Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

# Hashing para pesquisa de documentos

BCC202 - Estrutura de Dados I

Lucas Chagas, Nicolas Mendes, Pedro Morais

Professor: Pedro Henrique Lopes Silva

Ouro Preto
13 de março de 2023

# Sumário



# Lista de Figuras

# 1 Introdução

O objetivo deste trabalho pratico foi implementar uma tabela hash que ao inves de armazenar quais palavras existem em um documento, armazena quais documentos possuem uma certa palavra.

## 1.1 Especificações do problema

Nos arquivos proporcionados pelo professor, já havia algumas funções e constantes que definem tamanhos máximos e strings específicas, sendo elas:"VAZIO","N","D","ND","NM","M","MAX STR","h"e "pegarChaves". Tendo esta base foi exigido que implementássemos uma tabela hash que possui alguma palavra como chave e os documentos que possuem tal palavra dentro desta chave, além disso, os métodos desta tabela deverão permitir ao usuário ver todos os elementos da tabela hash ou buscar quais documentos possuem uma sequência de palavras informada pelo usuário.

## 1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: Visual Studio Code. [1]
- Linguagem utilizada: C.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [2]

## 1.3 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- *CLANG*: ferramentas de análise estática do código.
- *Valgrind*: ferramentas de análise dinâmica do código.

## 1.4 Especificações da máquina

A máquina onde o desenvolvimento e os testes foram realizados possui a seguinte configuração:

- Processador: Ryzen 7 3700x.
- Memória RAM: 8 Gb.
- Sistema Operacional: Ubuntu.

## 1.5 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

```
make
```

Usou-se para a compilação as seguintes opções:

- *-std=99*: para usar-se o padrão ANSI C 99, conforme exigido.
- *-g*: para compilar com informação de depuração e ser usado pelo Valgrind.
- *-Wall*: para mostrar todos os possível *warnings* do código.

Para a execução do programa basta digitar:

```
./exe ≪ arquivoDeEntrada.in
```

[1] Vscode está disponível em https://code.visualstudio.com/
[2] Disponível em https://www.overleaf.com/

# 2 Desenvolvimento

As funções de manipulação da hash implementadas no código(localizadas no arquivo "indiceInvertido.c"), consiste em zerar vetores, inserir valores em vetores, buscar elementos em vetores do tipo abstrato "IndiceInvertido", sendo elas:"inicia", "leDocumentos", "vazioTodosDocumentos", "inserePalavraChave","busca", "insereDocumento","h"e "consulta".

As funções auxiliares do código são responsáveis por imprimir, ler e ordenar vetores de variados tipos, sendo elas: "selecionaNaoVazio", "imprimeResultadoBusca", "ordena", "leOpcao"e "imprime".

## 2.1 Funções de hash

- "inicia": esta função recebe uma tabela hash do TAD "IndiceInvertido"coloca todas as posições do vetor em um estado vazio,zerando todos os campos "n"e igualando todos os campos "chave"a constante "VAZIO"definida anteriormente no programa.

```c
void inicia(IndiceInvertido dic)
{
    int i;
    for (i = 0; i < M; i++)
    {
        strcpy(dic[i].chave, VAZIO); // Copia vazio para as chaves
        dic[i].n = 0;                // Zera a quantidade de documentos em cada chave
    }
}
```

Figura 1: Função "inicia".

