UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO
INE5408 - ESTRUTURAS DE DADOS

LEONARDO DE SOUSA MARQUES

PROJETO II:
Identificação de prefixos e indexação de dicionários

Docente:
Dr. Alexandre Gonçalves Silva

Florianópolis
2024

Sumário

1 INTRODUÇÃO

O seguinte relatório tem como objetivo analisar e resolver dois problemas relacionados à identificação de prefixos e à indexação de dicionários utilizando a estrutura de dados **Trie** (árvore digital). Essa estrutura, também conhecida como R-trie, é utilizada para para armazenar e pesquisar conjuntos de strings/palavras de forma eficiente, em árvore. Nesse sentido, como característica iremos ter uma raíz vazia (*root*), utilizada apenas para inicializar a estrutura e nós, tal que cada um representa um caractere (letra, para nosso problema). Com isso, ao perseguir a Trie a partir de um caractere após a raiz até o final, vamos montar uma palavra.

Em nossos problemas, vamos considerar entradas que representam dicionários. Sendo assim, em cada linha vamos ter uma palavra seguida de sua definição, conforme modelo abaixo.

1.1 Modelo de Arquivo

Abaixo, o arquivo .dic define os significados das palavras bear, bell, bid, bull, buy, sell, stock e stop.

```
1 [bear]The definition of bear is a large mammal found in America and
      Eurasia which has thick fur or a big person or a person who is
     cranky and grumpy.
2 [bell]A hollow metal musical instrument, usually cup-shaped with a
     flared opening, that emits a metallic tone when struck.
3 [bid]The definition of bid means an offer of what someone will give
      for something.
4 [bull]The definition of a bull is an uncastrated male bovine animal
     , or is slang for nonsensical and untrue talk.
5 [buy]The definition of buy means to purchase or to get by exchange.
6 [sell]Sell is defined as to exchange something for money, act as a
     sales clerk or offer for sale.
7 [stock]The definition of stock is something that is in normal
     supply or common.
8 [stop]To stop is defined as to block, close, defeat, prevent from
     moving or bring to an end.
```

Para nossos problemas, vamos armezenar apenas as palavras do dicionário, sem suas definições. Vamos estar interessados em armazenar também a posição inicial delas no arquivo e o comprimento que sua linha total ocupa.

2 PROBLEMA 1

Nosso primeiro problema será relacionado à identificação de prefixos a partir da construção da Trie. Para isso, vamos primeiro implementar a *struct* que representa um Nó na nossa

árvore, sabendo que ela tem como variáveis: posição inicial, comprimento e filhos (26, no máximo).

```
struct NoTrie {
        NoTrie* filhos[26];        // Ponteiros para filhos
        unsigned long posicao;     // Posicao no arquivo
        unsigned long comprimento; // Comprimento da linha
        bool final_da_palavra;

        NoTrie() : posicao(0), comprimento(0), final_da_palavra(
           false) {
            for (int i = 0; i < 26; ++i) {
                filhos[i] = nullptr;
            }
        }
    };
```

Vamos inicializar todas variáveis de tamanho com zero, filhos como ponteiros nulos e o booleano que representa o final da palavra como *false*. Para fazer a inserção dos caractéres foi implementado o método **inserir** na classe **Trie**. Este método percorre os caracteres da palavra, identificando o índice correspondente no vetor de filhos do nó atual, criando novos nós, caso necessário. Assim, a palavra é inserida na estrutura caractere por caractere. O trecho do código responsável por esta operação está apresentado abaixo:

```
void inserir(const string& palavra, unsigned long posicao, unsigned
    long comprimento) {
    NoTrie* curr = raiz;
    for (char c : palavra) {
        int index = c - 'a'; // retorna o valor ASCII de 'c';
        if (curr->filhos[index] == nullptr) {
            curr->filhos[index] = new NoTrie();
        }
        curr = curr->filhos[index];
    }
    curr->final_da_palavra = true;
    curr->posicao = posicao;
    curr->comprimento = comprimento;
}
```

Cada nó da Trie contém um vetor de 26 filhos, representando as letras do alfabeto em ordem. Para cada palavra inserida, o índice de cada letra é calculado subtraindo `'a'` do caractere, o que garante que `'a'` seja mapeado para 0, `'b'` para 1 e assim por diante.

