Trabalho Prático - Em busca do maior Primo

Arquitetura e organização de computadores

Grupo:

1. Gabrielle de Oliveira Fonseca – 0072379;
2. Maria Eduarda Rodrigues Alves Morais - 0072382.

Introdução

O grupo recebeu um diretório com diversos arquivos, o objetivo principal era criar um código que analisasse os arquivos .txt e encontrasse o maior número primo entre eles, tendo como resultado esse número primo e seu diretório.

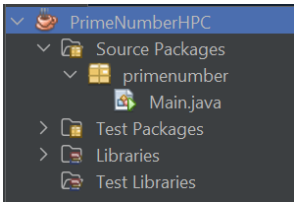
Problemas do desafio:

- Nem todos os arquivos eram .txt;
- Haviam letras, símbolos e espaços junto com os números;
- Alguns números tinham casas decimais (exemplo: 147,5667899);
- Era necessário utilizar computação paralela;

Implementação

Tendo em vista todas as características citadas acima, foi criada a seguinte solução:

1. Classes do projeto



2. Bibliotecas utilizadas

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.filechooser.FileFilter;
```

3. Main

3.1. selecionaDiretorioRaiz

Esse método já estava no projeto e não foi modificado pelo grupo. Tem o objetivo de selecionar a pasta que contém os arquivos a serem analisados, nesse método não são aceitos arquivos compactados, então foi necessário descompactar o arquivo .zip disponibilizado para teste dos resultados.

```
public static File selecionaDiretorioRaiz() {
    JFileChooser janelaSelecao = new JFileChooser(currentDirectoryPath: ".");
    //janelaSelecao.setControlButtonsAreShown(false);

    //conf. do filtro de selecao
    janelaSelecao.setFileFilter(new FileFilter() {
        @Override
        public boolean accept(File arquivo) {
            return arquivo.isDirectory();
        }

        @Override
        public String getDescription() {
            return "Diretório";
        }
    });
```

```
        janelaSelecao.setFileSelectionMode(mode: JFileChooser.DIRECTORIES_ONLY);

        //avaliando a acao do usuario na selecao da pasta de inicio da busca
        int acao = janelaSelecao.showOpenDialog(parent: null);

        if (acao == JFileChooser.APPROVE_OPTION) {
            return janelaSelecao.getSelectedFile();
        } else {
            return null;
        }
    }
```

3.2. obterArquivos

Nesse método, o código percorre toda pasta selecionada, criando uma lista com os arquivos de interesse (.txt) encontrados e no final retorna essa lista para ser utilizada no próximo método.

```
    //percorre recursivamente todos os arquivos e subpastas do diretorio e retorna uma lista com os arquivos encontrados
    public static List<File> obterArquivos(File diretorio) {
        List<File> arquivos = new ArrayList<>();

        if (diretorio.isDirectory()) {
            File[] listaArquivos = diretorio.listFiles();
            if (listaArquivos != null) {
                for (File arquivo : listaArquivos) {
                    if (arquivo.isFile()) {
                        arquivos.add(e: arquivo);
                    } else if (arquivo.isDirectory()) {
                        arquivos.addAll(c: obterArquivos(diretorio: arquivo));
                    }
                }
            }
        }

        return arquivos;
    }
```

3.3. isPrimo

O método “isPrimo” é responsável calcular se os números dos arquivos contidos na lista são ou não primos.

```
    //calcula se os numeros encontrados sao primos
    public static boolean isPrimo(int number) {
        if (number < 2) {
            return false;
        }
        for (int i = 2; i <= number / 2; i++) {
            if (number % i == 0) {
                return false;
            }
        }
        return true;
    }
```

3.4. achaMaiorPrimo

O método “achaMaiorPrimo” recebe a lista criada com todos os arquivos, separa os números inteiros (ignora todos os valores que não são INT, assim como vírgulas, pontos, espaços e ponto e vírgulas) e utilizando o método “isPrimo” analisa todos os números, armazenando o maior número primo encontrado no método “maiorPrimoInfo”. Ao terminar de analisar todos os números de todos os arquivos, ele interrompe as threads evitando que ocorra um loop na computação paralela e retorna o valor encontrado.

