**Pontifícia Universidade Católica de Minas Gerais**
**Curso de Ciência da Computação**
**Disciplina: Algoritmos e Estruturas de Dados II**
**Profs.: Felipe Domingos da Cunha, Max do Val Machado e Rodrigo Richard Gomes**

# Trabalho Prático II

## Regras Básicas

1. extends Trabalho Prático 01
2. Fique atento ao Charset dos arquivos de entrada e saída.

## Classe + Registro

Spotify é um serviço de streaming de música, podcast e vídeo que foi lançado oficialmente em 7 de outubro de 2008. É o serviço de streaming mais popular e usado do mundo. Ele é desenvolvido pela startup Spotify AB em Estocolmo, Suécia. Ele fornece conteúdo protegido de conteúdo provido de restrição de gestão de direitos digitais de gravadoras e empresas de mídia. O Spotify é um serviço freemium; com recursos básicos sendo gratuitos com propagandas ou limitações, enquanto recursos adicionais, como qualidade de transmissão aprimorada e downloads de música, são oferecidos para assinaturas pagas.

O Spotify está disponível na maior parte da Europa, parte da América, Austrália, Nova Zelândia e partes da Ásia. Está disponível para a maioria dos dispositivos modernos, incluindo computadores Windows, macOS e Linux, bem como smartphones e tablets com iOS, Windows Phone e Android. As músicas podem ser navegadas ou pesquisadas por artista, álbum, gênero, lista de reprodução ou gravadora. Usuários podem criar, editar ou compartilhar playlists, compartilhar faixas em redes sociais ou fazer playlists com outros usuários. O Spotify fornece acesso a mais de 30 milhões de músicas. Em julho de 2019, contava mais de 232 milhões de usuários ativos, incluindo 108 milhões de assinantes pagantes.

O arquivo **data.csv** contém mais de 175.000 músicas coletadas da plataforma Spotify Web API, que podem ser agrupados por artista, ano ou genero. Tal arquivo deve ser copiado para a pasta /tmp/. Quando reiniciamos o Linux, ele normalmente apaga os arquivos existentes na pasta /tmp/.

Implemente os itens pedidos a seguir.

1. **Classe em Java**: Crie uma classe *Música* seguindo todas as regras apresentadas no slide unidade001_nivelamento_introducao00.pdf.

   Sua classe terá os atributos privados `id`(String), `nome`(String), `key`(String), `artists`(List), `realease_date`(Date), `acousticness`(double), `danceability` (double), `energy`(double), `duration_ms`(int), `instrumentalness`(double), `valence`(double), `popularity`(int), `tempo`(float), `liveness`(double), `loudness`(double), `speechiness`(double), `year`(int). Sua classe também terá pelo menos dois construtores, e os métodos *gets*, *sets*, *clone*, *imprimir* e *ler*. O método *imprimir* mostra os atributos do registro (ver cada linha da saída padrão) e o *ler* lê os atributos de um registro. A entrada padrão é composta por várias linhas e cada uma contém uma string indicando o `id` do Música a ser lido. A última linha da entrada contém a palavra FIM. A saída padrão também contém várias linhas, uma para cada registro contido em uma linha da entrada padrão.

   A saída padrão deve obedecer o seguinte formato:
   `id ## artists ## name ## realease_date ## acousticness ## danceability ## instrumentalness ## liveness ## loudness ## speechiness ## energy ## duration`

   **Observação:** Para o campo `realease_date`, fica definido o valor padrão de 01 para a ausência de algum valor de mes ou dia.

2. **Registro em C**: Repita a anterior criando o registro *Música* na linguagem C.

