1. Trabalho prático - Etapa 1 - Listas encadeadas e Análise de Complexidade

Nesta etapa vamos implementar bibliotecas de listas ordenadas e não-ordenadas e analisar as ordens de complexidade e os tempos de execução dos métodos de inserção e de pesquisa em ambas estruturas.
Esta etapa é composta por duas partes principais: a implementação e a elaboração do relatório.

1.1. Implementação

1.1.1. Implementação das bibliotecas de Listas

- Devem ser implementadas duas bibliotecas:
  - Lista não ordenada
  - Lista ordenada.
- Ambas devem usar Generics de forma que na instanciação da lista seja possível definir o tipo (T) dos objetos a serem armazenados na lista.
- Ambas bibliotecas devem prover 3 métodos, a saber:
  - Um método construtor que instancie e inicialize uma lista vazia.
  - Um método `void adicionar(T novoValor)` que receba um elemento do tipo T e o adicione na lista;
  - Um método `T pesquisar(T valor)` que receba um elemento do tipo T contendo o valor a ser buscado na lista e o busque o valor na lista retornando o objeto encontrado ou null caso não encontre um objeto com a chave dada.
- A lista ordenada deve fazer uso de um `Comparator<T>` para realizar as comparações nos métodos `void adicionar(T novoValor)` e `T pesquisar(T valor)`. Assim, o método construtor da lista ordenada deve receber o `Comparator<T>` como parâmetro e o armazenar para uso nos métodos.
- O método `void adicionar(T novoValor)` da lista não ordenada pode inserir o novo elemento na lista na posição que o desenvolvedor julgar mais conveniente.
- O método `void adicionar(T novoValor)` da lista ordenada deve inserir o novo elemento na lista, a mantendo sempre ordenada de acordo com o critério definido no comparador.
- O método `T pesquisar(T valor)` de lista ordenada deve fazer uso do fato de que a lista está ordenada para, sempre que for possível, não buscar por toda a lista.

1.1.2. Implementação do programa de teste

Para elaborar o relatório será necessário executar alguns testes nas suas bibliotecas. Para isso vamos desenvolver um programa de testes.

- O programa de testes processará dados de Alunos. Assim, você precisará desenvolver uma classe Aluno, que tenha os atributos matricula (int), nome (String) e nota (float).

- A ideia é que a lista ordenada mantenha os alunos ordenados por matrícula. Assim, você precisará desenvolver um `Comparator<Aluno>` que use a matrícula como critério de comparação de alunos.
- Seu programa de testes deve ler um arquivo texto com o seguinte formato:
    - A primeira linha terá apenas um número inteiro que indicará a quantidade de registros de alunos que há no arquivo;
    - Da segunda linha em diante, cada linha trará a matrícula, o nome e a nota de um aluno, nesta ordem, separados por ponto-e-vírgula(;)
    - Para gerar arquivos de entrada você pode utilizar o "GeradorArquivosOrdenados.java" disponível em https://github.com/victoriocarvalho/GeradorArquivos. Neste mesmo repositório pode ser encontrado um exemplo de leitor de arquivo. O objetivo deste leitor é só exemplificar a leitura de arquivos. Note que ele não gera instâncias de aluno; ele apenas imprime os dados na tela.
- Ao iniciar, seu programa deve criar duas listas, uma ordenada e outra não ordenada, e ler o arquivo de entrada. Para cada registro de aluno lido do arquivo, o programa deve instanciar um novo aluno e adicionar a instância às duas listas. Assim, ao final da leitura do arquivo seu programa terá em memória uma lista ordenada e uma não ordenada, ambas contendo os registros de todos os alunos. Ao finalizar a leitura do arquivo e a criação das listas seu programa deve exibir na tela o tempo gasto neste processo (pesquise como medir o tempo de execução em Java).
- Então seu programa deve exibir um menu com três opções:
    - **Pesquisar na lista não ordenada**: nesta opção o programa deve solicitar a matrícula do aluno a ser pesquisado, chamar o método "pesquisar" da lista não ordenada e imprimir os dados do aluno retornado ou uma mensagem de erro caso o retorno seja NULL. O programa deve imprimir na tela o tempo gasto entre a chamada e o retorno do método pesquisar (pesquise como medir o tempo de execução em Java). Esta informação será necessária para a elaboração do relatório.
    - **Pesquisar na lista ordenada:** nesta opção o programa deve solicitar a matrícula do aluno a ser pesquisado, chamar o método "pesquisar" da lista ordenada e imprimir os dados do aluno retornado ou uma mensagem de erro caso o retorno seja NULL. O programa deve imprimir na tela o tempo gasto entre a chamada e o retorno do método pesquisar. Esta informação será necessária para a elaboração do relatório.
    - **Sair:** A execução do programa é finalizada.
- Organize seu código de forma que a biblioteca fique em uma pasta enquanto o programa de teste, a classe Aluno e os Comparators fiquem em outra.

