Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
DCC205 - Estrutura de Dados

Trabalho Prático 2

Prazo de Entrega: 01/07/2022
Valor: 15 pontos

1- Objetivo

O objetivo deste trabalho é realizar uma análise do número de ocorrências das palavras usadas em um texto baseada em uma nova ordem lexicográfica.

2- O Analisador

O programa deve ser capaz de ler todas as palavras contidas num arquivo de entrada e relatar no arquivo de saída todas as palavras que ocorrem no texto, seguido do número de vezes que esta palavra foi encontrada.

As palavras unidas por hífen (“-”) devem ser consideradas como uma só durante a análise e suas partes não devem ser contabilizadas separadamente.

Palavras que sejam encontradas mais de uma vez, com diferenciação de caixa alta ou baixa, devem ser contabilizadas como a mesma palavra. No arquivo de saída, as palavras encontradas devem sempre estar em caixa baixa.

Por exemplo, as palavras “TRABALHO” e “tRaBaLhO” devem contabilizar duas ocorrências da mesma palavra no arquivo de saída - “trabalho”.

O programa deve ser capaz de ignorar espaços em branco e elementos de pontuação (“,”, “.”, “!”, “?”, “:”, “;” e “_”) no final das palavras. Estes elementos devem ser removidos da saída do programa.

3- Entrada

A entrada do programa será feita através de um arquivo de entrada, passado como parâmetro durante a execução utilizando a flag abaixo:

-[i|l][1] : arquivo de entrada com o texto a ser processado e a nova ordem lexicográfica a ser usada

O arquivo de entrada será composto por dois blocos de informação: um bloco que define a nova

[1] Os parâmetros podem ser informados com letras maiúsculas ou minúsculas.

ordem lexicográfica e outro que conterá um texto não formatado, composto de ao menos um caractere, que deverá ser tratado pelo programa. O bloco de definição da nova ordem lexicográfica será sempre iniciado pelo identificador "#ORDEM"[2] seguido de uma quebra de linha. Em seguida, será informado a nova ordem lexicográfica para os caracteres, separados por um ou mais espaçamentos, seguindo o formato abaixo, seguido de uma quebra de linha:

$(CARACTERE)_1$ $(CARACTERE)_2$ ..... $(CARACTERE)_{26}$

CARACTERE: Representa um caractere dentro intervalo de caracteres A-Z[3].

O bloco com o texto será iniciado pelo identificador "#TEXTO"[2], seguido de uma quebra de linha. Em seguida será informado um texto que poderá conter uma ou mais linhas, sendo que cada linha será composta de pelo menos um caractere. O número de #ORDEM e #TEXTO é arbitrário.

3.1- Ordenação

A ordenação das palavras da saída do programa deve ser feita utilizando a nova ordem lexicográfica disponível no arquivo de entrada.

A ordenação dos demais elementos que possam aparecer no texto (como o hífen "-"), devem seguir a ordem ascendente da tabela ASCII.

Ex.: #ORDEM = Z Y X W V U T S R Q P O N M L K J I H G F E D C B A

"-teste" vem antes de "7up", que vem antes de "trabalho", que vem antes de "pratico"

3.2- Algoritmos

Você deve estender o algoritmo Quicksort para realizar ordenações usando as ordens alternativas descritas. Você deve também implementar duas otimizações:

- Mediana de M elementos, que escolhe o pivô como sendo a mediana de M elementos escolhidos a partir do vetor de entrada. O valor de M é um parâmetro informado utilizando a opção -[m|M] da linha de comando.
- Uso de um algoritmo simples (Inserção e Seleção) para partições de menor tamanho. O tamanho máximo das partições a serem consideradas para essa otimização é informado pela opção -[s|S] da linha de comando.

3.3- Avaliação Experimental

---

[2] *Case sensitive*
[3] *Não é Case sensitive, portanto para a nova ordem lexicográfica 'a' == ' A'*

Você deve considerar pelo menos as seguintes dimensões na sua avaliação experimental:

- Tamanho do vetor a ser ordenado
- Número de elementos considerados na escolha do pivô como mediana (note que a escolha tradicional pode ser considerada como "Mediana de 1".
- Tamanho da partição a partir do qual será usado um método simples. Nesse caso, é interessante entender o compromisso envolvido, ou seja, há um tamanho ótimo de partição em termos de desempenho.

4- Saída

A saída do programa deve ser direcionada a um arquivo de saída cujo caminho será fornecido através da seguinte opção de linha de comando:

-[o|O] : endereço do arquivo de saída

O arquivo de saída deverá conter todas as palavras distintas do arquivo de entrada, em formato minúsculo, seguindo as especificações descritas na seção 2, seguido do número de vezes que determinada palavra aparece no texto de entrada. A comparação de ocorrências de uma palavra no texto não é *case sensitive*.

