Projeto Integrador I

Busca de palavras e expressões em livros e artigos com ranqueamento por aparição e relevância

Alunos:

- Aenã Eloi Martinelli – 2012130016
- Davi Pereira Speck Alves – 2012130053
- Eduardo de Oliveira Alvim - 2012130076

Ideia do software: Montar um código capaz de implementar em um software a função de busca por palavras e expressões, não apenas mediante ao comum já encontrado na função nativa de navegadores **(CTRL+F)** ou de editores de texto **(CTRL+L – no Word)**. Além disso, apesar de ser criado para uma certa realidade, o software não será restritivo à utilização em livros e artigos, sua funcionalidade pode se encaixar no padrão em que o usuário deseja, mas a indicação mais eficaz será em torno dos mesmos. Então, sabe-se que com a busca dada como finalidade principal, torna-se possível tratar do diferencial em relação aos outros softwares já existentes no mercado, a capacidade em ranquear as páginas em que a entrada **(palavra ou expressão)** foi encontrada com maior frequência e relevância e a associação das mesmas com os períodos **(frase iniciada em letra maiúscula e terminada em ponto)** em que estão inseridas.

FUNCIONALIDADES:

1. Abrir em conjunto com arquivos relacionados à texto (TXT, DOC, PDF, PPT).
2. Definir um atalho para "busca bruta" - funcionalidade nativa de navegadores e editores de texto (CTRL+F / CTRL+L).
3. Informar palavra que deseja buscar.
4. A "busca bruta" apresentará a quantidade de palavras encontradas e o número da palavra em que você está no arquivo **(palavra em que esta / total de palavras)** com possibilidade de ir para a próxima e voltar para a anterior.
5. Definir um atalho para busca ranqueada – funcionalidade - diferencial.
6. Informar palavra que deseja buscar.
7. Apresentar um botão "Gerar Arquivo" em que terá a saída da busca ranqueada (no formato PDF).
8. O arquivo PDF gerado apresentará o ranking de páginas com o quantitativo em que a palavra foi encontrada e em seguida mostrará todas as palavras em seus respectivos períodos.

OBS: Dessa forma o software além de funcionar como uma ferramenta de direcionamento e localização em textos, possibilitará uma visualização suscinta do assunto em torno da palavra buscada, trazendo a possibilidade de auxílio na confecção de resumos e interpretação de textos. Demais funcionalidades podem surgir caso sobre tempo para produção. (Download estará disponível em site do software)

Algoritmos referências

Boyer-Moore – Versão 1 (1977):

Esse algoritmo tenta fazer menos comparações usando uma característica do padrão a ser procurado variável “a”. Quando se compara a[1..m] com b[i..k] (k=i+m-1 para i=1,..., n-m+1), isto é, quando se compara a com um segmento qualquer dentro de b, a próxima comparação não precisa ser com b[i+1..k+1]. Pode ser com b[i+d..k+d] onde d é calculado de forma que b[k+1] coincida com a última ocorrência de b[k+1] em a. Assim, podemos deslocar a comparação com o próximo segmento em mais de um elemento. Não importa o resultado da comparação anterior.

Exemplo:

Procurar `abcd` em `abacacbabcdcdabd`

Veja como podemos fazer a busca avançando no modo proposto:

a	b	a	c	a	c	b	a	b	c	d	c	d	a	b	d
a	b	c	d												
				a	b	c	d								
							a	b	c	d					
									a	b	c	d			

Outro exemplo – Procurar `aba`:

a	b	a	c	a	c	b	a	b	c	d	c	d	a	b	d
a	b	a													
				a	b	a									
					a	b	a								
							a	b	a						
											a	b	a		
													a	b	a

O problema então consiste em saber qual a última ocorrência de `b[k+1]` em `a`.
Se soubermos todos os valores possíveis de `b[k+1]`, podemos calcular este valor para cada elemento de `a`.
Aqui está a particularidade do algoritmo: é necessário conhecer o alfabeto.
Como estamos lidando com caracteres, o alfabeto são todos os caracteres de 0 a 255.
Podemos então previamente calcular qual a última ocorrência de cada um dos caracteres de `a`.