1.2. Elaboração do Relatório

Elabore um relatório respondendo às seguintes perguntas:

1. Qual a ordem de complexidade, no pior caso, do método de inserir de lista não-ordenada? Em quais condições o pior caso ocorre? Faça a análise do código que você implementou.

2. Qual a ordem de complexidade, no pior caso, do método de pesquisar de lista não-ordenada? Em quais condições o pior caso ocorre? Faça a análise do código que você implementou.
3. Faça pesquisas, pelo pior caso, em listas não ordenadas de tamanhos variados e tome nota dos tempos de execução das buscas. Faça um gráfico da variação do tempo de execução pela variação do tamanho da entrada. Comente se a variação do tempo pela variação do tamanho de entrada correspondeu ao que era esperado considerando a ordem de complexidade definida na análise do exercício anterior. Obs.1: Como o tempo de busca em uma mesma lista pode variar devido a fatores externos, rode a pesquisa ao menos 3 vezes para cada lista e use a média dos tempos para essa análise.
4. Obs.2: Varie o tamanho do arquivo de entrada de forma que seja perceptível a variação no tempo de execução das buscas.
5. Qual a ordem de complexidade, no pior caso, do método de inserir de lista ordenada? Em quais condições o pior caso ocorre? Faça a análise do código que você implementou.
6. Qual a ordem de complexidade, no pior caso, do método de pesquisar na lista não-ordenada? Em quais condições o pior caso ocorre? Faça a análise do código que você implementou.
7. Faça pesquisas, pelo pior caso, em listas ordenadas de tamanhos variados e tome nota dos tempos de execução das buscas. Faça um gráfico da variação do tempo de execução pela variação do tamanho da entrada. Comente se a variação do tempo pela variação do tamanho de entrada correspondeu ao que era esperado considerando a ordem de complexidade definida na análise do exercício anterior. Obs.1: Como o tempo de busca em uma mesma lista pode variar devido a fatores externos, rode a pesquisa ao menos 3 vezes para cada lista e use a média dos tempos para essa análise.
8. Obs.2: Varie o tamanho do arquivo de entrada de forma que seja perceptível a variação no tempo de execução das buscas.
9. Como foi a variação do tempo de leitura e construção das listas em função da variação do tamanho do arquivo de entrada? Faça um gráfico do tempo de execução em função do tamanho da entrada. Comente se essa variação ficou dentro do esperado, considerando as ordens de complexidade dos métodos de inserção das listas.

   Finalize o relatório indicando a URL do repositório do github em que seu trabalho está disponível.

Regras Gerais

- Os trabalhos devem ser desenvolvidos em grupos de até 4 pessoas. Como os trabalhos são sequenciais e incrementais, os grupos não devem sofrer alterações durante o semestre.

- Os trabalhos serão avaliados considerando:
    - ***Completude do código***: todos os requisitos foram atendidos?
    - ***Corretude do código***: tudo funciona como deveria?
    - ***Qualidade do código***: serão observados critérios de qualidade como modularidade (o código está bem encapsulado em classes/métodos?), eficiência (foram observadas as características das estruturas para otimizar o desempenho?) , comentários (o raciocínio seguido está documentado em comentários?).
    - ***Qualidade do relatório***: o relatório responde a todas as questões solicitadas de forma clara e correta?
    - ***Desempenho na arguição oral ou escrita:*** o professor pode solicitar que o grupo ou determinados componentes apresentem o trabalho, ou ainda, poderá optar por fazer uma arguição escrita.
    - ***Pontualidade:*** O trabalho foi entregue dentro do prazo?

- Os códigos fontes dos trabalhos devem ser armazenados em repositórios do Github e os endereços dos repositórios indicados nos relatórios de entrega.
- Todas as entregas de relatórios serão feitas pelo AVA. Apenas um componente de cada grupo deve enviar o relatório, onde deve constar os nomes de todos os componentes do grupo.
- O relatório deve ser enviado em formato .pdf.
- O prazo para entrega do relatório está definido na respectiva atividade no Ava.
- Esta primeira etapa do trabalho prático valerá 10 pontos, sendo 5 para a implementação e 5 para o relatório. As arguições orais/escritas influenciam a nota como um todo.