Desenvolvimento de Software para visualização e análise de Árvore genéricas na distribuição de terras em tribos bárbaras

André Luiz Machado, Hélio Strappazzon, Nicolas Vargas

Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Avenida Ipiranga, 6681 – Prédio 32 – CEP 90619-900 – Porto Alegre – RS – Brasil

Resumo. *Este trabalho descreve a implementação de um software desenvolvido para criação de árvores genéricas para visualização e análise de pergaminhos que são representados através de arquivos de texto(.txt). Baseando-se nesses arquivos, o presente trabalho busca implementar o algoritmo para analisar a distribuição de terras de tribos e montar uma árvore, onde mostra as relações de pai para filho e a quantidade de terras que cada guerreiro conquistou, a quantidade de terra que cada guerreiro herdou e qual guerreiro da última geração possuí mais terras a partir da soma de herança e conquista. Os resultados obtidos contribuem para melhor análise e compreensão de cada tribo analisada.*

1. Introdução

Uma arvore é um tipo abstrato de dados(TAD) e tem como definição armazenar nodos de forma pai-filho, ou seja, hierárquica. Se uma arvore não é vazia, precisa existir um elemento pai, chamado de raiz com zero ou mais filhos, mas que não possua nenhum pai [1]. Esse presente trabalho, descreve o desenvolvimento de um algoritmo que realiza a construção de árvores visando analisar e interpretar pergaminhos de tribos em formas de arquivos de texto(.txt) e verificar as heranças de cada elemento da árvore, além do guerreiro da última geração das tribos com a maior quantidade de terras acumuladas.

2. Metodologia

Para fins da construção do programa visando a análise das tribos e suas distribuições de terras, foram criadas quatro classes, são elas, Node, Tree, Tribo e App. Onde cada uma possuía um fim específico.

O programa foi desenvolvido utilizando a ferramenta Visual Studio Code.

A classe Node, foi criada para representar os nodos da árvore, onde cada Nodo possuí as informações como o guerreiro, quantidade de terras, pai, a camada que ele pertence na árvore e os seus filhos.

A classe Tree representa a árvore genérica que é construída a partir dos nodos que são a representação de guerreiros e suas relações pai-filho. Essa classe possui um foco maior na organização hierárquica da tribo. Ela possui métodos para montar a arvore e organizá-la, encontrar o raiz e escrevê-la em formato de string para melhor compreensão visual.

A classe Tribo é a classe que lida com a leitura dos arquivos e armazena os nodos, também é responsável pelo cálculo das terras e heranças. A classe Tribo lida com forma eficiente com a leitura do arquivo e com o cálculo final e parcial das terras.

A classe App foi criada para a execução do programa.

3. Algoritmo

O algoritmo feito está representado na figura 1 em forma de fluxograma realizado pela plataforma Miro[1].

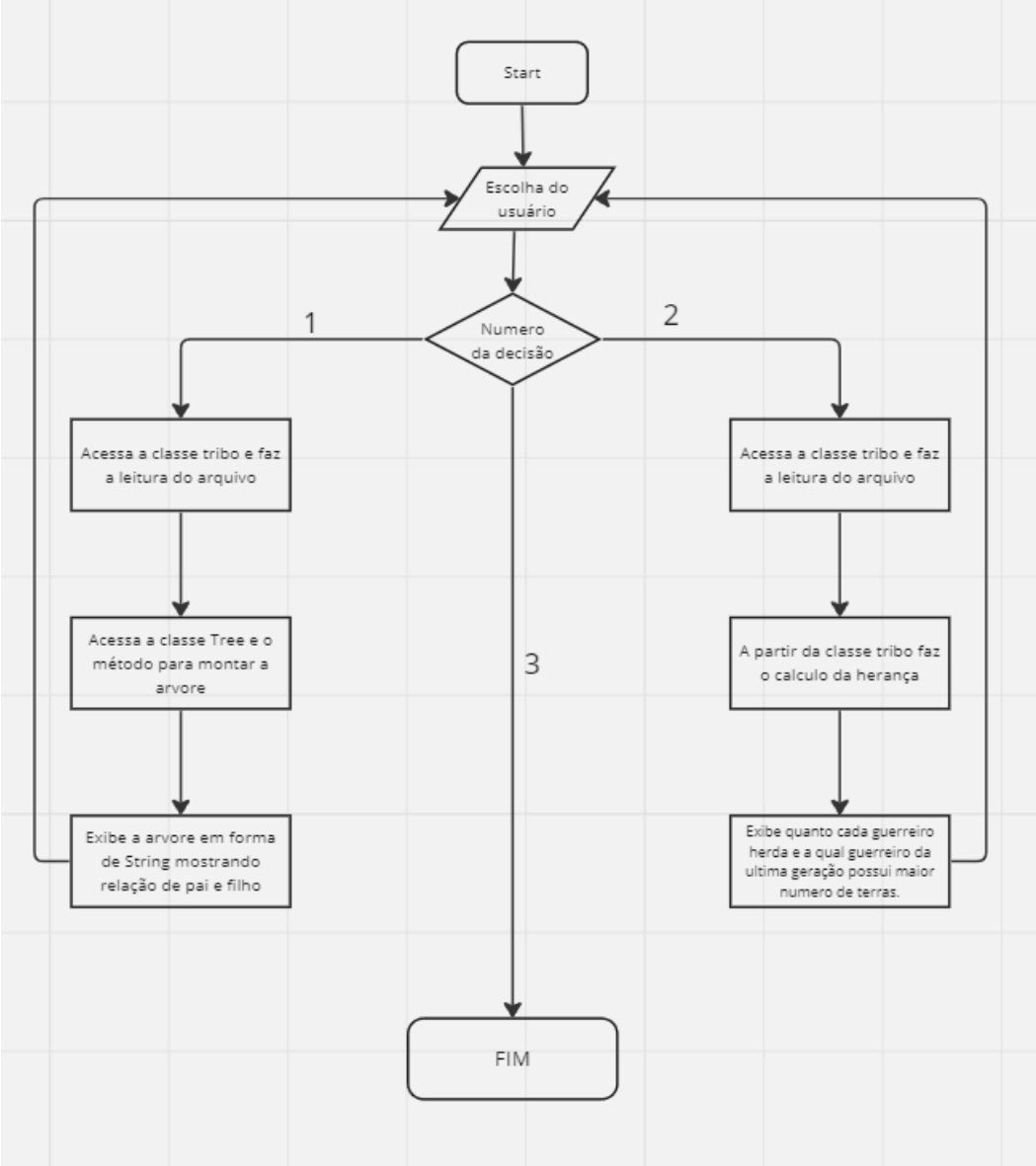

Figura 1 - Fluxograma do algoritmo

Os métodos apresentados no serão explicados na sequência desse relatório.

