Ccomp | Ciência da Computação

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL REI

AEDS II Trabalho Prático

Alunos:
Vinicius de Almeida Lima (212050070)
Matheus Duraes da Cunha Pereira (212050094)

Professor: Daniel Madeira

Introdução

No trabalho prático de AEDS 2 nos foi proposto que criássemos um programa para uma empresa de software que tem como principal produto uma ferramenta de análise de textos na Web. O trabalho consiste em fazer um programa que conte quantas vezes a mesma palavra aparece num texto de entrada, determinando assim a sua frequência.

Para solucionar o problema, foi criado um programa que cataloga e armazena todas as palavras de um texto de entrada, as linhas e a frequência em estão presentes em uma tabela. Em seguida, a partir de uma lista de palavras, ele apresenta os resultados de acordo com os parâmetros listados anteriormente.

Implementação

Para a implementação do programa que resolva o problema proposto, nós dividimos a implementação em 3 etapas

1.0 A primeira etapa: Construção da tabela hash e suas funções primordiais

Sendo a primeira delas referente a: Inicializar tabela, inserção de elementos na tabela, busca de elementos na tabela e pela natureza da tabela, uma função responsável por calcular o hash das palavras a serem inseridas, e pesquisadas na tabela. A seguir vamos apresentar e aprofundar a implementação do código.

1.1 Inicialização da tabela:

Para implementar uma tabela hash de forma a reduzir ao máximo seu número de colisões, utilizamos o número máximo de entradas (256) a multiplicamos por 2, e encontramos o número primo mais próximo (521), assim definindo o tamanho da tabela, a seguir criamos um “array” de 521 posições e rodamos o loop “for” para zerar cada posição da tabela, visando evitar memory dumps.

```
void inicializarTabela(Palavra t[]){
    int i;
    for(i = 0; i < TAM; i++)
        strcpy(t[i].palavra, "");
}
```

Figura 1: função de inicializar a tabela
Fonte: Elaborada pelos autores, 2022.

1.2 Estrutura das células da tabela:

Visando garantir maior otimização possível, a solução encontrada para armazenar a palavra, sua frequência e as linhas em que estão presentes, foi uma tabela hash de structs, onde cada struct representa uma palavra e contém as informações de uma string referentes à

palavra, um inteiro que indica a frequência e um array de inteiros referente as linhas onde se encontram as palavras.

```
typedef struct {
    char palavra[TAMPALAVRA];
    int frequencia;
    int linhas[TAMLINHA];
}Palavra;
```

Figura 2: Struct palavra
Fonte: Elaborada pelos autores, 2022.

Foram definidos como constantes, o tamanho da palavra e o número de linhas máximas do texto. Caso o texto possua mais linhas que o definido previamente deve se alterar a variável “TAMLINHA” para um tamanho adequado ao número de linhas do texto.

```
#define TAMPALAVRA 21
#define TAMLINHA 1000
```

Figura 3: Tamanho dos vetores de palavra e linhas
Fonte: Elaborada pelos autores, 2022.

1.3 Função Hash:

Quanto à implementação da função hash, foram utilizadas duas funções. A primeira função hash é a responsável por gerar um índice à partir da palavra que seráinserida, esse índice é gerado por meio do somatório do código ASCII de cada caractere multiplicado pela sua posição na string. Ao término do somatório utilizaremos a segunda função essa responsável por calcular o resto da divisão do somatório total da palavra pelo tamanho da tabela (521).

```
int funcaoHashString(char str[]){
    int i, tamS = strlen(str);
    unsigned int hash = 0;

    for(i = 0; i < tamS; i++)
        hash += str[i] * (i + 1);
    return hash % TAM;
}

int funcaoHash(int chave){
    return chave % TAM;
}
```

