UNIVERSIDADE FEDERAL DE MINAS GERAIS – UFMG
GRADUAÇÃO EM ENGENHARIA ELÉTRICA, 2° PERÍODO
Algoritmos e Estruturas de Dados II – TA2
Profª Gisele L. Pappa

Relatório - Trabalho Prático 03

Máquina de Busca

Nander Santos do Carmo - 2018019931

05 de dezembro de 2018

Belo Horizonte

Sumário

Introdução

Um dos desafios da computação está centrado basicamente na manipulação de dados de forma mais eficiente o possível, gastando o menor tempo para execução de um determinado algoritmo, busca-se obter um resultado satisfatório. Contudo, como cada vez mais, junto com o desenvolvimento de novas máquinas dotadas de maior capacidade de processamento, estamos criando programas mais complexos podemos afirmar que se torna cada vez mais necessário que as ações realizadas por nós nos computadores sejam mais eficientes.

Quando pensamos em eficiência de algoritmos computacionais geralmente acabamos por cair em dois grupos básicos, porém muito importantes para qualquer programa mais avançado: manipulação e busca de dados. Assim, visando aumentar a eficiência de um determinado programa que está em desenvolvimento, além de otimizar a forma como os dados serão manipulados, também é preciso otimizar a velocidade com a qual o programa encontra um dado salvo, para poder assim, processá-lo e executar seus comandos.

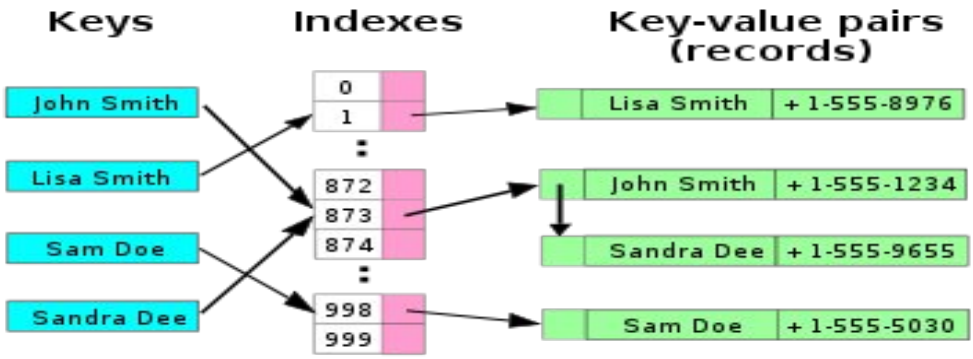
Com isso em mente, este trabalho teve como finalidade realizar o desenvolvimento de uma "Máquina de Busca" de dados, na qual, dado arquivos de consulta pode-se retornar rapidamente se uma palavra (sentença) está ou não presente nesses arquivos. Neste caso, a máquina não seguirá o modelo padrão proposto para internet, mas sim, será baseada em uma tabela Hash para armazenagem dos dados.

Implementação

Estrutura de Dados

Como precisaremos de uma tabela hash (exemplo abaixo na imagem) para realizar a procura das palavras da máquina de busca, para a implementação deste trabalho prático foi criado um Tipo Abstrato e Dado, chamado Hash, contendo os seguintes campos principais:

- Vetor de Listas Encadeadas: cada posição do vetor representa uma chave do Hash, onde as listas encadeadas serão usadas para tratar as possíveis colisões;
- Contador de colisões: Deverá ser incrementado sempre que uma colisão diferente acontecer (uma nova célula for inserida na lista).



Como serão avaliados, inicialmente, um número já bastante alto de arquivos, foi considerado um tamanho grande para o vetor do Hash, 55001, sendo que este número foi escolhido por ser grande o suficiente para armazenar com um número controlado de colisões as palavras que serão lidas dos arquivos para serem indexadas e por ser primo.

Funções do TAD Hash

O TAD *Hash* criado apresenta as seguintes funções vinculadas, cujos cabeçalhos estão listados abaixo:

- ✓ void TrataArquivo(char*);
- ✓ int TamanhoArquivo(FILE*);
- ✓ char** LeArquivo(FILE*, char[][40], int*);
- ✓ void IndexaArquivo(FILE*, char[][40], Hash*, int);
- ✓ void FazListaVazia(TipoLista*);
- ✓ void InsereLista(int, int, TipoChave, Hash*, TipoIndice);
- ✓ void CriaHash(Hash*);
- ✓ void InsereHash(TipoChave, TipoIndice, int, Hash*);
- ✓ void ImprimeHash(FILE*, Hash*);
- ✓ void OrdenaHash(int*, int*, int);
- ✓ void PesquisaHash(Hash*, char*);

Sendo que estas podem ser divididas em 3 grupos diferentes: Funções relacionadas à manipulação e leitura de arquivos, funções relacionadas à manipulação de listas encadeadas e, por fim, funções relacionadas à criação e manipulação de uma tabela Hash.