4. O programa

O programa foi desenvolvido na linguagem Java apresenta três classes principais descritas anteriormente onde cada uma tem sua finalidade especifica para desempenhar no programa. Para construção do programa foi utilizada a literatura [3].

Inicialmente foi criada classe Node, descrita a seguir.

```
1. import java.util.HashMap;
 2. import java.util.Map;
 3.
 4. public class Node {
 5.
 6.     private String guerreiro;
 7.     private int terras;
 8.     private Map<String, Node> filhos;
 9.     private Node pai;
10.     private int camada;
11.
12.     public Node(String guerreiro, int terras) {
13.         this.guerreiro = guerreiro;
14.         this.terras = terras;
15.         this.filhos = new HashMap<>();
16.         this.pai = null;
17.         this.camada = 0;
18.     }
19.
20.     public String getGuerreiro() {
21.         return guerreiro;
22.     }
23.
24.     public void setTerras(int terras) {
25.         this.terras = terras;
26.     }
27.
28.     public int getTerras() {
29.         return terras;
```

```
30.     }
31.
32.     public Map<String, Node> getFilhos() {
33.         return filhos;
34.     }
35.
36.     public Node getPai() {
37.         return pai;
38.     }
39.
40.     public void setPai(Node pai) {
41.         this.pai = pai;
42.     }
43.
44.     public void adicionarFilho(String nome, Node filho) {
45.         filhos.put(nome, filho);
46.         filho.setPai(this);
47.     }
48.
49.     public int getCamada() {
50.         return camada;
51.     }
52.
53.     public void setCamada(int camada) {
54.         this.camada = camada;
55.     }
56.
57. }
```

Percebe-se que a classe Node possui os atributos guerreiro que é a representação em String do guerreiro, Terras que é a quantidade de terras, filhos que é um map que associa nomes de filhos, pai que referencia o pai, onde esse atributo foi criado para encontrar o raiz da árvore e o atributo camada, onde diz de qual geração o guerreiro faz parte. O atributo camada foi criado para facilitar encontrar os guerreiros da última geração e assim descobrir qual teria mais terras. A classe também possui seus setters e getters para acessar e modificar os atributos.

A classe Tree foi criada para montar a árvore, onde cada nó da árvore é uma instancia da classe Node.

```
1. import java.util.Map;
2.
3. public class Tree {
4.     Node root;
5.
6.     public Tree() {
7.         this.root = null;
8.     }
9.
10.     boolean isEmpty() {
11.         return this.root == null;
12.     }
13.
14.     public Node encontraRaiz(Map<String, Node> nodes) {
15.         for (Node node : nodes.values()) {
16.             if (node.getPai() == null) {
17.                 return node;
18.             }
19.         }
20.         return null;
21.     }
22.
23.     public void montarArvore(Map<String, Node> nodes) {
24.         Tribo terrasRoot = new Tribo();
25.         for (Node node : nodes.values()) {
26.             if (node.getPai() == null) {
27.                 this.root = encontraRaiz(nodes);
28.                 root.setTerras(terrasRoot.terrasPai());
29.             } else {
30.                 Node pai = nodes.get(node.getPai().getGuerreiro());
31.                 if (pai != null) {
32.                     pai.adicionarFilho(node.getGuerreiro(), node);
```

```
33.             } else {
34.                 System.out.println("Erro: Nó pai não encontrado para " + node.getGuerreiro());
35.             }
36.         }
37.     }
38. }
39. // ai o recursivo dos guri
40.
41. private void imprimirArvore(Node node, StringBuilder r, String prefix) {
42.     if (node != null) {
43.         r.append(prefix).append("Guerreiro: ").append(node.getGuerreiro())
44.                 .append(", Terras conquistadas(não é a finall): ").append(node.getTerras());
45.
46.         Map<String, Node> filhos = node.getFilhos();
47.         if (!filhos.isEmpty()) {
48.             r.append(" Filhos: ");
49.             for (Node filho : filhos.values()) {
50.                 r.append(filho.getGuerreiro()).append(", ");
51.             }
52.         }
53.
54.         r.append("\n");
55.
56.         // recursivooo
57.         for (Node filho : filhos.values()) {
58.             imprimirArvore(filho, r, prefix + "  ");
59.         }
60.     }
61. }
62.
63. @Override
64. public String toString() {
```

```
65.         StringBuilder r = new StringBuilder();
66.         imprimirArvore(this.root, r, "");
67.         return r.toString();
68.     }
69. }
```

Verifica-se que a classe Tree tem o atributo root que é a raiz da árvore e assim que a árvore é criada a raiz é definida como Null.

A classe possui os métodos necessários para verificação da árvore e montagem, assim como a impressão em forma de String.

Método isEmpty(): Verifica se a árvore está vazia, ou seja, se não tem raiz.

Método encontraRaiz(Map<String, Node> nodes): Percorre os nós e retorna aquele que não possui pai, logo sendo o raiz.

Método montarArvore(Map<String, Node> nodes): Constrói a árvore a partir de um dos nós, associando cada nó ao seu pai correspondente. Além disso, ele descobre o raiz e atribui a primeira linha do pergaminho ao seu numero de terras.

Método imprimirArvore(Node node, StringBuilder r, String prefix): Método privado que realiza de forma recursiva e imprime a informação de cada Node. Esse método é chamado através do método publico toString().

Método toString(): Chama o método imprimir árvore.

