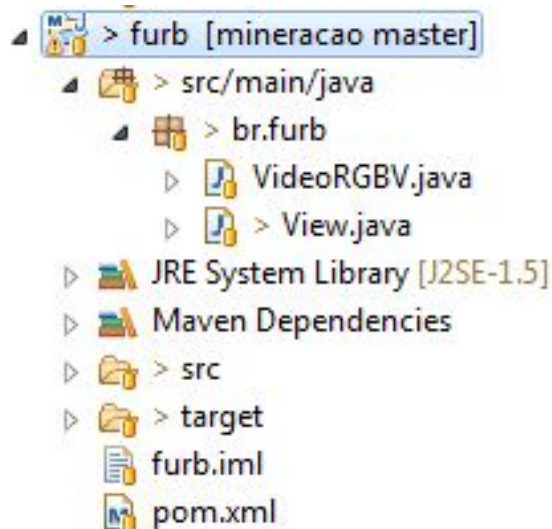


Mineração em vídeo

André Sestari
Antônio Marco
Tiago Pereira

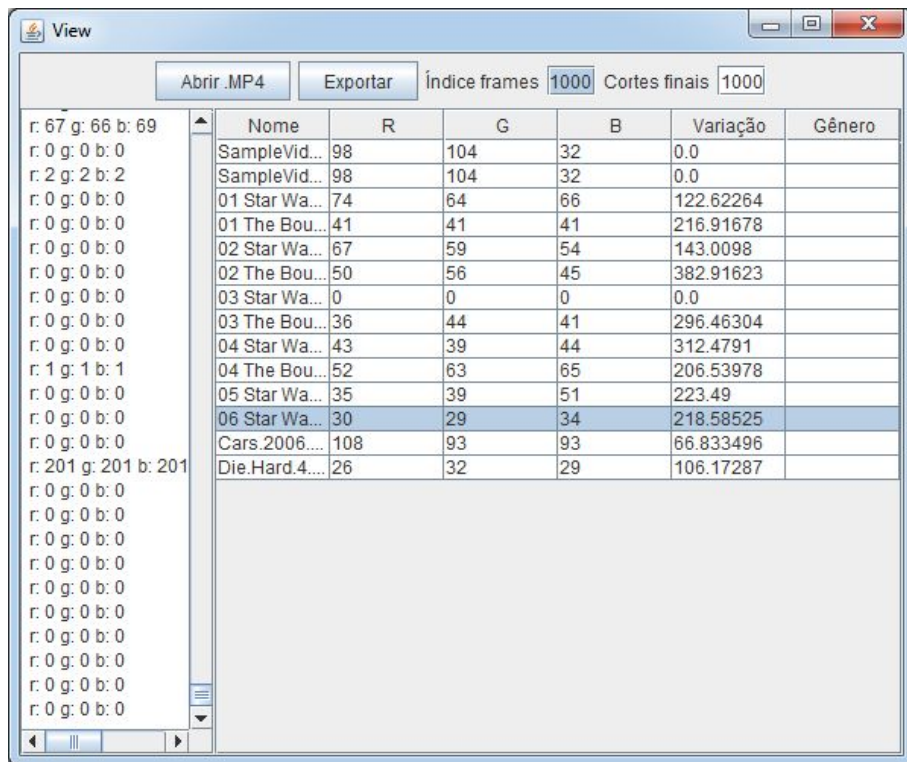
Estrutura do projeto



```
<dependencies>
  <dependency>
    <groupId>org.bytedeco</groupId>
    <artifactId>javacv</artifactId>
    <version>1.0</version>
  </dependency>

  <dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>3.14</version>
  </dependency>
</dependencies>
```

View



The screenshot shows a window titled "View" with a toolbar containing "Abrir .MP4" and "Exportar" buttons. It also features input fields for "Índice frames" (set to 1000) and "Cortes finais" (set to 1000). The main area displays a table with 7 columns: "Nome", "R", "G", "B", "Variação", and "Gênero". To the left of the table is a vertical list of frame identifiers in the format "r: [R] g: [G] b: [B]", with the 1000th frame ("r: 100 g: 0 b: 0") highlighted in blue. The table contains 12 rows of data, with the 1000th row also highlighted in blue.