- "leDocumento": esta função recebe uma tabela hash do TAD "IndiceIvertido"e a quantidade de elementos que serão colocados na tabela. Com manipulação de strings, a função separa todas as chaves e documentos através de tokenização, após a separaçao, com o auxilio das funções: "insereDocumento"e "inserePalavraChave", a função insere a chave e seus respectivos documentos em uma posição da tabela.

```c
void leDocumento(IndiceInvertido dic, int n)
{
    for (int i = 0; i < n; i++)
    {
        // Leitura da linha
        char str[MAX_STR];
        fgets(str, MAX_STR, stdin);

        // Remove o \n
        str[strcspn(str, "\n")] = 0;

        // Tokenização da string
        char *aux = strtok(str, " ");

        // Nome do documento
        char nomeDocumento[D];
        int qtdPalavras = 0;

        // Inserções
        while (aux != NULL)
        {
            if (qtdPalavras == 0) // Nome do documentos
            {
                strcpy(nomeDocumento, aux);
                qtdPalavras++;
            }
            else // Palavras chaves associadas ao documento
            {
                // Insere as palavras chaves e o nome do documento associado
                inserePalavraChave(dic, aux);
                insereDocumento(dic, aux, nomeDocumento);
                qtdPalavras++;
            }

            aux = strtok(NULL, " ");
        }
    }
}
```

Figura 2: Função de leitura.

- "inserePalavraChave": esta função recebe uma tabela hash do TAD "IndiceIvertido"e uma "Chave". Com auxílio da função "busca"a função verifica se a chave já existe na tabela e onde há um local válido para a inserção da palavra chave caso ela não esteja lá ainda.


Figura 3: Função de inserção.

- "insereDocumento": esta função recebe uma tabela hash do TAD "IndiceIvertido", uma "Chave"e uma string de documentos. com auxílio da função "busca", a funçao acha a posição da chave recebida por parâmetro, verifica se a quantidade de documentos naquela chave já atingiu seu limite e caso não tenha atingido, insere os documentos no campo da chave e incrementa a quantidade de documentos no campo "n".

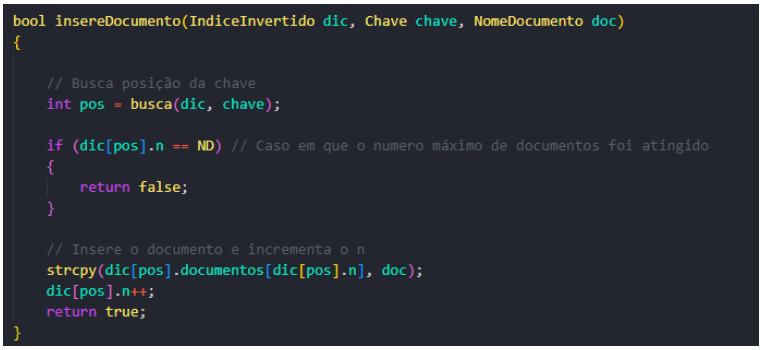



Figura 4: Função de inserção.

- "busca":esta função recebe uma tabela hash do TAD "IndiceIvertido"e uma "Chave". Usando da função "h"que retorna o local ideal para a chave estar, a função verifica se a posição ideal já está ocupada por outra chave e procura por outra posição disponível ou se a chave já está inserida na tabela. O retorno da função é a posição válida ou a posição da chave caso ela já esteja na tabela.


Figura 5: Função de busca.

- "h": função implementada para calcular a posição ideal de uma dada chave como parâmetro baseado em pesos definidos dentro da função.

```c
int h(char * chave) {
    float p[] = {0.8326030060567271, 0.3224428884580177,
                 0.6964223353369197, 0.1966079596929834,
                 0.8949283476433433, 0.4587297824155836,
                 0.5100785238948532, 0.05356055934904358,
                 0.9157270141062215, 0.7278472432221632};
    int tamP = 10;
    unsigned int soma = 0;
    for (int i=0;i<strlen ( chave );i++)
        soma += (unsigned int) chave [i] * p[i % tamP];
    return soma % M;
}
```

Figura 6: Função de hash.

- "consulta": esta função recebe uma tabela hash do TAD "IndiceIvertido", um vetor de "Chave", a quantidade de chaves no vetor, e um vetor de documentos passado por referência. Em primeiro lugar, a função busca a primeira chave do vetor na tabela hash, caso ela ache, ela coloca todos os documentos da primeira chave no vetor de documentos passado por referência(resultados). Após este passo, a próxima chave do vetor é buscada, e quando achada, os documentos do vetor "resultado"são comparados com os da chave atual, aqueles documentos que pertencerem aos 2 vetores permaneceram no vetor "resultado", e aquelas não pertencerem aos 2, serão substituídos por "VAZIO". este processo é repetido para todas as chaves, fazendo com que no final da execução o vetor "resultado"contenha apenas os documentos que estão presentes em todas as chaves.

