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
17 de março de 2023

Sumário

Lista de Códigos Fonte

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.

```
void inicia(IndiceInvertido dic)
{
    int i;
    for (i = 0; i < M; i++)
    {
        strcpy(dic[i].chave, VAZIO);
        dic[i].n = 0;
    }
}
```

Código 1: Código da função inicia indice invertido

- "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.Quanto ao gasto de memória da função,isto é utilizado pelo vetor "str",com tamanho "MAX-STR"(20001) e também pelo vetor "nomeDocumento",com o tamanho "D"(51).

```
void leDocumento(IndiceInvertido dic, int n, int *colisoes)
{
    for (int i = 0; i < n; i++)
    {

        char str[MAX_STR];
        fgets(str, MAX_STR, stdin);


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


        char *aux = strtok(str, " ");


        char nomeDocumento[D];
        int qtdPalavras = 0;


        while (aux != NULL)
        {
            if (qtdPalavras == 0)
            {
                strcpy(nomeDocumento, aux);
                qtdPalavras++;
            }
```

```
            else
            {

                inserePalavraChave(dic, aux, colisoes);
                insereDocumento(dic, aux, nomeDocumento);
                qtdPalavras++;
            }

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

Código 2: Código da função leDocumento

- "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.

```
bool inserePalavraChave(IndiceInvertido dic, Chave chave, int *colisoes)
{

    int pos = busca(dic, chave);
    if (pos != -1)
    {
        return false;
    }

    int j = 0;
    int ini = h(chave);
    while ((strcmp(dic[(ini + j) % M].chave, VAZIO) != 0) && (j < M))
    {
        j++;
        *colisoes = *colisoes + 1;
    }

    if (j < M)
    {
        strcpy(dic[(ini + j) % M].chave, chave);
        return true;
    }

    return false;
}
```

Código 3: Código da função inserePalavraChave

- "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".

```
    bool insereDocumento(IndiceInvertido dic, Chave chave, NomeDocumento
        doc)
{

    int pos = busca(dic, chave);

    if (dic[pos].n == ND)
    {
        return false;
```

```
    }

    strcpy(dic[pos].documentos[dic[pos].n], doc);
    dic[pos].n++;
    memoria++;
    return true;
}
```

Código 4: Código da função insereDocumento

- "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.

```
int busca(IndiceInvertido dic, Chave chave)
{
    int j = 0;
    int ini = h(chave);


    while ((strcmp(dic[(ini + j) % M].chave, VAZIO) != 0) && (strcmp(dic
        [(ini + j) % M].chave, chave) != 0) && (j < M))
    {
        j++;
    }

    if (strcmp(dic[(ini + j) % M].chave, chave) == 0)
    {
        return (ini + j) % M;
    }

    return -1;
}
```

Código 5: Código da função busca

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

```
    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;
}
```

Código 6: Código da função h

- "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.

```
    void consulta(IndiceInvertido dic, Chave *chaves, int n,
        NomeDocumento *resultados)
{

    for (int i = 0; i < n; i++)
    {
        int pos = 0;
        for (int j = 0; j < M; j++)
        {
            if (i == 0)
            {
                if (strcmp(dic[j].chave, chaves[i]) == 0)
                {
                    for (int k = 0; k < dic[j].n; k++)
                    {
                        strcpy(resultados[pos], dic[j].documentos[k]);
                        pos++;
                    }
                }
            }
            else
            {
                if (strcmp(dic[j].chave, chaves[i]) == 0)
                {
                    for (int l = 0; l < ND; l++)
                    {
                        int docExistente = 0;
                        for (int k = 0; k < dic[j].n; k++)
                        {

                            if (strcmp(resultados[l], dic[j].documentos[k
                                ]) == 0)
                            {
                                docExistente = 1;
                            }
                        }
                        if (!docExistente)
                        {
                            strcpy(resultados[l], VAZIO);
                        }
                    }
                }
            }
        }
    }
}
```

Código 7: Código da função consulta

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

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

```
}
```

Código 8: Código da função vazioTodosDocumentos

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").

```
    void selecionaNaoVazio(NomeDocumento *origem, NomeDocumento *destino,
         int *qtd)
{
    int pos = 0;
    for (int i = 0; i < ND; i++)
    {
        if (strcmp(origem[i], VAZIO) != 0)
        {
            strcpy(destino[pos], origem[i]);
            pos++;
        }
    }
    *qtd = pos;
}
```

Código 9: Código da função selecionaNaoVazio

- "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.

