BCC202 – Estruturas de Dados I (2024-01)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (`www.decom.ufop.br/`)

Trabalho Prático III (TP III) - 10 pontos, peso 1.

- Submissão com data e hora de entrega disponíveis na plataforma da disciplina. O que vale é o horário do *Moodle*, e não do *seu*, ou do *meu* relógio!!!

- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.

- O padrão de entrada e saída deve ser respeitado exatamente como determinado no enunciado. Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos.

- Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.

- A avaliação considerará o tempo de execução e o percentual de respostas corretas.

- Eventualmente serão realizadas entrevistas sobre o trabalho para complementar a avaliação;

- O trabalho é em grupo de até 2 (duas) pessoas.

- Será aceito trabalhos após a data de entrega, todavia com um decréscimo de 0,05 a cada 10min.

- Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.

- Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.

- Códigos ou funções prontas específicas de algoritmos para solução dos problemas elencados não são aceitos

- Não serão considerados algoritmos parcialmente implementados.

- Procedimento para a entrega:.

1. Submissão: via ***Moodle***.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via ***Moodle***.
6. Você deve submeter os arquivos *.h*, *.c* e o *.pdf* (relatório) na raiz do arquivo *.zip*. Use os nomes dos arquivos *.h* e *.c* exatamente como pedido.
7. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.

- Bom trabalho!

Hashing para pesquisa de documentos

Um banco de dados textual é uma coleção de documentos. Cada documento é constituído por uma lista de palavras. Uma estratégia de pesquisa a estes documentos é comparar a lista de palavras da consulta com a lista de palavras dos documentos. Entretanto, em grandes coleções a comparação direta é computacionalmente cara.

A base para o funcionamento de uma máquina de busca, tais como *Google* e *Yahoo!*, é a construção de um **índice invertido** para uma coleção de documentos. Esse índice pode ser entendido como ao invés de realizar a busca pelo nome do arquivo, ou buscar sequencialmente, salva-se quais as palavras em quais arquivos e a busca ocorre por essas palavras. O objetivo deste trabalho é construir um **índice invertido** utilizando tabela *hash* como estrutura de dados.

Considere que dado um banco de dados textual, exista um arquivo denominado *palavras_chave.txt* que armazena as principais palavras (palavras-chave) associadas a cada documento, conforme mostrado no exemplo abaixo.

```
prog.doc   algoritmo   selecao
aeds1.doc   algoritmo   estrutura   dados
darwin.doc   selecao   natural
```

palavras_chave.txt

Destaca-se que em cada linha do arquivo *palavras_chave.txt* estão, respectivamente, o nome de um documento seguido pelas suas palavras-chave.

O **índice invertido** possui duas partes principais: um vocabulário, contendo todos os termos distintos existentes no arquivo de palavras-chave e, para cada termo, uma lista que armazena os identificadores dos documentos que o contém. Supondo os identificadores 1, 2 e 3 para os documentos **prog.doc**, **aeds1.doc** e **darwin.doc**, respectivamente, o **índice invertido** para o arquivo *palavras_chave.txt* mostrado anterior é mostrado na Tabela 1.

Vocabulário	Lista de documentos
algoritmo	1 2
dados	2
estrutura	2
selecao	1 3
natural	3

Tabela 1: Resultado do processamento do arquivo *palavras_chave.txt*.

Observação: Para facilitar, considere que as palavras-chave dos documentos são constituídas por no máximo 20 caracteres. Além disso, os caracteres usados são apenas letras minúsculas e sem acento. Também, para facilitar, não será usado o arquivo *palavras_chave.txt*, ele será simulado por meio de entrada via terminal.

A inserção das palavras do arquivo *palavras_chave.txt* no índice invertido deverá ser implementada com a estrutura de *hash* com endereçamento aberto.

O seu programa deve ser adaptado para medir o desempenho, em termos do tempo de execução e do consumo de memória. Além disso, utilize contadores para determinar o número de comparações de chaves (colisões) para montar o índice invertido.