```
    //recebe uma lista de arquivos e percorre cada arquivo
    public static int achaMaiorPrimo(List<File> files) {
        maiorPrimoInfo maiorPrimoInfo = new maiorPrimoInfo();

        //utiliza a execucao paralela
        ExecutorService executor = Executors.newFixedThreadPool(nThreads);
```

```
        for (File file : files) {
            if (file.isFile() && file.getName().endsWith(suffix: ".txt")) {
                executor.submit(() -> {
                    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
                        String line;
                        while ((line = reader.readLine()) != null) {
                            //separa os numeros com espaço e virgula dos valores que nao sao int
                            String[] numbers = line.split(regex: "[,\\s]+");
```

```
                        for (String numberStr : numbers) {
                            try {
                                int number = Integer.parseInt(s: numberStr);
                                if (isPrimo(number)) {
                                    maiorPrimoInfo.atualizar(primo: number, arquivo: file);
                                }
                            } catch (NumberFormatException e) {
                                // Ignora valores que não forem int
                            }
                        }
                    }
                } catch (IOException e) {
                }
            });
        }
    }
```

```
        //interrompe a computacao pararela ao encontrar o resultado
        //impede looping
        executor.shutdown();
        try {
            executor.awaitTermination(timeout: Long.MAX_VALUE, unit: TimeUnit.NANOSECONDS);
        } catch (InterruptedException e) {
        }
        File arquivoMaiorPrimo = maiorPrimoInfo.getArquivoMaiorPrimo();
        if (arquivoMaiorPrimo != null) {
            System.out.println("O maior número primo encontrado está no arquivo: " + arquivoMaiorPrimo.getAbsolutePath());
        }
        return maiorPrimoInfo.getMaiorPrimo();
    }
```

3.5. maiorPrimoInfo

Já nesse método, cada vez que um novo maior primo é encontrado no método “achaMaiorPrimo”, esse número é atualizado, para garantir sincronização entre as threads.

```
    //armazena o maior numero primo e garante sincronizacao correta das threads
    public static class maiorPrimoInfo {

        private int maiorPrimo = 0;
        private File arquivoMaiorPrimo = null;

        public synchronized void atualizar(int primo, File arquivo) {
            if (primo > maiorPrimo) {
                maiorPrimo = primo;
                arquivoMaiorPrimo = arquivo;
            }
        }

        public int getMaiorPrimo() {
            return maiorPrimo;
        }

        public File getArquivoMaiorPrimo() {
            return arquivoMaiorPrimo;
        }
    }
```

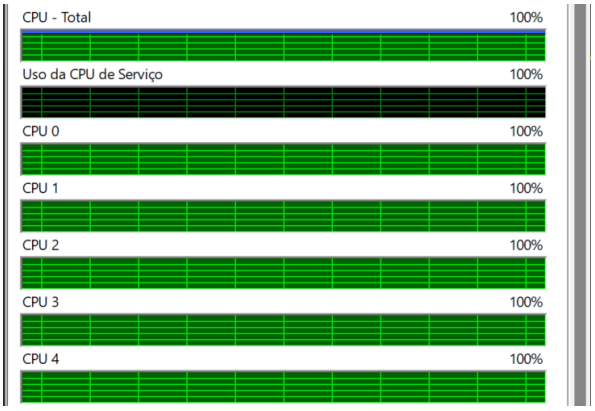
4. Main 2

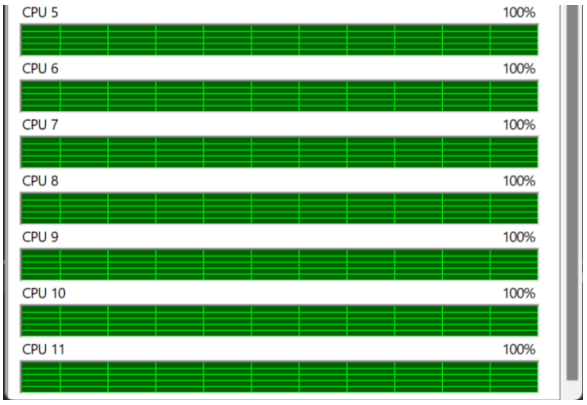
No último método, o código mostra na tela os resultados obtidos, são apresentados tanto o valor encontrado, quanto o seu respectivo diretório. Nele também se encontra a interface de seleção de diretório e a interface de demonstração de resultado.

```
    public static void main(String[] args) {

        //selecao de um diretorio para iniciar a busca
        File pastaInicial = selecionaDiretorioRaiz();

