SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL

Trabalho 1 - AEDS 2

Hash e Árvore Patrícia: Pesquisa de Ingredientes

Daniel Martins De Abreu [05798]
Gabriel Marcus de Oliveira Félix. [05792]
Manoel [05379]

Florestal - MG
2024

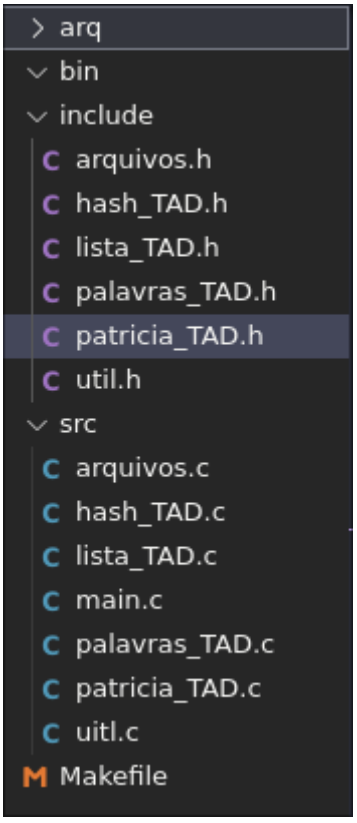
Sumário

1. Introdução

O trabalho visa realizar um estudo comparativo entre duas estruturas de dados amplamente utilizadas para a implementação de índices invertidos: a árvore Patricia e a tabela hash. O objetivo é construir índices invertidos para uma máquina de busca que recupera informações sobre ingredientes usados em poções no universo de Harry Potter. Após a indexação dos arquivos da coleção fornecida, que contém descrições de poções e seus ingredientes, serão realizados testes para verificar o desempenho das duas estruturas na inserção e consulta dos dados.

2. Organização

A estrutura do repositório do projeto está organizada para suportar a implementação e teste dos TADs de Árvores PATRICIA e Tabelas HASH para o sistema de índices invertidos. A seguir, está o panorama geral das pastas e arquivos incluídos no projeto:



Descrição das Pastas e Arquivos:

- **`arq/`**: Contém os arquivos de entrada e dados do projeto.
    - **`ArquivosEntrada/`**: Contém os arquivos de texto de entrada (`arquivo1.txt`, `arquivo2.txt`, etc.) que serão processados pelo sistema.
    - **`entrada.txt`**: Arquivo de entrada que lista os arquivos a serem processados.
- **`bin/`**: Contém o executável principal do projeto.
    - **`main`**: Arquivo executável gerado após a compilação do projeto.
- **`include/`**: Contém os arquivos de cabeçalho para as diversas partes do projeto.
    - **`arquivos.h`**: Declarações das funções para manipulação de arquivos.
    - **`hash_TAD.h`**: Declarações das funções e estruturas da Tabela Hash.
    - **`lista_TAD.h`**: Declarações das funções e estruturas para lista encadeada.
    - **`palavras_TAD.h`**: Declarações das funções e estruturas relacionadas às palavras.
    - **`patricia_TAD.h`**: Declarações das funções e estruturas da Árvore Patricia.
    - **`util.h`**: Declarações de funções utilitárias usadas no projeto.
- **`src/`**: Contém o código-fonte do projeto.
    - **`arquivos.c`**: Implementação das funções para manipulação de arquivos.
    - **`hash_TAD.c`**: Implementação das funções e operações da Tabela Hash.
    - **`lista_TAD.c`**: Implementação das funções e operações para lista encadeada.
    - **`main.c`**: Função principal do programa, que coordena a execução.

    - **`palavras_TAD.c`**: Implementação das funções e operações relacionadas às palavras.
    - **`patricia_TAD.c`**: Implementação das funções e operações da Árvore Patricia.
    - **`util.c`**: Implementação de funções utilitárias.
- **`Makefile`**: Arquivo de configuração para a construção do projeto. Define os comandos necessários para compilar o código-fonte e gerar o executável.

