Trabalho Prátioco de AEDS II

Gustavo Henriques da Cunha[1]

[1]Ciência da Computação
Universidade Federal de São João del-Rei (UFSJ)

gustavohenrique13579@gmail.com

Resumo. *Este trabalho tem como objetivo usar do conteúdo visto em sala de aula, mais especificamente a pesquisa usando da técnica Hash.*

1. Introdução

Neste trabalho, é pedido que seja implementado um sistema de análise de textos, no qual deve ser capaz de pesquisar a ocorrência e as linhas em que apareceram qualquer palavra que seja solicitada. Para tal, deve ser construido uma tabela *hash* com endereçamento aberto. Além disso, é importante notar que o texto de entrada não possui palavras com acentos, e que palavras com uma única letra são ignoradas, e o programa é *case sensitive*.

O arquivo de entrada contém sempre no máximo 256 palavras diferentes, com cada palavra tendo 20 letras ou menos, além de que cada linha terá no máximo 80 caracteres. O programa foi feito tendo esses dados em mente, e poderá ocorrer erros caso executado fora dessas condições.

Para a compilação do programa, foi feito um arquivo ”Makefile”, que permite compilar tudo com apenas um comando. Já na parte de execução, deve ser passado dois arquivos, onde o primeiro deverá possuir o texto a ser lido e o segundo uma lista com as palavras a serem buscadas.

2. Implementação

O programa foi divido em dois módulos mais o *main*.

O primeiro módulo é o responsável pela função *hash*, e por criar, adicionar e pesquisar os elementos na tabela. A função *hash*, provavelmente a parte mais importante de todo o trabalho, foi feita de uma maneira bem simples. Esse código é baseado na função *hash* 'djb2' de Daniel J. Bernstein, e utiliza o numero 33 para multiplicar o resultado do *hash* em cada iteração, passando por cada caracter da palavra, além de ser somado o código da tabela ASCII correspondente a cada letra. Inicialmente o valor de *hash* recebe 257, pois como 33, é um primo com bons resultados e não é muito grande.

A tabela possui um tamanho de 521. Esse valor foi escolhido por ser o primeiro primo após o dobro do número máximo de palavras no texto. A tabela ficou grande em relação ao número de dados, mas julguei ser adequado, já que o conjunto de dados é pequeno, e dobrar esse valor não ocuparia uma quantidade grande de memória.

Para inserir uma palavra na tabela, usamos o módulo da divisão do *hash* calculado por 521, o tamanho da tabela. Para resolver as colisões foi usado endereçamento aberto, assim se a posição já estiver ocupada, vai se procurar a próxima posição livre para inserir o dado.

Cada palavra de uma mesma linha são tratadas como a mesma. Para contar o número de aparições de uma palavra no texto, foi usado um contador de aparições em uma linha. Já se uma mesma palavra aparece em diferentes linhas, ela é adicionada uma vez por cada linha. Seu *hash*, no entanto, não muda, e o programa utiliza do endereçamento aberto como vantagem na busca de uma palavra. Para exibir em quais linhas cada palavra aparece, também é salvo a linha em que a palavra apareceu.

Para procurar uma palavra, o programa calcula seu *hash* e busca pela palavra diretamente na posição, e só para essa busca quando acha um espaço vazio. Essa busca irá retornar quantas vezes a palavra apareceu e as linhas em que ela apareceu.

Para guardar e imprimir as linhas em que cada palavra aparecem, foi utilizada de uma lista dinâmica, implementada no *header* 'list.h'. Esse arquivo possui a declaração de uma lista duplamente encadeada. Para a resolução desse problema, no entanto, não era necessário que essa lista fosse duplamente encadeada, ou possuise algumas funcionalidades que estão presentes no arquivo, mas eu apenas aproveitei de um código que implementei alguns meses atrás, e não vi problemas em ela ter funcionalidades a mais.

O arquivo *main* apenas lê os arquivos e chama as funções para criar, adicionar e pesquisar as palavras na tabela, imprimindo tudo no final.

3. Resultados e Discussões

Através de alguns testes, pude verificar que o código funciona bem com diferentes textos, e não deve apresentar nenhum *bug* dentro das especificações dadas.

Por se tratar de um número pequeno de dados, é difícil verificar a eficácia do *hash* apenas com o tempo de execução, pois tudo ocorre praticamente de forma instantánea. No entanto, exatamente por se tratar de um número pequeno de dados, pude imprimir toda a tabela e verificar facilmente o número de colisões. No geral, minha reação foi positiva. Apesar de em alguns momentos se formarem alguns blocos de palavras com o hash similar, ao todo, a distribuição das palavras foi muito boa, com grande parte das buscas podendo ser realizadas com menos de cinco comparacões.

4. Conclusão

Devido a simplicidade do código e aos resultados observados, considero está uma solução muito boa. Não tive grandes problemas na realização desse trabalho, apenas alguns poucos contratempos e problemas que ocorreram devido a eu não estar totalmente acostumado a alguns aspectos da lingaguem C.

Mas no final de tudo, um problema tão simples como o proposto não se encaixava em uma solução complexa. No entanto, caso o número de dados fosse maior, ou existe mais condições na entrada dos textos, seria necessário melhorar essa solução, ou recomeçar do zero, de acordo com as novas especificações, pois como foi dito anteriormente, essa solução levou em conta os dados de entrada, e é uma bem especifica a esse problema.

Referências

D. Patrick (https://math.stackexchange.com/users/12551/d-patrick). (26 ago. 2011). *Curious Properties of 33*. em: https://math.stackexchange.com/q/59920. Acesso em: 8

dez. 2022.

UEHARA, L. *Introdução ao Makefile*. Disponível em: https://embarcados.com.br/introducao-ao-makefile/. Acesso em: 8 dez. 2022.

York University (CA). *Hash Functions*. Disponível em: http://www.cse.yorku.ca/ oz/hash.html. Acesso em: 8 dez. 2022.