**Instituto Federal de Minas Gerais**

**Nome:** Marcus Vinicius Ribeiro Andrade.

**Matrícula:** 0064013.

**Tema:** Documentação do trabalho referente a programação paralela.

**Campus:** Ouro Branco, MG.

**Professor:** Saulo Henrique Cabral Silva.

**Período:** Terceiro**.**

**Turno:** Noturno.

**Introdução**

O processamento paralelo, cada vez mais, vem sendo visto como o único meio para a rápida solução de problemas computacionalmente grandes e intensivos GRAMA et al. (2003).Em decorrência disso, e com a redução no valor dos computadores, de modo geral, e, mais ainda, devido a muitas linguagens de programação hodiernas possuírem condições para que se possa trabalhar de forma paralela, é válido dizer que o uso de concorrência e paralelismo é cada vez mais apreciado, principalmente, por áreas recentes e que são, sobretudo, emergentes no mundo da tecnologia de informação.A saber, as áreas que usam, demasiadamente, programação paralela,são, por exemplo,o caso dos aplicativos que fazem processamento de transações que envolvem muita informação, a mineração de dados, os servidores web, os sistemas gerenciadores de bancos de dados (SGBDs),as inteligências artificiais, a nanotecnologia e muitas outras, que, em comum, partilham da realidade de se trabalhar com uma quantidade massiva de dados e executar, evidentemente, algoritmos bastante complexos.

Logo, diante do exposto, o presente trabalho visa esclarecer e implementar algoritmos de resolução de problemas que, por demais, podem ser bastante onerosos, no que tange ao tempo de execução e processamento de dados.As implementações, escritas na linguagem de programação Java, que, aqui, serão apresentadas ,visam usar o máximo do potencial computacional hospedeiro, isto é, objetivam evitar que o processador fique ocioso mediante a trabalhos pendentes.

Em detrimento disso, é imperioso ressaltar que para atingir o paralelismo nas atividades ,a seguir, conta-se apenas com o uso de threads, mais especificamente, multithreads, herdadas da classe Thread do java.E, primeiramente,antes de descrever, de fato, a implementação, torna-se importante, definir, de forma breve, o que são threads e seu papel no paralelismo.

Então, para que fiquem evidentes alguns conceitos, pode-se caracterizar programação paralela como a divisão da aplicação em partes e a execução simultânea dessas partes, por elementos de processamento, sendo esses, as threads, por exemplo, que são, por definição, um único fluxo do programa GRAMA et al. (2003), elas permitem a execução do programa fora da thread principal, main thread, que é criada ao executar o Java, por exemplo.Essas threads criadas, quando mais de uma, multithreads, permitem a execução paralela do sistema, o que, por sua vez, resulta em inúmeras vantagens e aplicações, como supracitado.

**Implementação Do Primeiro Programa:**

O primeiro programa visa encontrar o maior número primo, pela definição, um número é considerado primo quando é divisível por si próprio e , redundantemente, por um, contido em um arquivo de texto (extensão .txt).Logo, é claro que se trata, não apenas de um problema matemático , mas também, de uma questão sobre manipulação de arquivos. Assim, seguindo o conceito “dividir para conquistar”, tornou-se essencial racionar os dados para obter o melhor processamento paralelo.Em vista disso, como foi realizado a execução em uma máquina de seis núcleos (cores) e 2.90GHz de clock, dividiu-se todos os caminhos dos diretórios, pastas, de forma igualitária, para as threads criadas, no caso, seis, consoante a quantidade de núcleos do computador.

A organização do programa, vale ressaltar, espelha-se um pouco no modelo MVC (Model, Views, Controllers),mas, de forma simples, adaptada para ter um diretório controlador, um de dados, onde serão manipulados locais dos arquivos e , fora desses diretórios, a classe principal, Main.Diante disso, é válido dizer que o diretório Controllers possuí a parte lógica da aplicação e faz, inclusive, a aplicação das threads, o diretório Data contém a manipulação dos arquivos, no que tange ao path(caminho para o arquivo), e a classe Main invoca os métodos necessário para resolver os problemas.

Para uma visão mais abrangente da implementação e das estruturas abstratas de dados utilizadas, o diagrama abaixo ilustra a aplicação , como um todo.

AQUI VEM A ESTRUTURA DE DADOS

1. MAIN
2. CONTROLLER FILTER
3. CONTROLLER PARSER
4. CONTROLLER THREADS

Agora, consoante as estruturas de dados built-in (estruturas já existentes na linguagem), foram utilizadas, principalmente,os métodos da classe File, para manipulação de arquivos, variáveis dos tipos inteiro, string, double e booleana, ArrayList, Vector, List, métodos da classe Path e expressões regulares (regex).

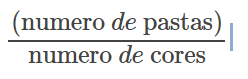
Para explicitar bem o funcionamento de cada método, abaixo está a descrição de cada um dos métodos implementados para resolução do problema.

**Classe Main e seus métodos:**

* **main(String[] args)**

Este método é padrão na linguagem java, é o chamado método principal e, no contexto do algoritmo, ele visa criar as instâncias (objetos) da classe ThreadsController para a criação de multithreads e, ainda, realizar a distribuição racionalizada dos dados, que , no caso, como supracitado, são os diretórios.

