UNIVERSIDADE FEDERAL DE VIÇOSA - CAMPUS FLORESTAL
HERON FILLIPE, FELIPE TAVARES, OTÁVIO AUGUSTO, MATHEUS AUGUSTO

"TRABALHO PRÁTICO - 1”

Trabalho Prático de Algoritmo e Estrutura de Dados II.

FLORESTAL
2024

Florestal - MG
2024

"TRABALHO PRÁTICO - 1”

Trabalho Prático de Algoritmo e Estrutura de Dados II.

Heron Fillipe Silveira Santos[4211]
Felipe Tavares de Sousa[5367]
Otávio Augusto Gomes e Souza[5376]
Matheus Augusto Nogueira Ferreira[5359]

FLORESTAL

2024

Sumário

Introdução

Este trabalho tem como objetivo o desenvolvimento de índices invertidos para uma máquina de busca de ingredientes contidos em poções do universo de Harry Potter. A indexação dos arquivos de entradas fornecidos será realizada utilizando TADs árvore PATRICIA e tabela HASH, com as buscas a partir de termos contendo ingredientes das poções, com o intuito de recuperar a(s) poção(ões) mais relevante(s), assim, faremos um comparativo entre a busca na árvore PATRICIA e na tabela HASH, para a implementação do dos TAD foram usados códigos do Ziviani como referência.

Metodologia

O trabalho foi realizado em linguagem C de programação utilizando GitHub e VS Code, para facilitar o desenvolvimento em conjunto. Também utilizamos a extensão LiveShare do VS Code, para que todos pudessem escrever no código ao mesmo tempo.

No desenvolvimento do trabalho separamos em etapas, primeiro, criamos os TADs, tabela Hash e árvore Patricia. Em seguida, fomos dividindo os problemas que foram aparecendo de acordo com a afinidade de cada um.

Desenvolvimento

Folder e Entradas:

No desenvolvimento deste trabalho prático dividimos os códigos em quatro TAD's, além do file de execução "Main.c", a fim de um desenvolvimento mais organizado e prático.

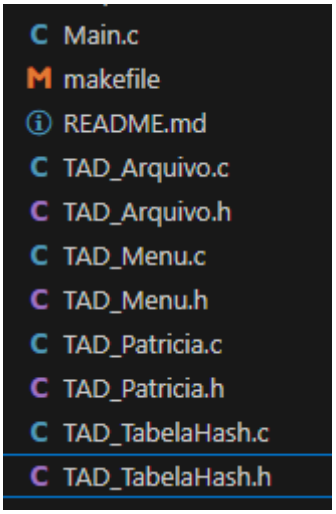

Fig 1.0: Folder do projeto

TAD_TabelaHash

Para armazenamento de palavras na tabela Hash, utilizamos o método de Hashing aberto por meio de encadeamento utilizando a estrutura de Lista Encadeada. Para a

formação do Hash/Índice da Tabela Hash, utilizamos a função “GeraPesos” baseada no código de Hash de Nivio Ziviani assim como as outras funções.

Para comportar os índices invertidos, uma estrutura chamada “InvertedIndex” foi criada contendo o idDoc(Referente ao ID do documento lido) e qtde(Referente a quantas vezes a palavra aparece neste documento). Um ponteiro para o primeiro valor dessa lista encadeada foi adicionado a estrutura básica do nó do Hash.

A função “GeraPesos” atua em conjunto com a função”hash_function” para a formação do valor de hash, onde “GeraPesos” fornece valores para as letras e determinadas posições na palavra, assim fazendo com que cada palavra tenha seu valor específico.

Para a inserção, possuímos um vetor de ponteiros para nós hash, onde fazemos o mod do valor hash de uma palavra, e inserimos ela na tabela, na posição do mod. Desse modo, notamos que pode haver mais de uma palavra que resulta em um valor de mod, e para resolver, utilizamos as listas encadeadas, logo, quando uma palavra tem valor de mod, semelhante a uma outra já inserida na tabela, essa nova palavra é apontada pela que esta inserida, pelo ponteiro “nextHashNode”, e então calculamos o número de colisões, que seria quantas vezes uma palavra a mais foi inserida em um mesmo espaço desse vetor.

