**1-INTRODUÇÃO**

No cotidiano atual a utilização da internet é indispensável, e a forma mais eficiente de comunicação na mesma é por meio de mensagens, seja utilizando bate-papos ou aplicativos especializados para celular. Ao pesquisarmos ao na internet sempre nos deparamos com o auto-complete seja nos buscadores como Google ou em sites que nos oferecem pesquisar pelo mesmo, também podemos utilizar de recurso parecido por meio de editores textuais como o sublime que quando estamos digitando algoritmos nos oferece a provável continuação da palavra, bastando apertar a tecla TAB do teclado para completar a mesma.

Pensando nisso foi o feito o trabalho prático da disciplina de algoritmos e estrutura de dados três.

**2-PROBLEMA PROPOSTO**

**3-SOLUÇÕES IMPLENTADAS**

**3.1-FORÇA BRUTA**

Um algoritmo força bruta basicamente compara todos os caracteres, a fim de achar a pesquisa que nos interessa. Contudo estamos interessados apenas nos prefixos, ou seja, a parte inicial da palavra, a busca pelos caracteres deve ser exata, deve-se pegar a palavra de entrada digitada, e achar outras palavras que batem exatamente com aquela até o ponto em que a mesma foi digitada.

Por exemplo, ao digitarmos o prefixo “com”, podemos ter como resultado, como, comics, computador, mas ao aumentarmos em uma letra tal prefixo, deixando de com e passando a ser “comp”, apenas um dos resultados acima nos será satisfatório, que será computador. Por tal motivo comparamos as posições casa a casa.

**3.1.2-SOLUÇÃO PROPOSTA E IMPLEMENTAÇÃO**

Após a leitura dos parâmetros de entrada do programa e leitura da base de dados fornecida, verifica-se o tamanho da palavra digitada como parâmetro para que não peguemos palavras que sejam menores que a mesma, ou seja, para entrar como possível retorno no auto-complete, a palavra deve ter pelo os primeiros caracteres iguais o da palavra digitada, e ser de tamanho maior ou igual aquilo que se deseja pesquisar.

Depois de feito isso, verificamos se a palavra digitada pelo usuário é maiúscula, caso seja, a mesma é transformada em minúscula, utilizando a função min, que aumenta em 32 os valores, com base na tabela ASCII, deixando os mesmos com a letra minúscula.

|  |  |  |  |
| --- | --- | --- | --- |
| Valor em decimal | Letra maiúscula | Valor em decimal | Letra minúscula |
| 65 | A | 97 | a |
| 66 | B | 98 | b |
| 67 | C | 99 | c |
| 68 | D | 100 | d |
| 69 | E | 101 | e |

Tabela 1: Mostra basicamente o funcionamento da tabela ASCII

Para que isso ocorra a função seleciona, compara a palavra fornecida pelo usuário com as palavras que temos em nosso dicionário, como fora explicado mais acima, enquanto os caracteres da palavra a ser pesquisada bater com os caracteres de uma palavra qualquer do dicionário, temos uma contador que se modifica, caso ao final da comparação o contador tem valor igual ao tamanho da palavra pesquisada, então tal palavra, é uma candidata para o auto-complete, dessa forma, está palavra é armazena em um outro vetor, que contém apenas as palavras selecionadas, tal processo se repete até que tenham sido comparadas todas as palavras.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Palavra Pesquisada: | B | A | T |  |  |  |
| Palavra do Dicionário: | B | A | T | A | T | A |
| Resultado: | Caractere igual | Caractere igual | Caractere igual |  |  |  |
| Palavra do Dicionário: | B | A | T | M | A | N |
| Resultado: | Caractere igual | Caractere igual | Caractere igual |  |  |  |
| Palavra do Dicionário: | B | A | N | A | N | A |
| Resultado: | Caractere igual | Caractere igual | Caractere  diferente |  |  |  |
| Palavra do Dicionário: | M | Ú | S | I | C | A |
| Resultado: | Caractere  diferente | Caractere  diferente | Caractere  diferente |  |  |  |

Tabela 2: exemplifica basicamente como funciona a busca no algoritmo força bruta.

Olhando a tabela 2 podemos ver como o força bruta realiza suas comparações, caractere por caractere, caso um dos caracteres seja diferente, desconsidera-se tal palavra.