Figura 4: Funções de Hash
Fonte: Elaborada pelos autores, 2022

1.4 Inserção na tabela Hash:

A função de inserção recebe como parâmetros a tabela Hash, a palavra a ser inserida na tabela e por último a linha em que ela se encontra.A partir dos parâmetros, é utilizada a função hash para encontrar o índice da palavra na tabela. Inicialmente, o programa assume que essa é a primeira ocorrência da palavra, logo preenche o número de frequência como 1. E a posição inicial do array com a linha que a palavra aparece. A seguir, verifica se a posição referente ao ID gerado pela função Hash está ocupado:

1.4.1: Caso não esteja, ela preenche a struct dessa posição de forma inicialmente sugerida, com a palavra, frequência 1 e o número da linha na posição inicial do array;

1.4.2: Caso esteja, é verificado se a palavra que ocupa essa posição é a mesma palavra a ser inserida, se sim, o número de frequência é incrementado e inserimos a linha recebida como parâmetro, no array de linhas da struct Palavra.

1.4.3: Se não, caso a palavra que se encontra na posição gerada pela função hash seja diferente da palavra a ser inserida, temos uma colisão. As colisões serão tratadas por endereçamento aberto, sendo assim o processo de inserção se repetirá na próxima célula da tabela hash, até que aconteça um dos dois casos acima (1.4.1 e 1.4.2).

1.5 Busca na tabela Hash:

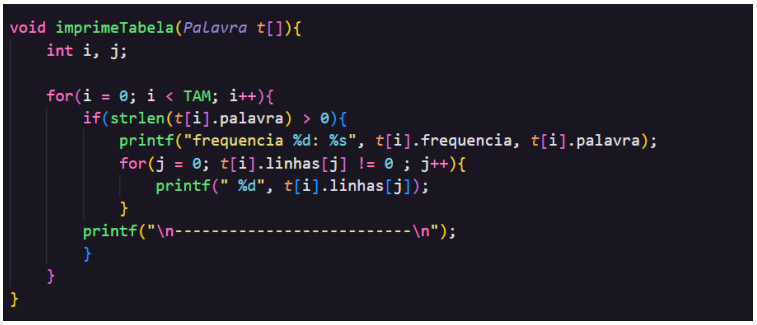
A função de busca foi implementada tendo em vista que ela recebe dois parâmetros a tabela hash e a palavra a ser pesquisada, é utilizado o ID da palavra por meio da função de hash e a partir disso, rodamos um loop “while” até que a palavra seja encontrada, o que nos leva a dois casos:

1.5.1 Caso encontrada: O programa vai imprimir, seu número de frequência que está contido em t[id].frequencia, a palavra que esta contida em t[id].palavra e as linhas que estão contidas em t[id].linhas

1.5.2 Caso não encontrada: O programa apenas imprime que a palavra não foi encontrada.

1.6 Imprimir tabela:

A função de imprimir a tabela, foi implementada apenas com o intuito de facilitar o desenvolvimento do código, não sendo utilizada em todo o código, caso deseje imprimir toda a tabela, basta descomentar a função “imprimirTabela” , que está na função main do código. A função percorre toda a tabela, e ao encontrar uma célula que não está preenchida, a imprime.



```
void imprimeTabela(Palavra t[]){
    int i, j;

    for(i = 0; i < TAM; i++){
        if(strlen(t[i].palavra) > 0){
            printf("frequencia %d: %s", t[i].frequencia, t[i].palavra);
            for(j = 0; t[i].linhas[j] != 0 ; j++){
                printf(" %d", t[i].linhas[j]);
            }
        printf("\n-------------------------\n");
        }
    }
}
```

Figura 5: Função de inicializar tabela Fonte: Elaborada pelos autores, 2022

2.0 A segunda etapa: Manipulação e filtragem das palavras do texto de entrada

É referente a implementação como todas as palavras do texto são transformadas para apenas letras minúsculas possibilitando que elas se tornem “case insensitive” , outra funcionalidade desta etapa incluiu que o programa ignorasse caracteres especiais como solicitado no trabalho.

Vale ressaltar que, como as instruções apontavam que acentuação não seria importante, não foi implementado uma funcionalidade que se relacione a isso.