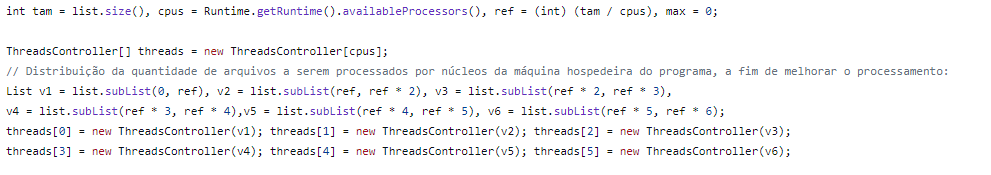
Essa divisão é feita quantitativamente, ou seja, dado um número x de núcleos,



no dispositivo, é feito a divisão de

Dessa maneira, pode-se ocupar o processador de forma equivalente e garantir que o mesmo não fique ocioso, enquanto a tarefa não terminar.Como consequência, têm-se também, o aumento expressivo do desempenho, ou seja, ocorre a redução do tempo de forma considerável.Contudo, o uso do processador torna-se exponencial, inviabilizando, durante o processo, a utilização de tarefas muito pesadas.

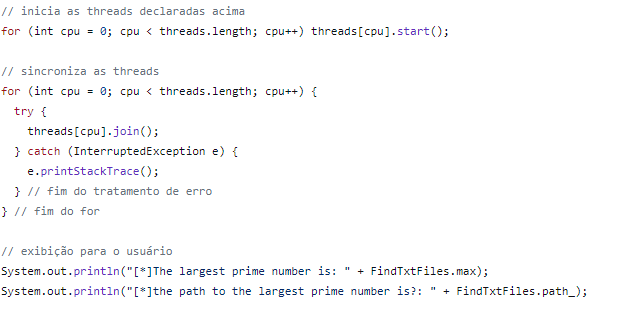
**Distribuição das pastas com os arquivos ,no método main , conforme a quantidade de núcleos**



Feito essas divisões, é chamado os métodos **start()** que dispara a execução da thread, mas, chamando o método run() - que logo será explicado, antes.

E também, é feito um loop, chamando o método **join(),** de cada thread que foi iniciada, para que se espere a thread terminar a execução, mantendo, desse modo, a sincronização.

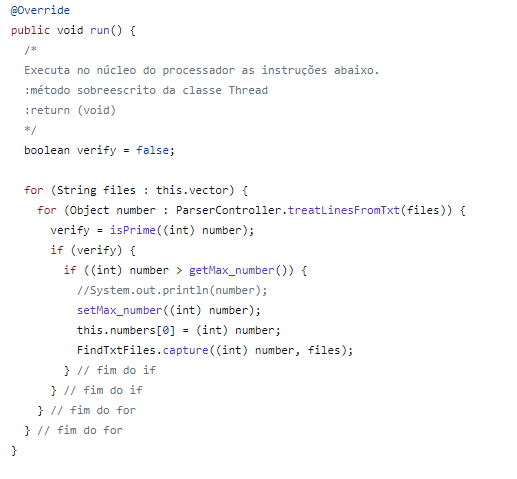
Vale a atenção de que, todos os demais programas, baseiam-se na mesma estrutura, no que tange a utilização das threads, ou seja, sempre se chamará os métodos acima de forma semelhante, como se segue na imagem abaixo:

**Uso comum de multithreads e exibição dos resultados**

**Classe ThreadsController e seus métodos:**

Explicado o método principal do programa, na classe ThreadsController, classe que herda os métodos da classe Thread do java, há a sobreescrita (@override) do método **run()**, que não recebe parâmetro e tem retorno do tipo void.Sumariamente, ela consiste dois loops, como os que percorrem matrizes,mas, no segundo, há a chamada do método estático da classe ParserController e a checagem se o número é primo ou não pelo método isPrime, da classe vigente.

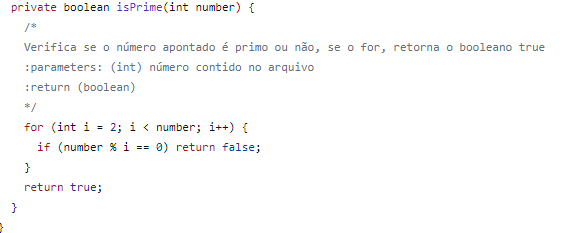
**Método run, sobreescrito, que é chamado pelo método start na classe Main**



* **isPrime(int number)**

Este método verifica se o número inteiro passado como parâmetro é primo ou não, retornando, desse modo, um valor booleano que indica se o é.

