UNIVERSIDADE FEDERAL DE SANTA CATARINA

CAMPUS TRINDADE

INE - DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

INE5408 - ESTRUTURAS DE DADOS

Alunos: João Victor Cabral Machado e Pedro Alfeu Wolff Lemos

22204113 **22200373**

Relatório do Projeto 2

Florianópolis

2023

Sumário

1. Introdução ao problema

O projeto 2 aborda a construção e utilização de uma estrutura hierárquica chamada trie(árvore de prefixos) para indexar e recuperar palavras em grandes arquivos de dicionários que estão armazenados em memórias secundárias.

Este trabalho tem como objetivo não apenas a implementação eficiente da trie, mas a resolução prática dos problemas relacionados à indexação e recuperação de palavras em dicionários.

O problema 1 consiste na construção da trie a partir das palavras definidas no arquivo do dicionário. Depois de construir a trie, a aplicação recebe várias palavras e determina se cada uma delas é um prefixo das outras palavras do dicionário. A saída padrão informa se a palavra é um prefixo e quantas palavras ele têm.

O problema 2 consiste em determinar a localização exata de cada palavra no arquivo do dicionário. Ao criar o nó correspondente ao último caractere da palavra, a posição inicial e o comprimento da linha são atribuídas à palavra no arquivo. Se essa palavra existir, a aplicação vai determinar essas informações. Também tem o caso, que caso a palavra exista no dicionário ela também pode ser um prefixo de outras, e daí a saída padrão inclui a informação adicional sobre a posição e o comprimento da palavra.

2. Algoritmos

Estrutura TrieNode:

```cpp
// Estrutura para representar um nó da Trie
struct TrieNode {
    TrieNode* children[26];  // Filhos para as letras do alfabeto
    unsigned long position;  // Posição inicial da palavra no arquivo de dicionário
    unsigned long length;    // Comprimento da linha que contém a palavra no arquivo de dicionário
    int wordCount;           // Número de palavras que têm este prefixo
    TrieNode() : position(0), length(0), wordCount(0) {
        fill_n(children, 26, nullptr);
    }
};
```

A trie foi modelada através da estrutura “TrieNode”, onde cada nó representa uma letro alfabeto, sendo inicializada com ponteiros para os filhos correspondentes a cada letra. Cada nó mantém informações cruciais para a indexação, como a posição inicial da palavra no arquivo de dicionário, o comprimento da linha que a define e o número de palavras que compartilham o mesmo prefixo.

Inserção na Trie:

```cpp
// Função para inserir uma palavra na Trie
void insertWord(TrieNode* root, const string& word, unsigned long position, unsigned long length) {
    TrieNode* node = root;
    for (char ch : word) {
        int index = ch - 'a';
        if (!node->children[index]) {
            node->children[index] = new TrieNode();
        }
        node = node->children[index];
        node->wordCount++;
    }
    // Atualizar a posição e comprimento se a posição ainda não foi definida ou a nova posição for menor
    if ((node->position == 0 && node->length == 0) || (position < node->position && length > 0)) {
        node->position = position;
        node->length = length;
    }
}
```

A função “inserword” percorre a trie, criando novos nós conforme necessário para representar cada letra da palavra, durante esse processo o contador de palavra, a posição e o comprimento são atualizados com base nas novas posições. Fazendo isso, garantimos que no final da inserção, cada nó tem informações sobre as palavras que compartilham o mesmo prefixo.

Busca na Trie:

```
// Função para procurar uma palavra na Trie
TrieNode* searchWord(TrieNode* root, const string& word) {
    TrieNode* node = root;
    for (char ch : word) {
        int index = ch - 'a';
        if (!node->children[index]) {
            return nullptr; // Palavra não encontrada na Trie
        }
        node = node->children[index];
    }
    return node;
}
```

A função “searchWord” permite a busca por uma palavra específica na Trie. Se a palavra não existir no dicionário, ela vai retornar “nullptr”.

Verificação na Trie:

```
// Função para verificar uma palavra na Trie e imprimir os resultados
void checkWord(TrieNode* root, const string& word) {
    TrieNode* node = searchWord(root, word);
    if (node) {
        cout << word << " is prefix of " << node->wordCount << " words" << endl;
        // Imprimir posição e comprimento se ambos forem maiores que zero ou apenas o comprimento for maior que zero
        if (word.length() > 2 && (node->position > 0 || node->length > 0)) {
            cout << word << " is at (" << node->position << "," << node->length << ")" << endl;
        }
    } else {
        cout << word << " is not prefix" << endl;
    }
}
```

A função “checkwaord” utiliza a busca na trie para determinarmos se uma palavra é de fato um prefixo das outras e também imprime os resultados. Se a palavra for um prefixo, ela vai mostrar de quantas palavras ela é um prefixo e imprime a sua posição e comprimento da linha do dicionário.

Função main:

```cpp
int main() {
    // Receber o nome do arquivo de dicionário
    string dictionaryFileName;
    cin >> dictionaryFileName;

    // Criar a raiz da Trie
    TrieNode* root = new TrieNode();

    // Abrir o arquivo de dicionário
    ifstream dictionaryFile(dictionaryFileName);
    if (dictionaryFile.is_open()) {
        string line;
        unsigned long position = 0;

        // Ler cada linha do arquivo de dicionário
        while (getline(dictionaryFile, line)) {
            if (line.empty()) {
                continue; // Ignorar linhas vazias
            }

            string word;
            size_t pos = line.find('[');
            if (pos != string::npos) {
                // Extrair a palavra entre colchetes
                word = line.substr(pos + 1, line.find(']') - pos - 1);
                // Inserir a palavra na Trie com a posição e comprimento da linha no arquivo de dicionário
                insertWord(root, word, position, line.length());
            }

            position += line.length() + 1; // Atualizar posição considerando o caractere de nova linha
        }

        // Fechar o arquivo de dicionário
        dictionaryFile.close();
    } else {
        cerr << "Erro ao abrir o arquivo de dicionario." << endl;
        return 1;
    }

    // Ler palavras da entrada padrão e verificar/imprimir os resultados
    string word;
    while (cin >> word && word != "0") {
        checkWord(root, word);
    }

    // Liberar a memória alocada para a Trie
    delete root;

    return 0;
}
```

A função main é o ponto de entrada do programa e ela é faz o papel mais importante na implementação da trie para a indexação e recuperação de palavras nos arquivos de dicionário.

Ela inicia com a leitura do nome do arquivo do dicionário e criando a raiz da trie. Essa raiz fornece a base para a estrutura hierárquica que será usada para colocarmos as palavras do dicionário.

Em seguida, a função abre o dicionário, processando cada linha. Durante esse processo, a função “insertword“ é chamada para inserirmos as palavras na trie, considerando a posição e o comprimento de cada palavra. Também lemos as linhas vazias, a fim de manter a execução correta do programa.

Depois de lermos o dicionário, ele é fechado. Em seguida, o programa entra em um loop onde ele lê cada palavra da entrada até encontrar o 0(que finaliza o programa). Para cada palavra lida, a função “checkword” é chamada, para verificar se a palavra é um prefixo das outras.

Por fim, a memória alocada para a trie é liberada para evitar vazamentos de memória.

3. Dificuldades

O principal problema que tivemos com o trabalho foi a interpretação (que foi resolvida conversando com nossos colegas) e também o manuseamento da memória, por causa dos ponteiros.

4. Referências

https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/tries.html

https://towardsdatascience.com/implementing-a-trie-data-structure-in-python-in-less-than-100-lines-of-code-a877ea23c1a1

https://www.geeksforgeeks.org/trie-insert-and-search/