3. Desenvolvimento

Nesta seção, serão destacados os principais detalhes das implementações cruciais para o projeto, focando nos pontos determinantes que permitiram a criação e funcionamento dos TADs (Tabela Hash e Árvore Patricia) para a indexação invertida. Os nomes das funções e variáveis foram escolhidos de maneira autoexplicativa para facilitar a compreensão do código sem a necessidade de descrições detalhadas de cada uma delas.

3.1 Nomes

Nós optamos por determinar um certo padrão:

- (f):Para funções importantes

```
void fPrintPatricia(tNodePT *raiz);
```

- (p):Para ponteiros explícitos

```
typedef struct tNodePT pNodePT;
```

- (t):Para qualquer struct criado para um certo “TAD”

```
typedef struct tNodeH {
   tPalavra palavra;      // Estrutura de dados que representa a palavra
   struct tNodeH *next;   // Ponteiro para o próximo nó na lista encadeada
} tNodeH;
```

3.2 Funções

Todas as funções tem um comentário de documentação

```
/**
* @brief Função de hash usando o algoritmo djb2.
*
* @param str String para gerar o hash.
* @param size Tamanho da tabela hash (número de buckets).
* @return unsigned int Índice da tabela hash onde a string deve ser
armazenada.
*/
unsigned int fHash(char *str, unsigned int size);
```

Onde

- @brief: é uma leve descrição do que é a função
- @param: é o parâmetro que ela recebe
- @return: é o que ela vai retornar caso

isso caso exista um parâmetro ou retorno de algo.

3.3 Mini Biblioteca de Arquivos

Para gerenciar os arquivos de forma eficiente, criamos uma mini biblioteca de arquivos. Essa abordagem facilita a manipulação do código e a obtenção de informações úteis para o projeto. Algumas das funcionalidades principais incluem:

- **Leitura da Quantidade de Arquivos**: A biblioteca pode ler a primeira linha de um arquivo de entrada e determinar a quantidade de arquivos de receitas a serem processados.

- **Criação de Vetores de Nomes de Arquivos**: A biblioteca cria um vetor contendo os nomes dos arquivos de receitas a serem abertos.

- **Geração de Caminhos Completos**: A biblioteca gera um vetor contendo os caminhos completos dos arquivos de receitas.

- **Processamento de Arquivos e Criação de Lista de Palavras**: A biblioteca processa os arquivos de receitas e cria uma lista de palavras (ingredientes).

- **Contagem de Ingredientes**: A biblioteca conta a quantidade de aparições de ingredientes na lista de palavras.

Os códigos que documentam e implementam essas funcionalidades estão nos arquivos’ arquivos.h’ e ‘arquivos.c’.

3.4 Implementação de uma Lista Encadeada de Palavras

Para organizar e gerenciar as palavras antes de inseri-las nas estruturas de pesquisa (Tabela Hash e Árvore Patricia), implementamos uma lista encadeada de palavras. Esta lista desempenha funções importantes, garantindo uma manipulação eficiente e ordenada das palavras.

A lista encadeada foi escolhida por sua flexibilidade em operações de inserção e remoção, permitindo a fácil manipulação dos dados sem necessidade de realocação de memória.

As principais funcionalidades da lista encadeada de palavras incluem:

- **Inserção de Palavras**: As palavras são adicionadas à lista encadeada conforme são lidas dos arquivos de texto. A inserção é realizada de forma eficiente, mantendo a integridade da lista.

- **Ordenação Alfabética**: Após a inserção, a lista pode ser ordenada em ordem alfabética. Esta ordenação facilita a busca sequencial e permite a inserção das palavras nas estruturas de pesquisa de forma organizada.

- **Tratamento de Palavras**: Antes de inserir as palavras na lista, realizamos um tratamento para remover caracteres especiais e normalizar o texto. Isso garante que as palavras sejam comparáveis e que as buscas subsequentes sejam precisas.

- **Obtenção do Tamanho da Lista**: Funções para determinar o tamanho da lista foram implementadas, permitindo que possamos medir o número de palavras processadas e garantir que todas foram corretamente inseridas e tratadas.