Boyer-Moore – Horspool (1980):

Pré-processamento: computar tabela de saltos. Tabelas pré computadas servem para a reutilização de busca em texto diante palavras chaves bastante requisitadas e pesquisadas... Essas tabelas serão usadas para saltar em palavras especificas para que haja maior rapidez na pesquisa, não sendo preciso então, o processo de busca em algoritmo por algoritmo.

Um exemplo para esse tipo de busca são os vírus, geralmente são usadas tabelas pré-computadas com assinaturas de vírus e quando é feita a varredura no sistema, utiliza-se das mesmas, mas tem um ponto controverso, quando existe uma tabela (padrão) muito grande, ela ocupa muito espaço e geralmente as pessoas não buscam padrões muito grandes em um texto ou em outros cenários.

Boyer-Moore – Versão 2 (?):

A versão 2 do algoritmo é intuitiva, mas tem uma implementação mais engenhosa. Não é necessário conhecer-se o alfabeto de a. Também só depende de a. **Neste algoritmo é necessário que a comparação de a com b, seja feita da direita para a esquerda:**

Para i = m, m + 1, ..., n
Comparar a[m] **com** b[i]; a[m-1] **com** b[i-1]; ...; a[1] **com** b[i-m+1]

A ideia básica é a seguinte: Suponha que numa das comparações já descobrimos que a[h..m] é igual a b[k-m+h..k], ou seja, descobrimos que existe um trecho (parcial ou total) no meio de b que é igual a um trecho corresponde dente de a. Só haverá casamento se a tiver um trecho igual em a[1..m-1]. Se não houver tal trecho em a, podemos deslocar de m elementos.

Exemplo:

Procurar `abcd` em `abacacbabcdcdabd`

Veja como podemos fazer a busca avançando no modo proposto:

a	b	a	c	a	c	b	a	b	c	d	c	d	a	b	d
a	b	c	d												
				a	b	c	d								
							a	b	c	d					
									a	b	c	d			

Outro exemplo – Procurar `aba`:

a	b	a	c	a	c	b	a	b	c	d	c	d	a	b	d
a	b	a													
				a	b	a									
					a	b	a								
							a	b	a						
											a	b	a		
													a	b	a

O problema então consiste em saber qual a última ocorrência de `b[k+1]` em `a`.
Se soubermos todos os valores possíveis de `b[k+1]`, podemos calcular este valor para cada elemento de `a`.
Aqui está a particularidade do algoritmo: é necessário conhecer o alfabeto.
Como estamos lidando com caracteres, o alfabeto são todos os caracteres de 0 a 255.
Podemos então previamente calcular qual a última ocorrência de cada um dos caracteres de `a`.

Busca em Texto – Força bruta

Um algoritmo natural e fácil de entender, é bem simples considerando o que muitos algoritmos de grande eficiência se baseiam. Ele faz uma busca em texto na tentativa de localizar o padrão, ao achar a primeira ocorrência ele retornara a mesma. O algoritmo faz com que busque no texto inteiro todas as ocorrências padronizadas (palavra, frase, letra). Não é um algoritmo recomendado, pois não é eficiente, já que não resolve o problema da melhor forma, para textos pequenos ele pode até resolver

problemas, mas para textos grandes, tem suas problemáticas. Ele é literalmente um algoritmo básico de busca em texto, fácil de entender e fácil de fazer, mas não eficiente para utilizar.

Referências Bibliográficas

- https://www.ime.usp.br/~mms/mac1222s2013/18%20%20Algoritmos%20de%20Busca%20de%20Palavras%20em%20Texto.pdf
- https://www.ime.usp.br/~pf/algoritmos/aulas/strma.html
- https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/kmp.html
- https://www.youtube.com/watch?v=hTFFo49hcvU