Sua construção é bem simples e visa, sobretudo, indicar se o número possui algum divisor, além disso, o método é privado e ,portanto, está disponível somente nesta classe:



**Método isPrime verifica se o inteiro passado é primo ou não, retornando um boolean correspondente**

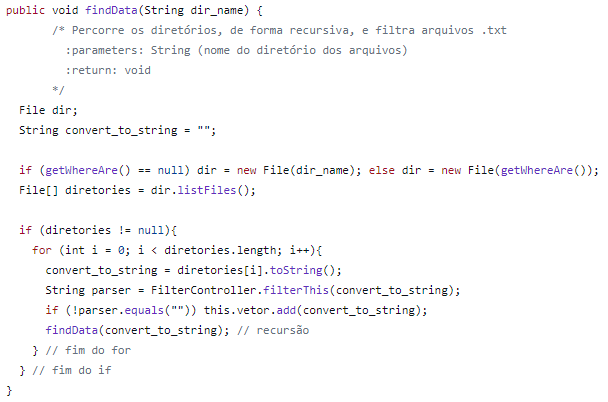
**Classe FindTxtFiles e seus métodos:**

Esta classe percorre o diretório indicado e seus sub-diretórios, de forma recursiva, pois, é de fácil percepção que escrever este método na forma recursiva permite, não apenas redução de código, mas também, ganho no desempenho.

* **findData(String dir\_name)**

O presente método recebe o nome do diretório como parâmetro e tem retorno do tipo void.Mais ainda, enquanto percorre os diretórios ,recursivamente, o caminho de cada arquivo passa pelo método **filterThis()** da classe FilterController, que logo será explicado,e,após serem feitas as modificações necessárias, é adicionado ao vetor do tipo de dado Vector.

Abaixo, está a imagem da implementação do método, na última linha, ele chama a si próprio, criando um ciclo recursivo, até o término da leitura dos arquivos.



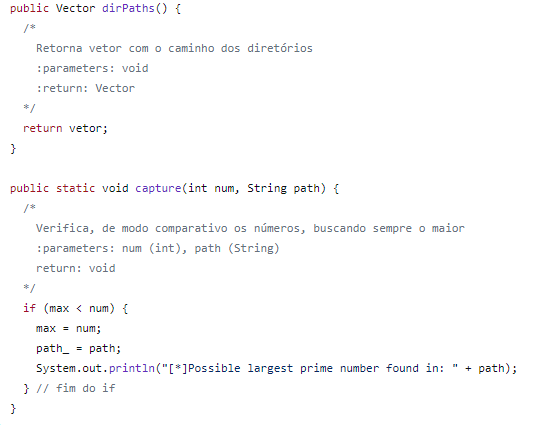
**Método findData que pesquisa arquivos recursivamente**

* **dirPaths()**

Método que retorna um vetor, do tipo Vector, e não recebe parâmetro.Este visa retornar o vetor com os caminhos do diretório.

* **capture(int num, String path)**

Método sem retorno que recebe um número inteiro, o número primo, e uma string com o path do mesmo.Este visa, exclusivamente, determinar o caminho e o maior número primo encontrado.Ou seja, é o método que retorna uma resposta para o problema.



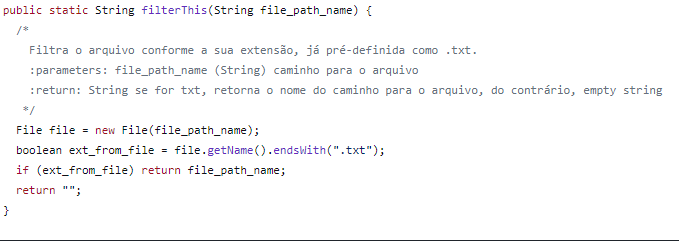
**Classe FilterController e seus métodos:**

Classe que contém o método responsável por considerar apenas arquivos do tipo texto (.txt) como proposto.

* **filterThis(String file\_path\_name)**

Este método é responsável pela separação dos arquivos de diferentes tipos, no repositório, para considerar, apenas, arquivos do tipo texto.Recebe, portanto, como parâmetro, o path do arquivo e analisa, por meio de manipulação de strings, quais arquivos terminam com .txt, retornando os que responde, afirmativamente, a questão.

**Método filterThis seleciona arquivos .txt**



**Classe ParseController e seus métodos:**

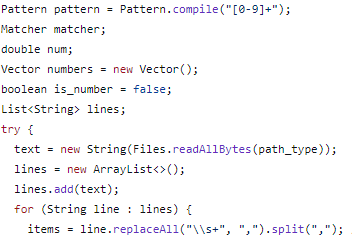
Finalmente, eis uma das classes mais importantes do programa, a classe parse contém o método responsável por analisar, minuciosamente, cada linha dos arquivos texto passados para ele, a fim de identificar números, para isso, é feito várias tratativas nos arquivos.

* **treatLinesFromTxt(String path\_and\_file)**

O método treatLinesFromTxt trata as linhas do arquivo, ou seja, captura apenas dados de interesse, isto é, todos os números para analisar se são primos e se forem, qual o maior.

O método , primeiramente, separa os itens na linha por vírgula e substitui espaços, entre os conteúdos, por vírgulas também para capturar como um vetor, cada item e , assim, melhor analisá-los; Para tal análise, é utilizado uma expressão regular **[0-9]+** esta expressão considera números de 0 a 9, ou seja, qualquer número em uma expressão que contenha textos, desse modo, fica mais favorável encontrar os algarismos que formam o maior número primo.

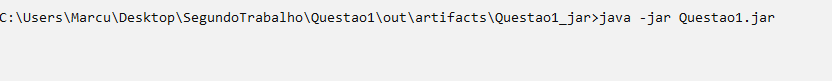
Um fato observado, e também bastante interessante, é que nesse método, é utilizado um vetor para armazenar cada linha do arquivo durante a leitura do mesmo.Contudo, é necessário salientar que este vetor deverá ser limpo, com o método **clear()**, pois, do contrário, em poucos segundos, acarretará em um erro de bufferOverFlow na memória ram da máquina, tornando o sistema inoperável.