A classe Tribo é responsável por realizar as operações de leitura do arquivo que será analisado e transformado em árvore, além de realizar os cálculos da herança de terras e de verificar quem é o guerreiro da ultima geração com a maior quantidade de terras. Possui os atributos currDir, onde verifica o diretório atual e a String relativo, onde representa o caminho relativo até o arquivo.

```
1. import java.io.BufferedReader;
2. import java.io.IOException;
3. import java.nio.charset.Charset;
4. import java.nio.file.Files;
5. import java.nio.file.Path;
6. import java.nio.file.Paths;
7. import java.util.HashMap;
8. import java.util.LinkedList;
9. import java.util.Map;
10. import java.util.Queue;
11.
12. public class Tribo {
13.     String currDir = System.getProperty("user.dir") + "\\";
```

```
14.    String relativo = "Alest-I-T2\\tribo\\src\\teste.txt";
15.
16.    public Map<String, Node> armazenaNodos() {
17.        Map<String, Node> nodes = new HashMap<>();
18.
19.        Path path = Paths.get(currDir + relativo);
20.
21.                    try (BufferedReader reader = Files.newBufferedReader(path,
Charset.forName("utf8"))) {
22.            String line;
23.            int numTerrasPai = Integer.parseInt(reader.readLine());
24.
25.            while ((line = reader.readLine()) != null) {
26.                String[] partes = line.split("\\s+");
27.                if (partes.length == 3) {
28.                    String pai = partes[0];
29.                    String filho = partes[1];
30.                    int terrasFilho = Integer.parseInt(partes[2]);
31.
32.                        Node paiNode = nodes.computeIfAbsent(pai, k -> new Node(pai,
terrasFilho));
33.                     Node filhoNode = nodes.computeIfAbsent(filho, k -> new Node(filho,
terrasFilho));
34.
35.                    paiNode.adicionarFilho(filho, filhoNode);
36.                }
37.            }
38.
39.        } catch (IOException e) {
40.            System.out.println("Erro na leitura: " + e.getMessage());
41.        }
42.
43.        return nodes;
44.    }
45.
```

```
46.    public int terrasPai() {
47.
48.        int numTerrasPai = -1;
49.        Path path = Paths.get(currDir + relativo);
50.
51.                    try (BufferedReader reader = Files.newBufferedReader(path,
Charset.forName("utf8"))) {
52.
53.            numTerrasPai = Integer.parseInt(reader.readLine());
54.
55.        } catch (IOException e) {
56.            System.out.println("Erro na leitura: " + e.getMessage());
57.        }
58.        return numTerrasPai;
59.    }
60.
61.    public void calculaHerancaFilho(Map<String, Node> nodes) {
62.        Tree calc = new Tree();
63.        calc.montarArvore(nodes);
64.        Node ultimaGeraMaiorTerras = null;
65.        int ultimoMaisTerras = 0;
66.        int maiorCamada = 0;
67.        Node raiz = calc.encontraRaiz(nodes);
68.
69.        Queue<Node> fila = new LinkedList<>();
70.        fila.add(raiz);
71.
72.        while (!fila.isEmpty()) {
73.            int tamanhoNivel = fila.size();
74.
75.            for (int i = 0; i < tamanhoNivel; i++) {
76.                Node node = fila.poll();
77.                node.setCamada(maiorCamada);
78.
```

```
79.             if (!node.getFilhos().isEmpty()) {
80.                 int herancaPorFilho = node.getTerras() / node.getFilhos().size();
81.                 int resto = node.getTerras() % node.getFilhos().size();
82.
83.                 for (Node filho : node.getFilhos().values()) {
84.                     int herancaIndividual = herancaPorFilho;
85.                     filho.setTerras(filho.getTerras() + herancaIndividual);
86.
87.                     if (resto > 0) {
88.                         filho.setTerras(filho.getTerras() + 1);
89.                         herancaIndividual += 1;
90.                         resto--;
91.                     }
92.
93.                 System.out.println(filho.getGuerreiro() + " herda: " + herancaIndividual
+ " /depois da herança: " + filho.getTerras());
94.
95.                     fila.add(filho);
96.                 }
97.             }
98.         }
99.         maiorCamada++;
100.     }
101.
102.     for (Node node : nodes.values()) {
103.                 if (node.getCamada() == maiorCamada - 1 && node.getTerras() >
ultimoMaisTerras) {
104.             ultimaGeraMaiorTerras = node;
105.             ultimoMaisTerras = node.getTerras();
106.         }
107.     }
108.
109.         System.out.println("\nO guerreiro com o maior numero de terras na ultima
geração: ");
110.     if (ultimaGeraMaiorTerras != null) {
```

```
111.                System.out.println(ultimaGeraMaiorTerras.getGuerreiro() + "\nTerras conquistadas mais a herança: " + ultimaGeraMaiorTerras.getTerras());
112.        }
113.    }
114. }
```

A classe possui os métodos armazenaNodos, terrasPai e calculaHerancaFilho.

Método armazenaNodos(): lê o arquivo de texto(.txt) do formato definido anteriormente que é: pai – filho- número de terras do filho e a primeira linha é o número de terras do pai. Então cria um mapa da classe Node, onde cada node representa um guerreiro e já associa seus pais.

Método terrasPai(): Lê o numero de terras do pai para atribuir ao Node certo esse número quando for montar a árvore. Esse método é chamado quando a árvore é montada.

Método calculaHerancaFilho (Map<String, Node> nodes): Utiliza um objeto da classe Tree e monta a árvore, após montar a árvore o raiz é encontrado, pois o algoritmo utiliza uma fila e adiciona o raiz. Então um loop até que a fila esteja vazia, logo o algoritmo no início do while verifica o tamanho da fila para ler os nodes por camadas e remove o elemento lido da fila e verifica se o Nodo atual possui filhos, e faz o cálculo da herança, caso a divisão de terras do pai pelo número de filhos tenha um resto diferente de 0, adiciona uma terra para cada filho até que o resto seja 0. Após isso o filho é adicionado na fila para ser lido e sua herança distribuída caso ele tenha herdeiros. Quando todos os nodos forem lidos, a fila estará vazia e sairá do while. Logo depois, entra em outro loop para fazer a verificação de qual guerreiro da última geração possui mais terras e imprime a informação após a varredura.