TAD_Arquivo:

Assim como pedido na especificação do trabalho, nosso código precisava ler um arquivo, que possui a quantidade de arquivos a serem lidos futuramente e esses arquivos precisamente. Dessa forma, decidimos por criar um TAD exclusivamente para a manipulação desses arquivos, e dos dados fornecidos por eles.

Nesse TAD, nós possuímos funções tanto para ler o primeiro arquivo que o código recebe, e assim extrair os próximos a serem lidos, como também os nomes das poções e seus ingredientes, assim como pede o trabalho. Para isso, utilizamos de funções de leitura de arquivo, com especificações necessárias, como a quebra de leitura a cada “;”, para podermos armazenar de forma correta cada ingrediente dentro da poção, e além disso, cada leitura de arquivo com ingredientes já executava direto a função de inseri-los tanto na tabela Hash, quanto na árvore Patricia.

Algumas funções foram adicionadas para formatar os ingredientes, aqui chamados de tokens, obtidos. As funções “removeLeadingSpaces”, “removeFinalDot” e “removePunctuation” foram criadas para remover, respectivamente, os espaços no início da string lida, o ponto final e qualquer pontuação presente no token a ser armazenado.

Para realizar comparações, implementamos a função “strcasestr”. Esta função é utilizada para localizar uma substring dentro de uma string, ignorando as diferenças entre maiúsculas e minúsculas. Embora strcasestr não faça parte do padrão ANSI C, ela é uma extensão comum em várias implementações da biblioteca C, como na GNU C Library (glibc). Como não estava disponível em todas as implementações, incluímos o código necessário para essa funcionalidade no nosso projeto.

TAD_Patricia:

A implementação da árvore Patrícia foi baseada no código de Ziviani, disponível neste link. Para que a árvore pudesse lidar com strings, foram feitas algumas adaptações importantes:

1. Alteração do TipoChave: O tipo char foi modificado para um ponteiro, permitindo o armazenamento de strings.
2. Adição do Campo char Char_maior: Foi introduzido um campo char_maior na estrutura do nó interno. Esse caractere é responsável por identificar o caractere divergente entre os apontadores esquerdo e direito. Optamos por manter esse caractere como o maior entre os dois comparados na mesma posição.
3. Índice Invertido: Adicionamos uma estrutura chamada InvertedIndexPatricia dos nós externos, que é responsável por armazenar uma lista encadeada com os valores dos índices invertidos.

A função de inserção percorre a árvore comparando o caractere da string na posição index com o char_maior dos nós internos até encontrar um nó externo. Quando um nó externo é encontrado, é feita uma verificação para determinar se a string já está presente na árvore. Caso contrário, a função InsereEntre é chamada. Essa função percorre a árvore para encontrar o local adequado onde serão criados e inseridos os novos nós externos e internos.De maneira similar, a função de pesquisa percorre a árvore comparando os caracteres até encontrar o nó externo que contém a string desejada. Se a string estiver presente na árvore, o nó externo correspondente é retornado.

TAD_Menu:

Para melhor a organização do trabalho, optamos pelo desenvolvimento desse TAD. Nesse TAD temos a criação do menu de opções do trabalho, onde a cada escolha o menu realiza uma ação, desde leitura dos arquivos de entrada ao cálculo de relevância do arquivo baseado no termo de busca. Ao decorrer da criação foi cogitado o desenvolvimento de mais TADs, onde estaria contido os cálculos de relevância da tabela Hash e da Árvore Patricia, entretanto, estava dando algum erro com malloc e calloc (esse erro não foi identificado).

Resultado

Como pedido no trabalho, executamos o código e ele nos traz o menu de opções, a seguir será mostrado o que cada opção realiza, sendo a primeira para ler o arquivo de entrada e a segunda para ler os .txt contendo a receita das poções e criar os índices invertidos.