```
void imprimeResultadoBusca(NomeDocumento *documentos, int tamNaoVazio)
{

    if (tamNaoVazio == 0)
    {
        printf("none\n");
        return;
    }
    else
    {
        for (int i = 0; i < tamNaoVazio; i++)
        {
            printf("%s\n", documentos[i]);
        }
        return;
    }
}
```

Código 10: Código da função imprimeResultadoBusca

- "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 para evitar erros de overlap. Como o algoritmo em questão é um quicksort, ele acaba gastando log(n) em espaço, sendo "n"um vetor de documentos com um máximo de 100 documentos sendo cada documento um vetor tipo string com 51 casas.

```
void ordena(NomeDocumento *documentos, int inicio, int final)
{
    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])
            {
                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);
    }
}
```

Código 11: Código da função ordena

- "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. O custo de memoria desta função cai sobre a criaçao do vetor "str"com tamanho "MAX STR"(20001).

```
void leOpcao(char *c, Chave *chaves, int *quantChaves)
{

    char str[MAX_STR];
    fgets(str, MAX_STR, stdin);


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


```

```
    char *aux = strtok(str, " ");


    *c = aux[0];

    aux = strtok(NULL, " ");

    int posChaves = 0;

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

    *quantChaves = posChaves;
}
```

Código 12: Código da função leOpção

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

```
void imprime(IndiceInvertido dic)
{
    for (int i = 0; i < M; i++)
    {
        if (strcmp(dic[i].chave, VAZIO) != 0)
        {
            printf("%s -", dic[i].chave);
            for (int j = 0; j < dic[i].n; j++)
            {
                printf(" %s", dic[i].documentos[j]);
            }
            printf("\n");
        }
    }
}
```

Código 13: Código da função imprime

2.3 Funções para coleta de informações

- "imprimeColisoes": esta função irá imprimir a quantidade de colisões ocorrida na execução de cada caso de teste no código.A função é estruturada a partir de uma variável inicializada na main e é passada por referência para a função "busca"e cada colisão é imcrementado 1 unidade para a variável.

```
void imprimeColisoes(int colisoes)
{
    printf("Quantidade de colisoes: %d\n", colisoes);
}
```

Código 14: Código da função imprimeColisoes

- "tempoDeExecução": esta funcionalidade implentada na main,irá calcular o tempo de execução de cada caso de teste feito no código.O cálculo do tempo é feito pela subtração do final pelo início, dividido pelo "CLOCKS-PER-SEC".

```
    double tempo_gasto = 0.0;
    clock_t begin = clock();
```

```
3
4       clock_t end = clock();
5       tempo_gasto += (double) (end - begin) / CLOCKS_PER_SEC;
6       printf("Tempo gasto %f s\n", tempo_gasto);
```

Código 15: Parte do código da "main"para o cálculo do tempo de execução

- "imprimeMemoria": esta irá imprimir a quantidade de memória gasta em cada caso de teste executado no código, à vista disso, o cálculo do gasto de memória será igual ao tamanho do vetor multiplicado pelos bytes de cada elemento do vetor. Sendo o primeiro vetor "Chaves chave"utilizado na função "leOpcao"e na função "consulta", o vetor "IndiceInvertido dic"utilizado na maioria das funções, o vetor auxliar para o nome do documento e vetor de string utilizado na função "leDocumento", o vetor "NomeDocumento resultados"utilizado na função de "consulta"e na "selecionaNaoVazios"e o vetor "NomeDocumento naoVazios"utilizado na função de ordenação(ordena) e na função de "imprime".

```
1   void imprimeMemoria()
2   {
3       int memoria = (M * sizeof(Item)) + (D * sizeof(char)) + (MAX_STR *
                  sizeof(char)) + (ND * sizeof(NomeDocumento)) + (ND *
              sizeof(NomeDocumento)) + (M * sizeof(Chave));
4       printf("Memoria gasta: %d bytes\n", memoria);
5   }
```

Código 16: Código da função imprimeMemoria

3 Análise do Experimento e Resultados

Foram realizados vários testes com diversos tipos de input, com isso conseguimos corrigir alguns erros ao longo do processo,tendo entraves principalmente na função "consulta", tendo em vista que há praticamente toda a lógica do trabalho prático nesta função e também na função "imprime", 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.Com isso, conclui-se que o código implementado cumpriu todos os requisitos propostos e também todos os casos de teste, sendo assim mostrado que cada tópico da disciplina foi aplicado de maneira produtiva e eficiente para atingir o melhor funciomento do código e alcançar a consolidação do conteúdo entre os membros do grupo.

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.