Pontificia Universidade Católica de Minas Gerais
Curso de Ciência da Computação
Disciplina: Algoritmos e Estruturas de Dados II

Trabalho Prático II

Regras Básicas

1. extends Trabalho Prático 01

2. Fique atento ao Charset dos arquivos de entrada e saída.

Classe + Registro

O Disney+ é um serviço de streaming de vídeo lançado pela The Walt Disney Company em 12 de novembro de 2019. A plataforma oferece um catálogo extenso de filmes e séries das marcas da Disney, incluindo Marvel, Star Wars, Pixar, National Geographic e 20th Century Studios. Desde seu lançamento, tem se consolidado como uma das principais plataformas do mercado de streaming, competindo diretamente com serviços como Netflix, Amazon Prime Video, HBO Max e Apple TV+.

Um dos grandes diferenciais do Disney+ é a oferta de filmes e séries exclusivos, que não estão disponíveis em outras plataformas. Entre os destaques estão produções originais como The Mandalorian (Star Wars), WandaVision (Marvel), Loki, Encanto e muitas outras. Além disso, a plataforma mantém um extenso catálogo de clássicos da Disney, como O Rei Leão, A Bela e a Fera, Aladdin, entre outros, permitindo que diferentes gerações revisitem conteúdos icônicos da empresa.

Desde o seu lançamento, o Disney+ teve um crescimento impressionante, atingindo mais de 100 milhões de assinantes em menos de dois anos. A plataforma está disponível em diversas partes do mundo, com suporte a múltiplos idiomas e adaptações do catálogo conforme a região. Essa expansão rápida reflete a força das marcas que compõem o serviço e o interesse global pelo conteúdo oferecido.

O arquivo DISNEYPLUS.CSV contém um conjunto de dados extraídos do site Kaggle. Este conjunto de dados contém listagens de todos os filmes e programas de TV disponíveis, juntamente com detalhes como elenco, diretores, classificações, ano de lançamento, duração, entre outros. 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 SHOW seguindo todas as regras apresentadas no slide unidade00l_conceitosBasicos_introducaoOO.pdf. Sua classe terá os atributos privado SHOW_ID: `string`, TYPE: `string`, TITLE: `string`, DIRECTOR: `string`, CAST: `string[]`, COUNTRY: `string`, DATE_ADDED: `date`, RELEASE_YEAR: `int`, RATING: `string`, DURATION: `string`, LISTED_IN: `string[]`. 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. Atenção para o arquivo de entrada, pois em alguns registros faltam valores e esse deve ser substituído pelo valor `NaN`. A entrada padrão é composta por várias linhas e cada uma contém um número inteiro indicando o SHOW_ID do SHOW 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, no seguinte formato: `[=> id ## type ## title ## director ## [cast] ## country ## date_added ## release_year ## rating ## duration ## [listed_in]`.

   Exemplo:`=> s1367 ## TV Show ## Disney Mech-X4 ## NaN ## [Kamran Lucas, Nathaniel Potvin, Pearce Joza, Raymond Cham] ## Canada ## NaN ## 2016 ## TV-Y7 ## 2 Seasons ## [Action-Adventure, Comedy, Science Fiction ]##`

   OBSERVAÇÃO:

   - As variáveis do tipo Lista `(cast,listed_in)` deverão ter os seus itens também ORDENADOS.
   - Caso algum campo não tenha o seu valor informado, favor atribuir `NaN`.

2. **Registro em C**: Repita a anterior criando o registro SHOW 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 **title**. 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 **title**. 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 à 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 **type**. O nome do arquivo de log será matrícula_insercao.txt.

   (Lembre-se: em caso de empate, o critério de ordenação deverá ser o TITLE do SHOW.)

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 **type**. O nome do arquivo de log será matrícula_shellsort.txt. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o TITLE do SHOW.)

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 **director**. O nome do arquivo de

log será matrícula_heapsort.txt. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o TITLE do SHOW.)

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 **date_added**. O nome do arquivo de log será matrícula_quicksort.txt.

    (Lembre-se: em caso de empate, o critério de ordenação deverá ser o TITLE do SHOW.)

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 **release_year**. O nome do arquivo de log será matrícula_countingsort.txt. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o TITLE do SHOW.)

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 **date_added**. O nome do arquivo de log será matrícula_bolha.txt. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o TITLE do SHOW.)

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 **duration**. O nome do arquivo de log será matrícula_mergesort.txt. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o TITLE do SHOW.)

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 **release_year**. O nome do arquivo de log será matrícula_radixsort.txt. Além disso, forneça uma explicação detalhada sobre a aplicação do Radixsort nesse contexto e discuta a complexidade temporal do algoritmo em comparação com outros algoritmos de ordenação. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o TITLE do SHOW.)

15. **Ordenação PARCIAL por Seleção em Java**: Refaça a Questão "Ordenação por Seleção" considerando a ordenação parcial com k igual a 10. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o nome do SHOW.)

16. **Ordenação PARCIAL por Inserção em C**: Refaça a Questão "Ordenação por Inserção" considerando a ordenação parcial com k igual a 10. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o nome do SHOW.)

17. **Heapsort PARCIAL em C**: Refaça a Questão "Heapsort" considerando a ordenação parcial com k igual a 10. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o nome do SHOW.)

18. **Quicksort PARCIAL em Java**: Refaça a Questão "Quicksort" considerando a ordenação parcial com k igual a 10. (Lembre-se: em caso de empate, o critério de ordenação deverá ser o nome do SHOW.)

19. **Comparativo de Ordenação**: Você foi designado para realizar uma análise de desempenho de algoritmos clássicos de ordenação. A tarefa consiste em comparar os tempos de execução, o número de comparações e o número de movimentações realizadas pelos seguintes algoritmos de ordenação: **Ordenação por Seleção (Selection Sort), Ordenação por Inserção (Insertion Sort), Ordenação por Bolha (Bubble Sort) e Quicksort**.

    A partir de uma vetor de números inteiros aleatórios, execute os quatro algoritmos para entradas de diferentes tamanhos: 100, 1000, 10000 E 100000 números aleatórios. Para cada execução, registre o tempo de execução, o número de comparações e o número de movimentações realizadas pelos algoritmos.

    Primeiramente, você deverá implementar cada um dos algoritmos de ordenação mencionados. Em seguida, gere quatro vetores diferentes com 100, 1000, 10000 E 100000 números inteiros aleatórios e execute cada algoritmo nas respectivas vetores. Durante a execução, registre o tempo de execução (em milissegundos), o número de comparações realizadas e o número de movimentações realizadas durante o processo de ordenação.

    Após realizar as execuções, construa gráficos que comparem o tempo de execução de cada algoritmo para as diferentes entradas, o número de comparações realizadas e o número de movimentações realizadas por cada algoritmo. Utilize a biblioteca `matplotlib` (ou outra similar) para gerar os gráficos.

    Por fim, elabore um relatório em LaTeX com a análise crítica dos resultados obtidos. Use como modelo o template overleaf da SBC [1]. Comente sobre o comportamento de cada algoritmo conforme o tamanho da entrada aumenta. Discuta quais algoritmos são mais eficientes em termos de tempo, comparações e movimentações, e explique as razões pelas quais certos algoritmos se comportam de maneira diferente com vetores maiores e menores.

    A ENTREGA DESTA QUESTÃO SERÁ FEITA NO CANVAS PELA TURMA TEÓRICA.

[1] Link para o template: https://www.overleaf.com/latex/templates/sbc-conferences-template/blbxwjwzdngr