	Nome	R	G	B	Variação	Gênero
r: 67 g: 66 b: 69	SampleVid...	98	104	32	0.0	
r: 0 g: 0 b: 0	SampleVid...	98	104	32	0.0	
r: 2 g: 2 b: 2	01 Star Wa...	74	64	66	122.62264	
r: 0 g: 0 b: 0	01 The Bou...	41	41	41	216.91678	
r: 0 g: 0 b: 0	02 Star Wa...	67	59	54	143.0098	
r: 0 g: 0 b: 0	02 The Bou...	50	56	45	382.91623	
r: 0 g: 0 b: 0	03 Star Wa...	0	0	0	0.0	
r: 0 g: 0 b: 0	03 The Bou...	36	44	41	296.46304	
r: 0 g: 0 b: 0	04 Star Wa...	43	39	44	312.4791	
r: 1 g: 1 b: 1	04 The Bou...	52	63	65	206.53978	
r: 0 g: 0 b: 0	05 Star Wa...	35	39	51	223.49	
r: 0 g: 0 b: 0	06 Star Wa...	30	29	34	218.58525	
r: 0 g: 0 b: 0	Cars.2006...	108	93	93	66.833496	
r: 201 g: 201 b: 201	Die.Hard.4...	26	32	29	106.17287	

Responsável por criar a tela e buscar as cores de cada frame.

View

Inicialização do
frameGrabber e
bytedeco.

Busca as cores dos
quatro pontos do
frame e cria a
instância de
VideoRGB.

```
private void tratarVideo(File file) {  
  
    FFmpegFrameGrabber g = new FFmpegFrameGrabber(file);  
    try {  
        g.start();  
        org.bytedeco.javacv.Frame frame = g.grabImage();
```

```
    while (frame != null) {  
  
        Long a = new Date().getTime();  
  
        if (qtFrames >= indice) {  
            qtFrames = 0;  
            frame = g.grabImage();  
  
            BufferedImage image = paintConverter.getBufferedImage(frame, 2.2 / g.getGamma());  
  
            if (image != null) {  
  
                // pega 4 cores do frame  
                int[] rgb1 = getRGBImage(40, 40, image);  
                int[] rgb2 = getRGBImage(image.getHeight() - 40, image.getHeight() - 40, image);  
                int[] rgb3 = getRGBImage(40, image.getHeight() - 40, image);  
                int[] rgb4 = getRGBImage(image.getHeight() - 40, 40, image);  
  
                colors.add(rgb1);  
                colors.add(rgb2);  
                colors.add(rgb3);  
                colors.add(rgb4);  
  
            }  
  
        } else {  
            Long b = new Date().getTime();  
            frame = g.grabFrame(false);  
        }  
  
        qtFrames++;  
    }  
}
```

VideoRGBV

Guarda os valores das cores e calcula a variação.

```
public VideoRGBV(ArrayList<int[]> rgbList, int qtdCorte){
    this.qtdCorte = qtdCorte;

    int count = rgbList.size();
    int slice = (int)(count / qtdCorte);
    int[] rgb = null;
    for(int i = 1; i < qtdCorte-1; i++){

        if(rgb!=null)
            this.variacao += calculaDiferenca(rgb, rgbList.get(slice*i));
        rgb = rgbList.get(slice*i);
        this.red += rgb[0];
        this.green += rgb[1];
        this.blue += rgb[2];

    }
    this.red /= qtdCorte;
    this.green /= qtdCorte;
    this.blue /= qtdCorte;
    this.variacao /= qtdCorte-1;
}

private float calculaDiferenca(int[] rgbOld, int[] rgbNew){
    try {
        float r = (rgbNew[0] - rgbOld[0]) / rgbOld[0] * 100;
        float g = (rgbNew[1] - rgbOld[1]) / rgbOld[1] * 100;
        float b = (rgbNew[2] - rgbOld[2]) / rgbOld[2] * 100;

        float rgb = (r+g+b)/3;

        return rgb > 0 ? rgb : (rgb * -1);
    }catch (ArithmeticException e){
        return 0;
    }
}
```

