Universidade Federal do Piauí - Campus SHNB
Sistemas de Informação
Algoritmos e Programação II
Nome: Marcos André Leal Silva
Nome: Pedro Vitor Passos Pereira
Nome: Gabriel Leal de Lima

Trabalho Prático

1 Introdução

Foi solicitado aos alunos a criação de um programa que pudesse identificar e extrair as palavras mais frequentes em um texto, além das stopwords inseridas nos textos, permitindo que o usuário escolha a quantidade de palavras que deseja encontrar e que pudesse selecionar entre textos em inglês ou português

Dessa forma, o desenvolvimento de programas como esse é uma tarefa muito interessante e desafiadora, pois envolve conceitos de algoritmos de ordenação e contagem de palavras, além de manipulação de arquivos. Além disso, a capacidade de processar textos em diferentes idiomas torna o programa ainda mais útil e versátil. Com a quantidade crescente de dados textuais disponíveis na internet e em outras fontes, a demanda por ferramentas de análise de texto eficientes e personalizáveis está em constante crescimento.

2 Solução Proposta

A solução gerada para tal, foi desenvolver um código em que fosse possível que o usuário\testador escolhesse qualquer texto, sendo em inglês ou português, com o intuito de descobrir as palavras mais frequentes do texto de acordo com quantas palavras o usuário digitasse, a exemplo, caso o usuário digitasse 2 (duas), seriam apresentados em um arquivo .txt, ou seja, um arquivo de texto, as (duas) palavras mais frequentes do texto inserido. Dessa maneira, o código-fonte foi modularizado em 7(sete) funções definidas para cada ação que seria feita no código. Sendo elas (para melhor explicação serão colocadas abaixo as funções em linguagem C e pseudocódigo em portugol das técnicas e algoritmos utilizados):

- **void leitura_texto(char *file_path, tabela *t);**

  Esta função foi desenvolvida para ler o texto que fosse inserido pelo usuário separando linha por linha em strings na tabela 1 (t) e colocando em minúsculos para facilitar na manipulação dos dados.

- **void leitura_stopwords(char *stopwords_path, tabela *t2);**

  Esta função foi desenvolvida para ler o texto, contendo stopwords (palavras irrelevantes para a compreensão final do sumarizador), que fosse inserido pelo usuário

separando linha por linha em strings na tabela 2 (t) e colocando em minúsculos para facilitar na manipulação dos dados.

- **void remover_stopwords(tabela *t, tabela *t2, tabela *t3);**

  Esta função serve para comparar a tabela 1 e a tabela 2, removendo as stopwords do texto, ou seja, serão inseridas em uma tabela de string (t3) todas as palavras que podem ser consideradas relevantes em um texto para o conjunto de resultados a ser exibido.

- **void contador_palavras(tabela *t3, tabela *t4);**

  Será utilizada para contar as palavras mais frequentes do texto e encaixar essas palavras na tabela 4 (t4).

- **void ordenacao(tabela *t4);**

  A função de ordenar será utilizada para fazer a ordenação das palavras mais frequentes utilizando um contador como referência.

- **void remover_duplicadas(tabela *t4);**

  Utilizada para remover palavras duplicadas.

- **void resumo(char *output_path, int tam, tabela *t4);**

  A função resumo faz com que a tabela 4 (t4) seja passada para um arquivo .txt que o programa irá criar.

- **Técnicas de programação utilizadas:**

- **Funções de manipulação de string;**

```
algoritmo ExemploString
var
   palavra: string
   tamanho: inteiro
inicio
   escreva("Digite uma palavra: ")
   leia(palavra)
   tamanho <- strlen(palavra)
```

```
    escreva("A palavra '", palavra, "' tem ", tamanho, " caracteres.")
fimalgoritmo
```

- **Funções de manipulação de struct;**

```
struct Pessoa
    nome: string
    idade: inteiro
    altura: real
fimstruct

algoritmo ExemploStruct
var
    pessoa1: Pessoa
inicio
    pessoa1.nome <- "João"
    pessoa1.idade <- 30
    pessoa1.altura <- 1.75

fimalgoritmo
```

