Hash de Índice Remissivo
IFMG - Campus Formiga

Guilherme Cardoso Silva

[1] Algoritmos e Estrutura de Dados II – Instituto Federal de Minas Gerais (IFMG)
Caixa Postal 35570-000 – Formiga – MG – Brasil

godgcs@gmail.com.br

1. Introdução

O objetivo geral deste trabalho é o projeto e a implementação de um sistema que construa um índice remissivo. Sendo que para implementação deste método deve ser utilizado Tabelas Hash no formato ExternHash, Arvore Binaria de Busca não balanceada, Lista Encadeada e Arvore AVL.

Com a implementação de ambos os metodos foi possivel comparar ambas as estruturas por meio de medidas de: tempo e complexidade computacional.

O metodo HashExtern apresentou um desempenho muito melhor a OpenHash devido ao seu melhor tratamento com relação a colisões.

2. Implementação

Para compilar é necessario apenas utilizar *gcc *.c -o "NomeExecutavel"* para gerar o executavel.

Para executar são necessarios passar como parametro o nome do arquivo que contém as palavras chaves, o nome do arquivo que contém o texto a ser consultado e o nome do arquivo onde serão gravados os indices das palavras lidas.

./"executavel" "arquivoPalavrasChaves" "arquivoTexto" "arquivoSaida"

A implementação deste projeto foi divido em quatro TADs, sendo elas:

- *Hash Externa*
- *Arvore Binária Não Balanceada*
- *Lista Encadeada*
- *Arvore AVL*

O motivo para esta separação entre os codigos é que futuramente caso seja necessario reaproveitar alguma das duas, preferencialmente a que apresentar melhor desempenho, para utilizar para construção de algum indice se tornaria mais facil a separação entre as funções.

Todas os metodos seguem a mesma extrutura:

- Fazem a leitura das palavras chaves em suas determinadas estruturas;
- Faz a leitura de todas as palavras em um texto, sendo que a cada palavra é buscado na estrutura se ela está presente, em caso positivo é armazenado o indice em que ela se encontra;
- É gerado um arquivo de saida com todas as palavras chaves que contém um indice de ocorrencia no texto.

2.1. HashExtern

A estrutura de dados implementada para este metodo foi a seguinte:

```
    struct linhas{
    int Linha;
    struct linhas *Prox;
};
struct nodo{
    char Palavra[50];
    struct linhas *IndiceI;
    struct linhas *IndiceF;
    struct nodo *Prox;
};
struct tabela{
    struct nodo *Inicio;
    struct nodo *Fim;
    int Vazio;
    int Colisoes;
};
typedef struct nodo *Nodo;
typedef struct linhas *Linha;
typedef struct tabela *TabelaExtern;
```

A TAD *externHash.c* foi implementada de acordo que pudesse armazenar e consultar esta lista de palavras com indereçamento externo e para isto foi disponibilizado as seguintes funções:

- **TabelaExtern ELeChaves(char Arq[], int Op, int* vetor)**;
- **int ETamanhoHash(void)**;
- **void EBuscaPalavras(TabelaExtern EHash, char Arq[], int Op, FILE *arquivo, int* vetor)**;
- **void EResultadoBusca(TabelaExtern EHash, FILE *arquivo)**;
- **void ELiberaHash(TabelaExtern EHash)**;
- **int* EPreencheVetorPesos(void)**.

2.2. Arvore Binaria de Busca não Balanceada

A estrutura de dados implementada para este metodo foi a seguinte:

```
struct linhas{
    int Linha;
    struct linhas *Prox;
};
struct nodo{
```

```
        char Palavra[50];
        struct linhas *IndiceI;
        struct linhas *IndiceF;
        struct nodo *Esq;
        struct nodo *Dir;
};

        typedef struct nodo *Nodo;
typedef struct linhas *Linha;
typedef struct nodo *Arv;
```

A TAD *arvoreBinaria.c* foi implementada de acordo que pudesse armazenar e consultar esta lista de palavras em uma arvore binaria e para isto foi disponibilizado as seguintes funções:

- **Arv ABLeChaves(char Arq[])**;
- **void ABResultadoBusca(Arv AArvore, FILE *arquivo)**;
- **void ABBuscaPalavras(Arv AArvore, char Arq[])**;
- **void ABLiberaArvore(Arv AArvore)**;