Então a classe App é a responsável pela logica que executa o programa.

```
1. import java.util.Map;
 2. import java.util.Scanner;
 3.
 4. public class App {
 5.     public static void main(String[] args) throws Exception {
 6.         Scanner sc = new Scanner(System.in);
 7.
 8.         while (true) {
 9.                 System.out.println("Menu: \n1. Mostrar pai-filho\n2. Mostrar as terras herdadas e o guerreiro da última geração que obteve a maior quantidade de terras finais\n3. Sair");
10.             int escolha = Integer.parseInt(sc.nextLine());
11.             if (escolha == 1) {
12.                 Tribo teste = new Tribo();
```

```
13.             Map<String, Node> nodes = teste.armazenaNodos();
14.             Tree arvore = new Tree();
15.             arvore.montarArvore(nodes);
16.             System.out.println(arvore.toString());
17.         } else if (escolha == 2) {
18.             Tribo teste = new Tribo();
19.             Map<String, Node> nodes = teste.armazenaNodos();
20.             Tree arvore = new Tree();
21.             arvore.montarArvore(nodes);
22.             teste.calculaHerancaFilho(nodes);
23.         } else if (escolha == 3) {
24.             break;
25.         } else {
26.             System.out.println("Escolha uma opção válida");
27.         }
28.     }
29.     sc.close();
30. }
```

Onde o programa é executado e a aplicação inicia e para o usuário vai direto a um loop, onde ele determina se quer ver a árvore em formato de String, ver as heranças e o guerreiro com mais terras ou se ele quer sair do programa.

5. Como utilizar o programa

Para utilizar o programa é necessário abrir o arquivo onde estão as classes e o pergaminho (arquivo de texto) a ser analisado na IDE java de sua preferência, mas como esse programa foi desenvolvido através do VSCode é recomendado utilizar essa IDE, pois essa seção irá descrever o uso do programa através dele.

Primeiramente, é necessário abrir o Visual Studio Code e em seguida abrir a pasta onde está localizado o arquivo, conforme a figura 2.

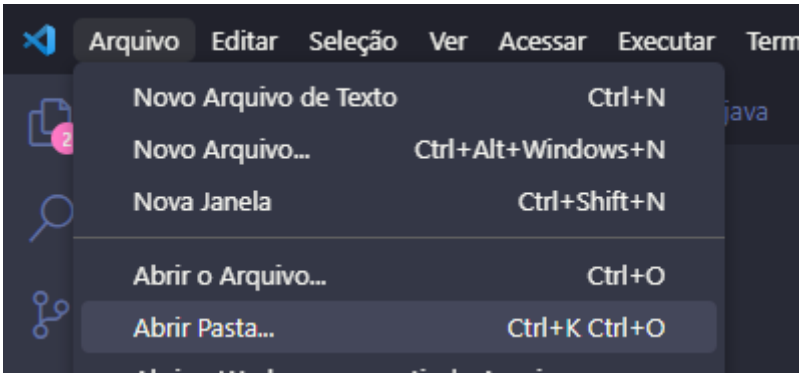

Figura 2 - Abrir pasta

Após esses passos o explorador de arquivos estará aberto com a pasta do programa, então será necessário ir em AlestI e em seguida abrir o arquivo tribo conforme a figura 3. Esse passo é muito importante, pois é necessário abrir o arquivo tribo no Visual Studio Code para funcionar corretamente.

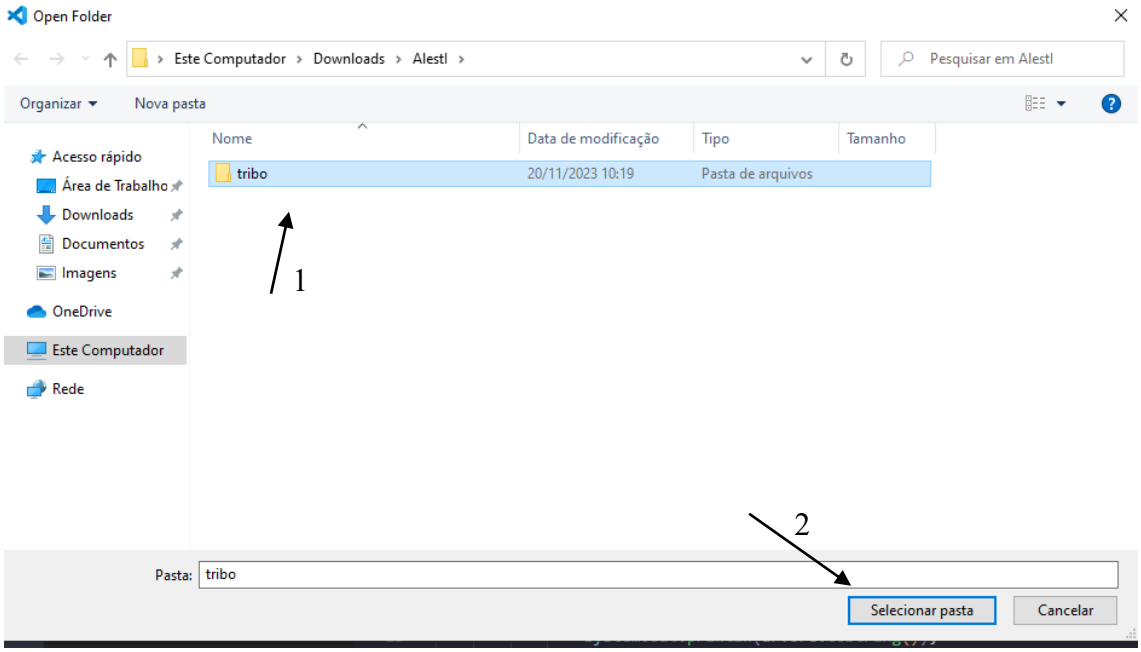

Figura 3 - Selecionando arquivo

Após isso selecione “src” e depois a classe App e o arquivo teste.txt você deve inserir os dados da tribo que o usuário deseja no txt teste, conforme a figura 4. Observação, o arquivo a ser lido sempre deve ser esse e deve ser modificado para cada tribo. Caso o usuário queira ler outro arquivo é preciso modificar o caminho do arquivo na classe tribo e, então, seguir com a execução do programa. Onde alterar está descrito na figura 7.