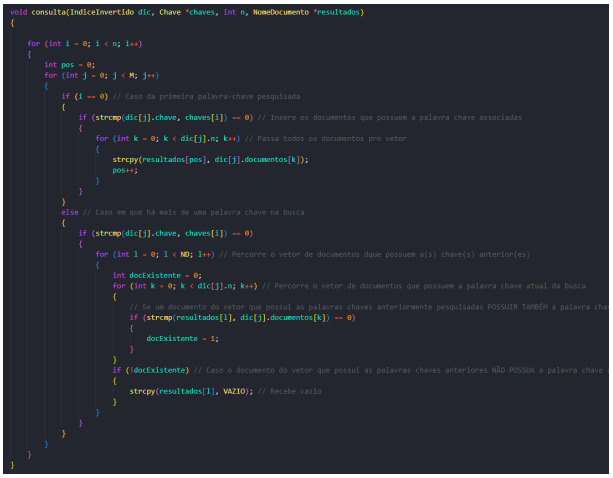



Figura 7: Função de consulta.

- "vazioTodosDocumentos": esta função recebe um vetor do TAD "NomeDocumentos"e o preenche com a string "VAZIO".

```c
void vazioTodosDocumentos(NomeDocumento *documentos)
{
    for (int i = 0; i < ND; i++)
    {
        strcpy(documentos[i], VAZIO);
    }
}
```

Figura 8: Função de reset.

## 2.2 Funções auxiliares

- "selecionaNaoVazio": esta função recebe 2 vetores do TAD "NomeDocumento"e um inteiro representando quantidade. Um dos vetores contém informação e o outro o vetor irá receber apenas os valores válidos(aqueles que não estão preenchidos com "VAZIO").

```c
void selecionaNaoVazio(NomeDocumento *origem, NomeDocumento *destino, int *qtd)
{
    int pos = 0;
    // For que percorre todo o vetor de documentos
    for (int i = 0; i < ND; i++)
    {
        if (strcmp(origem[i], VAZIO) != 0) // Caso o documento seja válido, ele será copiado para o próximo vetor
        {
            strcpy(destino[pos], origem[i]);
            pos++;
        }
    }
    // Retorna a quantidade de documentos válidos por referência
    *qtd = pos;
}
```

Figura 9: Função de busca.

- "imprimeResultadoBusca": esta função recebe um vetor do TAD "NomeDocumento"e um inteiro representando quantidade de elementos. Quando a quantidade de elementos for 0 a função imprime "none"no terminal, caso contrário a função imprime todo o vetor no terminal.

```c
void imprimeResultadoBusca(NomeDocumento *documentos, int tamNaoVazio)
{
    // Caso em que não há nenhum documento que corresponde atenda as demandas da busca
    if (tamNaoVazio == 0)
    {
        printf("none\n");
        return;
    }
    else // Caso em que há um ou mais documentos que atendem as demandas da busca
    {
        for (int i = 0; i < tamNaoVazio; i++)
        {
            printf("%s\n", documentos[i]);
        }
        return;
    }
}
```

Figura 10: Função de impressão.