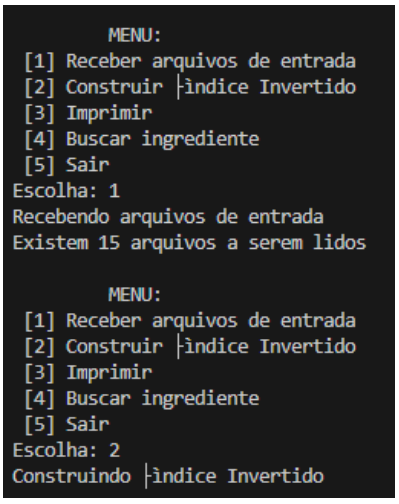

Fig 2.0: Teste do Buscar ingredientes

Para concluir o trabalho, chegamos na parte de teste, no exemplo a seguir, buscamos por 5 termos e retornamos o cálculo de relevância de cada artigo, feito pela busca na tabela Hash e na árvore Patricia.

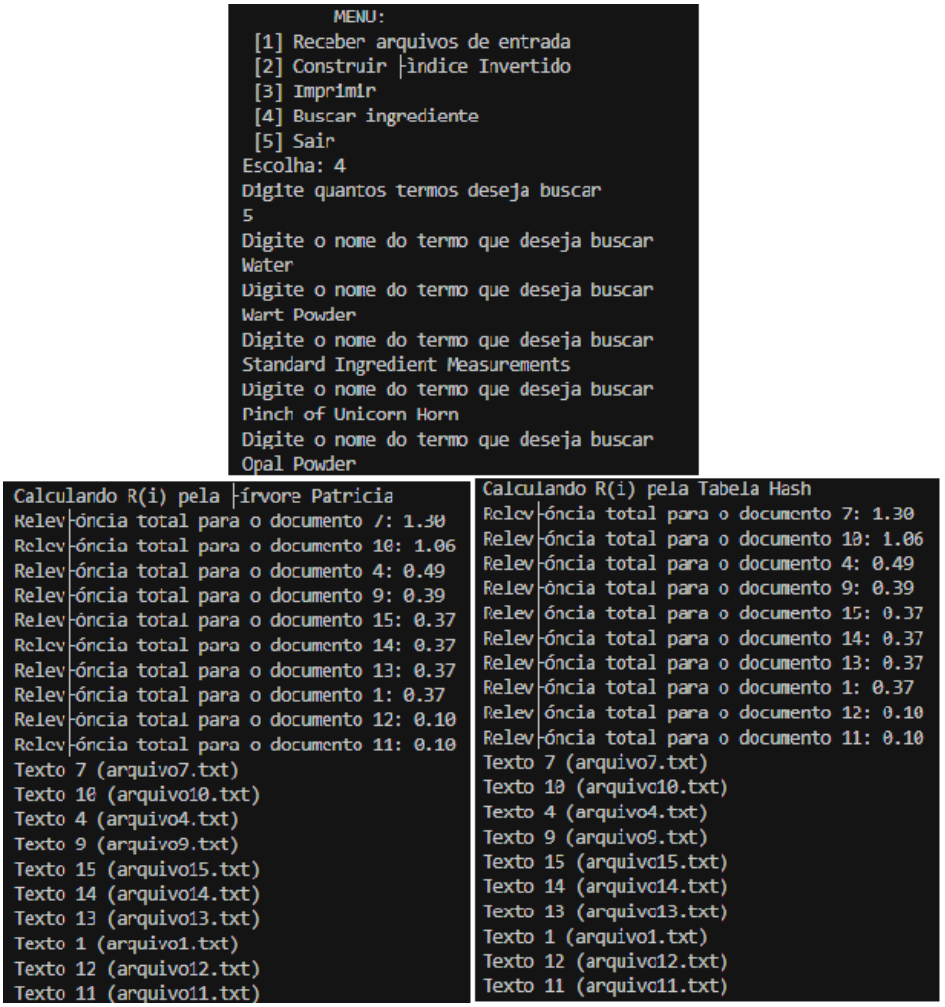

Fig 3.0: Teste do Buscar ingredientes

Para finalizar, fizemos o teste comparativo entre as colisões da tabela Hash e os seekings da árvore Patricia,

```
Tabela Hash
Chaves inseridas na tabela hash em ordem alfab|®tica:
Chave: Asphodel Powder -> (Colisoes: 0) (Doc: 11, Qtde: 1)
Chave: Berry Juice -> (Colisoes: 1) (Doc: 4, Qtde: 1)
Chave: Bezoar -> (Colisoes: 0) (Doc: 0, Qtde: 2)
Chave: Bezoar Powder -> (Colisoes: 0) (Doc: 8, Qtde: 1)
Chave: Blindworm Mucus -> (Colisoes: 0) (Doc: 4, Qtde: 1)
Chave: Blindworm Mucus Bubbles -> (Colisoes: 0) (Doc: 3, Qtde: 1)
Chave: Boomslang Skin -> (Colisoes: 0) (Doc: 7, Qtde: 1)
Chave: Castor Oil -> (Colisoes: 0) (Doc: 2, Qtde: 1) (Doc: 11, Qtde: 1)
Chave: Cedarwood Ash -> (Colisoes: 0) (Doc: 10, Qtde: 1)
Chave: Creeping Fig -> (Colisoes: 0) (Doc: 5, Qtde: 1)
Chave: Dandelion Juice -> (Colisoes: 0) (Doc: 12, Qtde: 1)
Chave: Dragon Bone Powder -> (Colisoes: 0) (Doc: 14, Qtde: 1)
Chave: Dragon's Blood -> (Colisoes: 0) (Doc: 14, Qtde: 0)
Chave: Elderberry Juice -> (Colisoes: 0) (Doc: 11, Qtde: 1)
Chave: Essence of Murtlap Extract -> (Colisoes: 0) (Doc: 7, Qtde: 1)
Chave: Essence of Wormwood -> (Colisoes: 0) (Doc: 5, Qtde: 1)
Chave: Fresh Mint -> (Colisoes: 0) (Doc: 8, Qtde: 1)
Chave: Ginger Root -> (Colisoes: 0) (Doc: 11, Qtde: 1) (Doc: 13, Qtde: 1)
Chave: Honey Water -> (Colisoes: 0) (Doc: 4, Qtde: 1)
Chave: Horned Slugs -> (Colisoes: 0) (Doc: 1, Qtde: 1)
Chave: Lavender Sprigs -> (Colisoes: 0) (Doc: 3, Qtde: 1)
Chave: Lionfish Spines -> (Colisoes: 1) (Doc: 4, Qtde: 1)
Chave: Mandrake Root -> (Colisoes: 0) (Doc: 7, Qtde: 1) (Doc: 8, Qtde: 1)
Chave: Mistletoe Berries -> (Colisoes: 0) (Doc: 0, Qtde: 1)
Chave: Moldy Bark -> (Colisoes: 0) (Doc: 8, Qtde: 1)
Chave: Moonstone Powder -> (Colisoes: 0) (Doc: 9, Qtde: 1)
Chave: Mushrooms -> (Colisoes: 0) (Doc: 6, Qtde: 1)
Chave: Opal Powder -> (Colisoes: 0) (Doc: 9, Qtde: 1)