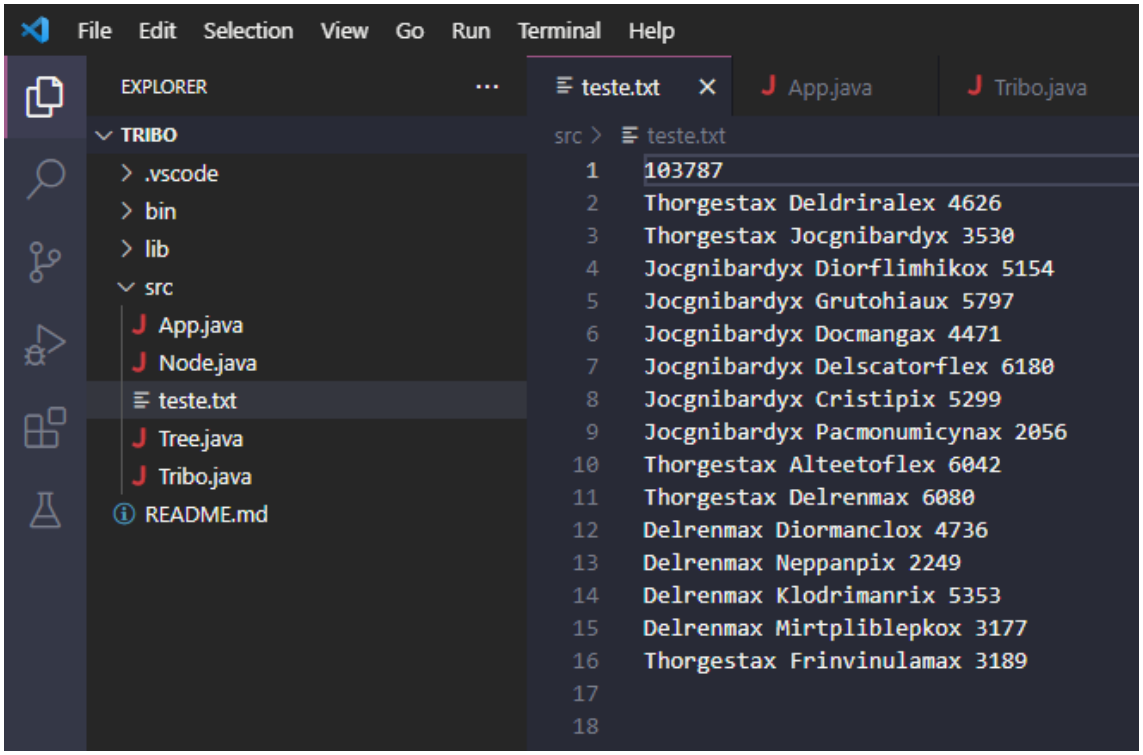

Figura 4 - Abrindo o arquivo de texto

Após configurar o arquivo você deve ir para a classe App e para fins de facilitar para o usuário final, a própria IDE fornece uma ferramenta de compilação sem que o usuário precise compilar e executar o programa no terminal. Essa ferramenta está disponível no canto superior direito da IDE como mostra a figura 5.

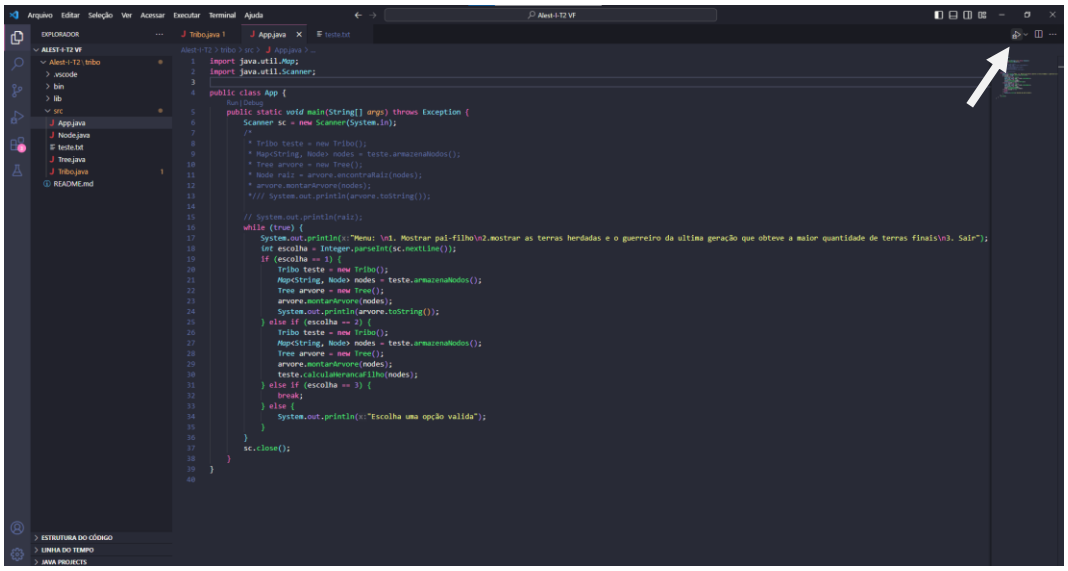

Figura 5 - Iniciar e compilar com a ferramenta o programa

Após iniciar o programa aparecera o menu no terminal. Então é só escolher as opções que deseja analisar na tribo, como mostra a figura 6.

