

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

DIM0124 - PROGRAMAÇÃO CONCORRENTE

Relatório de Desenvolvimento de Aplicação Concorrente

KNN

Matrícula: 20180063677

Nome: Gabriel Martins Spínola

1. **Introdução**

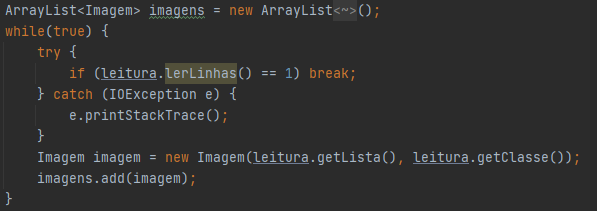
<Incluir uma breve introdução sobre o algoritmo implementado e resultados obtidos>

O algoritmo implementado foi o algoritmo do KNN, que tem o intuito de checar se existem ou não pessoas nas fotos que serão analisadas, para isso temos um arquivo em CSV contendo metadados de fotos com e sem pessoas para que possamos achar as fotos mais próximas quando processarmos.

Como resultado obtido, o programa imprime na tela se há ou não pessoas na foto a cada imagem que for processada. Com um dataset de 1gb o programa roda serialmente em 10 segundos em média, já com threads o cenário melhora, com semáforos e utilizando 5 threads a média cai para 7 segundos, com visibilidade e variáveis atômicas e utilizando também 5 threads a média continua em 7 segundos.

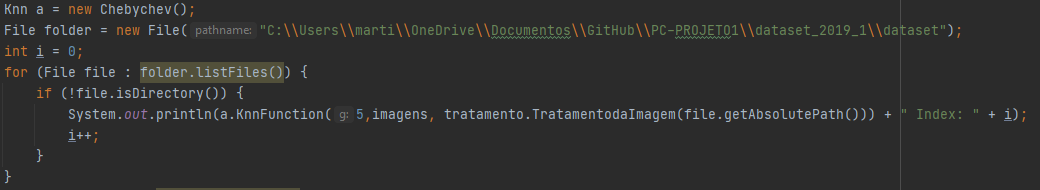
1. **Implementação Serial** 
   1. Descrição

O algoritmo começa lendo um dataset em csv que possui vários metadados de algumas imagens que tenha ou não pessoas. Esse dataset é usado para calcular os vizinhos mais próximos da imagem que será processada posteriormente. Cada linha do csv é transformado em um objeto do tipo Imagem e armazenado em uma lista.



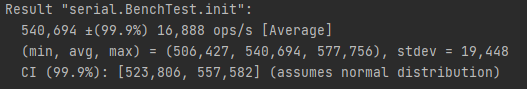
Após isso, criamos um objeto do tipo Chebchev (classe que tem o método que executa o calculo de quals imagens são mais próximas) e percorremos a pasta do dataset executando a função de calculo para

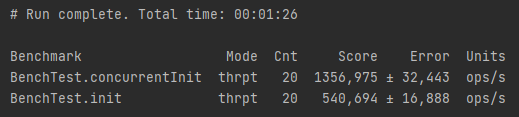
cada imagem presente no dataset e imprimindo o resultado.



* 1. Avaliação com Microbenchmark

Foi realizado o teste de benchmark com o jmh com as funções init e concurrent init que foram disponibilizadas no sigaa. A execução do teste retornou o seguinte resultado:





O método concurrentInit foi executado 1356 vezes por segundo, com um intervalo de confiança de 99,9%, também foi possível notar que o método apresentou taxa de erro de 32 operações por segundo.