- **Funções de manipulação de arquivo;**

```
algoritmo ExemploEscritaArquivo
var
  arquivo: arquivo
  nome: string
  idade: inteiro
inicio
  nome <- "João"
  idade <- 30

  // Abrindo o arquivo para escrita
  arquivo <- abrirArquivo("pessoas.txt", "escrita")

  // Escrevendo no arquivo
  escrever(arquivo, "Nome: ", nome, " - Idade: ", idade)
```

```
  // Fechando o arquivo
  fecharArquivo(arquivo)
fimalgoritmo


algoritmo ExemploLeituraArquivo
var
  arquivo: arquivo
  nome: string
  idade: inteiro
inicio
  // Abrindo o arquivo para leitura
  arquivo <- abrirArquivo("pessoas.txt", "leitura")

  // Lendo do arquivo
  leia(arquivo, nome)
  leia(arquivo, idade)

  // Fechando o arquivo
  fecharArquivo(arquivo)

  // Exibindo os dados lidos
  escreva("Nome: ", nome, " - Idade: ", idade)
fimalgoritmo
```

- **Funções de manipulação de vetores;**

```
algoritmo ExemploVetor
var
  numeros: vetor[5] de inteiro
  tamanho, limite, posicao: inteiro
inicio
  limite <- 100
  tamanho <- 5

  PreencherVetorAleatorio(numeros, tamanho, limite)

  escreva("Vetor original: ")
  para i de 0 ate tamanho-1 faca
    escreva(numeros[i], " ")
  fimpara
```

```
    OrdenarVetor(numeros, tamanho)

    escreva("\nVetor ordenado: ")
    para i de 0 ate tamanho-1 faca
      escreva(numeros[i], " ")
    fimpara

    posicao <- BuscarValor(numeros, tamanho, 50)
    se posicao <> -1 entao
      escreva("\nO valor 50 está na posição ", posicao)
    senao
      escreva("\nO valor 50 não foi encontrado no vetor.")
    fimse
  fimalgoritmo
```

- **Função de ordenação;**

```
funcao ordenarPorInsercao(vetor: vetor[1..n] de inteiro, n: inteiro)
inicio
   para i de 2 ate n faca
      chave := vetor[i]
      j := i-1
      enquanto j >= 1 e vetor[j] > chave faca
         vetor[j+1] := vetor[j]
         j := j-1
      fimEnquanto
      vetor[j+1] := chave
   fimPara
fimFuncao
```

---

3 Análise de desempenho

O código em questão é responsável por realizar o processamento de um arquivo de texto puro, removendo as palavras consideradas stopwords (palavras que não são importantes para o contexto) e gerando um resumo contendo as palavras mais relevantes do texto.

Outrossim, outro fator que pode afetar o desempenho do código é o fato de que as funções "strcmp" e "strcpy" são chamadas diversas vezes dentro das funções "remover_stopwords",

"contador_palavras" e "ordenacao". Essas funções possuem um custo elevado em termos de processamento, o que pode impactar o tempo de execução do código. Uma forma de contornar esse problema seria armazenar o resultado dessas funções em variáveis temporárias, evitando chamá-las múltiplas vezes.

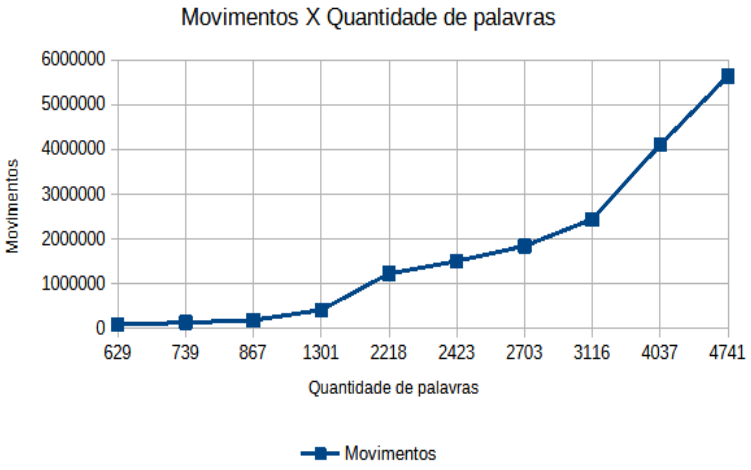
O desempenho do programa pode ser avaliado considerando-se os casos melhor, médio e pior de execução.