Após a construção do **índice invertido**, implemente um programa que permita que sejam realizadas consultas de um ou mais termos na coleção de documentos considerada. O objetivo é, dado uma consulta, retornar um vetor de documentos **ordenados lexicograficamente pelo nome**. Portanto, o programa além de realizar a operação de criar um **índice invertido** dado um arquivo de palavras-chave e deve também realizar as seguintes operações:

a) Realizar consultas com um ou mais termos: Deve retornar uma lista com os nomes dos documentos que contêm qualquer um dos termos.

b) Imprimir o índice invertido: Imprime as palavras, uma por linha e à frente de cada palavra, você deve imprimir a lista das ocorrências. Para cada elemento da lista, você deve imprimir o nome do documento (e não o identificador).

Imposições e comentários gerais

Neste trabalho, as seguintes regras devem ser seguidas:

- Seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada)
- Um grande número de *Warnings* ocasionará a redução na nota final.

O que deve ser entregue

- Código fonte do programa em C (**bem identado e comentado**).
- Documentação do trabalho (relatório[1]). A documentação deve conter:
  1. **Implementação:** descrição sobre a implementação do programa. Não faça "*print screens*" de telas. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Muito importante: os códigos utilizados na implementação devem ser inseridos na documentação.
  2. **Impressões gerais**: descreva o seu processo de implementação deste trabalho. Aponte coisas que gostou bem como aquelas que o desagradou. Avalie o que o motivou, conhecimentos que adquiriu, entre outros.
  3. **Análise**: deve ser feita uma análise dos resultados obtidos com este trabalho.
  4. **Conclusão**: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  5. **Formato:** PDF ou HTML.

Como deve ser feita a entrega

Verifique se seu programa compila e executa na linha de comando antes de efetuar a entrega. Quando o resultado for correto, entregue via *Moodle* até D, um arquivo **.ZIP** com o nome e sobrenome do aluno. Esse arquivo deve conter: (i) os arquivos *.c* e *.h* utilizados na implementação, (ii) instruções de como compilar e executar o programa no terminal, e (iii) o relatório em **PDF**.

Detalhes da implementação

Para atingir o seu objetivo, você deverá construir um Tipo Abstrato de Dados (TAD) `IndiceInvertido` como representação de um índice invertido implementado com *Hash* de endereçamento aberto.

O TAD *IndiceInvertido* deverá implementar, pelo menos, as seguintes operações:

1. `aloca`: aloca o TAD `IndiceInvertido`.
2. `libera`: libera o TAD `IndiceInvertido`.
3. `insereDocumento`: insere um documento baseado na chave no TAD `IndiceInvertido`.
4. `busca`: retorna o índice de uma chave no TAD `IndiceInvertido`.

[1]Exemplo de relatório: `https://www.overleaf.com/latex/templates/modelo-relatorio/vprmcsdgmcgd`.

5. `consulta`: baseado em uma ou mais chaves, retorna o nome dos documentos que contêm todas as chaves no índice invertido presente no TAD `IndiceInvertido`.

6. `imprime`: imprime o índice invertido presente no TAD `IndiceInvertido`.

Alocação de um ou mais TADs *IndiceInvertido* fica a critério do aluno.

O TAD deve ser implementado utilizando a separação interface no *.h* e implementação *.c* discutida em sala, bem como as convenções de tradução. Caso a operação possa dar errado, devem ser definidos retornos com erro, tratados no corpo principal.

Fica a seu critério o uso ou não de uma estrutura para armazenar o vocabulário e se serão usados os nomes dos arquivos ou identificadores numéricos dentro do **índice invertido**.

O código-fonte deve ser modularizado corretamente em cinco arquivos: *tp.c*, *indiceInvertido.h*, *indiceInvertido.c*, *hash.c* e *hash.c*. O arquivo *tp.c* deve apenas invocar, tratar as respostas das funções e procedimentos definidos no arquivo *indiceInvertido.h* e impressões necessárias. A separação das operações em funções e procedimentos está a cargo do aluno, porém, **não deve haver acúmulo** de operações dentro de uma mesma função/procedimento.

Os arquivos *hash.h* e *hash.c* possuem as definições das constantes, a função *hash* (`h`) e a função para pegar as palavras de cada documento na linha.