Os códigos que documentam e implementam essas funcionalidades estão nos arquivos' lista_TAD.h' e 'lista_TAD.c'.

3.5 TAD tPalavra

Ao observarmos o trabalho prático, percebemos que seria mais eficiente criar um TAD que armazena uma string e uma lista encadeada de IDs. Isso facilita o

gerenciamento e armazenamento das palavras (ingredientes) e suas respectivas aparições nos documentos, economizando memória e melhorando a organização dos dados. As seguintes estruturas foram utilizadas:

```
// Estrutura para armazenar a quantidade e o número do arquivo
typedef struct tID {
  int qtd; // Quantidade de aparições
  int arq; // Número do arquivo
} tID;

// Estrutura para o nó da lista encadeada que contém tID
typedef struct tNodeID {
   tID data;               // Dados do nó (tID)
   struct tNodeID *next;   // Ponteiro para o próximo nó
} tNodeID;

// Estrutura para armazenar uma palavra e a lista encadeada de tID
typedef struct {
   char *nome;         // Nome da palavra
   tNodeID *node;      // Ponteiro para o início da lista encadeada
de tID
} tPalavra;
```

Com o uso da lista encadeada de IDs, ficou muito mais fácil inserir cada palavra (ingrediente) com seus respectivos IDs, invertendo e inserindo apenas quando a palavra existe em algum documento, economizando memória.

Funções Essenciais do TAD `tPalavra`

Para os IDs:

- **Criação de IDs**: Inicializa um novo ID.

- **Inserção de IDs na lista encadeada**: Adiciona um novo ID à lista encadeada.

- **Liberação de memória dos IDs**: Libera a memória alocada para os IDs.

Para `tPalavra`:

- **Salvar uma palavra e sua lista encadeada de tID**: Armazena uma nova palavra com seus IDs associados.

- **Processar uma lista de palavras e armazenar suas quantidades e números de arquivo em uma lista de `tPalavra`**: Processa as palavras e suas aparições nos documentos.

- **Exibir o vetor de `tPalavra` com sua palavra e seus IDs**: Mostra as palavras e suas informações associadas.

- **Implementação do TF-IDF**: (A ser verificada) Calcula o maior ID baseado no TF-IDF.

Os códigos que documentam e implementam essas funcionalidades estão nos arquivos ’palavras_TAD.h’ e ‘palavras_TAD.c’.

3.6 Tabela Hash

Para a implementação da tabela hash, utilizamos uma hash fechada, ajustada para inserir estruturas tPalavra usando o campo nome como chave. As funções principais incluem:

- **Função de hash:** Utilizando uma implementação simples apelidada de djb2.
    - O djb2 é um algoritmo de hash simples e eficiente, criado por Daniel J. Bernstein. Ele é especialmente útil para hashing de strings e é bastante popular por sua velocidade e boa distribuição de hashes.

- **Criação da tabela hash.**

- **Inserção de tPalavra na tabela hash.**

- **Pesquisa de tPalavra na tabela hash.**

- **Exibição da tabela hash.**

Os códigos que documentam e implementam essas funcionalidades estão nos arquivos ‘hash_TAD.h’ e ‘hash_TAD.c’.

3.7 Patricia

A implementação da árvore Patricia foi modularizada para facilitar a organização e manutenção do código. A estrutura da árvore Patricia foi ajustada para trabalhar com a estrutura tPalavra ao invés de strings diretas. As principais funcionalidades incluem:

- **Inserção de tPalavra.**
- **Pesquisa de tPalavra.**
- **Exibição da árvore Patricia.**

O tipo de estrutura da árvore patrícia ficou da seguinte forma:

```
// Enum para tipo de nó
typedef enum {
   INTERNO,  ///< Nó interno da árvore Patricia
   EXTERNO   ///< Nó externo da árvore Patricia
} TipoNode;

// Estrutura para o nó da árvore Patricia
typedef struct tNodePT pNodePT;
typedef struct tNodePT {
   TipoNode tipo;  ///< Tipo do nó (INTERNO ou EXTERNO)
   union {
       struct {
           char letra;       ///< Letra que divide os nós internos
           int index;        ///< Índice da letra na palavra
           pNodePT *esq;     ///< Ponteiro para a subárvore esquerda
           pNodePT *dir;     ///< Ponteiro para a subárvore direita
       } nodeInterno;
       tPalavra item;  ///< Palavra armazenada no nó externo
   } node;
} tNodePT;
```

