Universidade Federal do Piauí - Campus SHNB
Sistemas de Informação
Algoritmos e Programação II
Adriano de Oliveira Leal Rocha
Hélio Rocha Vieira de Couto Junior
Myllena Caetano de Oliveira
TRABALHO PRÁTICO

1. Introdução.

O problema proposto em sala de aula trata-se da elaboração de um programa que seja capaz de gerar um sumarizador de notícias. Este deve ser composto pelas palavras que possuem maior número de repetições e que são consideradas as palavras chaves para compreender o foco principal de um texto.

O programa deve ser capaz de percorrer um documento escolhido pelo usuário e gerar a lista de palavras, sem levar em consideração preposições, artigos, conectivos, pronomes ou símbolos de pontuação. A quantidade de palavras presentes na lista também deve ser escolhida pelo usuário.

2. Solucionando o problema.

Após analisar o problema proposto chegamos a conclusão de que a melhor alternativa seria percorrer todo o arquivo, armazenando as palavras encontradas, excluindo as stopwords, e logo depois verificar a quantidade de vezes que cada palavra irá se repetir.

Usando o princípio da abordagem Top-Down, separamos o programa em diversas funções. Foram utilizadas também duas bibliotecas criadas por um dos próprios membros do grupo.

Inicialmente foram criadas duas estruturas, **Palavra**, que serve para guardar uma string e a quantidade de vezes que ela se repete no texto, e **WordList**, que serve para armazenar um vetor de **Palavra** e o tamanho utilizado desse vetor**.** As estruturas estão relacionadas abaixo, em pseudo-código:

estrutura Palavra:
    valor: **caractere[**50**]**
    repeticoes: **inteiro**
estrutura WordList:
    palavras: **Palavra[**10000**]**
    tamanho: **inteiro**

Além da função principal, foram criadas três funções básicas para o funcionamento do programa: **countWords**, **sortWords** e **showImportantWords**. Além destas foram criadas outras duas funções: **searchWordInList** e **isStopWord**.

Função Principal (main())

Na função main, acrescentou-se apenas as opções que devem ser informadas pelo usuário, e estas foram utilizadas como parâmetro para as demais funções.

Primeiramente, foi declarada uma variável do tipo vetor de caractere para armazenar o caminho do arquivo contendo o texto, além de uma variável do tipo inteiro para armazenar o tamanho do resumo e de uma variável do tipo ponteiro para a estrutura **WordList**.

Após a declaração destas variáveis, é realizada uma operação de alocação de memória ocupando o tamanho de uma **WordList**, em seguida, o endereço de memória desta alocação é atribuído ao ponteiro para **WordList**, e o valor do atributo tamanho, dentro do ponteiro de **WordList**, é atribuído como 0. Depois, é utilizada a função **input()**, proveniente da biblioteca **yolib.h**, para imprimir uma mensagem na tela e requisitar a entrada do usuário, e o retorno desta função é atribuído à variável que guarda o caminho do arquivo de texto. A função **input()** é usada novamente, mas desta vez, seu retorno é convertido em inteiro, através da função **toint()**, também da biblioteca **yolib.h**, e então atribuído à variável que armazena o tamanho do resumo.

Por fim, são chamadas as funções **countWords()**, para adicionar as palavras do arquivo de texto à estrutura **WordList**, **sortWords()**, para ordenar as palavras dentro do vetor da **WordList** de forma decrescente de acordo com o seu índice de repetições, e **showImportantWords()**, que exibe na tela as n primeiras palavras da **WordList**, onde n é o tamanho do resumo.

Função coundWords()

A função **countWords()** funciona da seguinte forma: são passadas por parâmetro o nome do arquivo que contém o texto e um ponteiro para a estrutura **WordList**. Essa função irá tentar abrir o arquivo com o nome passado no parâmetro, caso não consiga abrir, é imprimida uma mensagem de erro e o programa é encerrado, caso contrário, ela irá ler todas as palavras encontradas no texto uma a uma, usando a função **searchWordInList(),** para verificar se elas já foram incluídas na **WordList,** e a função **isStopWord()** para verificar se a palavra está na lista de stopwords. Caso a palavra não seja uma stopword e já tenha sido incluída, a variável que guarda a quantidade de repetições daquela palavra é incrementada, caso contrário a palavra é incluída na **WordList**.