**Parte do código que exibe o uso das expressões regulares e da separação dos arquivos por espaços e vírgulas**

**Utilização Do Primeiro Programa:**

Após discorrer sobre a implementação do programa, é necessário, também, informar seu funcionamento.Para isso, basta executar o arquivo com a extensão .jar, que é um Java Archive, ou seja, uma compilação do java,como segue na imagem:

****

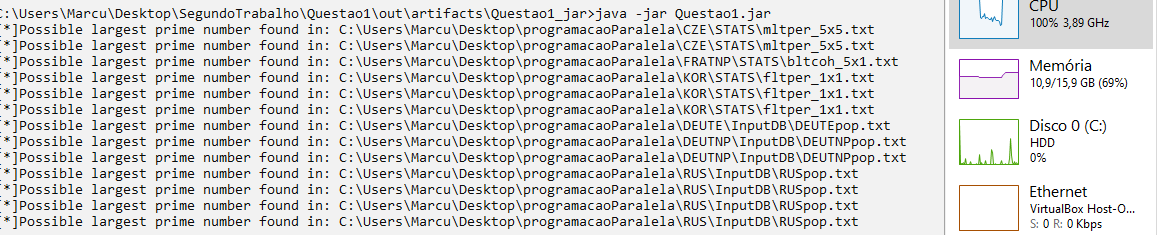
**No terminal, basta fazer o comando java -jar [nome do arquivo.jar] para que ocorra a execução.**

**Execução E Testes Do Primeiro Programa:**

Com a utilização da programação paralela, a execução do programa não é tão dispendiosa, pois, por volta de **um minuto e meio ,** o programa termina sua execução com sucesso.Um pouco mais,salienta-se que as configurações da máquina utilizada na execução, já foram supra-mencionadas.

Para confirmar o resultado, além de conversa com os colegas, foi feito um script na linguagem de programação python que obteve o mesmo resultado.O mesmo pode ser acessado em: **https://github.com/PyMarcus/FindPrimeNumber**

Abaixo, está o resultado da execução do teste no terminal windows:



**Imagem que exibe durante o processo, vale notar, o aproveitamento total de processamento.**

**resultadoNumeroPrimo**

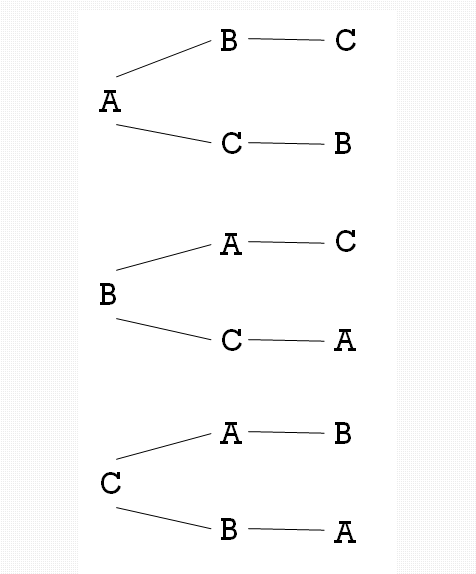
**Imagem que exibe o resultado, ao término do processo**

**Implementação Do Segundo Programa:**

O segundo programa visa encontrar a senha de um arquivo zip.Para isso, é necessário que seja realizado um teste de força bruta,técnica utilizada que se dá através de sucessivas tentativas de acertar uma combinação de senha (uma chave), e assim conseguir acesso às informações e dados que deseja, (Fernandes). para tentar quebrar a senha.No mundo da segurança da computação, esse tipo de programa é bastante comum e, normalmente, é de implementação simples.Ademais, foi dito que a senha tem 7 caracteres, o que, por sua vez, ajuda no direcionamento da implementação do algoritmo.

As threads, nessa implementação, foram usadas para combinar as letras e números de forma diferente.Contudo, vale ressaltar que a senha não foi descoberta, pois, o sistema reiniciou algumas vezes, possivelmente, devido a instabilidade do próprio windows utilizado.

Simplificadamente, a ideia do algoritmo é baseada na lógica de permutação, como exemplifica a imagem abaixo, consistindo também, no número de possibilidades **fatorial = 7!** :



AQUI VEM A ESTRUTURA DE DADOS

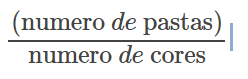
1. MAIN
2. CONTROLLER FILTER
3. CONTROLLER PARSER
4. CONTROLLER THREADS

**Classe Main e seus métodos:**

* **main(String[] args)**

Este método é padrão na linguagem java, é o chamado método principal e, no contexto do algoritmo, ele visa criar as instâncias (objetos) da classe PerformerController, que contém a implementação de threads, para a criação de multithreads e, ainda, realizar a distribuição racionalizada dos dados, que , no caso, como supracitado, são os diretórios.