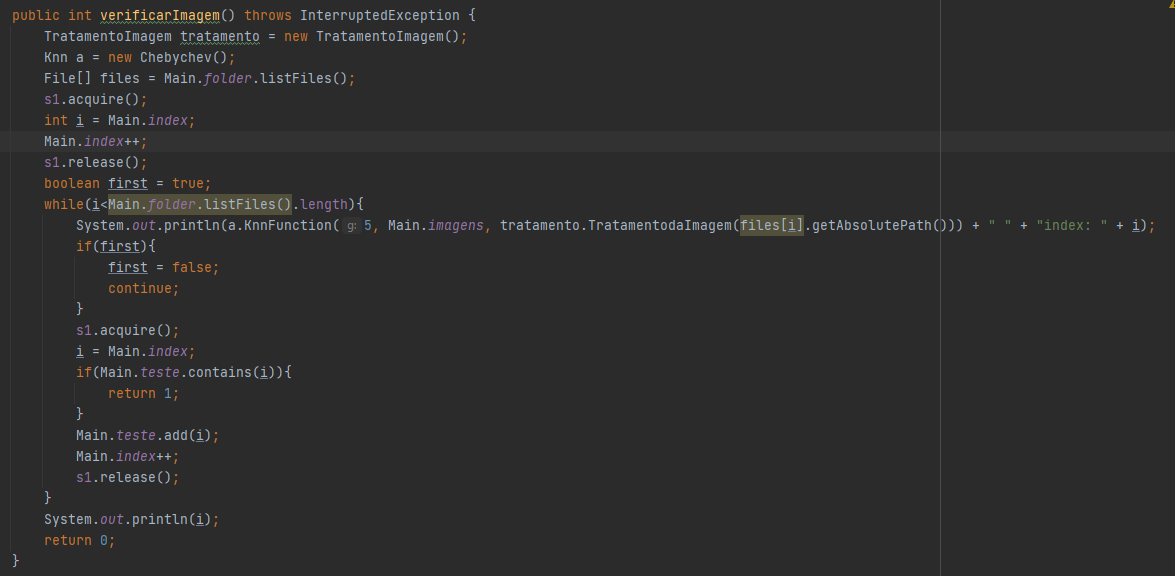
* 1. Avaliação com Macrobenchmark

<Mostrar o setup do teste do Jmeter. Inserir resultados dos testes com o Jmeter e discutir os resultados. Inserir gráfico com Número de Threads vs Throughput e discutir os resultados. **Observação: lembrar de realizar os testes de JMeter utilizando três GCs: G1, ZGC e Shenandoah. Aproveitar este momento para usar o JFR para fazer o profile das aplicações**>

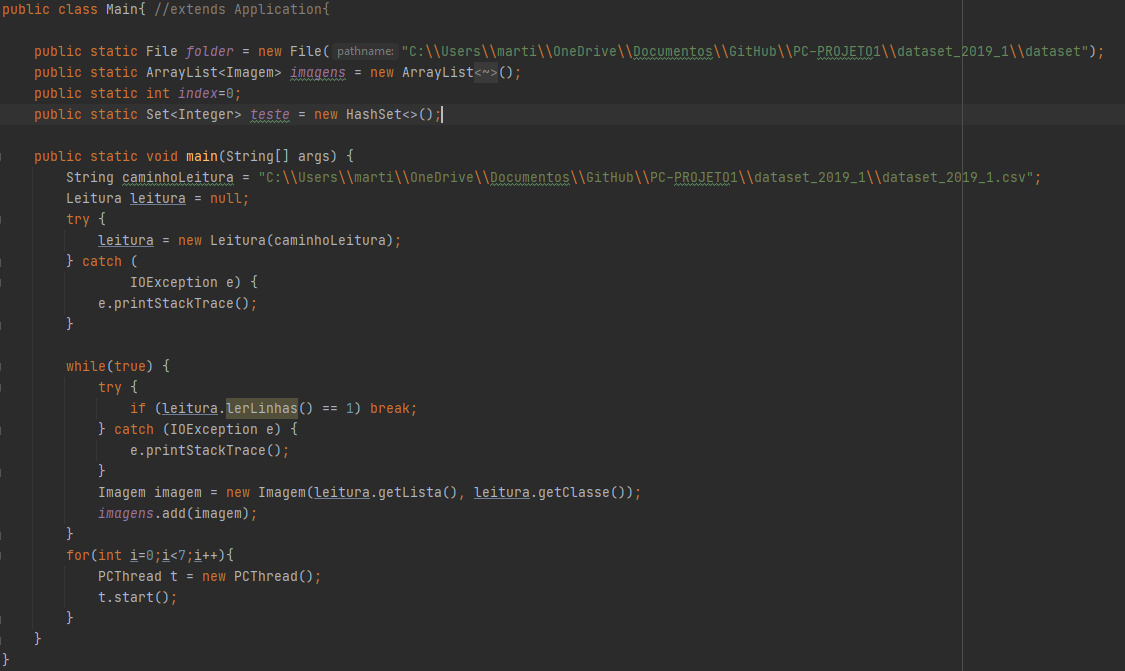
* 1. Avaliação de Profile

<Descrever uma análise do profile com o JFR e JMC. Mostrar dados de Hot methods, threads, contenção e dados do GC. Apresentar e discutir eventuais gargalos>

1. **Descrição da Implementação Concorrente - Abordagem Mutex/Semáforo (Vale 4.0)**
   1. Descrição

A primeira abordagem concorrente foi realizada utilizando semáforo, onde o semáforo controla o acesso a região critica que é o arquivo(imagem) que vai ser processado naquele momento pela thread, cada thread precisa percorrer a pasta do dataset processando as imagens, para isso, quando uma thread acessa um arquivo através de um indice, esse indice é bloqueado e então o algoritmo armazena o valor do indice e depois incrementa para poder liberar o acesso.