Adaptamos o nó interno para ter um index + letra que difere para sabermos onde vamos inserir

Os códigos que documentam e implementam essas funcionalidades estão nos arquivos ‘patricia_TAD.h’ e ‘patricia_TAD.c’.

3.8 Main

O arquivo main.c serve como ponto central do programa, unindo todas as funcionalidades e proporcionando ao usuário uma interface interativa por meio de um menu recursivo. Essa estrutura permite múltiplas interações com o sistema, oferecendo uma experiência mais dinâmica.

```
=== Menu ===
1. Listar arquivos
2. Listar Ingredientes
3. Mostrar tabela hash
4. Mostrar árvore Patricia
5. Buscar por palavra (Hash e Patricia)
0. Sair
Digite a opção:
```

4. Compilação e Execução

Para compilar o programa é necessário ter as dependências necessárias que são o gcc, make. Para compilar basta usar um “make”.

```
[~/facul/semestre3/AEDS-II/TPs/TP-1/TP_CODIGOS : 刀] make
Compilado com sucesso
```

Para execução basta “make run”:

```
[~/facul/semestre3/AEDS-II/TPs/TP-1/TP_CODIGOS : 刀] make run
Digite o nome do arquivo de entrada:
```

5. Resultados

Como resultado tivemos um programa que consegue receber uma entrada.

```
Digite o nome do arquivo de entrada: entrada.txt
```

Ter um menu que:

- Lista arquivos
- Lista ingredientes em ordem alfabética
- Mostra tabela Hash
- Mostra Árvore Patricia
- Busca por palavra e mostra o melhor resultado e abre o arquivo tanto no Hash quanto na Árvore Patricia

```
=== Menu ===
1. Listar arquivos
2. Listar Ingredientes
3. Mostrar tabela hash
4. Mostrar árvore Patricia
5. Buscar por palavra (Hash e Patricia)
0. Sair
Digite a opção:
```

1. Listar ingredientes

```
Palavra[1]: Asphodel Powder
<1,12>
Palavra[2]: Berry Juice
<1,12><1,5>
Palavra[3]: Bezoar
<1,9><2,1>
Palavra[4]: Bezoar Powder
<1,9>
Palavra[5]: Blindworm Mucus
<1,5><1,4>
Palavra[6]: Blindworm Mucus Bubbles
<1,4>
Palavra[7]: Boomslang Skin
<1,8>
Palavra[8]: Castor Oil
<1,12><1,3>
Palavra[9]: Cedarwood Ash
<1,11>
Palavra[10]: Creeping Fig
<1,6>
Palavra[11]: Dandelion Juice
<1,13>
Palavra[12]: Dragon Bone Powder
<1,15>
Palavra[13]: Dragon's Blood
<1,15>
Palavra[14]: Elderberry Juice
<1,12>
Palavra[15]: Essence of Murtlap Extract
<1,8>
Palavra[16]: Essence of Wormwood
<1,6>
Palavra[17]: Fresh Mint
<1,9>
Palavra[18]: Ginger Root
<1,14><1,12>
Palavra[19]: Honey Water
```