2.3. Lista Encadeada

A estrutura de dados implementada para este metodo foi a seguinte:

```
        struct linhas{
        int Linha;
        struct linhas *Prox;
};
struct nodo{
        char Palavra[50];
        struct linhas *IndiceI;
        struct linhas *IndiceF;
        struct nodo *Prox;
};
typedef struct nodo *Nodo;
typedef struct linhas *Linha;
typedef struct nodo *Lista;
```

A TAD *listaEncadeada.c* foi implementada de acordo que pudesse armazenar e consultar esta lista de palavras com uma lista encadeada e para isto foi disponibilizado as seguintes funções:

- **Lista LELeChaves(char Arq[])**;
- **void LEResultadoBusca(Lista PLista, FILE *arquivo)**;
- **void LEBuscaPalavras(Lista PLista, char Arq[])**;

- **void LELiberaLista(Lista PLista)**.

Para que não fosse necessario realizar uma ordenação dos indices todas as palavras chaves lidas eram inseridas na lista de forma ordenada, para diminuir o custo de ordenação.

2.4. Arvore AVL

A estrutura de dados implementada para este metodo foi a seguinte:

```
struct linhas{
        int Linha;
        struct linhas *Prox;
};
struct nodo{
        char Palavra[50];
        struct linhas *IndiceI;
        struct linhas *IndiceF;
        struct nodo *Esq;
        struct nodo *Dir;
        int Altura;
};

        typedef struct nodo *Nodo;
typedef struct linhas *Linha;
typedef struct nodo *ArvAVL;
```

A TAD *arvoreAVL.c* foi implementada de acordo que pudesse armazenar e consultar esta lista de palavras em uma arvore do Tipo AVL e para isto foi disponibilizado as seguintes funções:

- **ArvAVL AAVLLeChaves(char Arq[])**;
- **void AAVLResultadoBusca(ArvAVL AArvore, FILE *arquivo)**;
- **void AAVLBuscaPalavras(ArvAVL AArvore, char Arq[])**;
- **void AAVLLiberaArvore(ArvAVL AArvore)**;

3. Descrição dos Testes

Para que fosse possivel avaliar os metodos implementados, foram realizados uma bateria de testes, variando as funções de construção de indices, medindo o tempo de execução de cada uma,com um arquivo de texto similar a todas.

OBS. O tempo de escrita no arquivo de saida também foi contabilizado em cada um dos metodos.

4. Resultados

Os resultados de todos os testes podem ser analizados nas Tabelas 1, 2.

Table 1. Testes Texto 4

Metodo	Tempo
HashExterna	0.003081
ArvoreBinaria	0.003691
ListaEncadeada	0.003699
ArvoreAVL	0.003681

Table 2. Testes Texto 5

Metodo	Tempo
HashExterna	0.038584
ArvoreBinaria	0.075913
ListaEncadeada	0.129468
ArvoreAVL	0.052296

Pode observar que o tempo gasto pela tabela hash externa foi bem menor que os outros três metodos. Como sua consulta pode ser realizada em O(1) em seu melhor caso, isto já seria um resultado previamente suposto. É importante também constatar que além de ter o melhor tempo, ela utilizou de um metodo de ordenação externo, quicksort, já os outros metodos a inserção já era feita em ordem alfabetica.

Dois metodos foram relacionados a Arvores binarias, uma não balanceada e outra AVL, sendo esta ultma a que apresentou um melhor desempenho. Um pior caso para a arvore não balanceada seria a inserção dos valores ordenados, o que transformaria a arvore em uma lista encadeada e tornaria seu custo equivalente a lista. Já a arvore AVL que é constantemente balanceada ao inserir uma nova palavra tem um pequeno custo com as rotações de balanceamento, porém como após balanceada ela é constantemente verificada resulta em uma pequena melhora no tempo geral da estrutura de dados.

5. Conclusão

Este trabalho resultou em uma grande experiencia e conhecimento com relação a formas de implementar um indice redimissivo com diversas estruturas de dados. Além disto pode-se exercitar com um problema pratico, e comparar quatro formas de armazenamento para solucionar o mesmo problema

Neste caso o HashExterno apresentou um melhor resultado, um custo de tempo menor . Um destaque sobre a arvore AVL em relação a não balanceada também pode ser notado.

References

Ziviani, N. (2011). *Projeto de Algoritmos: Com Implementacoes em Pascal e C*. Cengage Learning, 3 edition.