Trie and Ternary Search Tree Implementation Report

Tousif

November 13, 2023

1 Trie Implementation

1.1 Overview

The Trie data structure is used to compile an alphabetical index of words and the lines on which they appear. The implementation includes the following functions:

- `Trie()`
- `void insert(const string& word, int line)`
- `void printIndex()`
- `void printIndexHelper(TrieNode* node, string wordSoFar)`
- `~TrieNode()`
- `~Trie()`

1.2 Function Details

1. `Trie()`

   Description: Constructor for the Trie class. Initializes the root node of the Trie.

   Time Complexity: $O(1)$

2. `void insert(const string& word, int line)`

   Description: Inserts a word and the line number into the Trie.

   Time Complexity: $O(L)$, where L is the length of the word.

3. `void printIndex()`

   Description: Prints the alphabetical index using the Trie.

   Time Complexity: $O(N)$, where N is the total number of nodes in the Trie.

4. `void printIndexHelper(TrieNode* node, string wordSoFar)`

   Description: Recursive helper function for printing the alphabetical index.

   Time Complexity: $O(N)$, where N is the total number of nodes in the Trie.

5. `~TrieNode()`

   Description: Destructor for the TrieNode class. Frees the memory allocated for the TrieNode and its children.

   Time Complexity: $O(1)$

6. `~Trie()`

   Description: Destructor for the Trie class. Calls the destructor for the root TrieNode, freeing all memory.

   Time Complexity: $O(N)$, where N is the total number of nodes in the Trie.

2 Ternary Search Tree (TST) Implementation

2.1 Overview

The Ternary Search Tree (TST) data structure is used for the same purpose as the Trie. The implementation includes the following functions:

- `TernarySearchTree()`
- `void insert(const string& word, int line)`
- `void printIndex()`
- `void printIndexHelper(TSTNode* node, string wordSoFar)`
- `~TSTNode()`
- `~TernarySearchTree()`

2.2 Function Details

1. `TernarySearchTree()`

   Description: Constructor for the TernarySearchTree class. Initializes the root node of the TST.

   Time Complexity: $O(1)$

2. `void insert(const string& word, int line)`

   Description: Inserts a word and the line number into the TST.

   Time Complexity: $O(L)$, where L is the length of the word.

3. `void printIndex()`

   Description: Prints the alphabetical index using the TST.

   Time Complexity: $O(N)$, where N is the total number of nodes in the TST.

4. `void printIndexHelper(TSTNode* node, string wordSoFar)`

   Description: Recursive helper function for printing the alphabetical index.

   Time Complexity: $O(N)$, where N is the total number of nodes in the TST.

5. `~TSTNode()`

   Description: Destructor for the TSTNode class. Frees the memory allocated for the TSTNode and its children.

   Time Complexity: $O(1)$

6. `~TernarySearchTree()`

   Description: Destructor for the TernarySearchTree class. Calls the destructor for the root TSTNode, freeing all memory.

   Time Complexity: $O(N)$, where N is the total number of nodes in the TST.

3 Memory Management

3.1 Trie

3.2 TST

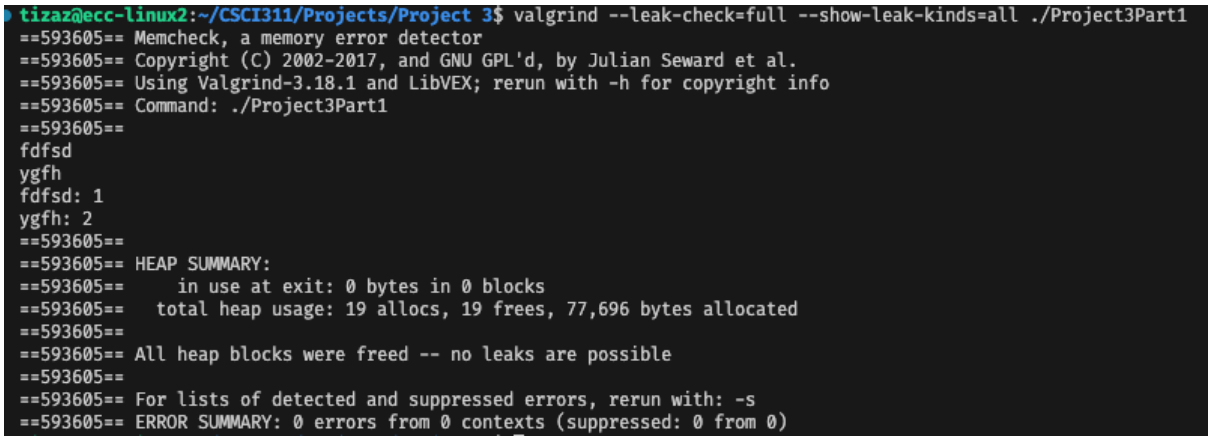

Figure 1: Valgrind output for Trie

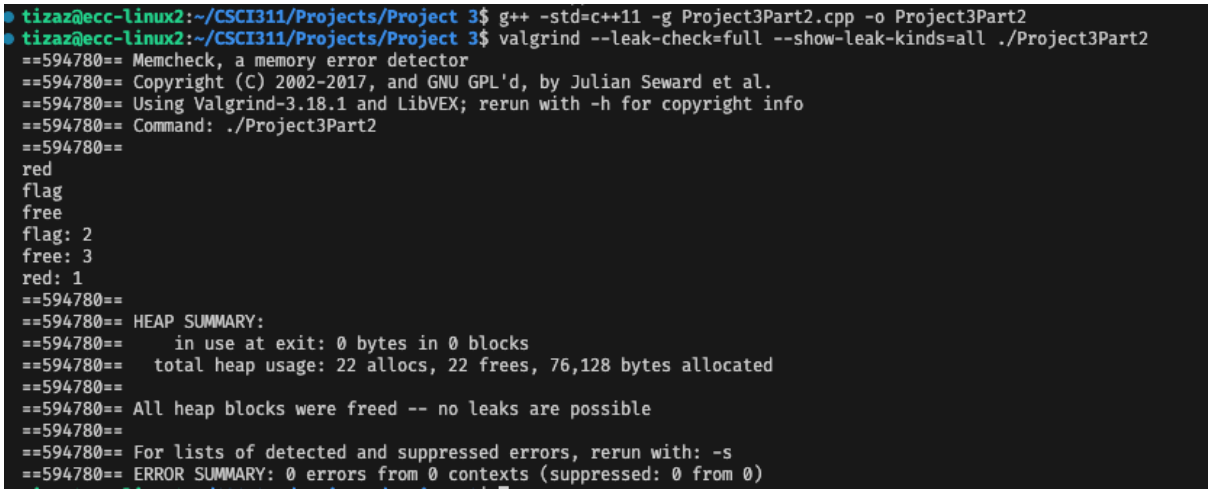

Figure 2: Valgrind output for TST