Project 3 of CSCI 311:
Indexing a Book using a Trie AND a Ternary Search Tree

https://github.com/amacharla15/Indexing-a-Book-using-a-Trie-AND-a-Ternary-Search-Tree.git

Objective

The goal of this project is to implement an alphabetical index of words appearing in a text file and the respective line numbers where they occur. The index was built using two data structures:

1. **Trie**
2. **Ternary Search Tree (TST)**

Implementation

1. Trie

A Trie is a tree-based data structure used to store a dynamic set of strings. Each node represents a single character of a word, and paths in the Trie represent words.

Node Structure

Each node in the Trie contains:

- An array of size 26 for links to child nodes (corresponding to each lowercase alphabet letter).
- A set<int> to store the line numbers where the word appears.
- A Boolean 'isEndOfWord' to indicate the end of a word.

Key Functions

- **Insert:** This function takes a word and its line number as inputs. It traverses the Trie and creates new nodes as necessary. At the end of the word, it marks the node as the end of the word and stores the line number in the set.
- **Print Index:** Recursively traverses the Trie in lexicographical order, printing each word and its associated line numbers.
- **Destructor:** A recursive function to free memory allocated for each node.

Time Complexity

- **Insertion:** O(M), where M is the length of the word being inserted.
- **Printing Index:** O(N), where N is the total number of characters across all stored words.

Space Complexity

- O(N), where N is the total number of characters across all words stored in the Trie.

---

2. Ternary Search Tree (TST)

A TST is a hybrid of a binary search tree and a Trie. Nodes are structured to store characters, with pointers for left, middle, and right children.

Node Structure

Each node contains:

- A char data field to store a character.
- A set<int> to store line numbers.
- Three pointers: left, middle, and right.
- A Boolean 'isEndOfWord' to indicate the end of a word.

Key Functions

- **Insert:** Recursively traverses the TST, inserting characters into the appropriate branches. If the word ends, the node is marked as the end, and the line number is stored in the set.
- **Print Index:** Performs an in-order traversal of the TST to print words in lexicographical order.
- **Destructor:** Frees memory by recursively deleting all nodes.

Time Complexity

- **Insertion:** O(M), where M is the length of the word.
- **Printing Index:** O(N), where N is the total number of characters across all words.

Space Complexity

- O(N), where N is the total number of nodes created for characters.

Recurrence Relations:

1. Trie (Project3Part1.cpp)

1. Insert Function

- Recurrence Relation: $T(M)=T(M-1)+O(1)$, where M is the length of the word being inserted.
- Explanation: Each recursive call processes one character, reducing the problem size by 1.
- Iterative Expansion:
  - At step M: $T(M)=T(M-1)+O(1)$
  - Substitute $T(M-1)$: $T(M) = (T(M-2)+O(1))+O(1)$
  - Continue substituting: $T(M)= T(M-k)+k \cdot O(1)$
  - Stop when $k= M$: $T(M)= T(0)+M \cdot O(1)$
  - Base case: $T(0)=O(1)$
- Time Complexity: O(M), where MMM is the length of the word.
- Space Complexity: O(C), where C is the total number of characters across all words.

2. PrintIndex Function:

- Recurrence Relation: $T(N)=26 \cdot T(N/26)+O(1)$, where Nis the number of nodes in the Trie.
- Explanation: Each node has up to 26 children, and the function recursively processes all children.
- Iterative Expansion:
  - At the top level: $T(N)=26 \cdot T(N/26)+O(1)$
  - Substitute $T(N/26)= 26^2 \cdot T(N/26^2)+26 \cdot O(1)+O(1$
  - Continue substituting: $T(N)= 26k \cdot T(26kN)+i=0\sum(k-1)26^i \cdot O(1)$
  - Stop when $N/26k=1$; K = log26N

- Total work: T(N)=O(N)

- Time Complexity: O(N), where N is the number of nodes in the Trie.
- Space Complexity: O(M), where M is the height of the Trie (equal to the length of the longest word).