O limite de tempo para solução de cada caso de teste é de apenas **um segundo**. Além disso, o seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada). *Warnings* ocasionará a redução na nota final. Assim sendo, utilize suas habilidades de programação e de análise de algoritmos para desenvolver um algoritmo correto e rápido!

Entrada

A entrada é dada por:

- Número N de documentos que serão inseridos.
- Em seguida, serão lidos N documentos, onde cada linha possui:
  - Nome do arquivo.
  - Palavras do documento.
- Opção O do que fazer: “B” e “I”:
  - “B” buscar palavras no *índice invertido*, além de “B” é fornecido as palavras a serem buscadas.
  - “I” imprimir o *índice invertido*.

Restrições

O problema possui algumas restrições:

- O número de documentos (N) é no máximo 100.
- O número máximo de palavras buscadas é 100.
- O número máximo de palavras por documento é de 1000.
- O vocabulário é de no máximo 1000 palavras.
- As palavras tem no máximo 20 caracteres.
- O nome do documento tem no máximo 50 caracteres.

Não haverá remoção de palavras (chave) ou documentos da *hash*.

Saída

A saída deve ser de acordo com a opção fornecida. Se for busca, basta informar os documentos com todas as palavras de forma ordenada. Se por acaso nenhum dos documentos tenham todas as palavras informadas, imprima "none". Se for imprimir, imprima o índice invertido na ordem em que eles aparecem na *hash*.

A IMPRESSÃO DA RESPOSTA DA FUNÇÃO DE BUSCA DEVE SER ORDENADA LEXICOGRAFICAMENTE.

A IMPRESSÃO DO *ÍNDICE INVERTIDO* DEVE OCORRER NA ORDEM EM QUE AS CHAVES APARECEM NA *HASH*. A IMPRESSÃO DOS DOCUMENTOS DEVEM RESPEITAR A ORDEM DE INSERÇÃO DOS MESMOS.

Exemplo de casos de teste

Exemplo da saída esperada dada uma entrada:

Entrada	**Saída**
3 prog.doc algoritmo selecao aeds1.doc algoritmo estrutura dados darwin.doc selecao natural B algoritmo selecao	prog.doc

Entrada	**Saída**
3 prog.doc algoritmo selecao aeds1.doc algoritmo estrutura dados darwin.doc selecao natural B algoritmo natural	none

Entrada	**Saída**
3 prog.doc algoritmo selecao aeds1.doc algoritmo estrutura dados darwin.doc selecao natural B algoritmo	aeds1.doc prog.doc

Entrada	**Saída**
3 prog.doc algoritmo selecao aeds1.doc algoritmo estrutura dados darwin.doc selecao natural I	dados - aeds1.doc selecao - prog.doc darwin.doc natural - darwin.doc algoritmo - prog.doc aeds1.doc estrutura - aeds1.doc

A SAÍDA DA SUA IMPLEMENTAÇÃO DEVE SEGUIR EXATAMENTE A SAÍDA PROPOSTA.

Diretivas de Compilação

As seguintes diretivas de compilação devem ser usadas (essas são as mesmas usadas no run.codes).

```
$ gcc -c hash.c -Wall
$ gcc -c indiceInvertido.c -Wall
$ gcc -c tp.c -Wall
$ gcc hash.o indiceInvertido.o tp.o -o exe
```

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta *valgrind*. O *valgrind* é um *framework* de instrumentação para análise dinâmica de um código e é muito útil para resolver dois problemas em seus programas: **vazamento de memória e acesso a posições inválidas de memória** (o que pode levar a *segmentation fault*). Um exemplo de uso é:

```
1 gcc -g -o exe ordenacao.c tp.c -Wall
2 valgrind --leak-check=full -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
1 ==xxxxx== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.

PONTO EXTRA

Será concedido 2 pontos extras (nos dez do TP, ou seja, 0,2 nos dez do período) para quem realizar a mesma implementação utilizando uma *hash* com listas encadeadas e *hashing* duplo. **Para este caso, devem ser considerados somente os casos com a opção "B" e deve ser incluída uma análise sobre o número de colisões no relatório.**