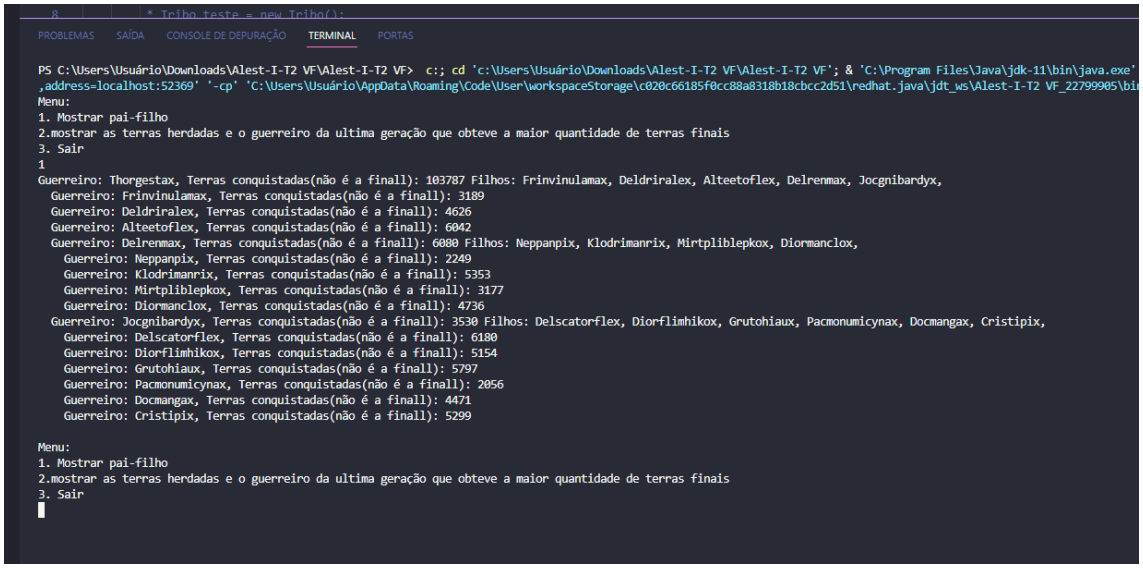

Figura 6 - Programa em execução

Pode ocorrer algum erro na leitura de arquivo, caso o programa seja aberto de forma diferente ou para a leitura de outro arquivo, mas é facilmente corrigido acessando a classe Tribo e corrigindo os caminhos para a leitura do arquivo nas variáveis currDir e relativo indicados na figura 7.

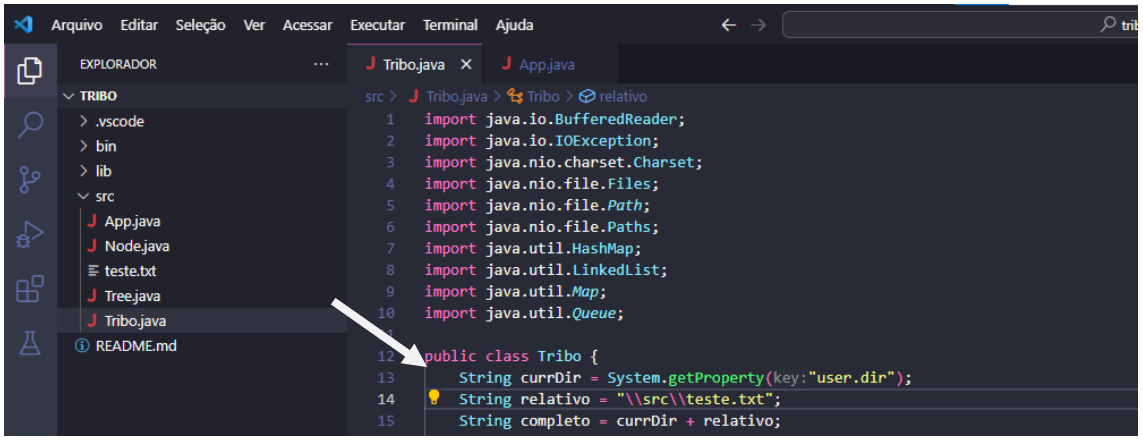

Figura 7 - Alteração do caminho do arquivo

6. Testes

6.1 Primeiro teste

O primeiro teste realizado foi para analisar a tribo exemplo da documentação do trabalho. A figura 8 representa a descrição da tribo analisada.

103787
Thorgestax Deldriralex 4626
Thorgestax Jocgnibardyx 3530

Jocgnibardyx Diorflimhikox 5154

Jocgnibardyx Grutohiaux 5797

Jocgnibardyx Docmangax 4471

Jocgnibardyx Delscatorflex 6180

Jocgnibardyx Cristipix 5299

Jocgnibardyx Pacmonumicynax 2056

Thorgestax Alteetoflex 6042

Thorgestax Delrenmax 6080

Delrenmax Diormanclox 4736

Delrenmax Neppanpix 2249

Delrenmax Klodrimanrix 5353

Delrenmax Mirtpliblepkox 3177

Thorgestax Frinvinulamax 3189

Figura 8 - Tribo analisada no primeiro teste

Após a execução do programa obteve-se os resultados esperados, assim mostrou a árvore corretamente e as heranças foram distribuídas igualmente, exceto quando o resto da divisão era diferente de zero, assim conforme descrito anteriormente, adiciona 1 para os primeiros filhos lidos, até zerar o resto de terras.