Caso melhor:
O caso melhor ocorre quando o arquivo de entrada é pequeno e não contém muitas palavras. Nesse caso, a execução do programa será rápida, pois as operações de leitura, processamento e escrita serão concluídas rapidamente.

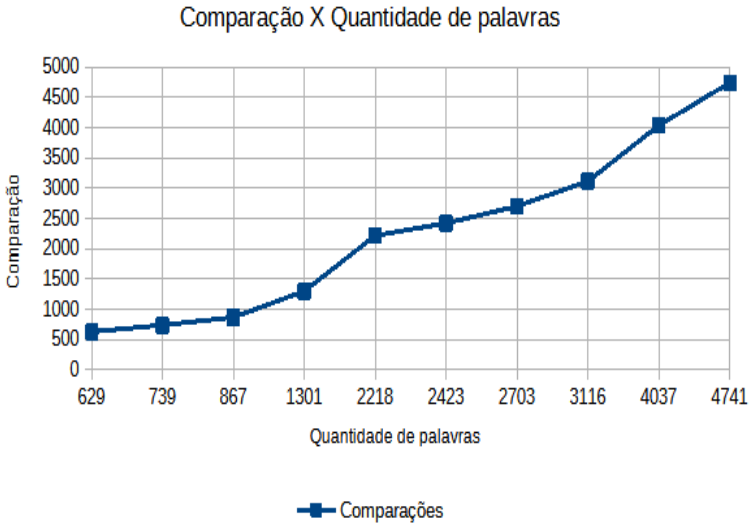
Caso médio:
O caso médio ocorre quando o arquivo de entrada é de tamanho moderado, contendo várias palavras. Nesse caso, o programa pode levar mais tempo para processar as palavras, mas ainda deve ser concluído em tempo razoável.

Caso pior:
O caso pior ocorre quando o arquivo de entrada é grande e contém muitas palavras. Nesse caso, o programa pode demorar consideravelmente mais para processar as palavras e gerar o resumo. Isso ocorre porque o programa precisa fazer várias interações sobre as palavras, o que aumenta a complexidade do algoritmo.

Abaixo estão dois gráficos e duas tabelas com a relação entre os testes que foram feitos no algoritmo em que fosse utilizada o três casos e utilizando o algoritmo de inserção como base para o gráfico, assim pode-se notar que a quantidade de movimentos que são feitas é relativamente elevada, ou seja, é um custo elevado de tempo para que possa ser totalmente retornado o resultado esperado.



Quantidade de palavras	Movimentos
629	96504
739	134846
867	191656
1301	417544
2218	1229090
2423	1507072
2703	1845034
3116	2443862
4037	4103185
4741	5636388



Quantidade de Palavras	Comparações
629	628
739	738
867	867
1301	1300
2218	2217
2423	2422
2703	2702
3116	3115
4037	4036
4741	4740

.

4 Conclusão

Portanto, o problema proposto de retirar as stopwords de um texto e inserir as palavras mais frequentes do mesmo texto em um arquivo .txt foram solucionados com uma estrutura de código simples. Um dos aspectos positivos para solucionar o problema, foi a utilização de códigos fontes e técnicas de programação aprendidas no decorrer da disciplina, o que tornou de fácil entendimento para o mesmo.

No entanto, alguns aspectos podem ser melhorados, principalmente na função de “ordenacao”, que está sendo utilizada no modelo de inserção. Assim, para uma alternativa seria utilizar algoritmos de ordenação mais eficientes, como o quicksort ou o mergesort.

No geral, o código possui uma estrutura simples, com funções bem definidas e separação clara de responsabilidades. Entretanto, há algumas melhorias de desempenho que podem ser feitas em relação à sua eficiência e organização.