Instituto de Matemática e Estatística

Hélio Hideki Assakura Moreira

EP3 - Índice Remissivo

São Paulo

2014

Localização das palavras desejada:

Para a criação das listas e da árvore, para cada palavra do arquivo de entrada das palavras, temos que buscar no texto o numero de ocorrências da palavra e as linhas em que ela ocorre. Para isso, recebemos cada linha do texto e verificamos se essa palavra ocorre usando a função *strstr* (da biblioteca string.h). Porém, foi usada a função *strtok* (string.h) para excluir possíveis “erros”, como a palavra estar seguida de um sinal de pontuação, e também verificamos se não foi achado apenas uma parte da palavra, como *contar* em *descontar*.

Criação do elemento:

Se localizada a palavra na linha, guardamos essa ocorrência em uma lista ligada com a função *InsereNoFimLinha*, de complexidade O(n), n é o numero de elementos na lista.

A estrutura da lista é:

```
struct cel{
        int info;
        struct cel * prox;
};
typedef struct cel celula;
typedef celula * linha;
```

Sendo *info* a linha da ocorrência da palavra.

Registradas as ocorrências e o número de vezes em que a palavra apareceu, criamos o elemento.

Criação da Tabela de Símbolos:

Logo que foi criado o elemento, ele é inserido na Tabela de Símbolos, que é uma lista com os elementos criados.

Lista Desordenada

Para a criação da lista, usamos a função *InsereNoFimTabela,* que percorre a lista até achar o ultimo elemento, colocando o novo elemento na ultima posição. Isso tem complexidade O(n), sendo n o tamanho da lista.

Porém, ao percorrer o texto *p* vezes, sendo p o numero de palavras, e *m* o número de linhas no texto, nos dá um algoritmo O(m^p). Mesmo o algoritmo tendo a complexidade ruim, pelos resultados empíricos, não há uma diferença tão grande que inviabilize a implementação do código.

Lista Ordenada

Segue o mesmo princípio da lista desordenada. Porém, os elementos são criados a partir de um vetor ordenado lexicograficamente. Logo, ao inserí-los, a Tabela já será impressa ordenadamente.

Arvore Binária

Para a implementação da árvore binária, foi usada a função *InsereArvore*, passada pelo próprio professor na aula. Essa função percorre a arvore ***a-1*** vezes, sendo ***a*** a altura da árvore. Essa função já contém o algoritmo de busca, e como a inserção não é ordenada, a árvore tem formato mais próximo de $\ln n$, sendo n o numero de elementos a serem inseridos. Como a forma de obtenção do elemento é semelhante a Lista, temos complexidade O(m^p).

Exemplos Empíricos:

Primeiramente, foi criado uma espécie de “menu” para facilitar a manipulação do programa:

```
|----------------------------------------|

 Escolha o tipo de Tabela a ser utilizada:
0 - Sair
1 - Lista Desordenada
2 - Lista Ordenada (lexicografica)
3 - Arvore de Busca Binaria (Saida Pre-Ordem)
4 - Arvore de Busca Binaria (Saida In-Ordem)
5 - Arvore de Busca Binaria (Saida Pos-Ordem)

|----------------------------------------|
```

Sendo:
m = numero de linhas no texto
p = numero de palavras

Para m = 26175, p = 100:

Desordenada:

```
real    0m10.126s
user    0m0.590s
sys     0m0.016s
```

Ordenada:

```
real    0m7.356s
user    0m0.587s
sys     0m0.016s
```

Arvore:

```
real    0m10.596s
user    0m0.583s
sys     0m0.016s
```

Para m = 104703, p = 100:

Desordenada:

```
real    0m14.854s
user    0m2.877s
sys     0m0.081s
```

Ordenada:

```
real    0m11.844s
user    0m2.701s
sys     0m0.068s
```

Arvore:

```
real    0m14.045s
user    0m2.702s
sys     0m0.067s
```

Para m = 104703, p = 20:

Desordenada:

```
real    0m7.952s
user    0m0.450s
sys     0m0.014s
```

Ordenada:

```
real    0m6.749s
user    0m0.440s
sys     0m0.014s
```

Arvore:

```
real    0m12.192s
user    0m0.452s
sys     0m0.015s
```

Para m = 26175, p = 20:

Desordenada:

```
real    0m7.644s
user    0m0.124s
sys     0m0.004s
```

Ordenada:

```
real    0m7.644s
user    0m0.124s
sys     0m0.004s
```

Arvore:

```
real    0m8.044s
user    0m0.123s
sys     0m0.005s
```