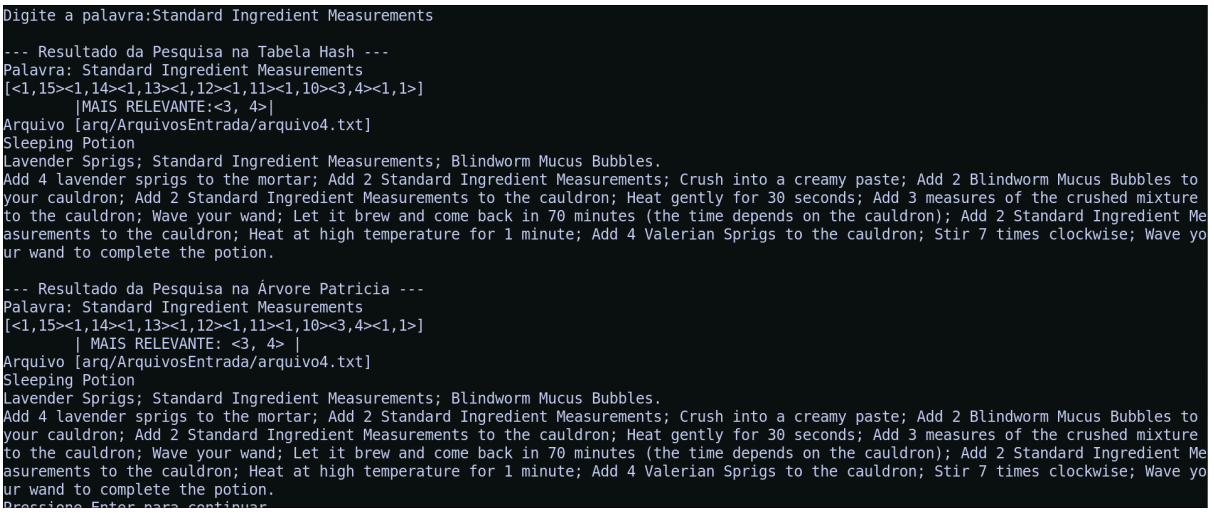
2. Mostrar tabela Hash

```
Tabela Hash:
Indice 0:  -> Water: [<1,12><2,11><1,10><1,9><1,5>] -> Peppermint Sprig: [<1,6>]
Indice 1:  -> Standard Ingredient Measurements: [<1,15><1,14><1,13><1,12><1,11><1,10><3,4><1,1>] -> Porcupine Quills: [<1,6><1,2>] -> Pep
permint Leaf: [<1,11>]
Indice 2:  -> Dragon Bone Powder: [<1,15>]
Indice 3:  -> Rose Water: [<1,11>] -> Moonstone Powder: [<1,10>] -> Lionfish Spines: [<1,5>] -> Ginger Root: [<1,14><1,12>] -> Berry Juic
e: [<1,12><1,5>]
Indice 4:  -> Wiggentree Twigs: [<2,3>] -> Powdered Dragonfly Wing Root: [<1,13>] -> Horned Slugs: [<1,2>]
Indice 5:  -> Squill Bulb Extract: [<1,3>] -> Mandrake Root: [<1,9><1,8>] -> Essence of Murtlap Extract: [<1,8>]
Indice 6:  -> Mistletoe Berries: [<1,1>] -> Castor Oil: [<1,12><1,3>] -> Bezoar: [<1,9><2,1>]
Indice 7:  -> Wart Powder: [<1,7>] -> Opal Powder: [<1,10>] -> Blindworm Mucus Bubbles: [<1,4>]
Indice 8:  -> Lavender Sprigs: [<1,4>] -> Asphodel Powder: [<1,12>]
Indice 9:  -> Pearl Powder: [<1,14><1,11>] -> Elderberry Juice: [<1,12>] -> Cedarwood Ash: [<1,11>] -> Boomslang Skin: [<1,8>]
Indice 10:  -> Sopophorous Beans: [<1,6>] -> Salamander Blood: [<1,8><1,7><1,5>] -> Essence of Wormwood: [<1,6>] -> Blindworm Mucus: [<1,
5><1,4>] -> Bezoar Powder: [<1,9>]
Indice 11:  -> Moldy Bark: [<1,9>]
Indice 12:  -> Dandelion Juice: [<1,13>]
Indice 13:  -> Valerian Syrup: [<1,10>] -> Powdered Bicorn Horn: [<1,8>] -> Pinch of Unicorn Horn: [<1,15><1,14><1,13><1,1>] -> Mushrooms
: [<1,7>] -> Honey Water: [<1,5>] -> Creeping Fig: [<1,6>]
Indice 14:  -> Unicorn Horn Powder: [<1,10>] -> Snake Fangs: [<1,2>] -> Fresh Mint: [<1,9>] -> Dragon's Blood: [<1,15>]
Pressione Enter para continuar...
```