- "ordena": esta função é um quicksort adaptado para vetores de strings do TAD "NomeDocumento", que ordena um vetor recebido por parâmetro de maneira crescente,tendo como escolha do pivô,para a ordenação,o elemento inicial do vetor. Por conta do comportamento da tabela hash, a função também verifica os endereços de memória válidos para evitar erros de overlap.

```c
void ordena(NomeDocumento *documentos, int inicio, int final) // QuickSort para ordenar os documentos em ordem alfabética crescente
{
    int pivo, esquerda, direita;
    NomeDocumento aux;
    pivo = inicio;
    esquerda = inicio;
    direita = final;

    while (esquerda <= direita)
    {
        while ((esquerda < final) && (strcmp(documentos[esquerda], documentos[pivo]) < 0))
        {
            esquerda++;
        }
        while ((direita > inicio) && (strcmp(documentos[direita], documentos[pivo]) > 0))
        {
            direita--;
        }
        if (esquerda <= direita)
        {
            if (&aux != &documentos[esquerda]) //IF para evitar o overlap de strings - copiar um endereço para ele mesmo
            {
                strcpy(aux, documentos[esquerda]);
            }
            if (&documentos[esquerda] != &documentos[direita])
            {
                strcpy(documentos[esquerda], documentos[direita]);
            }
            if (&documentos[direita] != &aux)
            {
                strcpy(documentos[direita], aux);
            }
            esquerda++;
            direita--;
        }
    }
    if (direita > inicio)
    {
        ordena(documentos, inicio, direita);
    }
    if (esquerda < final)
    {
        ordena(documentos, esquerda, final);
    }
}
```

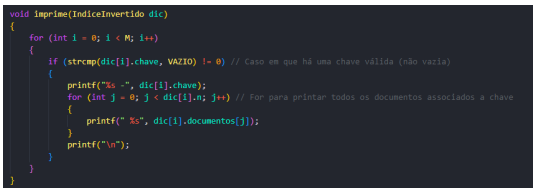
Figura 11: Função de ordenação.

- ”leOpcao”: esta função recebe uma variável tipo char por referência, um vetor de chaves e uma variável que representa a quantidade de chaves passada por referência. A função extrai o comando do usuário(I ou B) e armazena no char passado por parâmetro caso haja chaves.

```c
void leOpcao(char *c, Chave *chaves, int *quantChaves)
{
    // Le string
    char str[MAX_STR];
    fgets(str, MAX_STR, stdin);

    // Remove \n
    str[strcspn(str, "\n")] = 0;

    // Tokenização da string
    char *aux = strtok(str, " ");

    // Opção do user: B ou I
    *c = aux[0];

    aux = strtok(NULL, " ");

    int posChaves = 0;

    while (aux != NULL)
    {
        strcpy(chaves[posChaves], aux);
        aux = strtok(NULL, " ");
        posChaves++;
    }

    *quantChaves = posChaves;
}
```

Figura 12: Função de leitura.

- ”imprime”: esta função recebe uma tabela hash do TAD ”IndiceInvertido”. A função imprime todas as chaves válidas e seus respectivos documentos no terminal.



```c
void imprime(IndiceInvertido dic)
{
    for (int i = 0; i < M; i++)
    {
        if (strcmp(dic[i].chave, VAZIO) != 0) // Caso em que há uma chave válida (não vazia)
        {
            printf("%s -", dic[i].chave);
            for (int j = 0; j < dic[i].n; j++) // For para printar todos os documentos associados a chave
            {
                printf(" %s", dic[i].documentos[j]);
            }
            printf("\n");
        }
    }
}
```

Figura 13: Função de impressão.

# 3 Experimetos e Resultados

Foram realizados vários testes com diversos tipos de input, com isso conseguimos corrigir alguns erros ao longo do processo,tendo entraves principalmente função ”consulta”, tendo em vista que há praticamente toda a lógica do trabalho prático nesta função, também houve problemas com o método de ordenação, que deveria realizar todo o processo em menos de 1 segundo, e o erro ”overlap”apontado pelo valgrind, nos levando a tomar certas ações para evitar tal problema, o método de ordenação usado foi o quicksort devido sua simples implementação e complexidade adequada, e o que foi implementado para evitar o erro de overlap foi verificar dentro do próprio quicksort se as posições analisadas eram válidas.

# 4 Considerações Finais

O código foi capaz de lidar com todos os casos de teste propostos no ”runcodes”e não apresentou nenhum erro na verificação do valgrind. O código ficou suficientemente intuitivo, sendo comentado e indentado em toda sua constituição.