A saída do analisador deve ser ordenada de forma ascendente baseado na ordem lexicográfica informada.

Cada palavra, juntamente com a sua frequência, no arquivo de saída deve ser separada por uma quebra de linha. A última linha do arquivo de saída deve conter o identificador "#FIM"[2], seguido de uma quebra de linha.

5- Exemplo de Entrada e Saída

Entrada	**Saída**
#ORDEM Z Y X W V U T S R Q P O N M L K J I H G F E D C B A #TEXTO Era uma vez UMA gata xadrez.	xadrez 1 vez 1 uma 2 gata 1 era 1 #FIM

6- Premissas

1. Não existe ordem pré-definida da ocorrência do bloco da ordem lexicográfica e do texto

no arquivo de entrada. Ou seja, o ordem de ocorrência das tags "#ORDEM" e "#TEXTO" não é fixa.

2. Não existe ordem pré-definida da ocorrência dos parâmetros da linha de comando. Os caracteres do bloco da ordem lexicográfica sempre serão um dos caracteres do intervalo A-Z (incluindo “K”, “Y” e “W”).

3. A execução do programa não deve durar mais que 60 segundos.

4. Não existem limites para a quantidade de caracteres numa linha do arquivo de entrada

5. Não existem limites para a quantidade de linhas no arquivo de entrada

6. Palavras agrupadas por hífen não possuem espaços em branco

Ex.: “guarda-chuva” **CORRETO**

“guarda- chuva” **INCORRETO**

7. Nenhuma palavra do texto de entrada será acentuada[4]

8. Não existirão aspas simples ou duplas no texto de entrada

9. A nova ordem lexicográfica será apenas para o intervalo de caracteres A-Z, os demais seguirão sua ordem padrão.

10. As palavras “#ORDEM” e “#TEXTO” (*case sensitive*) são reservadas e não serão usadas no texto a ser processado

11.Nenhum sinal de pontuação estará presente no texto, exceto aqueles descritos na seção 2

7. Atividade Extra (OPCIONAL - 1 ponto)

Para prevenir erros de entrada, seu trabalho deve ser capaz de corrigir eventuais problemas no arquivo de entrada, antes de realizar o processamento sobre o texto.

Os possíveis erros que seu programa deve corrigir são:

- Conversão de caracteres acentuados para seu respectivo par, sem acentos.
- União de palavras com hífen separadas por espaços em branco
  - Todas as palavras terminadas com hífen deverão ser unidas à palavra subsequente
- Remoção de caracteres de pontuação (“,”, “.”, “!”, “?”, “:”, “;” e “_”) do início e meio de palavras
  - Quando estes caracteres ocorrerem no meio da palavra, as duas “metades” devem ser unidas

Após corrigir os problemas encontrados, o processamento do texto deve ser o mesmo descrito na Seção 2.

7.1- Correção de palavras com hífen

[4] Esta premissa não é válida para aqueles que implementarem a parte extra do trabalho, descrita na seção 8

Toda palavra termina no caractere hífen (“-”) deve ser unida a próxima palavra presente no texto. Esta regra, porém, não deve ser aplicada para uma ocorrência de um hífen livre no texto. Desta forma, para aplicar esta regra, a palavra terminada em hífen deve ter tamanho maior que 1.

A correção de palavras com hífen não é recursiva, ou seja, caso uma palavra seja unida a outra que também termine em hífen, esta **não** deve ser unida à palavra seguinte.

Caso uma palavra termine em hífen e seja a última palavra da linha do texto, a regra de união não se aplicará.

8- Entregáveis

Você deve utilizar a linguagem C ou C++ para o desenvolvimento do seu sistema. O uso de estruturas pré-implementadas pelas bibliotecas-padrão da linguagem ou terceiros é **terminantemente vetado**. Caso seja necessário, use as estruturas que **você** implementou nos Trabalhos Práticos anteriores para criar **suas próprias implementações** para todas as classes, estruturas de dados, e algoritmos.

Você **DEVE utilizar** a estrutura de projeto abaixo junto ao *Makefile* :

```
- TP
   |- src
   |- bin
   |- obj
   |- include
   Makefile
```

A pasta **TP** é a raiz do projeto; a pasta **bin** deve estar vazia; src deve armazenar arquivos de código (*.c, *.cpp ou *.cc); a pasta include, os cabeçalhos (*headers*) do projeto, com extensão *.h, por fim a pasta **obj** deve estar vazia. O **Makefile** deve estar na **raiz do projeto**. A execução do **Makefile** deve gerar os códigos objeto *.o no diretório **obj,** e o executável do TP no diretório **bin**.