3. Destructor (deleteTrie Function)

- Recurrence Relation: T(N)=26·T(N/26)+O(1), where N is the number of nodes.
- Explanation: Each node is recursively deleted after its children.
- Time Complexity: O(N), where N is the total number of nodes.
- Space Complexity: O(M), where M is the height of the Trie.

2. Ternary Search Tree (Project3Part2.cpp)

1**. Insert Function**

- Recurrence Relation: T(M)=T(M−1)+O(1) where M is the length of the word.
- Explanation: Each recursive call processes one character.
- Iterative Expansion:
  - T(M)=T(0)+M·O(1)
  - Base case: T(0)=O(1)
- Time Complexity: O(M), where M is the length of the word.
- Space Complexity: O(N), where N is the total number of nodes in the TST.

2. PrintIndex Function

- Recurrence Relation: T(N)= 3·T(N/3)+O(1),, where N is the total number of nodes.
- Explanation: Each node has three children (left, middle, right), and the function processes all of them recursively.
- Iterative Expansion:
  - At the top level: T(N)=3·T(N/3)+O(1)

    - Substitute T(N/3): $T(N)= 32 \cdot T(N/32)+3 \cdot O(1)+O(1)$
    - Continue substituting: $T(N)= 3k \cdot T(N/3k)+\sum i=0k-13i \cdot O(1)$
    - Stop when $N/3k=1$: $k= \log 3N$
    - Total work: $T(N)= O(N)$
- Time Complexity: O(N), where Nis the total number of nodes in the TST.
- Space Complexity: O(H), where H is the height of the TST (O(M), where M is the length of the longest word).

3**. Destructor (deleteTST Function)**

- Recurrence Relation: $T(N)= 3 \cdot T(N/3)+O(1)$,, where N is the number of nodes.
- Explanation: Each node is recursively deleted after its children.
- Time Complexity: O(N), where N is the total number of nodes.
- Space Complexity: O(H), where H is the height of the TST.

Memory Management:

Project3Part1.CPP (Trie):

Input1:

```
amacharla@LOANER-1FZNX33:/mnt/c/Users/amacharla/311project3$ valgrind --leak-check=full --show-leak-kinds=all ./part1 < input1.txt
==10960== Memcheck, a memory error detector
==10960== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10960== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==10960== Command: ./part1
==10960==
age 3, 4
best 1
foolishness 4
it 1, 2, 3, 4
of 1, 2, 3, 4
the 1, 2, 3, 4
times 1, 2
was 1, 2, 3, 4
wisdom 3
worst 2
==10960==
==10960== HEAP SUMMARY:
==10960==     in use at exit: 0 bytes in 0 blocks
==10960==   total heap usage: 74 allocs, 74 frees, 90,013 bytes allocated
==10960==
==10960== All heap blocks were freed -- no leaks are possible
==10960==
==10960== For lists of detected and suppressed errors, rerun with: -s
==10960== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Input2:

```
amacharla@LOANER-1FZNX33:/mnt/c/Users/amacharla/311project3$ valgrind --leak-check=full --show-leak-kinds=all ./part1 < input2.txt
==10994== Memcheck, a memory error detector
==10994== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10994== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==10994== Command: ./part1
==10994==
a 1
by 2
charles 2
cities 1
dickens 2
of 1
tale 1
two 1
==10994==
==10994== HEAP SUMMARY:
==10994==     in use at exit: 0 bytes in 0 blocks
==10994==   total heap usage: 45 allocs, 45 frees, 86,401 bytes allocated
==10994==
==10994== All heap blocks were freed -- no leaks are possible
==10994==
==10994== For lists of detected and suppressed errors, rerun with: -s
==10994== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Input3:

```
amacharla@LOANER-1FZNX33:/mnt/c/Users/amacharla/311project3$ valgrind --leak-check=full --show-leak-kinds=all ./part1 < input3.txt
==11027== Memcheck, a memory error detector
==11027== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==11027== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==11027== Command: ./part1
==11027==
a 1, 2, 3
away 2
far 2
in 2
land 2
lived 3
once 1
princess 3
there 3
time 1
upon 1
young 3
==11027==
==11027== HEAP SUMMARY:
==11027==     in use at exit: 0 bytes in 0 blocks
==11027==   total heap usage: 68 allocs, 68 frees, 90,891 bytes allocated
==11027==
==11027== All heap blocks were freed -- no leaks are possible
==11027==
==11027== For lists of detected and suppressed errors, rerun with: -s
```