Funções de Arquivos:

- void TrataArquivo(char*):

  Essa função trata as palavras lidas de um arquivo. Recebe por referência uma string contendo a palavra que se deseja tratar, retirando todos os caracteres não alfa-numéricos, com a exceção do hífen e do apóstrofe que serão mantidos, além de transformar todas as letras em minúsculas, para facilitar a comparação de palavras. Essa função não retorna nada.

- int TamanhoArquivo(FILE*):

  Essa função recebe um ponteiro para um arquivo e retorna o número de palavras contidas nesse arquivo.

- char** LeArquivo(FILE*, char[][40], int*):

  Essa função recebe um ponteiro para um arquivo, uma matriz de caracteres e uma variável do tipo inteiro. A função realiza a leitura das palavras do arquivo e as armazena em um vetor de strings (as quais têm o tamanho variável para

economia de espaço, através de alocação dinâmica). Essa função retorna um vetor de strings.

- void IndexaArquivo(FILE*, char[][40], Hash*, int):

  Essa função recebe como referência uma variável do tipo Hash, um arquivo, um vetor de strings e um inteiro. A função realiza a leitura palavra por palavra do arquivo recebido, trada as palavras lidas, compara com as palavras contidas no vetor de strings recebido e, caso, não esteja contida no vetor, essa palavras é então transformada em um índice da tabela Hash e inserida no Hash através da chamada de uma função que será descrita mais adiante. Essa função não retorna nada.

Funções de Listas :

- void FazListaVazia(TipoLista*):

  Essa função recebe por referência uma lista. A função cria a célula cabeça para a lista através da alocação dinâmica, faz com que o campo *primeiro* e *ultimo* da lista apontem para essa célula. Essa função não retorna nada.

- void InsereLista(int, int, TipoChave, Hash*, TipoIndice):

  Essa função recebe como parâmetro além de um hash como referência, e a posição do vetor do hash no qual se deseja inserir a nova célula, os dados necessários para preencher uma nova célula, realiza a inserção, após a última célula, de uma nova célula contendo as informações passadas como parâmetro na lista. Essa função não retorna nada.

Funções do Hash:

- void CriaHash(Hash*):

  Essa função basicamente recebe um hash como referência, define o campo *tamanho* do hash como sendo o tamanho do vetor do hash, aloca um vetor de listas com o número de posições correto e chama a Função *FazListaVazia* para criar as listas referentes a cada posição do vetor. Essa função não retorna nada.

- void InsereHash(TipoChave, TipoIndice, int, Hash*):

Essa função recebe um ponteiro para um hash, a chave que se deseja inserir no hash, o indice gerado para essa chave e o número do arquivo do qual a palavra foi retirada. Primeiro o programa verifica se a lista da posição do vetor indicada pelo indice gerado do hash está vazia, caso esteja ela inicia uma nova lista, com as informações recebidas como parâmetro. Se a lista já estiver preenchida, o programa verifica se alguma célula nessa lista possui a mesma chave que a que se deseja inserir. Em caso afirmativo o campo *frequencia* dessa célula é incrementado em 1 unidade. Em caso negativo é inserida uma nova célula no fim dessa lista e o campo *colisoes* do hash é incrementado em 1 unidade. Essa função não retorna nada.

- void ImprimeHash(FILE*, Hash*):

  Essa função recebe como parâmetro um arquivo, onde irá realizar a impressão do hash criado, e um hash. A função percorre posição por posição do vetor e dentro dessas posições célula por célula da lista até esta ser completamente percorrida . Para cada célula é criado dois vetores auxiliares contendo em quais arquivos a palavra representada pela célula aparece e qual o peso (frequência) dessa palavra desse arquivo. Depois disso, esses dois arquivos são enviados para serem ordenados e logo depois são impressos ordenadamente no arquivo. Essa função não retorna nada.

- void OrdenaHash(int*, int*, int):

  Essa função recebe por referência dois vetores e, através do método de ordenação por seleção (método escolhido por ser de fácil implementação e principalmente por ser estável) ordena os dois , baseado no vetor que representa a frequência das palavras dos arquivos. Essa função não retorna nada.

- void PesquisaHash(Hash*, char*):

  Essa função recebe por referência um hash e uma string como parâmetro. Primeiro a função transforma a string em um índice. Em seguida, dentro da lista indicada pela posição do vetor refrente ao índice gerado para a string recebida, a função procura pela célula quem contém a mesma chave. Caso encontre a função, assim como a *ImprimeHash* cria os dois vetores auxiliares, ordena os dados em relação à frequência e imprime a palavra, seguida dos arquivos em que ela aparece, com suas respectivas frequências. Caso não encontre a função imprime uma mensagem informando que a palavra não se encontra em nenhum arquivo. Essa função não retorna nada.

