Igor Pontes Tresolavy – NUSP 12553646

EP4 – Explicações Gerais

1. Organização do Programa

A tarefa a ser cumprida pelo programa pode ser dividida em duas partes: captura e armazenamento das palavras de um texto, e organização das palavras em ordem alfabética.

Tanto os protótipos das funções, quanto o corpo da função *main()* foram divididos entre essas duas partes, através de comentários no cabeçalho do programa e ao longo da função *main()*.

Além disso, é importante mencionar o significado das *macros* e das estruturas das listas ligadas utilizadas para armazenar as palavras e suas ocorrências:

- *Macro HASH_TABLE_MAX_SIZE*;
    - Define o tamanho máximo possível da tabela Hash do programa, que não necessariamente sempre será atingido, pois, como será explicado no capítulo 3, a tabela Hash é dinamicamente alocada ao longo do programa.
- *Macro SEED*
    - Utilizada na função hash, explicada no capítulo 3.
- Célula da lista ligada de palavras;
    - Ponteiro para vetor de caracteres (a palavra, efetivamente);
    - Ponteiro para primeira célula da lista ligada de ocorrências;
    - Ponteiro para a próxima célula da lista ligada de palavras.
- Célula da lista ligada de ocorrências.
    - Linha;
    - Número de aparições na linha;
    - Ponteiro para a próxima célula da lista ligada de ocorrências.

2. Funcionamento do Programa

a. Input

A entrada do programa é o caminho até o arquivo que será lido. Se o arquivo estiver no mesmo diretório que o executável, só o nome do arquivo como input já basta.

b. Parte 1

Tanto no desenvolvimento do programa, quanto no seu entendimento, a utilização de pseudocódigos é de grande auxílio. Abaixo encontra-se o pseudocódigo da primeira parte do programa (essa mesma ferramenta será usada para a Parte 2):

- Para cada caractere do arquivo.
    - Se o texto não tiver chegado ao fim;
        - Se o caractere for uma letra ASCII;
            - Contabiliza-o na palavra em análise.
        - Se o caractere NÃO for uma letra ASCII;
            - Na tabela Hash, adiciona a palavra e a linha na qual ela ocorre, ou contabiliza mais uma ocorrência na mesma linha.
        - Se houver mudança de linha no texto.
            - Contabiliza-a.

Apesar da implementação do programa possuir mais nuances e, portanto, maior complexidade que o pseudocódigo, o entendimento do programa é consideravelmente facilitado pela ferramenta.

c. Parte 2

Para a Parte 2, por sua vez:

- Elimina espaços vazios na tabela Hash;
- Resolve colisões presentes na tabela Hash;
- Organiza as palavras da tabela Hash em ordem alfabética utilizando algum algoritmo de ordenação*;
- Imprime tabela Hash e ocorrências de cada palavra em ordem de aparição.

* Algoritmo utilizado: Heapsort

O algoritmo de ordenação Heapsort foi escolhido para ordenar as palavras armazenadas na tabela em ordem alfabética por conta de sua rapidez garantida. Ou seja, pode-se sempre ter certeza de que a ordenação terá complexidade O(n logn), o que é extremamente vantajoso considerando que a tabela Hash terá, em média, 10000 ~ 20000 ou mais palavras distintas.

No entanto, uma desvantagem do algoritmo é a falta de estabilidade na ordenação, isso é, o algoritmo possui certa inconsistência na ordenação de palavras distintas que possuem valores iguais entre si. Por exemplo, as palavras “Amor” e “amor” são distintas, mas possuem valores iguais na ordenação por Heapsort, assim como as palavras “convento” e “Convento”. No entanto, o algoritmo pode ordenar as palavras da seguinte maneira:

Amor, amor, convento, Convento

Percebe-se que a versão da palavra "amor" com a primeira letra maiúscula pode preceder a versão de primeira letra minúscula. No entanto, o mesmo pode não acontecer para as palavras “convento” e “Convento”, como demonstrado acima.

Um outro algoritmo de ordenação que possui complexidade de caso médio consideravelmente baixa e não possui tal instabilidade e é o Quicksort. No entanto, como o pior caso do Quicksort tem complexidade $O(n^2)$ e a estabilidade na ordenação de palavras de letra minúscula e maiúscula não é crucial, escolheu-se o algoritmo de Heapsort por conta de sua baixa complexidade de tempo constante e garantida para qualquer tamanho da tabela Hash.

3. Explicação sobre a Função Hash

A função Hash escolhida foi a MurmurHash, uma função Hash não-criptográfica para tabelas Hash de propósito geral. O nome *Murmur* advém das operações lógicas de multiplicação (MU) e rotação de bits (R).