Chave: Pearl Powder -> (Colisoes: 0) (Doc: 10, Qtde: 1) (Doc: 13, Qtde: 1)
Chave: Peppermint Leaf -> (Colisoes: 0) (Doc: 10, Qtde: 1)
Chave: Peppermint Sprig -> (Colisoes: 0) (Doc: 5, Qtde: 1)
Chave: Pinch of Unicorn Horn -> (Colisoes: 0) (Doc: 0, Qtde: 1) (Doc: 12, Qtde: 1) (Doc: 13, Qtde: 1) (Doc: 14, Qtde: 1)
Chave: Porcupine Quills -> (Colisoes: 0) (Doc: 1, Qtde: 1) (Doc: 5, Qtde: 1)
Chave: Powdered Bicorn Horn -> (Colisoes: 0) (Doc: 7, Qtde: 1)
Chave: Powdered Dragonfly Wing Root -> (Colisoes: 0) (Doc: 12, Qtde: 1)
Chave: Rose Water -> (Colisoes: 0) (Doc: 10, Qtde: 1)
Chave: Salamander Blood -> (Colisoes: 0) (Doc: 4, Qtde: 1) (Doc: 6, Qtde: 1) (Doc: 7, Qtde: 1)
Chave: Snake Fangs -> (Colisoes: 0) (Doc: 1, Qtde: 1)
Chave: Sopophorous Beans -> (Colisoes: 0) (Doc: 5, Qtde: 1)
Chave: Squill Bulb Extract -> (Colisoes: 0) (Doc: 2, Qtde: 1)
Chave: Standard Ingredient Measurements -> (Colisoes: 0) (Doc: 0, Qtde: 1) (Doc: 3, Qtde: 3) (Doc: 9, Qtde: 1) (Doc: 10, Qtde: 1) (Doc: 11, Qtde: 1) (Doc: 12, Qtde: 1) (Doc: 13, Qtde: 1) (Doc: 14, Qtde
: 1)
Chave: Unicorn Horn Powder -> (Colisoes: 0) (Doc: 9, Qtde: 1)
Chave: Valerian Syrup -> (Colisoes: 0) (Doc: 9, Qtde: 1)
Chave: Wart Powder -> (Colisoes: 0) (Doc: 6, Qtde: 1)
Chave: Water -> (Colisoes: 0) (Doc: 8, Qtde: 1) (Doc: 9, Qtde: 1)
Chave: Wiggentree Twigs -> (Colisoes: 0) (Doc: 2, Qtde: 2)
```

```
|Árvore Patricia
Chave: Berry Juice -> (Seekings: 1)  (Doc: 4, Qtde: 1),
Chave: Bezoar -> (Seekings: 2)  (Doc: 0, Qtde: 2),
Chave: Bezoar Powder -> (Seekings: 1)  (Doc: 8, Qtde: 1),
Chave: Blindworm Mucus -> (Seekings: 1)  (Doc: 4, Qtde: 1),
Chave: Asphodel Powder -> (Seekings: 1)  (Doc: 11, Qtde: 1),
Chave: Boomslang Skin -> (Seekings: 1)  (Doc: 7, Qtde: 1),
Chave: Blindworm Mucus Bubbles -> (Seekings: 1)  (Doc: 3, Qtde: 1),
Chave: Castor Oil -> (Seekings: 2)  (Doc: 2, Qtde: 1), (Doc: 11, Qtde: 1),
Chave: Dandelion Juice -> (Seekings: 1)  (Doc: 12, Qtde: 1),
Chave: Cedarwood Ash -> (Seekings: 1)  (Doc: 10, Qtde: 1),
Chave: Elderberry Juice -> (Seekings: 1)  (Doc: 11, Qtde: 1),
Chave: Ginger Root -> (Seekings: 2)  (Doc: 11, Qtde: 1), (Doc: 13, Qtde: 1),
Chave: Creeping Fig -> (Seekings: 1)  (Doc: 5, Qtde: 1),
Chave: Dragon Bone Powder -> (Seekings: 1)  (Doc: 14, Qtde: 1),