## Pesquisa

3. **Pesquisa Sequencial em Java**: Faça a inserção de alguns registros no final de um vetor e, em seguida, faça algumas pesquisas sequenciais. A chave primária de pesquisa será o atributo **id**. A entrada padrão é composta por duas partes onde a primeira é igual a entrada da primeira questão. As demais linhas correspondem a segunda parte. A segunda parte é composta por várias linhas. Cada uma possui um elemento que deve ser pesquisado no vetor. A última linha terá a palavra FIM. A saída padrão será composta por várias linhas contendo as palavras SIM/NAO para indicar se existe cada um dos elementos pesquisados. Além disso, crie um arquivo de log na pasta corrente com o nome matrícula_sequencial.txt com uma única linha contendo sua matrícula, tempo de execução do seu algoritmo e número de comparações. Todas as informações do arquivo de log devem ser separadas por uma tabulação '\t'.

4. **Pesquisa Binária em C**: Repita a questão anterior, contudo, usando a Pesquisa Binária. A entrada e a saída padrão serão iguais as da questão anterior. O nome do arquivo de log será matrícula_binaria.txt. A entrada desta questão **não** está ordenada.

## Ordenação

**Observação: ATENÇÃO para os algoritmos de ordenação que já estão implementados no Github!**

5. **Ordenação por Seleção em Java**: Usando vetores, implemente o algoritmo de ordenação por seleção considerando que a chave de pesquisa é o atributo **nome**. A entrada e a saída padrão são iguais as da primeira questão, contudo, a saída corresponde aos registros ordenados. Além disso, crie um arquivo de log na pasta corrente com o nome matrícula_selecao.txt com uma única linha contendo sua matrícula, número de comparações (entre elementos do *array*), número de movimentações (entre elementos do *array*) e o tempo de execução do algoritmo de ordenação. Todas as informações do arquivo de log devem ser separadas por uma tabulação '\t'.

6. **Ordenação por Seleção Recursiva em C**: Repita a questão anterior, contudo, usando a Seleção Recursiva. A entrada e a saída padrão serão iguais as da questão anterior. O nome do arquivo de log será matrícula_selecaoRecursiva.txt.

7. **Ordenação por Inserção em Java**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo de Inserção, fazendo com que a chave de pesquisa seja o atributo **id**. O nome do arquivo de log será matrícula_insercao.txt.

   (Lembre-se: em caso de empate, o critério de ordenação deverá ser o nome do música)

8. **Shellsort em C**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Shellsort, fazendo com que a chave de pesquisa seja o atributo **id**. O nome do arquivo de log será matrícula_shellsort.txt.

9. **Heapsort em Java**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Heapsort, fazendo com que a chave de pesquisa seja o atributo **realease_date**. O nome do arquivo de log será matrícula_heapsort.txt.

10. **Quicksort em C**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Quicksort, fazendo com que a chave de pesquisa seja o atributo **duration**. O nome do arquivo de log será matrícula_quicksort.txt.

11. **Counting Sort em Java**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Counting Sort, fazendo com que a chave de pesquisa seja o atributo **popularity**. O nome do arquivo de log será matrícula_countingsort.txt.

12. **Bolha em C**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo da Bolha, fazendo com que a chave de pesquisa seja o atributo **danceability**. O nome do arquivo de log será matrícula_bolha.txt.

13. **Mergesort em Java**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Mergesort, fazendo com que a chave de pesquisa seja o atributo **energy**. O nome do arquivo de log será matrícula_mergesort.txt.

14. **Radixsort em C**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Radixsort, fazendo com que a chave de pesquisa seja o atributo **duration**. O nome do arquivo de log será matrícula_radixsort.txt.

15. **Ordenação PARCIAL por Seleção em Java**: Refaça a Questão "Ordenação por Seleção" considerando a ordenação parcial com k igual a 10.

16. **Ordenação PARCIAL por Inserção em C**: Refaça a Questão "Ordenação por Inserção" considerando a ordenação parcial com k igual a 10.

17. **Heapsort PARCIAL em C**: Refaça a Questão "Heapsort" considerando a ordenação parcial com k igual a 10.

18. **Quicksort PARCIAL em Java**: Refaça a Questão "Quicksort" considerando a ordenação parcial com k igual a 10.