Após a construção da Trie, foi implementada a funcionalidade de busca, que verifica se uma palavra ou prefixo está presente na estrutura. O método `buscar` percorre os nós da Trie

de acordo com os caracteres da palavra de entrada. Se um nó correspondente a um caractere não existir, a palavra não é prefixo.

```
void buscar(const string& prefixo) {
    NoTrie* curr = raiz;
    for (char c : prefixo) {
        int index = c - 'a';
        if (curr->filhos[index] == nullptr) {
            cout << prefixo << " is not prefix" << endl;
            return;
        }
        curr = curr->filhos[index];
    }
    unsigned long count = 0;
    contarPalavras(curr, count);
    cout << prefixo << " is prefix of " << count << " words" <<
       endl;

    if (curr->final_da_palavra) {
                cout << prefixo << " is at (" << curr->posicao
                << "," << curr->comprimento << ")" << endl;
            }
}
```

Caso o prefixo seja encontrado, o método adicional `contarPalavras` é usado para determinar quantas palavras têm o prefixo buscado. Este método realiza uma busca em profundidade (DFS) a partir do nó correspondente ao último caractere do prefixo, somando todas as palavras completas encontradas. Tal método é privado e da classe Trie.

```
void contarPalavras(NoTrie* no, unsigned long& count) {
            if (no->final_da_palavra) {
                count++;
            }
            for (int i = 0; i < 26; i++) {
                if (no->filhos[i]) {
                    contarPalavras(no->filhos[i], count);
                }
            }
        }
```

3 PROBLEMA 2

Já para o segundo problema, vamos retornar em qual posição a palavra de entrada está e o tamanho de sua linha. Durante a leitura do arquivo de entrada, a posição inicial da palavra

é calculada como o deslocamento em relação ao início do arquivo, enquanto o comprimento é definido como o tamanho total da linha (excluindo o caractere de nova linha). O seguinte trecho de código ilustra como as palavras são extraídas do arquivo e inseridas na Trie na função principal (main).

```
int main() {
    string filename;
    string palavra;

    cin >> filename;

    ifstream filedic(filename);
    if (!filedic.is_open()) {
        cerr << "Erro ao abrir o arquivo " << filename << endl;
        throw runtime_error("Erro no arquivo .dic");
    }

    string linha;
    structures::Trie trie;
    unsigned long posicao = 0;

    while (getline(filedic, linha)) {
        size_t inicio = linha.find('[');
        size_t fim = linha.find(']');
        if (inicio != string::npos && fim != string::npos) {
            string palavra = linha.substr(inicio + 1, fim - inicio
               - 1);
            unsigned long comprimento = linha.length();
            trie.inserir(palavra, posicao, comprimento);
        }
        posicao += linha.length() + 1; // Inclui o caractere de
           nova linha
    }
    filedic.close();
```

No método `buscar`, como vimos acima, se o prefixo corresponde a uma palavra completa no dicionário, os valores de posição e comprimento armazenados no nó final são exibidos. Na main, incluímos o seguinte bloco de código para fazer a busca.

```
while (1) {  // leitura das palavras ate' encontrar "0"
        cin >> palavra;
        if (palavra.compare("0") == 0) {
            break;
        }
        trie.buscar(palavra);
    }
```

4 EXEMPLO DE EXECUÇÃO

Vamos executar o código para o dicionário visto anteriormente, com as definições de bear, bell, bid, bull, buy, sell, stock e stop. Vamos dar como entrada as seguintes palavras: bear, bell, bid, bu, bull, buy, but, sell, stock e stop. Com isso, vamos ter a seguinte saída:

```
bear is prefix of 1 words
bear is at (0,149)
bell is prefix of 1 words
bell is at (150,122)
bid is prefix of 1 words
bid is at (273,82)
bu is prefix of 2 words
bull is prefix of 1 words
bull is at (356,113)
buy is prefix of 1 words
buy is at (470,67)
but is not prefix
sell is prefix of 1 words
sell is at (538,97)
stock is prefix of 1 words
stock is at (636,79)
stop is prefix of 1 words
stop is at (716,92)
```

5 CONCLUSÃO

Como conclusão, vamos apontar as dificuldades na resolução dos problemas.

Em ambos problemas, o principal desafio se deu na criação da classe Trie, mas foi possível implementar funções relativamente simples para a inserção e busca dos dados. Também, não foram apresentados por não fazer parte da discussão, mas implementamos uma função para deletar todos os nós e que é utilizada pelo destrutuor da classe.

No geral, por já existir bastante material de pesquisa sobre as árvores digitais, não tivemos muitos problemas na implementação geral.

6 REFERÊNCIAS

JOYANES AGUILAR, Luis. **Programação em C++: algoritmos, estruturas de dados e objetos**. São Paulo: McGraw Hill, 2008. xxxi, 768 p. ISBN 9788586804816.

FEOFILOFF, Paulo. **Tries (árvores digitais)**. Universidade de São Paulo. Disponível em: <link>. Acesso em 10 de dezemebro de 2024.

Trie Data Structure | Insert and Search. Geeks for Geeks. Disponível em: <link>. Acesso em 10 de dezemebro de 2024.