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()*.

Ademais, 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;*
  - Valor utilizado na Função Hash para aumentar sua aleatoriedade e evitar colisões.
- 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.

Apesar dessa característica, pode-se ter certeza que, para textos iguais, a saída do programa será sempre igual, independente do ambiente no qual o programa for compilado e executado.

Um outro algoritmo de ordenação que possui complexidade de caso médio consideravelmente baixa e não possui tal instabilidade 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 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. No caso, o valor escolhido foi o número primo 0x1b873593.

Seu funcionamento está descrito a seguir:

- Multiplica valor de cada caractere da palavra pelo número de sua posição e soma os resultados;
- Soma o valor de todos os caracteres;
- Multiplica os dois resultados anteriores pelo valor de *SEED*;
- Retorna o resultado modulado pela *macro HASH_TABLE_SIZE*, o que efetivamente impede que o resultado seja maior que o tamanho máximo permitido definido pelo valor da *macro*.

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.