Essa divisão é feita quantitativamente, ou seja, dado um número x de núcleos,



no dispositivo, é feito a divisão de

Dessa maneira, pode-se ocupar o processador de forma equivalente e garantir que o mesmo não fique ocioso, enquanto a tarefa não terminar.Como consequência, têm-se também, o aumento expressivo do desempenho, ou seja, ocorre a redução do tempo de forma considerável.Contudo, o uso do processador torna-se exponencial, inviabilizando, durante o processo, a utilização de tarefas muito pesadas.

Dito isso, esse método possui interação com o usuário.Ele oferece **duas opções para encontrar a senha** a primeira, segue a implementação original que consiste em buscar uma senha de tamanho fixo de 7 caracteres.A segunda, consiste em encontrar uma senha de tamanho variável, baseada na função product, da classe intertools, da linguagem de programação python.Contudo, nessa segunda opção, não houve implementação de programação paralela.

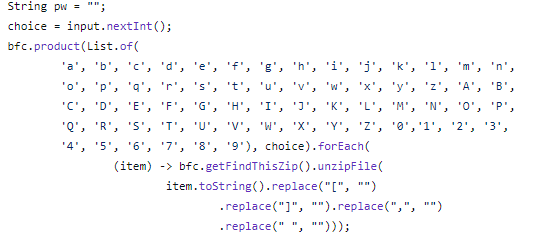


**Interação com o usuário para definir o método que lhe convém**

**Divisão de caracteres, a serem operados pelas threads, montados de forma diversificada**

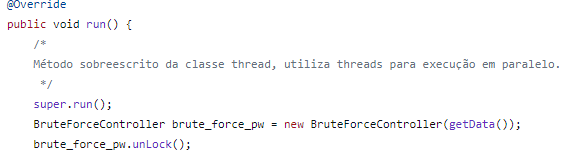
****

**Chamada do método da segunda opção que permite tamanhos variáveis de senha**

****

**Classe PerformerController e seus métodos:**

Em vista disso, o usuário pode escolher um dos dois métodos, a começar pelo primeiro, o método **main** chama a classe PerformerController, que herda da classe Threads, e faz a sobreescrita (@override) do método **run()**, que não recebe parâmetro e tem retorno do tipo void.A função principal deste, é instanciar um objeto da classe BruteForceController , que recebe o conjunto de caracteres como parâmetro, e executar o método **unLock()**

****

**Método run executa método unLock**

**Classe BruteForceController e seus métodos:**

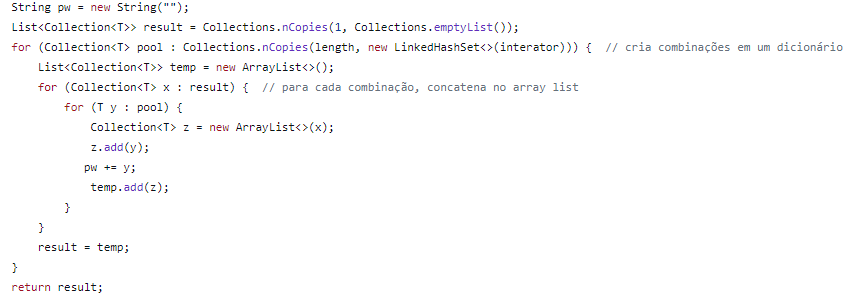
Esta classe faz as permutações, com a mesma lógica de crossing over da biologia, no assunto de genética.Enfim, ela combina caracteres para tentar descobrir uma senha de arquivo .zip.

* **unLock()**

Método principal, no que tange a funcionalidade, não recebe parâmetro e ,também, não possui retorno, é responsável por criar uma série de loops,pois, foi-se observado que o desempenho era melhor, e somas nas strings,com a finalidade de gerar as combinações necessárias.Além disso, para que funcione, é utilizado um contador, sob a ótica de ponteiros, da linguagem C, para alterar os valores nas posições do vetor.Ao realizar a combinação de 7 caracteres, é chamado o método **unZipFile**, da classe **FindZipFile**, que será ainda explicada, para enviar a senha ao arquivo zip.Por fim, a string recebe o valor vazio, para que, se não encontrada, possa ser gerada uma nova senha.

* **product(Collection<T> interator, int length)**

O método product, recebe um conjunto de caracteres e um número que informa o tamanho pretendido da senha. Ademais, como retorno, envia o tipo List<Collection<T>>, que, posteriormente, é tratado, no método **main** da classe Main, para remoção do formato de lista para string.Possui a implementação mais sofisticada, fazendo combinações em um tipo de dado **LinkedHashSet**, com a ideia de produtório, para com os caracteres que lhe é passado.Mais uma vez, este método, já foi implementado por outra pessoa, contudo, a fim de oferecer uma melhor versatilidade ao programa, tornou-se necessário, também, adequá-lo ao mesmo.



