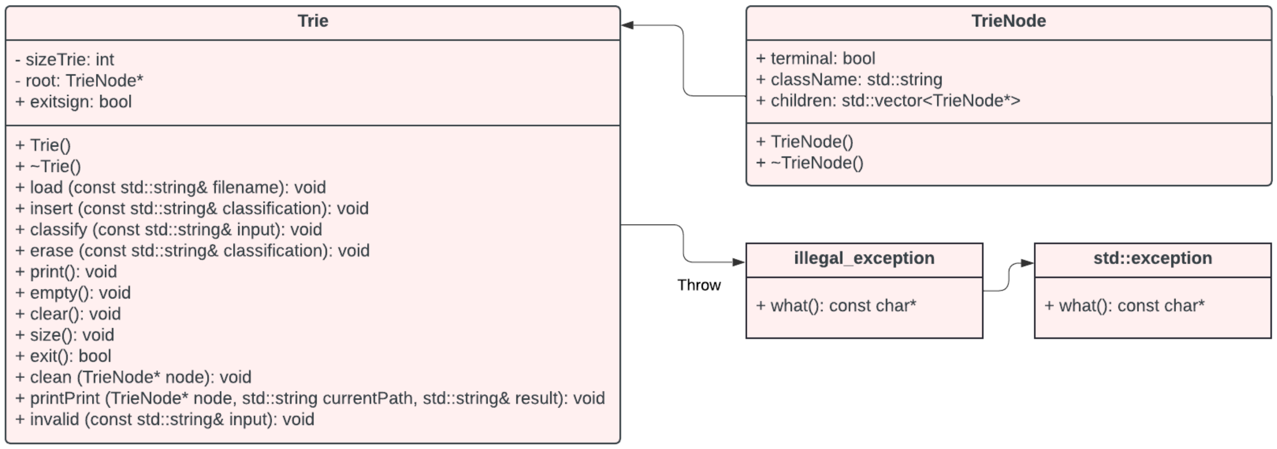
These are helper functions that will be used by main commands.
clean function will be used to recursively delete all nodes in the Trie, clearing its memory. This function will be called by the destructor as well as the clear() function.
printPrint function will travel the Trie and return "path" into a single string output. This will be called by the print() function. It will be easier to format, keeping the command print() function clean.
invalid function will be checking if the input contains any uppercase letters. If the input does contain upper case letters, it will throw illegal_exception. This will be called by insert(), classify(), and erase() to validate.

<u>**Private:**</u>

- TrieNode* root
- int sizeTrie

These two variables will be private because the user should not be directly accessing or modifying them.
root variable will point to the root node of the Trie. It will be a starting point for most operations.
sizeTrie variable will track the size of the Trie. It should only be modified in Trie class (load(), insert(), erase() function).

UML DIAGRAM:


Trie
- sizeTrie: int
- root: TrieNode*
+ exitsign: bool
+ Trie()
+ ~Trie()
+ load (const std::string& filename): void
+ insert (const std::string& classification): void
+ classify (const std::string& input): void
+ erase (const std::string& classification): void
+ print(): void
+ empty(): void
+ clear(): void
+ size(): void
+ exit(): bool
+ clean (TrieNode* node): void
+ printPrint (TrieNode* node, std::string currentPath, std::string& result): void
+ invalid (const std::string& input): void
TrieNode
+ terminal: bool
+ className: std::string
+ children: std::vector<TrieNode*>
+ TrieNode()
+ ~TrieNode()
Throw
illegal_exception
+ what(): const char*
std::exception
+ what(): const char*

Running Time:
→ n: number of class in the classification
→ N: number of classes in the trie

Insert **O(n)**	**First while loop:** The classification string is divided into classNames using commas as separators. ● *Let's say there are 'n' class names*. There is a **for loop that iterates 'n' times to check**

	if the node already exists. In both situations (node already exists, or insertion is needed), it is a **constant time operation because the number of child nodes is fixed** at 15. ● Since the function processes each of 'n' class names once, and each operation within this loop takes constant time $O(n) \times O(1) = O(n)$ **Second while loop:** This is used to update the terminality of the node ● Like the first while loop, even though there is a for loop within the while loop, since the number of child nodes is fixed, it will take constant time. $O(n) \times O(1) = O(n)$
Classify **O(N)**	**While loop:** The loop iterates from the root node to a terminal node. If the depth of the trie is 'N', the loop runs 'N' times. ● **For loop**: There are two for loops inside the while loop. **Both for loops are iterating 'n' times, where n is the number of children. Since each node has a fixed** maximum of 15 children, this is a constant time operation $O(1)$ $O(N) \times O(1) = O(N)$
Erase **O(n)**	**While loop:** Processes each class name in the classification ● **For loop**: It is iterating 'c' times (where c is the number of children), to find the matching child. Since the number of children has a fixed maximum of 15 children, it is **constant** $O(1)$**.** $O(n) \times O(1) = O(n)$ **For loop:** Iterating 'c' times to check if there any other children ● **Since each node has a fixed** maximum of 15 children, this is a constant time $O(1)$ **For loop:** The function iterates over the path vector in reverse order ● The path has all the nodes visited stored. If the classification contains 'n' class names, the path vector will contain 'n' nodes ● The **loop runs for the total number of nodes in the path vector** $O(n)$ $O(n) + O(1) + O(n) = O(n)$
Print **O(N)**	Print function calls the **helper function printPrint** ● First checks if it is nullptr $O(1)$ ● **For loop:** Iterating 'c' (c=number of children) times to check if the current node has any children. **Since the number of children has a fixed maximum, it is constant.** $O(1)$ ● Checks if the current node is terminal and adds to its output string depending on the condition ● **Recurvice call** → $O(N)$**)** $O(1) + O(1) + O(N) = O(N)$
Empty **O(1)**	The function either prints out "empty 0" or "empty 1" depending on the size of the Trie ● **If statement, simple comparison** → $O(1)$ $O(1)$
Clear **O(N)**	Clear function calls the **helper function clean** ● Checking if it is nullptr → $O(1)$ ● **For loop:** Recursively deleting the node. Since the number of children has fixed maximum, time is constant → $O(1)$ ● If the trie contains 'N' nodes, the **total number of recursive calls made by clean function is 'N'** → $O(N)$ $O(N)$
Size **O(1)**	**Simple print statement** → $O(1)$ $O(1)$