Funções do Programa Principal

Pra facilitar a leitura, decidiu-se que o programa principal seria dividido em 3 processos: *Indexador*, que realiza a leitura dos arquivos de busca e a criação da tabela hash baseada neles; *Processador*, que fica por conta de realizar a pesquisa de fato das palavras que se deseja verificar se estão presentes nos arquivos de busca; e o programa principal, que realiza a leitura dos arquivos contendo as palavras que serão desconsideradas para a criação do hash e as palavras que se deseja buscar. Assim, o programa principal é composto pelas seguintes funções:

- void main(int argc, char const*[]):

  A função recebe como parâmetro o arquivo de consulta e um contendo as palavras que devem ser desconsideradas na construção do hash. Em seguida ela chama a função *TamanhoArquivo* para descobrir quantas palavras tem no arquivo de stopwords e inclui as palavras contidas nesse arquivo em uma matriz. Na sequência é criada uma variável Hash, que é passada como referência para a função *Indexador*, que irá realizar a confecção do Hash e do arquivo de índice invertido. Após a criação do Hash é impresso na tela o número de colisões que ocorreram na criação do Hash. Por fim a função realiza a leitura do arquivo de consulta através da função *LeArquivo*, que retorna um vetor de strings contendo as palavras que serão pesquisadas. Na sequência a função *Processador* é chamada para fazer a pesquisa no Hash das palavras de consulta e a impressão da saída. O tempo de execução dessa função é medido e impresso na tela. Essa função não retorna nada.

- void Indexador(FILE*, Hash*, char[][40]):

  Essa função recebe como parâmetro um arquivo, um hash e uma matriz. A função realiza a leitura de todos os arquivos cujo nome sejam apenas algoritmos presentes na pasta *corpus* um a um. Para cada palavra lida de cada um desses arquivos a função chama a *IndexaArquivo* que irá adicionar a palavra caso seja necessário e possível no hash. Isso é feito para todos os arquivos da pasta. Por fim a função *ImprimeHash* é chamda para imprimir no arquivo *inverted_list.txt,* também presente na pasta *corpus.*

- void Processador(Hash*, char**, int):

  Essa função recebe como parâmetro o hash, um vetor de strings e um inteiro que representa o número de palavras que serão pesquisadas. Dentro de um loop então, a função chama para cada palavra de consulta a função *PesquisaHash.* Essa função não retorna nada.

Programa Principal

O programa recebe como parâmetro dois arquivos, sendo um deles o arquivo de consulta e outro contendo as palavras que devem ser desconsideradas na construção do hash (palavras que não possuem muito significado ou valor intrínseco como conjunções, pronomes, artigos...). Em seguida ela chama a função *TamanhoArquivo* para descobrir quantas palavras tem no arquivo de de palavras a serem desoconsideradas, criando na sequência um loop que irá ler todas as palavras do arquivo, uma a uma, e adicionar em uma matriz auxiliar que foi criada com exatamente o número de linhas necessárias para armazenar todas as palavras do arquivo. Esse número foi retirado da função *TamanhoArquivo*.

Na sequência uma variável do tipo Hash é criada e passada como referência para a função *Indexador*, que irá realizar a confecção do Hash e do arquivo de índice invertido. A criação do hash será feita desconsiderando todas as *stopwords* contidas e tratando as palavras para que fiquem em um mesmo padrão (sempre minúsculas e sem caracteres não alfanuméricos, com exceção do apóstrofe e do hífen que podem sim compor palavras). Esse processo de tratamento e verificação das palavras que irão entrar efetivamente no grafo são feitos internamente pelas funções criadas para o TAD Hash. A seguir, após a criação do Hash, o número de colisões que ocorreram na criação do Hash é impresso na tela para fins de consulta.

Encerrando o código a função realiza a leitura do arquivo de consulta através da função *LeArquivo*, que retorna um vetor de strings contendo as palavras que serão pesquisadas e inicia uma sequencia de comadnos que servem para medir o tempo de execução da função de busca. A função *Processador* é então chamada para fazer a pesquisa no Hash das palavras de consulta e a impressão da saída, relatando quando a busca tem sucesso em quais arquivos e com qual frequência a palavra procurada aparace em cada um deles, ou, em caso de insucesso na busca a função imprime uma mensagem informando que a palavra não está presente em nenhum arquivo. Após a conclusão da busca o tempo de execução dessa função é medido e impresso na tela.