Chave: Dragon's Blood -> (Seekings: 0)  (Doc: 14, Qtde: 0),
Chave: Essence of Murtlap Extract -> (Seekings: 1)  (Doc: 7, Qtde: 1),
Chave: Fresh Mint -> (Seekings: 1)  (Doc: 8, Qtde: 1),
Chave: Essence of Wormwood -> (Seekings: 1)  (Doc: 5, Qtde: 1),
Chave: Honey Water -> (Seekings: 1)  (Doc: 4, Qtde: 1),
Chave: Horned Slugs -> (Seekings: 1)  (Doc: 1, Qtde: 1),
Chave: Lavender Sprigs -> (Seekings: 1)  (Doc: 3, Qtde: 1),
Chave: Lionfish Spines -> (Seekings: 1)  (Doc: 4, Qtde: 1),
Chave: Mandrake Root -> (Seekings: 2)  (Doc: 7, Qtde: 1), (Doc: 8, Qtde: 1),
Chave: Mistletoe Berries -> (Seekings: 1)  (Doc: 0, Qtde: 1),
Chave: Moldy Bark -> (Seekings: 1)  (Doc: 8, Qtde: 1),
Chave: Moonstone Powder -> (Seekings: 1)  (Doc: 9, Qtde: 1),
Chave: Opal Powder -> (Seekings: 1)  (Doc: 9, Qtde: 1),
Chave: Mushrooms -> (Seekings: 1)  (Doc: 6, Qtde: 1),
Chave: Peppermint Leaf -> (Seekings: 1)  (Doc: 10, Qtde: 1),
Chave: Pearl Powder -> (Seekings: 2)  (Doc: 10, Qtde: 1), (Doc: 13, Qtde: 1),
Chave: Peppermint Sprig -> (Seekings: 1)  (Doc: 5, Qtde: 1),
Chave: Pinch of Unicorn Horn -> (Seekings: 4)  (Doc: 0, Qtde: 1), (Doc: 12, Qtde: 1), (Doc: 13, Qtde: 1), (Doc: 14, Qtde: 1),
Chave: Porcupine Quills -> (Seekings: 2)  (Doc: 1, Qtde: 1), (Doc: 5, Qtde: 1),
Chave: Rose Water -> (Seekings: 1)  (Doc: 10, Qtde: 1),
Chave: Powdered Bicorn Horn -> (Seekings: 1)  (Doc: 7, Qtde: 1),
Chave: Powdered Dragonfly Wing Root -> (Seekings: 1)  (Doc: 12, Qtde: 1),
Chave: Salamander Blood -> (Seekings: 3)  (Doc: 4, Qtde: 1), (Doc: 6, Qtde: 1), (Doc: 7, Qtde: 1),
Chave: Valerian Syrup -> (Seekings: 1)  (Doc: 9, Qtde: 1),
Chave: Snake Fangs -> (Seekings: 1)  (Doc: 1, Qtde: 1),
Chave: Unicorn Horn Powder -> (Seekings: 1)  (Doc: 9, Qtde: 1),
Chave: Sopophorous Beans -> (Seekings: 1)  (Doc: 5, Qtde: 1),
Chave: Squill Bulb Extract -> (Seekings: 1)  (Doc: 2, Qtde: 1),
Chave: Wart Powder -> (Seekings: 1)  (Doc: 6, Qtde: 1),
Chave: Water -> (Seekings: 2)  (Doc: 8, Qtde: 1), (Doc: 9, Qtde: 1),
Chave: Wiggentree Twigs -> (Seekings: 2)  (Doc: 2, Qtde: 2),
Chave: Standard Ingredient Measurements -> (Seekings: 10)  (Doc: 0, Qtde: 1), (Doc: 3, Qtde: 3), (Doc: 9, Qtde: 1), (Doc: 10, Qtde: 1), (Doc: 11, Qtde: 1), (Doc: 12, Qtde: 1), (Doc: 13, Qtde: 1), (Doc:
 14, Qtde: 1),
```