Função sortWords()

Para a função **sortWords()** são passados como parâmetro um ponteiro para a **WordList,** e ela realiza uma operação de insertion sort (Ordenação por inserção), que ordena o vetor de palavras da **WordList** de forma decrescente, fazendo com que as palavras que mais se repetem no texto fiquem nas posições iniciais do vetor.

Nesta função, foi utilizado o conceito de Insertion Sort, aplicado em sala de aula. Nela foi utilizado uma versão levemente modificada do algoritmo mostrado na disciplina, para que a mesma ordene o vetor de forma decrescente.

Função showImportantWords()

Para exibir a lista de palavras mais importantes, foi criada a função **showInportantWords()**, que recebe como parâmetro um ponteiro para a **WordList** e um inteiro **n** informado pelo usuário contendo o número de palavras que devem ser exibidas. A função então exibe as **n** primeiras palavras do vetor juntamente com a sua respectiva quantidade de repetições.

A função é composta por um laço de repetição, que utiliza um contador **i**, cuja iteração inicia de 1, e vai até **n**, onde n é o menor número entre o tamanho utilizado do vetor da **WordList** e o número passado por parâmetro. O escopo do laço de repetição contém apenas um comando que imprime a palavra encontrada no índice **i** do vetor, e sua respectiva quantidade de repetições.

Função searchWordInList()

A função **searchWordInList()** recebe como parâmetro um ponteiro para vetor de caracteres e um ponteiro para a **WordList**. A função então realiza uma busca sequencial na **WordList** para tentar encontrar a string que corresponde ao vetor de caracteres passado por parâmetro. Então a função retorna o índice onde se encontra a string dentro do vetor de palavras da **WordList**, ou retorna 0 caso a string não exista no vetor da **WordList**.

Função isStopWord()

isStopWord() é uma função que recebe como parâmetro uma string contendo uma palavra, a função abre um arquivo chamado **stopwords.txt**, que contém todas as stopwords, e verifica se a palavra está contida no arquivo, caso sim a função irá retornar 1, caso não, a função retorna 0.

Conhecimentos da Disciplina

Dos conhecimentos aplicados em sala de aula foram utilizados os códigos de ordenação por inserção e busca sequencial. Além também dos conhecimento de alocação dinâmica e ponteiros, passagem por parâmetros e manuseio de arquivos.

Documentação breve das bibliotecas utilizadas:

- yolib.h

  char * input(char *message)

  Imprime uma mensagem na tela e solicita a entrada do usuário, então é retornada uma string contendo a entrada digitada pelo usuário.

  int toint(char *charV)

  Converte uma string em inteiro. Caso a conversão seja falha, a função retorna 0.

  int tofloat(char *charV)

  Converte uma string em float.Caso a conversão seja falha, a função retorna 0.

- yostring.h

  int strLen(char *str)

  Retorna o tamanho de uma string.

  char * strtoup(char *str)

  Converte todas as letras de uma string para maiúsculo e retorna o ponteiro para essa string.

  char * strtolow(char *str)

  Converte todas as letras de uma string para minúsculo e retorna o ponteiro para essa string.

  char *removeSpecialChars(char *str)

  Remove quaisquer caracteres especiais de uma string e retorna o ponteiro para essa string.

