UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA

Disciplina: [1107186] - Estruturas de Dados – Turma 02
Prof. Christian Azambuja Pagot (christian@ci.ufpb.br)
Data de Entrega: 28/08/2019

Spell Checker

1. Objetivo

O objetivo deste trabalho consiste na implementação de um *spell checker* através do uso de tabelas *hash*.

2. Problema

No início dos anos 60 surgiram os primeiros programas de computador capazes de verificar a ortografia de palavras contidas em textos. Estes programas passaram a ser popularmente conhecidos como *spell checkers*.

O *spell checker* é um programa que testa as palavras de um texto contra um dicionário. Se uma determinada palavra do texto é encontrada no dicionário, presume-se que ela está escrita corretamente. Se a palavra não é encontrada no dicionário, considera-se que ela esteja escrita de forma incorreta ou que o dicionário em questão ainda não a contém.

Neste trabalho os alunos deverão implementar um *spell checker*. O programa receberá como entrada dois arquivos texto: 1) um dicionário; e 2) um texto a ser verificado. Cada palavra do texto deverá ser verificada contra o dicionário. Após o término da verificação, o programa deverá gerar um arquivo texto de saída contendo as seguintes estatísticas:

- Número total de palavras do texto.
- Tempo gasto na verificação (com precisão mínima de milisegundos).
- Número de palavras do texto que não encontraram correspondência no dicionário.
- Lista das palavras não encontradas.

3. Desenvolvimento

O trabalho pode ser desenvolvido individualmente, em duplas ou em trios. O programa deve ser escrito na linguagem C. Somente as bibliotecas e estruturas de dados padrão que acompanham a linguagem C podem ser utilizadas.

Como forma de tornar o teste das palavras contra o dicionário mais eficiente, o dicionário deverá ser implementado através de uma tabela *hash*. Desta forma, para se verificar a ortografia de uma palavra do texto, será necessário calcular o seu valor de *hash* e verificar se este valor já se encontra inserido na tabela de *hash* do dicionário.

Os alunos deverão definir quantos *bucktes* a tabela de hash deverá ter, qual a função de *hash* a ser utilizada e como serão tratados os eventuais casos de colisão.

Ao final do processo de verificação o programa deve gerar um relatório, em formato de arquivo

texto, com os resultados. O formato do relatório é apresentado no exemplo a seguir:

```
Número total de palavras do texto: 123
Tempo total da verificação: 54ms.
Número de palavras que falharam no spell check: 3
Lista de palavras que falharam no spell check:

Num. Ocorrencia - Palavra
----------------------------------------
1 - shjgdy
2 - ttttttttt
3 - rrrr
```

Observações sobre o arquivo de saída:

- **Número total de palavras do texto**: Corresponde ao número total de palavras encontradas no texto.
- **Tempo da verificação:** É o tempo gasto pelo programa para verificar todas as palavras do texto de entrada. Este tempo deve considerar **apenas** o tempo dos testes, ou seja, **não devem ser contabilizados os tempos de carga do dicionário, construção da tabela *hash* e o tempo de escrita do relatório final no disco rígido!** A precisão desta medida deve ser de, **pelo menos**, **milisegundos**.
- **Número de palavras que falharam no *spell check***: Corresponde ao número total de palavras do texto que não foram encontradas no dicionário.
- **Lista de palavras que falharam no *spell check***: Lista de todas as palavras que não passaram no teste de ortografia.

Observações sobre o dicionário e o arquivo texto de entrada:

- O arquivo de dicionário, contendo palavras em português, será fornecido pelo professor como parte do exercício e está codificado no formato ASCII e sem acentos.
- O *spell checker* deve diferenciar letras maiúsculas e minúsculas.
- O texto a ser analisado deve estar codificado em ASCII, caso contrário, o *spell checking* pode falhar.

Extra

A realização de comparações do sistema implementado com sistemas de *hash* existentes (e.g. aqueles encontrados em Lua, Python, Java, C++, etc.) pode render, dependendo da qualidade técnica desta comparação, **até 3 pontos extras** no trabalho. É aconselhável que os alunos que pretendam realizar a tarefa extra conversem antes com o professor para detalharem melhor como as comparações serão feitas.

Avaliação

Trabalhos que atendam adequadamente os requisitos mínimos definidos neste documento concorrerão a nota máxima de 10 pontos.

A avaliação do trabalho incluirá os pontos a seguir:

- Clareza e organização da apresentação.
- Eficiência e eficácia das soluções apresentadas.
- Embasamento técnico dos argumentos.
- A adequação dos algoritmos e estruturas de dados utilizadas na implementação.
- Relevância dos elementos apresentados (imagens, trechos de código, etc.) no contexto do trabalho.
- Experimentos realizados e capacidade de análise dos resultados obtidos.
- Qualidade técnica da atividade extra (se esta for incluída no trabalho).

Entrega

Entrega do trabalho

O trabalho será considerado entregue assim que este for enviado pelo SIGAA até o dia **28/08/2019, às 23h59min**.

A entrega consistirá no envio de um **arquivo ZIP** contendo o **código fonte** e um artigo de **3 páginas, em formato PDF**, contedo o desenvolvimento e resultados obtidos. O template para a escrita do artigo será divulgado pelo professor.

A não entrega do trabalho acarretará em nota zero.

Não serão aceitos trabalhos atrasados.

Se os alunos preferirem, poderão também postar o código fonte de suas implementações em repositórios tais como o Github.