A MurmurHash foi escolhida pois é capaz de retornar valores Hash para strings, além de ser rápida e eficiente, isso é, gera poucas colisões. Existem 3 versões da função e a MurmurHash3 foi a escolhida, por ser a mais atual e eficiente dentre as três.

A função aceita 3 argumentos de entrada:

1. *Key*: No caso, a palavra do texto que será inserida na tabela;
2. *Len*: quantidade de bytes armazenados no vetor *Key* que, no caso, é igual à quantidade de caracteres na palavra;
3. *Seed*: valor escolhido com o intuito de randomizar a função Hash. Pode assumir qualquer valor Inteiro positivo ou igual a 0, escolheu-se o valor 0xbc9f1d34 para a *macro SEED*, pois é a *Seed* utilizada pelo desenvolvedor do banco de dados open-source LevelDB, Jeff Dean.

Além disso, algumas constantes são utilizadas durante o algoritmo. Essas constantes foram determinadas pelo desenvolvedor da função MurmurHash, Austin Appleby, com o intuito de diminuir ao máximo a probabilidade de colisões ocorrerem:

- c1 = 0xcc9e2d51
  - Usada para multiplicar grupos de bytes presentes em *Key*.
- c2 = 0x1b873593
  - Usada para o mesmo fim de c1.
- r1 = 15
  - Usada para rotacionar grupos de bytes presentes em *Key*.
- r2 = 13
  - Usada para o mesmo fim de r1.
- m = 5
  - Usado em iterações no cálculo do valor Hash de *Key*
- n = 0xe6546b6
  - Usado para o mesmo fim de m.

Abaixo encontra-se o pseudocódigo da função MurmurHash utilizada no programa:

- Para cada grupo de 4 bytes presentes em *Key* (4 caracteres de uma palavra);
  - Multiplica grupo de 4 bytes por c1;
  - Rotaciona r1 bits de grupo do resultado para a esquerda;
  - Multiplica resultado por c2;
  - Atribui à *Seed* valor da operação lógica OU EXCLUSIVO de *Seed* com valor do resultado;

    - Rotaciona r2 bits de *Seed* para a esquerda;
    - Atribui à *Seed* valor da multiplicação aritmética de *Seed* por m, somado à n.
- Para qualquer byte restante, se houver;
    - “Arrasta” (logical shift) bytes restantes 8 bits para a esquerda;
    - Atribui à resultado valor da operação lógica OU com bytes restantes;
- Atribui à *Seed* o valor da operação lógica OU EXCLUSIVO de *Seed* com o resultado anterior;
- Atribui à *Seed* o valor da operação lógica OU EXCLUSIVO de *Seed* com o valor de *Len;*
- Atribui à *Seed* o valor da operação lógica OU EXCLUSIVO de *Seed* com o valor de *Seed* “arrastado” (logical shift) 16 bits para a direita;
- Multiplica *Seed* por 0x85ebca6b;
- Atribui à *Seed* o valor da operação lógica OU EXCLUSIVO de *Seed* com o valor de *Seed* “arrastado” (logical shift) 13 bits para a direita;
- Multiplica *Seed* por 0xc2b2ae35;
- Atribui à *Seed* o valor da operação lógica OU EXCLUSIVO de *Seed* com o valor de *Seed* “arrastado” (logical shift) 16 bits para a direita.
- Retorna *Seed*;

A função Hash é o resultado da função murmurHash após a operação módulo com a *macro MAX_HASH_TABLE_SIZE* (= 100000), o tamanho máximo possível para a tabela Hash. Foi escolhido um tamanho grande o suficiente para diminuir o número de colisões, mas pequeno o bastante para não utilizar espaços exorbitantes de memória.

Além disso, como mencionado anteriormente, como forma de otimização do uso de memória, o tamanho da tabela Hash é dinâmico, isto é, não é sempre igual à *MAX_HASH_TABLE_SIZE*, e sim sempre igual ao maior resultado de uma função Hash de alguma das palavras distintas do texto. Portanto, se o maior resultado da função Hash dentre as palavras do texto for igual à 1000, o tamanho da tabela Hash ao fim da Parte 1 do programa será igual à 1000.

Por exemplo, para um trecho do livro “Guerra e Paz”, de Leo Tolstoy, de 19021 palavras distintas, em inglês, houveram 1645 colisões, o que equivale à aproximadamente somente 8,64% das palavras.

Referências

- Wikipédia – MurmurHash;
- Explicação dos argumentos da função MurmurHash: Stackoverflow;
- Explicação dos parâmetros da função MurmurHash: Stackoverflow;
- Fonte da Seed escolhida: Stackoverflow;