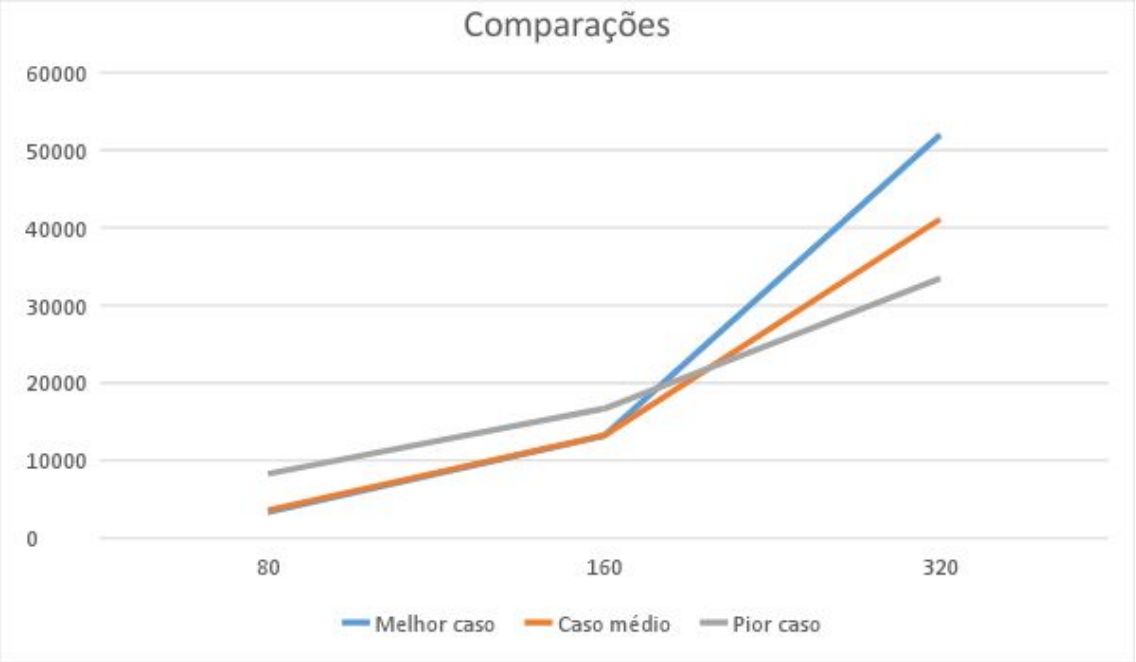
  int thereIsSpecialChar(char *str)

  Verifica se uma string contém caracteres especiais. Retorna 1 caso contenha ou 0 caso não contenha.

3. **Análise da solução.**
    - **Desempenho**

Para verificar o funcionamento do programa foram realizadas simulações para o melhor caso, caso médio e pior caso.

Como exemplo para se analisar o melhor caso foi utilizado um arquivo de texto onde todas as palavras contidas neste, eram diferentes. Para análise do caso utilizou-se um texto comum. E para análise de pior caso foi utilizado um arquivo de texto onde todas as palavras são iguais.



Analisando os números do gráfico, vemos que, caso haja menos de 160 palavras no texto, o melhor caso possui o menor custo, caso o texto possua mais palavras, o melhor caso passa a apresentar um custo maior. Já no pior caso, ele apresenta um pior desempenho para textos inferiores a 160 palavras, entretanto, nos textos com mais de 160 palavras, ele passa a apresentar um custo menor.

- **Limitações**

O programa possui limitações quanto à quantidade de palavras no texto. Como a solução se utiliza de um vetor de estrutura para armazenar as palavras do texto, foi definido um tamanho máximo para esse vetor de 10000, logo, o programa está limitado a textos com até 10000 palavras diferentes.

4. Conclusão.

O problema proposto foi um sumarizador de notícias que pudesse exibir as palavras mais mencionadas em um determinado texto ou notícia. Como solução, foi desenvolvido um programa capaz de abrir um arquivo de texto informado pelo usuário, e contar todas as palavras deste arquivo, bem como o número de repetições de cada uma delas.

Vantagens:

- Eficácia
- Fácil utilização

Desvantagens:

- Limite de 10000 palavras diferentes no texto

Possíveis Melhorias:

O programa atualmente utiliza uma busca sequencial para verificar se uma palavra do texto já foi incluída na WordList, esta busca poderia ser substituída por um algoritmo de busca binária, tornando o código mais eficiente.

O programa atualmente exibe apenas uma lista em ordem decrescente das palavras que mais aparecem no texto, uma melhoria possível seria fazer com que o programa gerasse uma nuvem de palavras, assim a visualização da saída do programa seria mais fácil e intuitiva.