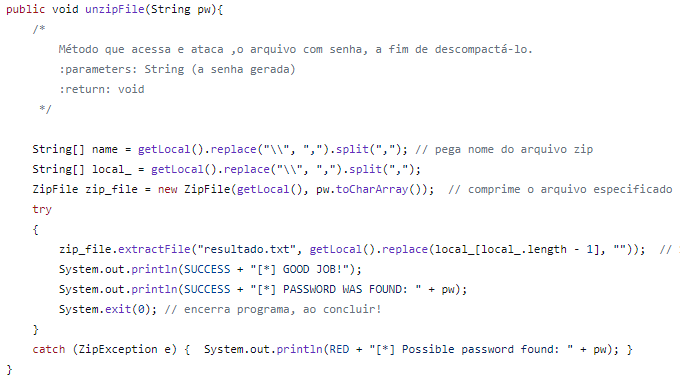
**Método product pode ser uma opção interessante para senha de diversos tamanhos.**

**Classe FindZipFile e seus métodos:**

Esta classe contém métodos que testarão as senhas no arquivo zip especificado.Porém, vale informar que, para funcionar, foi utilizado uma dependência que torna o processo mais simples.Tal dependência pode ser encontrada em:**https://javalibs.com/artifact/net.lingala.zip4j/zip4j**

* **unzipFile (String pw)**

O método que tenta extrair o arquivo zip.Para isso, utiliza os recursos advindos da dependência supradita e, mais ainda, testa cada senha passada por parâmetro, do tipo String, o método não possuí retorno.Se a senha é encontrada, ele termina a execução e exibe ,no terminal ,o resultado.



**Extrai arquivos zip com senhas, desde que a senha seja conhecida**

**Utilização Do Segundo Programa:**

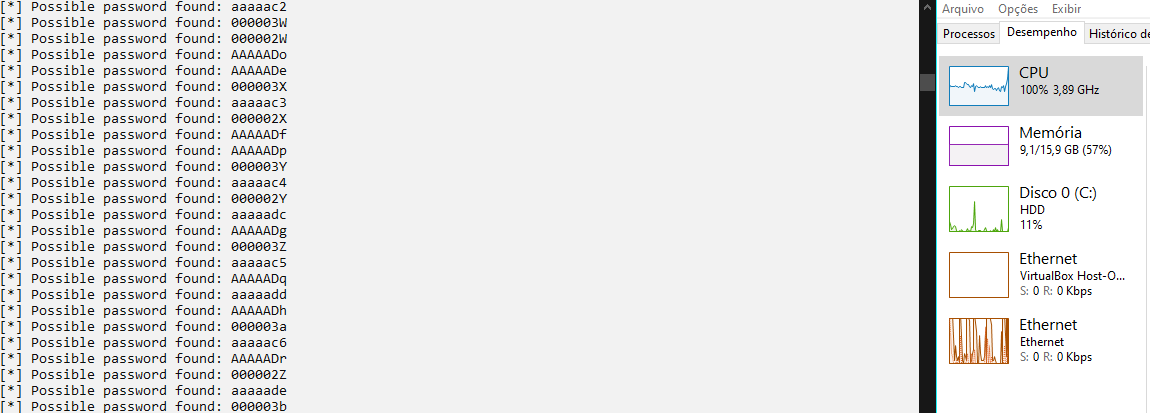
Após discorrer sobre a implementação do programa, é importante, também, informar seu funcionamento.Para isso, basta executar o arquivo com a extensão .jar, que é um Java Archive, ou seja, uma compilação do java,como segue na imagem:

**16**

**Utilização via terminal do windows**

**Execução E Testes Do Segundo Programa:**

A execução de testes de força bruta tendem a ser bastante demoradas.Foram feitos, antes, testes de senhas simples de 7 caracteres, funcionando, contudo, devido a alguns impencilhos, não foi possível manter a máquina ligada pelo tempo suficiente para encontrar a senha requerida.Abaixo, está o resultado da execução do teste no terminal windows:



**Implementação Do Terceiro Programa:**

O terceiro objetiva calcular todos os cpfs dos brasileiros, isto, por sua vez, implica em se trabalhar com uma quantidade muito grande de dados, na ordem dos bilhões, haja vista que cpfs podem ir até 99.999.999.999, que é um valor bastante alto e , inclusive, não é suportado pelo tipo int , no java.Em vista disso, durante este trabalho, foi-se necessário trabalhar, entre outros, com o tipo long,pois, este suporta números de ordens grandiosas.

Assim, seguindo o conceito “dividir para conquistar”, tornou-se essencial racionar os dados, intervalos candidatos a serem o cpf, para obter o melhor processamento paralelo.Em vista disso, como foi realizado a execução em uma máquina de seis núcleos (cores) e 2.90GHz de clock, dividiu-se todos os caminhos dos diretórios, pastas, de forma igualitária, para as threads criadas, no caso, seis, consoante a quantidade de núcleos do computador.

A organização do programa, vale ressaltar, espelha-se um pouco no modelo MVC (Model, Views, Controllers),mas, de forma simples, adaptada para ter um diretório controlador e , fora desse diretório, a classe principal, Main.Diante disso, é válido dizer que o diretório Controllers possuí a parte lógica da aplicação e faz, inclusive, a aplicação das threads, e a classe Main invoca os métodos necessário para resolver os problemas.

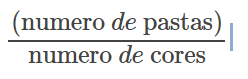
Para uma visão mais abrangente da implementação e das estruturas abstratas de dados utilizadas, o diagrama abaixo ilustra a aplicação , como um todo.

**Classe Main e seus métodos:**

* **main(String[] args)**

Este método é padrão na linguagem java, é o chamado método principal e, no contexto do algoritmo, ele visa criar as instâncias (objetos) da classe PerformerController, que contém a implementação de threads, para a criação de multithreads e, ainda, realizar a distribuição racionalizada dos dados, que , no caso, como supracitado, são os diretórios.