8.1 Documentação

A documentação do trabalho deve ser entregue em formato **pdf**. A documentação deve conter os itens descritos a seguir :

Título, nome, e matrícula.
Introdução: Contém a apresentação do contexto, problema, e qual solução será empregada.
Método: Descrição da implementação, detalhando as estruturas de dados, tipos abstratos de dados

(ou classes) e funções (ou métodos) implementados.
Análise de Complexidade: Contém a análise da complexidade de tempo e espaço dos procedimentos implementados, formalizada pela notação assintótica.
Estratégias de Robustez: Contém a descrição, justificativa e implementação dos mecanismos de programação defensiva e tolerância a falhas implementados.
Análise Experimental: Apresenta os experimentos realizados em termos de desempenho computacional e localidade de referência, assim como as análises dos resultados.
Conclusões: A Conclusão deve conter uma frase inicial sobre o que foi feito no trabalho. Posteriormente deve-se sumarizar o que foi aprendido.
Bibliografia: Contém fontes utilizadas para realização do trabalho. A citação deve estar em formato científico apropriado que deve ser escolhido por você.
Instruções para compilação e execução: Esta seção deve ser colocada em um apêndice ao fim do documento e em uma página exclusiva (não divide página com outras seções).
***Número máximo de páginas:** 20

A documentação deve conter a descrição do seu trabalho em termos funcionais, dando foco nos algoritmos, estruturas de dados e decisões de implementação importantes durante o desenvolvimento.

Evite a descrição literal do código-fonte na documentação do trabalho.

Dica: Sua documentação deve ser clara o suficiente para que uma pessoa (da área de Computação ou não) consiga ler, entender o problema tratado e como foi feita a solução.

8.2 Submissão

Todos os arquivos relacionados ao trabalho devem ser submetidos na atividade designada para tal no Moodle. A entrega deve ser feita **em um único arquivo** com extensão **.zip**, com nomenclatura nome_sobrenome_matricula.zip}, onde nome, sobrenome e matrícula devem ser substituídos por suas informações pessoais. O arquivo **.zip** deve conter a sua documentação no formato **.pdf** e a estrutura de projeto descrita no início da Seção 3.

9. Avaliação

O trabalho será avaliado de acordo com:

- A Corretude na execução dos casos de teste - (50% da nota total)
- Apresentação da análise de complexidade das implementações - (15% da nota total)
- Estrutura e conteúdo exigidos para a documentação - (20% da nota total)

- Indentação, comentários do código fonte e uso de boas práticas - (10% da nota total)
- Cumprimento total da especificação - (5% da nota total)

Se o programa submetido não compilar[5], seu trabalho não será avaliado e sua nota será 0. Trabalhos entregues com atrasos sofrerão penalização de 2^{d-1} pontos, com d = dias de atraso.

10. Considerações Finais

1. Comece a fazer esse trabalho prático o quanto antes, enquanto o prazo de entrega está tão distante quanto jamais estará.
2. Leia **atentamente** o documento de especificação, pois o descumprimento de quaisquer requisitos obrigatórios aqui descritos causará penalizações na nota final.
3. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.
4. Plágio é CRIME. Trabalho onde o plágio for identificado serão **automaticamente anulados** e as medidas administrativas cabíveis serão tomadas (em relação a todos os envolvidos). Discussões a respeito do trabalho entre colegas são permitidas. É permitido consultar fontes externas, desde que exclusivamente para fins didáticos e devidamente registradas na sessão de bibliografia da documentação. **Cópia e compartilhamento de código não são permitidos**.

FaQ (Frequently asked Questions)

1. **Posso utilizar alguma estrutura de dados do tipo Queue, Stack, Vector, List, e etc..., do C++? NÃO**
2. Posso utilizar o tipo String? SIM.
3. Posso utilizar o tipo String para simular minhas estruturas de dados? NÃO
4. Posso utilizar alguma biblioteca para tratar exceções? SIM.
5. Posso utilizar alguma biblioteca para gerenciar memória? SIM.
6. As análises e apresentação dos resultados são importantes na documentação? SIM.
7. Os meus princípios de programação ligados a C++ e relacionados a engenharia de software serão avaliados? NÃO
8. Posso fazer o trabalho em dupla ou em grupo? NÃO
9. Posso trocar informações com os colegas sobre a teoria? SIM.
10. Posso fazer o trabalho no Windows, Linux, ou MacOS? SIM.
11. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? SIM.

[5] Entende-se por compilar aquele programa que, independente de erros no Makefile ou relacionados a problemas na configuração do ambiente, funcione e atenda aos requisitos especificados neste documento em um ambiente Linux.