Project3part2.cpp (TST):

Input1:

```
amacharla@LOANER-1FZNX33:/mnt/c/Users/amacharla/311project3$ valgrind --leak-check=full --show-leak-kinds=all ./part2 < input1.txt
==11063== Memcheck, a memory error detector
==11063== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==11063== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==11063== Command: ./part2
==11063==
age 3, 4
best 1
foolishness 4
it 1, 2, 3, 4
of 1, 2, 3, 4
the 1, 2, 3, 4
times 1, 2
was 1, 2, 3, 4
wisdom 3
worst 2
==11063==
==11063== HEAP SUMMARY:
==11063==     in use at exit: 0 bytes in 0 blocks
==11063==   total heap usage: 73 allocs, 73 frees, 82,205 bytes allocated
==11063==
==11063== All heap blocks were freed -- no leaks are possible
==11063==
==11063== For lists of detected and suppressed errors, rerun with: -s
==11063== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Input2:

```
amacharla@LOANER-1FZNX33:/mnt/c/Users/amacharla/311project3$ valgrind --leak-check=full --show-leak-kinds=all ./part2 < input2.txt
==11082== Memcheck, a memory error detector
==11082== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==11082== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==11082== Command: ./part2
==11082==
a 1
by 2
charles 2
cities 1
dickens 2
of 1
tale 1
two 1
==11082==
==11082== HEAP SUMMARY:
==11082==     in use at exit: 0 bytes in 0 blocks
==11082==   total heap usage: 44 allocs, 44 frees, 80,617 bytes allocated
==11082==
==11082== All heap blocks were freed -- no leaks are possible
==11082==
==11082== For lists of detected and suppressed errors, rerun with: -s
==11082== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Input3:

```
amacharla@LOANER-1FZNX33:/mnt/c/Users/amacharla/311project3$ valgrind --leak-check=full --show-leak-kinds=all ./part2 < input3.txt
==11098== Memcheck, a memory error detector
==11098== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==11098== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==11098== Command: ./part2
==11098==
a 1, 2, 3
away 2
far 2
in 2
land 2
lived 3
once 1
princess 3
there 3
time 1
upon 1
young 3
==11098==
==11098== HEAP SUMMARY:
==11098==     in use at exit: 0 bytes in 0 blocks
==11098==   total heap usage: 67 allocs, 67 frees, 82,163 bytes allocated
==11098==
==11098== All heap blocks were freed -- no leaks are possible
==11098==
==11098== For lists of detected and suppressed errors, rerun with: -s
==11098== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Conclusion:

Performance Analysis:

1. **Trie:**
    - The Trie provided faster insertion and retrieval operations due to its fixed child-node structure, enabling direct access based on character positions.
    - However, it consumed more memory, as each node allocated space for 26 child pointers, regardless of actual usage.
2. **Ternary Search Tree (TST):**
    - The TST offered a more memory-efficient solution, allocating memory dynamically only when new branches were required.
    - Its recursive structure, however, made insertion and retrieval slightly slower compared to the Trie.

Key Observations:

- Both the Trie and TST achieved the expected time complexities:
    - Insertion: O(M), where M is the length of a word.
    - Index Printing: O(N), where N is the total number of nodes.
- Proper memory management was ensured through recursive destructors for both data structures, validated using Valgrind to confirm no memory leaks.