A figura 9 e a figura 10 mostram a execução de decisão do usuário no menu. Nesse contexto, exibe a árvore e as heranças, também o guerreiro com a maior herança. Nesse caso o guerreiro com a maior herança foi Klodrimanrix com 12062 terras conquistas mais a herança que vem desde a distribuição de terras do seu avô com seu pai e tios.

```
Menu:
1. Mostrar pai-filho
2.mostrar as terras herdadas e o guerreiro da ultima geração que obteve a maior quantidade de terras finais
3. Sair
1
Guerreiro: Thorgestax, Terras conquistadas(não é a finall): 103787 Filhos: Frinvinulamax, Deldriralex, Alteetoflex, Delrenmax, Jocgnibardyx,
  Guerreiro: Frinvinulamax, Terras conquistadas(não é a finall): 3189
  Guerreiro: Deldriralex, Terras conquistadas(não é a finall): 4626
  Guerreiro: Alteetoflex, Terras conquistadas(não é a finall): 6042
  Guerreiro: Delrenmax, Terras conquistadas(não é a finall): 6080 Filhos: Neppanpix, Klodrimanrix, Mirtpliblepkox, Diormanclox,
    Guerreiro: Neppanpix, Terras conquistadas(não é a finall): 2249
    Guerreiro: Klodrimanrix, Terras conquistadas(não é a finall): 5353
    Guerreiro: Mirtpliblepkox, Terras conquistadas(não é a finall): 3177
    Guerreiro: Diormanclox, Terras conquistadas(não é a finall): 4736
  Guerreiro: Jocgnibardyx, Terras conquistadas(não é a finall): 3530 Filhos: Delscatorflex, Diorflimhikox, Grutohiaux, Pacmonumicynax, Docmangax, Cristipix,
    Guerreiro: Delscatorflex, Terras conquistadas(não é a finall): 6180
    Guerreiro: Diorflimhikox, Terras conquistadas(não é a finall): 5154
    Guerreiro: Grutohiaux, Terras conquistadas(não é a finall): 5797
    Guerreiro: Pacmonumicynax, Terras conquistadas(não é a finall): 2056
    Guerreiro: Docmangax, Terras conquistadas(não é a finall): 4471
    Guerreiro: Cristipix, Terras conquistadas(não é a finall): 5299
```

Figura 9 - Exibição da árvore do primeiro teste

```
Menu:
1. Mostrar pai-filho
2.mostrar as terras herdadas e o guerreiro da ultima geração que obteve a maior quantidade de terras finais
3. Sair
2
Frinvinulamax herda: 20758 /depois da herança: 23947
Deldriralex herda: 20758 /depois da herança: 25384
Alteetoflex herda: 20757 /depois da herança: 26799
Delrenmax herda: 20757 /depois da herança: 26837
Jocgnibardyx herda: 20757 /depois da herança: 24287
Neppanpix herda: 6710 /depois da herança: 8959
Klodrimanrix herda: 6709 /depois da herança: 12062
Mirtpliblepkox herda: 6709 /depois da herança: 9886
Diormanclox herda: 6709 /depois da herança: 11445
Delscatorflex herda: 4048 /depois da herança: 10228
Diorflimhikox herda: 4048 /depois da herança: 9202
Grutohiaux herda: 4048 /depois da herança: 9845
Pacmonumicynax herda: 4048 /depois da herança: 6104
Docmangax herda: 4048 /depois da herança: 8519
Cristipix herda: 4047 /depois da herança: 9346

O guerreiro com o maior numero de terras na ultima geração:
Klodrimanrix
Terras conquistadas mais a herança: 12062
```

Figura 10 - Exibição de heranças e o guerreiro da última geração com mais terras

6.2 Segundo teste

Para realização do segundo teste, foi utilizada a tribo da figura 11. A tribo utilizada é baseada na mitologia grega.

```
80
Gaia Cronos 20
Erebo Eter 10
Erebo Hemera 8
Caos Gaia 20
Caos Erebo 18
Cronos Zeus 24
Cronos Hades 14
Gaia Japeto 12
Japeto Prometeu 4
Cronos poseidon 8
```

Figura 11 - Tribo segundo teste

Depois da execução do teste, também se obteve os resultados esperados conforme a figura 12 e a figura 13, logo constatou-se que o guerreiro da última geração com maior número de terras é o Prometou que teve a soma das terras conquistas mais a herança de 46 terras.

```
Menu:
1. Mostrar pai-filho
2.mostrar as terras herdadas e o guerreiro da ultima geração que obteve a maior quantidade de terras finais
3. Sair
1
Guerreiro: Caos, Terras conquistadas(não é a finall): 80 Filhos: Gaia, Erebo,
  Guerreiro: Gaia, Terras conquistadas(não é a finall): 20 Filhos: Japeto, Cronos,
    Guerreiro: Japeto, Terras conquistadas(não é a finall): 12 Filhos: Prometeu,
      Guerreiro: Prometeu, Terras conquistadas(não é a finall): 4
    Guerreiro: Cronos, Terras conquistadas(não é a finall): 20 Filhos: Zeus, Hades, poseidon,
      Guerreiro: Zeus, Terras conquistadas(não é a finall): 24
      Guerreiro: Hades, Terras conquistadas(não é a finall): 14
      Guerreiro: poseidon, Terras conquistadas(não é a finall): 8
  Guerreiro: Erebo, Terras conquistadas(não é a finall): 10 Filhos: Hemera, Eter,
    Guerreiro: Hemera, Terras conquistadas(não é a finall): 8
    Guerreiro: Eter, Terras conquistadas(não é a finall): 10
```

Figura 12 - Árvore do segundo teste

```
Menu:
1. Mostrar pai-filho
2.mostrar as terras herdadas e o guerreiro da ultima geração que obteve a maior quantidade de terras finais
3. Sair
2
Gaia herda: 40 /depois da herança: 60
Erebo herda: 40 /depois da herança: 50
Japeto herda: 30 /depois da herança: 42
Cronos herda: 30 /depois da herança: 50
Hemera herda: 25 /depois da herança: 33
Eter herda: 25 /depois da herança: 35
Prometeu herda: 42 /depois da herança: 46
Zeus herda: 17 /depois da herança: 41
Hades herda: 17 /depois da herança: 31
poseidon herda: 16 /depois da herança: 24

O guerreiro com o maior numero de terras na ultima geração:
Prometeu
Terras conquistadas mais a herança: 46
```

Figura 13 - Heranças do segundo teste

6.3 Terceiro teste

O terceiro teste foi realizado para uma tribo de anões, onde o pergaminho encontrado é referente a figura 14.

```
104
Thror Thrain 48
Dain Thror 30
Dain Fror 18
Thrain Thorin 4
Dain Borin 22
Thrain Frelin 2
Borin Farin 18
Farin Fundin 4
Farin Groin 10
```