Análise de Complexidade

As funções de complexidade, assim como a ordem de complexidade para cada função implementada nos projetos e no TAD serão analisadas em função de um número *n* que simboliza o tamanho da chave e *m* que simboliza o tamanho do vetor do Hash, além de *l* para representar o tamanho da lista. E para casos especiais de funções em função de *a*, onde *a* representa o tamanho do arquivo de stopword (número de palavras contidas), *b* representa o tamanho dos arquivos que serão lidos (arquivos de consulta) e c representa o número de palavras que serão pesquisadas, enquanto *k* representa o número de arquivos que serão consultados:

TAD Hash - Funções de Arquivo:

- **Função TrataArquivo:**

  A função apresenta, dentro outras coisas (dentro delas alguns outros loops O(n)), uma cadeia de loops ambos O(n), cujo resultado é uma função $O(n^2)$. Assim, esta função é $O(n^2)$.

- **Função TamanhoArquivo:**

  Essa função recebe um arquivo e conta o número de palavras que tem nele, através de um loop executado *a* vezes. Assim, a função é O(a).

- **Função LeArquivo:**

  Essa função é composta por várias O(1) de comparação e atribuição e um loop que será executado *a* vezes, onde a é o tamanho (ou número de palavras do arquivo stopword). Assim, a função é O(a).

- **Função IndexaArquivo:**

  Essa função executa *b* vezes alguns loops que serão executados *n* vezes e *a* vezes, respectivamente, cada um contendo funções O(1). Assim a função é O(n(a+b)).

TAD Hash - Funções de Lista :

- **Função FazListaVazia:**

  Essa função realiza apenas operações O(1) de atribuição. Assim, é O(1).

- **Função InsereLista:**

  Assim como a anterior essa função realiza apenas operações O(1) de atribuição. Assim, é O(1).

TAD Hash - Funções do Hash :

- **Função CriaHash:**

  Além de comandos O(1), essa função apresenta um loop que é executado *m* vezes. Assim, a função é O(m).

- **Função InsereHash:**

  Como essa função representa basicamente uma função de inserção de um elemento no Hash, sua complexidade pode ser descrita como O(1+n/m), quando consideramos que o tempo esperado para percorrer cada lista seja n/m.

- **Função ImprimeHash:**

  Essa função possui um loop executado *m* vezes, dentro do qual teremos um outro loop executado *l* vezes, dentro do qual teremos um loops executados *m* vezes. Dessa forma a função apresenta ordem de complexidade O(lm²).

- **Função OrdenaHash:**

  Essa função se trata de um método de ordenação por inserção. Assim sua ordem é O(n²), sendo que os motivos da escolha do método já foi esclarecido na descrição da função.

- **Função PesquisaHash:**

  Como essa função representa basicamente uma função de pesquisa de um elemento no Hash, sua complexidade pode ser descrita como O(1+n/m), quando consideramos que o tempo esperado para percorrer cada lista seja n/m. Contudo além disso a função apresenta uma série de loops que serão executados *m* vezes. Assim, a função apresenta, no fim, ordem de complexidade O(mn).

Programa Principal - Funções de Arquivo:

- **Função Indexador**:

  Além de funções e comandos O(1). Essa função executa um loop que será realizado *k* vezes, dentro do qual será chamada a função *IndexaArquivo* que é O(n(a+b)). No fim ainda temos a chamada da função *ImprimeHash* que é O(lm²). Dessa forma teremos uma ordem de complexidade O(k(na+nb+lm²)).

- **Função Processador**:

  Nessa função temos um loop que é executado *c* vezes, dentro do qual temos a chamada da função PesquisaHash que é O(mn). Assim, a ordem de complexidade dessa função é O(mnc).

- **Função main**:

  Nessa função além de comandos simples O(1), temos a chamada das funções *Indexador* e *Processador,* O(kna+knb+klm²) e O(mnc), respectivamente, e das funções TamanhoArquivo e LeArquivo, que são O(a). Assim, essa função é no fim O(kna + knb + klm² + mnc + a).

Detalhes Técnicos e Tomadas de decisão

Foram criados 4 arquivos principais: Hash .c e Hash .h que representam o TAD criado e um outro arquivo main.c contendo o código do programa principal. Durante a leitura, caracteres não alfa-numéricos foram desconsiderados nas palavras, com exceção do hífen e do apóstrofe. O tamanho do Hash escolhido foi para reduzir o número de colisões e não é recomendado alterar, pois o programa não foi testado com outros valores, devido ao tempo corrido no fim do semestre. Para facilitação do código optou-se por não utilizar a função *opendir* e realizar a leitura dos arquivos normalmente através de um loop. Por fim, O tempo de pesquisa está sendo contado para a realização da busca de todas as palavras presentes no arquivo fornecido. O programa não retorna um tempo pra cada palavra buscada.

O compilador utilizado foi o *GNU GCC Compiler* vinculado ao editor de texto *Atom.* O projeto foi desenvolvido e testado em Linux (Distribuição Ubuntu 18.04 LTS).