O indice é uma variável estática que está instanciada na classe Main



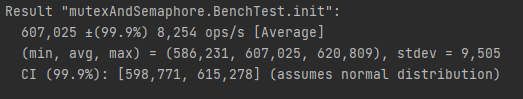
* 1. Avaliação de Condição de Corrida

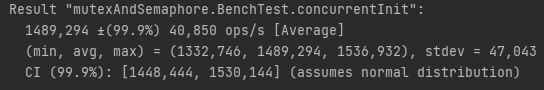
<Inserir setup dos testes com JCStress, descrever as regiões críticas do algoritmo e mostrar através de testes do JCStress que tais regiões são thread-safe. Mostrar tabela de resultados>

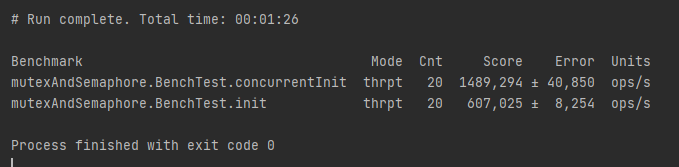
* 1. Avaliação com Microbenchmark

<Descrever setup dos testes. Inserir resultados dos testes com o JMH e discutir os **resultados comparando** com os resultados das soluções anteriores. Mostrar tabela de resultados e comparação via tabela>

Assim como a implementação serial, o teste foi realizado com os métodos concurrency init e init, e apresentaram os seguintes resultados:







Vemos então que a implementação concorrente teve 1489 operações por segundo com 99% de confiança com variação entre 1448 e 1530 vezes por segundos.

* 1. Avaliação com Macrobenchmark

<Mostrar o setup do teste do Jmeter. Inserir resultados dos testes com o Jmeter e discutir os resultados. Inserir gráfico com Número de Threads vs Throughput e discutir os resultados. Incluir comparação no gráfico com as abordagens anteriores. **Observação: lembrar de realizar os testes de JMeter utilizando três GCs: G1, ZGC e Shenandoah. Aproveitar este momento para usar o JFR para fazer o profile das aplicações**>

* 1. Avaliação de Profile

<Descrever uma análise do profile com o JFR/JMC e discutir os resultados comparando com os resultados das soluções anteriores. Mostrar dados de Hot methods, threads, contenção e dados do GC. Apresentar e discutir eventuais gargalos>

1. **Descrição da Implementação Concorrente - Abordagem Atomic/Volátil (4.0)**
   1. Descrição

<Descrever através de um diagrama de classes e trechos de código a implementação concorrente. Enfatizar qual parte do código e qual o percentual de código concorrente>

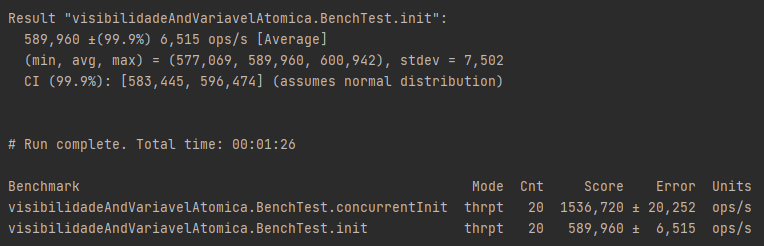
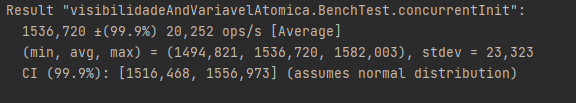
* 1. Avaliação de Condição de Corrida

<Descrever as regiões críticas do algoritmo e mostrar através de testes do JCStress que tais regiões são thread-safe. Mostrar tabela de resultados>

* 1. Avaliação com Microbenchmark

<Inserir resultados dos testes com o JMH e discutir os resultados comparando com os resultados das soluções anteriores. Mostrar tabela de resultados>

Por fim temos o teste com o JMH da implementação concorrente através de visibilidade e variáveis atômicas:



Onde podemos observar que foram realizadas 1536 operações por segundo, superando as 1489 da implementação anterior, a variação esteve ente 1516 a 1556 vezes por segundo e 20 operações de erro por segundo, bem menos que as 40 operações de erro da implementação anterior.

* 1. Avaliação com Macrobenchmark

<Inserir resultados dos testes com o Jmeter e discutir os resultados comparando com os resultados das soluções anteriores. Mostrar resultados de Throughput. **Observação: lembrar de realizar os testes de JMeter utilizando três GCs: G1, ZGC e Shenandoah. Aproveitar este momento para usar o JFR para fazer o profile das aplicações**>

* 1. Avaliação de Profile

<Descrever uma análise do profile com o JFR/JMC e discutir os resultados comparando com os resultados das soluções anteriores. Mostrar dados de Hot methods, threads, contenção e dados do GC. Apresentar e discutir eventuais gargalos>

1. **Discussão (Vale 1.0)**

<Comparar os resultados e descrever as lições aprendidas com os experimentos >