2.1 Manipulação das palavras do texto de entrada:

Para manipularmos apenas palavras implementamos um simples algoritmo que lê apenas caracteres que são letras do alfabeto, caracteres na tabela ASCII entre 65 e 90(letras maiúsculas) e entre 97 e 122(letras minúsculas), cada letra é salva num array de letras, após encontrar um carácter que não seja uma letra, ele forma a palavra com as letras que ele já haviam sido armazenadas. Caso seja uma palavra de apenas uma letra ela é descartada. O algoritmo então irá transformar todas as letras para minúsculas possibilitando que elas se tornem “case insensitive". Com a palavra toda em letras minúsculas ele utiliza a função de inserção da palavra na tabela, fazendo isso até o EOF onde se encerra a etapa de inserção das palavras do texto de entrada na tabela.

A função implementada para fazer esse processo recebe como parâmetros a tabela Hash e o input de texto de entrada.

```
void LeInput(Palavra t[], char **argv) {...}
```

Figura 6: Função de ler o texto de entrada
Fonte: Elabora pelos autores, 2022.

3.0 A terceira etapa: Manipulação, busca e apresentação do resultado obtido pela execução do programa no terminal

O arquivo de pesquisa é de estrutura fixa onde na primeira linha, apresenta um número inteiro N, que diz respeito ao número de palavras que vão ser pesquisadas. As palavras a serem pesquisadas então são apresentadas nas linhas subsequentes. Devido ao modelo fixo a implementação foi baseada em:

Utilizar o número de palavras a serem pesquisadas (Primeira linha) e então criar um loop “for” onde a cada repetição a palavra a ser pesquisada é lida e então inserida como parâmetro na função de busca previamente apresentada.

A função implementada para fazer esse processo recebe como parâmetros a tabela Hash e o input de texto de pesquisa.

```
void LePesquisa(Palavra t[], char **argv) {...}
```

Figura 7: Função de ler a lista de pesquisa
Fonte: Elaborada pelos autores, 2022

4.0 Resultados obtidos:

Para testarmos a funcionalidade do programa utilizamos os textos de entrada e pesquisa disponibilizados nas orientações do trabalho. Foi notado entretanto que havia uma divergência quanto ao tamanho das linhas presente no texto de entrada. Algumas linhas do texto de entrada possuíam mais do que 80 caracteres, diferente do informado pelas orientações do trabalho, devido a isso foi modificado o tamanho das linhas do texto de entrada para 100 caracteres, de forma a comportar o texto de teste fornecido.

Assim, os exemplos de entrada e saída obtivemos o mesmo resultado apresentado. O programa lê o arquivo de pesquisa, onde estão contidas as palavras para teste, e imprime suas informações conforme encontra-as na tabela hash.

```
./a.out input.txt pesquisa.txt
5 lagarto 1 2 3
5 papel 1 2 3 4
5 pedra 1 2 4
5 spock 1 3 4
5 tesoura 1 2 3 4
2 esmaga 2 3
```

Figura 8: Output da execução do programa
Fonte: Elaborada pelos autores, 2022

5.0 conclusão:

Em geral concluímos que a experiência da realização do trabalho foi proveitosa e envolveu de forma coerente os conteúdos trabalhados na matéria de AEDS 2. Sendo portanto uma maneira efetiva e desafiadora de aprendizado e prática.

A principal dificuldade encontrada foi implementar de maneira inteligente uma forma de armazenar a frequência, a palavra e as linhas em que estão presentes em um mesmo local, visando facilitar a busca. Além da implementação de um algoritmo que leia apenas palavras, ignorando os caracteres especiais.

Referências

Para realizar esse trabalho usamos os slides dispostos no Campus Virtual da Universidade Federal de São João Del-Rei (UFSJ), na disciplina de Algorítimo e Estrutura de Dados II: https://campusvirtual.ufsj.edu.br/portal/2022_2/course/view.php?id=431