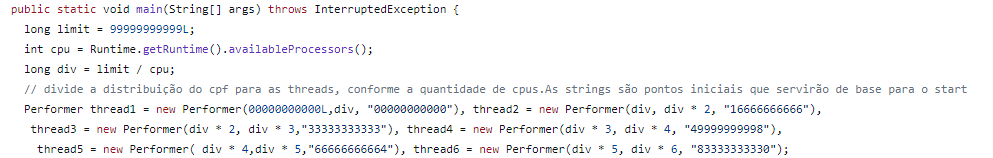
Essa divisão é feita quantitativamente, ou seja, dado um número x de núcleos,



no dispositivo, é feito a divisão de

Dessa maneira, pode-se ocupar o processador de forma equivalente e garantir que o mesmo não fique ocioso, enquanto a tarefa não terminar.Como consequência, têm-se também, o aumento expressivo do desempenho, ou seja, ocorre a redução do tempo de forma considerável.Contudo, o uso do processador torna-se exponencial, inviabilizando, durante o processo, a utilização de tarefas muito pesadas.

Com isso em mente, o método main começa por definir os limites inferiores e superiores dos intervalos de divisão para as threads.Salientando que, em si, deverão ser testados todos os números de 0 a 99.999.999.999.

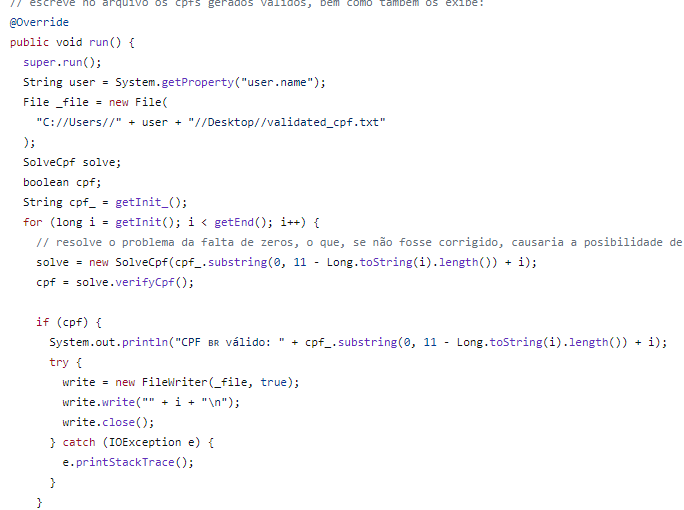


**Divisão dos limites dos intervalos para o cálculo dos cpfs entre as threads**

**Classe PerformerController e seus métodos:**

Em vista disso, o método **main** chama a classe PerformerController, que herda da classe Threads, e faz a sobreescrita (@override) do método **run()**, que não recebe parâmetro e tem retorno do tipo void.A função principal deste, é instanciar um objeto da classe SolveCpf , que recebe o conjunto de caracteres como parâmetro, e executar o método **verifyCpf()**

Além disso, exibe os CPF válidos no console e os salva em um arquivo de texto, cujo o final da execução resultou em um tamanho de cerca de 11GB, aproximadamente.

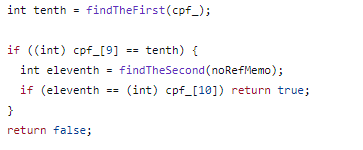


**Método run que gera os números a serem calculados e chama o método para testá-los**

**Classe SolveCpf e seus métodos:**

Por fim, a classe solve é responsável por analisar os números e verem se eles são candidatos a formarem um cpf válido, para isso, ela conta com métodos que realizam os cálculos necessários.

* **verifyCpf()**

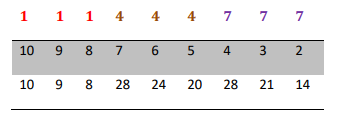
**** Método que não recebe entrada de dados,mas, retorna um boolean.Este método, por sua vez, chama outros dois métodos que calcularão os dois últimos dígitos do número em questão para confirmar se este pode ser um cpf válido, no Brasil, ou não.

Assim, se os dígitos calculados corresponderem à décima e à décima primeira posição, para cada cálculo, então, o cpf será considerado válido.

* **findTheFirst(int[] nineDigits)**

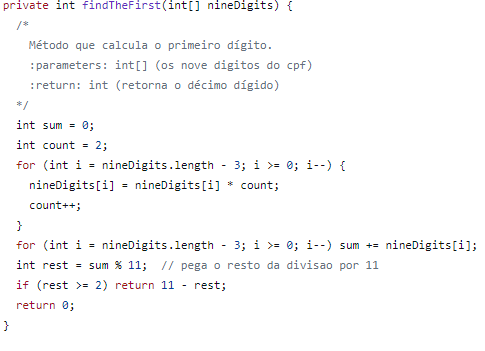
Este método calcula o primeiro dígito verificador, seu cálculo baseia-se em multiplicar cada um dos números, da direita para a esquerda por números crescentes a partir do número 2.A título de exemplo, considerar o número 11144477735, o método verifyCpf, pega apenas os nove dígitos do número, por isso,

o método em questão, precisa apenas fazer os cálculos no vetor estático de 9 posições como segue a imagem abaixo:

 **O vetor estático recebe sucessivas multiplicações**

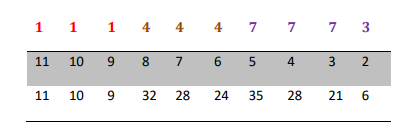
Feito isso, soma-se , sequencialmente os números dos resultados obtidos, para, então, dividir o mesmo por 11.Feito isso, basta analisar se o resto da divisão for menor que 2, então o dígito é igual a 0 (zero). Se o resto da divisão for maior ou igual a 2, então o dígito verificador é igual a 11 menos o resto da divisão (11 - resto)

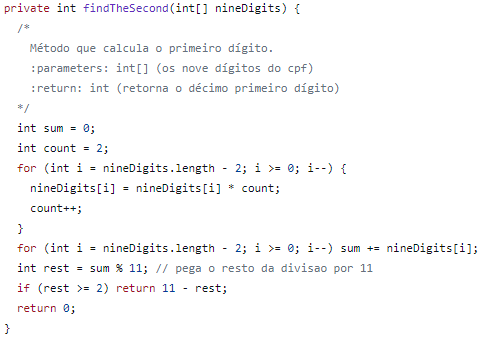
Logo,eis , abaixo, a execução do método:



* **findTheSecond(int[] nineDigits)**

Este método faz a verificação do último dígito do número candidato a CPF.Seu cálculo é semelhante ao método acima, mudando, apenas, a multiplicação, que se dá por meio dos valores 11, 10, 9, 8, 7, 6, 5, 4, 3, 2 já que se incluí mais um dígito no cálculo

**O vetor estático recebe sucessivas multiplicações para a última posição do número**

****

**Utilização Do Segundo Programa:**

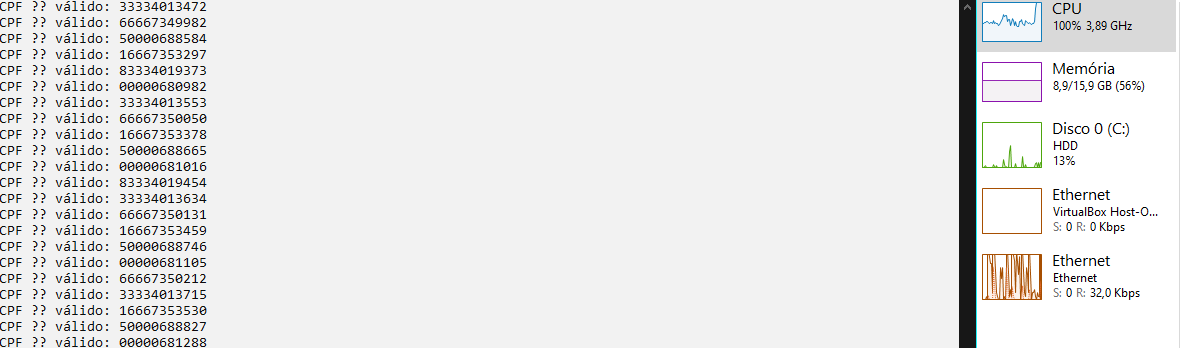
Após discorrer sobre a implementação do programa, é importante, também, informar seu funcionamento.Para isso, basta executar o arquivo com a extensão .jar, que é um Java Archive, ou seja, uma compilação do java,como segue na imagem:

**Execução via terminal**



**Execução E Testes Do Segundo Programa:**

A execução gera um arquivo na área de trabalho que receberá todos os números de cpf válidos, além disso, é exibido, durante o processo, o resultado no terminal.Para conferir os números do resultado produzido, foi utilizado o site **<https://consulta.guru/validar-cpf.>**

****

**Conclusão:**

Diante do exposto, é evidente, por si mesmo, a utilidade da programação paralela para a computação atual.Isso pois, o mundo moderno está repleto de informações e dados que são gerados a todo momento, tão logo, é perceptível a necessidade de métodos que trabalhem da forma mais otimizada possível para atender às demandas da sociedade vigente,que visa ter os recursos de modo quase que imediatos. Em detrimento a esses fatores, o presente trabalho agrega no que concerne a compreender melhor os meandros da programação paralela na resolução dos problemas que envolvem grandes processamentos de dados, de fato.

Um pouco mais, cabe elucidar que, as maiores dificuldades encontradas para executar esta tarefa, foram, ironicamente, alguns problemas gerais na própria programação paralela, em si.Ou seja, a limitação de hardware e, principalmente, a dificuldade em distribuir , adequadamente, os dados a serem processados em cada núcleo.Contudo, como afirmado acima, as tecnologias atuais, ainda sim, permitem ao programador, mesmo que acadêmico, usufruir de benefícios dessa forma de programar.Tão logo, diante dos resultados, é claro que o paralelismo deve ser utilizado sempre que possível.

**Bibliografia**

* **Documentação de Trabalhos Práticos** <https://homepages.dcc.ufmg.br/~nivio/cursos/aed2/roteiro/>. Acesso em: 14 de Maio de 2022.
* GRAMA, A. et al. **Introduction to Parallel Computing, Second Edition.** 2 edition. Team Lib, 2003