CS221L PROJECT

Dictionary implementation

Muhammad Baqar Raza 2021363

Sermad Mehdi 2021578

Introduction:

The CS224 Lab project objective was to code something productive with atleast two data structures. I code a ***Dictionary program*** which is basically a Spelling checker, Word completion and suggestion program. Data structures I used are ***Tries***, ***Queue*** and some knowledge of ***Array.***

- ***trie data structure*** to store the words in a dictionary and a priority queue to store the suggestions in order of frequency.
- The program reads the words from a text file called ***"dictionary.txt"*** and inserts them into the trie. Dictionary have *1000* words.
- Then, it prompts the user to enter a word to spell check and checks if the word is present in the trie. If the word is present or not, it suggests words based on the given prefix using the ***suggestWords()*** function.
- The ***suggestWordsUtil()*** function is a recursive function that traverses the trie and stores the suggestions in the priority queue.
- ***The priority queue*** stores the suggestions in decreasing order of frequency, so the most frequent suggestions are printed first.

Code:

- Library used in the code

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <queue>
#include <vector>
#include <iomanip>

```

- Struct TrieNode to initialize all to Zero and new variables

```cpp
const int ALPHABET_SIZE = 26;

// Trie node
struct TrieNode {
  TrieNode* children[ALPHABET_SIZE];
  bool isEndOfWord;
  int frequency;

  TrieNode() {
    for (int i = 0; i < ALPHABET_SIZE; i++) {
      children[i] = nullptr;
    }
    isEndOfWord = false;
    frequency = 0;
  }
};

```

```cpp
// Trie class
class Trie {
 public:
  TrieNode* root;
  Trie() {
    root = new TrieNode();
  }

  // Insert a word into the trie
  void insert(const std::string& word) {
  TrieNode* current = root;
  for (char ch : word) {
    int index = ch - 'a';
    if (current->children[index] == nullptr) {
      current->children[index] = new TrieNode();
    }
    current = current->children[index];
  }
  current->isEndOfWord = true;
  current->frequency++;
}


```

- the function inserts the word into the trie by creating new nodes for each character.
- **frequency** counter are used to store additional information about the word.

-
- The search() function checks if a given word is present in the trie or not.
- It does this by traversing the trie and following the path of the word.
- isEndOfWord flag is set for the last node of the path
- If the path does not exist or the isEndOfWord flag is not set for the last node, it returns false

```
bool search(const string& word) {
  TrieNode* current = root;
  for (char ch : word) {
    int index = ch - 'a';
    if (current->children[index] == nullptr) {
      return false;
    }
    current = current->children[index];
  }
  return current->isEndOfWord;
}

// Suggest words based on a given prefix
void suggestWords(const string& prefix) {
 TrieNode* current = root;
 for (char ch : prefix) {
   int index = ch - 'a';
   if (current->children[index] == nullptr) {
     cout << "No suggestions" << endl;
     return;
   }
   current = current->children[index];
 }

 // Use a priority queue to store the suggestions in order of frequency
 priority_queue<pair<int, string>> pq;
 string word = prefix;
 suggestWordsUtil(current, pq, word);

std::cout << std::left << std::setw(15) << "Word" << std::setw(15) << "Frequency" << std::endl;
while (!pq.empty()) {
  std::pair<int, std::string> top = pq.top();
  pq.pop();
  std::cout << std::left << std::setw(15) << top.second << std::setw(15) << top.first << std::endl;
}
}
```

- The suggestWords() function suggests words based on a given prefix. It does this by traversing the trie prefix.
- If the path exists, it uses ***a priority queue*** to store the suggestions in order of frequency and then prints them in a table form.
- **in a priority queue, the values are removed on the basis of priority**. The element with the highest priority is removed first.

- The suggestWordsUtil() function is a recursive function that is used to suggest words based on a given prefix. It takes a **TrieNode** pointer **node**, a reference to a **priority_queue pq**, and a reference to a **string word** as input.

```

 private:
  // Recursive function to suggest words
  void suggestWordsUtil(TrieNode* node, priority_queue<pair<int, string>>& pq, string& word) {
    if (node->isEndOfWord) {
  pq.push({node->frequency, word});
  }
  for (int i = 0; i < ALPHABET_SIZE; i++) {
  if (node->children[i] != nullptr) {
  word.push_back(i + 'a');
  suggestWordsUtil(node->children[i], pq, word);
  word.pop_back();
  }
  }
  }
  };
```

```
int main() {
  // Read the words from the text file and insert them into the trie
  Trie trie;
  ifstream infile("dictionary.txt");
  string word;
  while (infile >> word) {
    trie.insert(word);
  }

  int choice;
  do {
    cout << "Menu:" << endl;
    cout << "1. Spell check a word" << endl;
    cout << "2. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;

    if (choice == 1) {
      // Spell check a given word and suggest words
      string input;
      cout << "Enter a word to spell check: ";
      cin >> input;
      if (trie.search(input)) {
        cout << "Correct spelling" << endl;
      } else {
        cout << "Incorrect spelling" << endl;
      }
      trie.suggestWords(input);
    }
  } while (choice != 2);

  return 0;
}
```

- File named dictionary open and read words. File contains 1000 words.
- Main Menu for program
- Suggest word after every input.