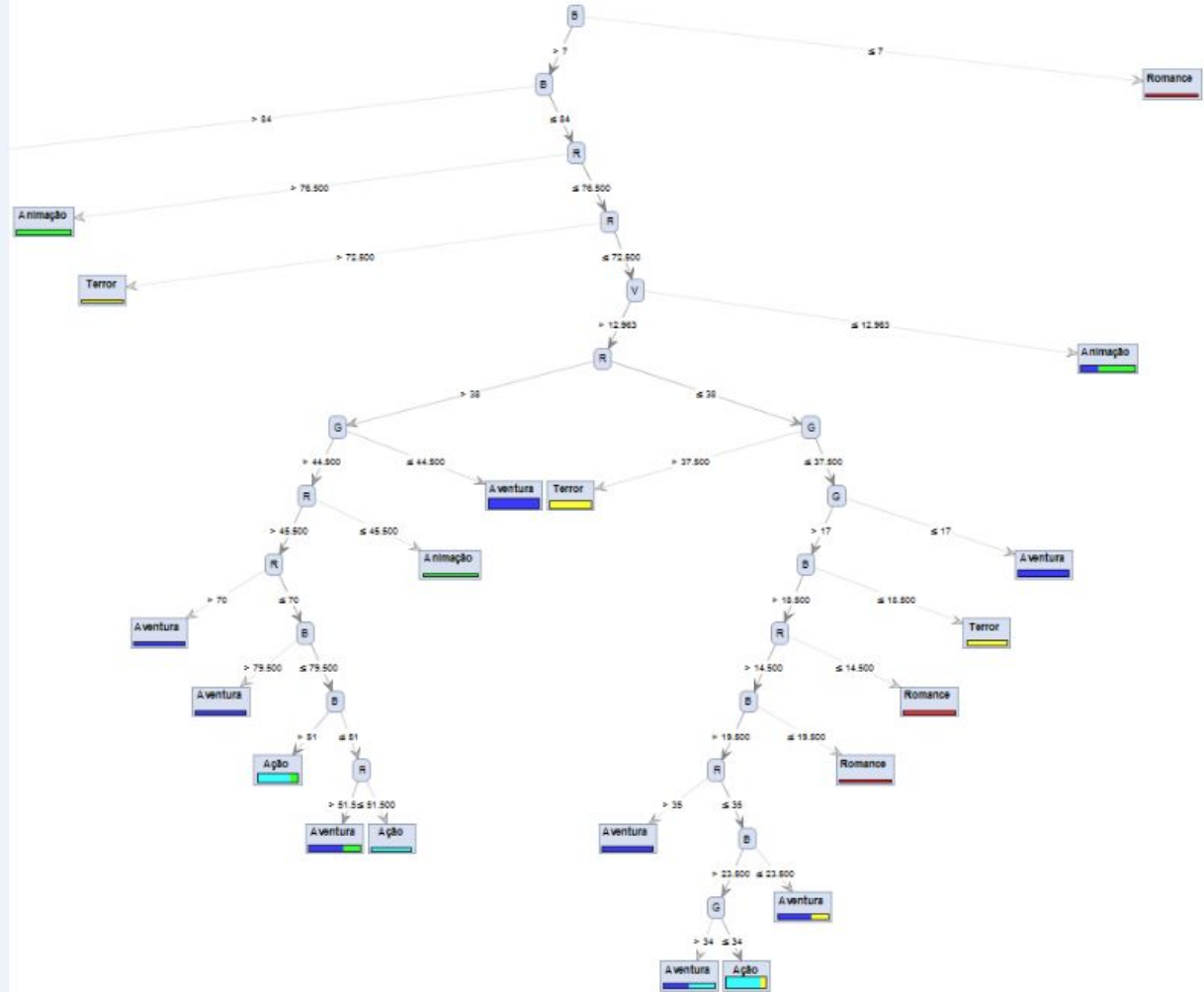
Exportação

Exportação dos
filmes lidos e adição
do gênero (manual).

		R	G	B	V	Gênero
1	Ice Age: The Great Escape	62	69	77	-25.925926	Animação
2	Kill Bill Vol. 2. 2004. 7x	42	40	40	-14.814815	Ação
3	Kung Fu Panda 2.x2	51	38	34	111.11111	Animação
4	Shrek	55	54	50	107.4074	Animação
5	Shrek 4	89	72	45	111.11111	Animação
6	Shrek 3	72	54	47	18.518517	Animação
7	Carros	121	107	102	290	Animação
8	Matrix revolution	39	39	36	133.33	Ação
9	Matrix	36	43	39	496	Ação
10	Kill Bill vol 2	73	64	52	66.666664	Ação

RapidMiner

Árvore criada com os dados do processamento de 54 filmes com gêneros variando entre ação, terror, aventura, animação e romance.



Resultado

- Filmes de animação e terror são mais assertivos (valores altos de RGB).
- Filmes de ação e aventura continuam difíceis de minerar, apesar da adição da variação.
- Variação depende do tamanho do filme.
- Lento para processar vídeos longos (como filmes);
- Rápida e fácil construção da árvore de decisão.

Dúvidas?