Depois que já temos as palavras pré-selecionadas para serem mostradas como resultado temos que ordená-las pelo seu peso. Entretanto antes de ordenar recebemos o peso das palavras em um vetor, e este é ordenado utilizando o famoso algoritmo de ordenação bubble sort. Basicamente o algoritmo de ordenação percorre o vetor comparando os elementos adjacentes, trocando as posições dos elementos que estiverem fora de ordem, repetindo o processo até que reste apenas um elemento.

Após feito isso temos a última função, responsável por basicamente imprimir na tela os resultados encontrados, para isso temos uma última comparação, onde comparando-se o peso das palavras selecionadas com o vetor ordenado, caso o valor seja igual, a palavra já ordenada é mostrada.

**3.2-OUTRO ALGORITMO QUE SERÁ IMPLEMENTADO**

**4-ANÁLISE DE COMPLEXIDADE**

Nesta seção analisaremos e compararemos os algoritmos implementados, também iremos mostra a diferença entre todos os uns dos algoritmos em termos de complexidade.

**4.1-ALGORITMO DE FORÇA BRUTA**

Para começar o algoritmo de força bruta utiliza-se de dois loops aninhados responsáveis pela leitura do arquivo, um que roda enquanto não atingir o número total de palavras contidas no arquivo, tendo outro loop que basicamente repete tal processo, sendo assim a complexidade da leitura é O(número\_de\_palavras\*número\_de\_palavras), sendo sua função O(n²), sendo n o número total de palavras lidas.

Depois de feita a leitura utiliza a função strlen, que é uma função que retorna o tamanho de um string, tal função está presente na biblioteca string.h, sendo complexidade para o retorno do tamanho O(n), sendo n o tamanho da palavra.

Após o retorno do tamanho correto da palavra, a função min verifica se a palavra digitada pelo usuário está em letra maiúscula ou minúscula, caso esteja minúscula ele simplesmente mantém a palavra, caso seja maiúscula, temos um for que percorre a palavra transformando as letras maiúsculas em minúsculas, a complexidade desta função é a mesma que da função strlen, sendo a mesma O(n), sendo n o tamanho da palavra.

A função seleciona palavras, como o próprio nome já diz, seleciona possíveis palavras que serão apresentadas no auto-complete, para isso temos dois loops aninhados que vão até o tamanho da palavra, seguido de uma condicional que verifica se a palavra do dicionário tem prefixo igual a palavra digitada pelo usuário, caso seja igual temos uma outra condicional que verifica se o valor do contador é igual ao tamanho da palavra pesquisada, que foi retornado pela função strlen, caso seja, temos um loop que irá copiar as palavras selecionadas para uma lista, sendo que este loop percorre até 100, sendo este definido como o tamanho da maior palavra. Desta forma a complexidade final é O(4\*numero\_total\_de\_palavras\*numero\_total\_de\_palavras\*100).

Depois de selecionadas as palavras que serão utilizadas, recebe-se através de um loop que vai até o número de palavras, em um vetor o peso das palavras, tal vetor é ordenado usando o algoritmo bubble sort que tem complexidade no pior caso de O(n²), sendo n o número de palavras, logo a complexidade final da função vetor maiores mais o da função bubble sort é O(numero\_total\_de\_palavras)+O(n²).

Após retornado o vetor já ordenado, temos a função seleção, que imprime as palavras já ordenadas para serem retornadas ao usuário, nesta função possui-se dois loops aninhados, sendo que os dois vão de 0 até o tamanho total de palavras, e uma estrutura condicional que verifica se o vetor ordenado tem o peso igual ao da palavra selecionada, caso tenha imprime-se tal palavra, logo a complexidade da função é O(2\*numero\_de\_palavras\*numero\_de\_palavras).

Com isso podemos concluir que a complexidade do algoritmo é O(max(numero\_de\_palavras\*numero\_de\_palavras)+O(n)+O(n)+O(4\*numero\_total\_de\_palavras\*numero\_total\_de\_palavras\*100)+O(numero\_total\_de\_palavras)+O(n²)+O(2\*numero\_de\_palavras\*numero\_de\_palavras), sendo sua complexidade final O(400\*numero\_de\_palavras\*numero\_de\_palavras) ou O(400\*n²).

**5-ANÁLISE DE RESULTADOS**

**6-TESTES**

**7-CONCLUSÃO**

**8-REFERẼNCIAS BIBLIOGRÁFICAS**