        if (pastaInicial == null) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Você deve selecionar uma pasta para o processamento",
                    title:"Selecione o arquivo", messageType: JOptionPane.WARNING_MESSAGE);
        } else {
            //retorna o numero primo encontrado e o diretorio onde esse numero se encontra
            List<File> files = obterArquivos(diretorio: pastaInicial);
            int maiorPrimo = achaMaiorPrimo(files);
            JOptionPane.showMessageDialog(parentComponent: null, "O maior número primo encontrado é: " + maiorPrimo,
                    title:"Resultado", messageType: JOptionPane.INFORMATION_MESSAGE);
        }
    }
}
```

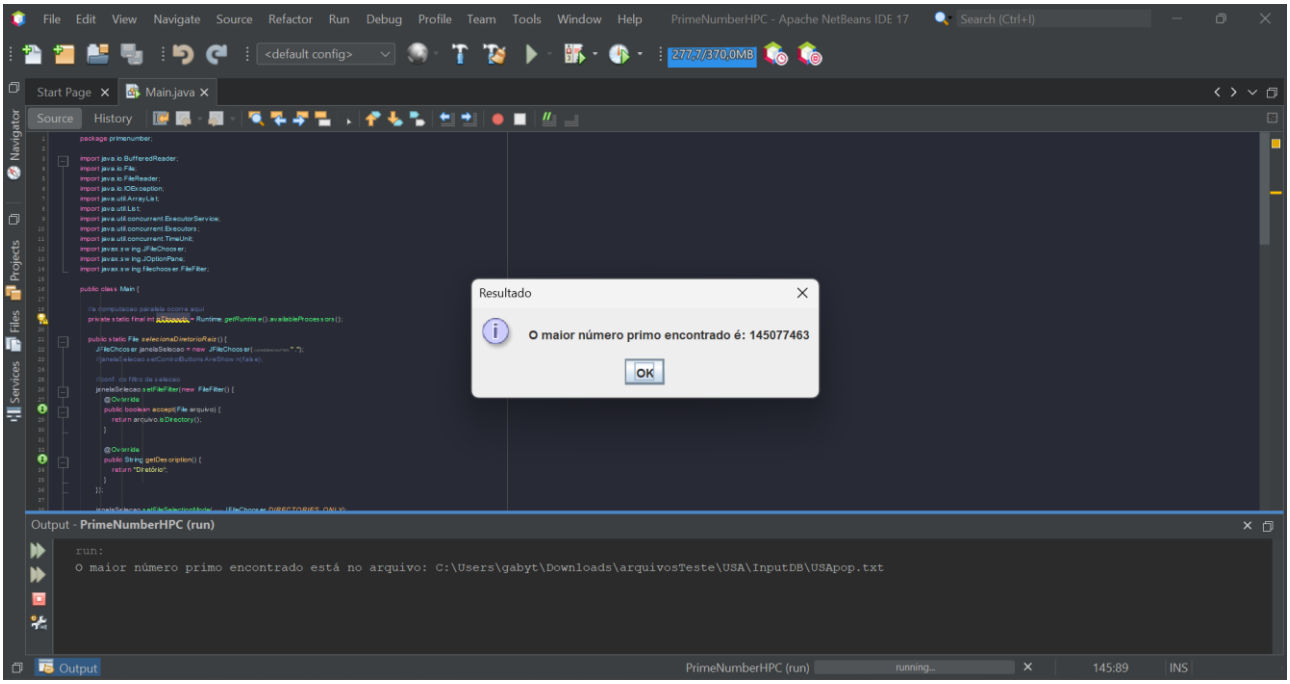
Resultados

O grupo teve um resultado bastante satisfatório com o código implementado. Durante a execução do código, todas as 12 CPUs foram utilizadas:

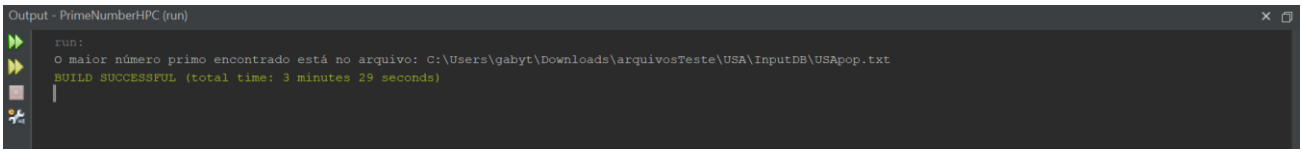




O resultado obtido foi:



E o tempo total de execução foi:



Conclusão

O código implementado pelo grupo funcionou de forma efetiva, chegando ao resultado esperando e em um tempo consideravelmente bom. Todos os problemas apresentados na introdução desse trabalho foram solucionados com sucesso.

A solução foi otimizada da melhor forma a utilizar todos os processos exigidos dentro do tempo aceitável e em todos os testes realizados foram obtidos a mesma média de tempo (3 minutos). Acreditasse que para diminuir esse tempo, seria necessário diminuir o número de CPUs utilizadas, tendo em vista que outro teste realizado sem a computação paralela levou cerca de 1 minuto e 30 segundos para encontrar o maior número primo do mesmo arquivo utilizado nos testes finais.

O grau de dificuldade desse trabalho foi médio, levando em consideração que houve um avanço no curso de Arquitetura e Organização de Computadores, se comparado com os outros 2 trabalhos já apresentados.

Bibliografia

https://www.devmedia.com.br/programacao-paralela-com-java/33062

https://pt.stackoverflow.com/questions/491553/numero-primo-java

http://www.dcc.ufmg.br/~nivio/cursos/aed2/roteiro

https://www.guj.com.br/t/descobrir-se-um-numero-e-primo-ou-nao/81156/2

https://receitasdecodigo.com.br/java/algoritmo-simples-em-java-para-verificar-se-um-numero-e-primo