Figura 14 - Tribo terceiro teste

Posteriormente a execução do teste, alcançou-se a exibição da árvore de acordo com a figura 15 conseguiu-se o resultado do ultimo guerreiro com mais terras foi Thorin com 69 terras conquistas, como mostra a figura 16.

```
Menu:
1. Mostrar pai-filho
2.mostrar as terras herdadas e o guerreiro da ultima geração que obteve a maior quantidade de terras finais
3. Sair
1
Guerreiro: Dain, Terras conquistadas(não é a finall): 104 Filhos: Thror, Borin, Fror,
  Guerreiro: Thror, Terras conquistadas(não é a finall): 48 Filhos: Thrain,
    Guerreiro: Thrain, Terras conquistadas(não é a finall): 48 Filhos: Frelin, Thorin,
      Guerreiro: Frelin, Terras conquistadas(não é a finall): 2
      Guerreiro: Thorin, Terras conquistadas(não é a finall): 4
  Guerreiro: Borin, Terras conquistadas(não é a finall): 22 Filhos: Farin,
    Guerreiro: Farin, Terras conquistadas(não é a finall): 18 Filhos: Groin, Fundin,
      Guerreiro: Groin, Terras conquistadas(não é a finall): 10
      Guerreiro: Fundin, Terras conquistadas(não é a finall): 4
  Guerreiro: Fror, Terras conquistadas(não é a finall): 18
```

Figura 15 - Árvore do terceiro teste

```
Menu:
1. Mostrar pai-filho
2.mostrar as terras herdadas e o guerreiro da ultima geração que obteve a maior quantidade de terras finais
3. Sair
2
Thror herda: 35 /depois da herança: 83
Borin herda: 35 /depois da herança: 57
Fror herda: 34 /depois da herança: 52
Thrain herda: 83 /depois da herança: 131
Farin herda: 57 /depois da herança: 75
Frelin herda: 66 /depois da herança: 68
Thorin herda: 65 /depois da herança: 69
Groin herda: 38 /depois da herança: 48
Fundin herda: 37 /depois da herança: 41

O guerreiro com o maior numero de terras na ultima geração:
Thorin
```

Figura 16 - Heranças segundo teste

6.4 Quarto teste

Em relação ao quarto teste, foi analisada o pergaminho que obtinha os dados em forma de hierarquia, por exemplo Avô – pai1 e Avô Pai2 conforme na figura 17, também continha um guerreiro apenas na última geração para verificar se ele iria constar no resultado esperado.

```
100
Pai1 Filho1P1 20
Avô Pai2 24
Avô Pai1 22
Pai1 Filho2P1 8
Pai1 filho3P1 12
Pai2 Filho1P2 10
Avô Pai3 18
```

```
Pai1 Filho4P1 8
Pai2 Filho2P2 4
Pai2 Filho3P2 12
Filho1P1 Neto 2
```

Figura 17 - Tribo quarto teste

Posteriormente aos testes, obteve-se os resultados esperados, com a distribuição da árvore de forma correta, de acordo com a figura 18 e também confirmou que o guerreiro da última geração seria o Neto, pois era o único integrante da última geração com 36 terras totais como é possível visualizar na figura 19.

```
Menu:
1. Mostrar pai-filho
2.mostrar as terras herdadas e o guerreiro da ultima geração que obteve a maior quantidade de terras finais
3. Sair
1
Guerreiro: Avô, Terras conquistadas(não é a finall): 100 Filhos: Pai1, Pai3, Pai2,
  Guerreiro: Pai1, Terras conquistadas(não é a finall): 20 Filhos: Filho1P1, filho3P1, Filho2P1, Filho4P1,
    Guerreiro: Filho1P1, Terras conquistadas(não é a finall): 20 Filhos: Neto,
      Guerreiro: Neto, Terras conquistadas(não é a finall): 2
    Guerreiro: filho3P1, Terras conquistadas(não é a finall): 12
    Guerreiro: Filho2P1, Terras conquistadas(não é a finall): 8
    Guerreiro: Filho4P1, Terras conquistadas(não é a finall): 8
  Guerreiro: Pai3, Terras conquistadas(não é a finall): 18
  Guerreiro: Pai2, Terras conquistadas(não é a finall): 24 Filhos: Filho1P2, Filho2P2, Filho3P2,
    Guerreiro: Filho1P2, Terras conquistadas(não é a finall): 10
    Guerreiro: Filho2P2, Terras conquistadas(não é a finall): 4
    Guerreiro: Filho3P2, Terras conquistadas(não é a finall): 12
```

Figura 18 - Árvore quarto teste

```
Menu:
1. Mostrar pai-filho
2.mostrar as terras herdadas e o guerreiro da ultima geração que obteve a maior quantidade de terras finais
3. Sair
2
Pai1 herda: 34 /depois da herança: 54
Pai3 herda: 33 /depois da herança: 51
Pai2 herda: 33 /depois da herança: 57
Filho1P1 herda: 14 /depois da herança: 34
filho3P1 herda: 14 /depois da herança: 26
Filho2P1 herda: 13 /depois da herança: 21
Filho4P1 herda: 13 /depois da herança: 21
Filho1P2 herda: 19 /depois da herança: 29
Filho2P2 herda: 19 /depois da herança: 23
Filho3P2 herda: 19 /depois da herança: 31
Neto herda: 34 /depois da herança: 36

O guerreiro com o maior numero de terras na ultima geração:
Neto
Terras conquistadas mais a herança: 36
```

Figura 19 - Heranças quarto teste

7. Conclusão

É evidente, portanto, que o programa desenvolvido é eficiente para analisar os pergaminhos de tribos antigas e, assim, criar uma árvore de forma correta e precisa com

a formatação do pergaminho, ademais, também é possível fazer o calculo das heranças de forma correta e responder quem é o guerreiro da ultima geração com a maior quantidade de terras.

Referencias

[1] GOODRICH,Michael.T. e TAMASSIA,Roberto. 2004. Projeto de Algoritmos. Porto Alegre. Bookman.

[2] MIRO(2023). Disponível em https://miro.com/app/dashboard/ Consultado Novembro 2023.

[3] HORSTMANN, C. Java for everyone: late objects. 2nd ed. Danvers: Wiley, 2013.