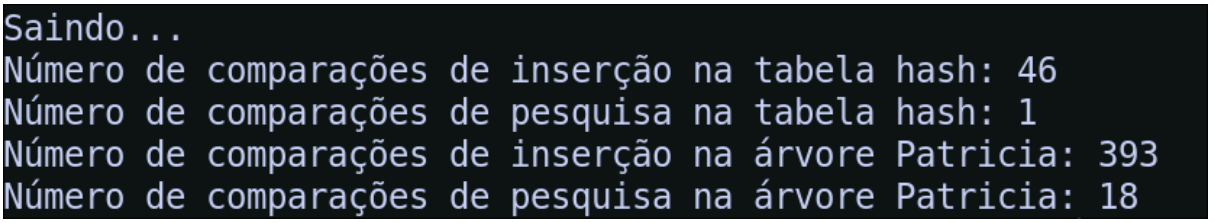
3. Mostrar arvore patricia

```
Árvore Patricia:
I:('A',0)
 E:'Asphodel Powder':[<1,12>]
 I:('B',0)
  I:('e',1)
   I:('r',2)
    E:'Berry Juice':[<1,12><1,5>]
    I:('',6)
     E:'Bezoar':[<1,9><2,1>]
     E:'Bezoar Powder':[<1,9>]
   I:('l',1)
    I:('',15)
     E:'Blindworm Mucus':[<1,5><1,4>]
     E:'Blindworm Mucus Bubbles':[<1,4>]
    E:'Boomslang Skin':[<1,8>]
  I:('C',0)
   I:('a',1)
    E:'Castor Oil':[<1,12><1,3>]
    I:('e',1)
     E:'Cedarwood Ash':[<1,11>]
     E:'Creeping Fig':[<1,6>]
   I:('D',0)
    I:('a',1)
     E:'Dandelion Juice':[<1,13>]
     I:(' ',6)
      E:'Dragon Bone Powder':[<1,15>]
      E:'Dragon's Blood':[<1,15>]
    I:('E',0)
     I:('l',1)
      E:'Elderberry Juice':[<1,12>]
      I:('M',11)
       E:'Essence of Murtlap Extract':[<1,8>]
       E:'Essence of Wormwood':[<1,6>]
     I:('F',0)
      E:'Fresh Mint':[<1,9>]
      I:('G',0)
       E:'Ginger Root':[<1,14><1,12>]
```

4. Pesquisa



5. Resultados de comparações de pesquisas e inserções Hash e Patricia



6. Conclusão

Considerando os objetivos, implementações e resultados obtidos neste trabalho, podemos concluir o seguinte: O objetivo principal de construir índices invertidos eficientes para uma máquina de busca de ingredientes de poções no universo de Harry Potter foi alcançado com sucesso. As estruturas de dados PATRICIA e HASH demonstraram ser adequadas para essa tarefa, permitindo a indexação rápida e eficaz de um grande volume de dados textuais. Além disso, observou-se que a estrutura PATRICIA requer um número maior de inserções devido à necessidade de criar nós internos e externos para cada inserção de tArquivo.

No entanto, algumas limitações foram identificadas durante o desenvolvimento, optamos por fazer o índice invertido de forma genérica ao invés de separado para cada um das estruturas. A implementação do TF-IDF não foi completamente validada, e as comparações de pesquisa realizadas apresentam inconsistências.

7. Referências

[1] Hash djb2. Disponível em: <https://theartincode.stanis.me/008-djb2/> Último acesso em: 03 de agosto de 2024.

[2] N. Ziviani, Projeto de Algoritmos com Implementações em Pascal e C, 3ª ed., Cengage Learning, 2010.

[3] N. Ziviani. Projeto de Algoritmos com Implementações em Pascal e C. Disponível em: <www2.dcc.ufmg.br/livros/algoritmos/>