Fig 4.0: Print tabela Hash e árvore Patricia

Na árvore Patricia, muitas chaves são encontradas com apenas uma ou duas buscas, mostrando uma estrutura eficiente para busca de strings específicas. Essa eficiência se deve à sua estrutura compactada que elimina nós redundantes, mantendo a complexidade próxima de O(log n).

Já na tabela hash, a presença de colisões exige múltiplas verificações no mesmo bucket. Essas colisões aumentam o tempo de busca dependendo do número de colisões.

A árvore Patricia se mostra mais eficiente na maioria dos casos, com buscas rápidas e poucas verificações necessárias. A tabela hash, apesar de eficiente, enfrenta desafios com colisões, o que pode impactar sua performance.

Considerações Finais

Durante a execução do trabalho não foi possível utilizar o comando make para criar o executável gerado, sendo necessário fazer alterações para que o mesmo fosse incluído na pasta output.

Após a criação da árvore Patrícia, houveram dificuldades para imprimir a mesma em ordem alfabética. Apesar disso foram conferidas todas as chaves inseridas, de forma que todas estão presentes ao final das inserções.

No desenvolvimento do cálculo de relevância, encontramos alguns empecilhos, como que tipo de estrutura usar para armazenar os valores, de início, seria utilizado uma matriz, mas a alocação de memória se tornaria mais um problema; problema na estruturação das repetições (while e for), onde entravam em loop ou fechavam a execução.

Referências

https://www2.dcc.ufmg.br/livros/algoritmos/cap5/codigo/c/5.16a5.21-patricia.c
https://code.visualstudio.com/
https://github.com/OtavioAugustoGomeseSouza/TP_AEDS_2.git