Tribo Bárbara

Gabriel F. Stundner[*], Gustavo L. Vidaletti[†]
Faculdade de Informática — PUCRS

08 de Março de 2019

Resumo

Este artigo descreve alternativas de solução para o primeiro problema proposto na disciplina de Algoritmos e Estrutura de Dados II no semestre 2019/1, que se trata sobre o número de terras herdadas e conquistadas por uma Tribo barbara, onde desejamos saber quantas terras o ultimo descendente da tribo possuiu em sua vida e qual foi a ultima geração, para isso foram feitos testes procurando uma solução eficiente para poder descobrir o herdeiro.

Introdução

Foram encontrados pergaminhos antigos de uma tribo Barbara que possui a informacao do numero de terras que o chefe mais velho da tribo, da primeira geração conquistou em sua vida e todos os filhos e pais das gerações seguintes, onde a primeira linha do pergaminho e o numero de terras que o primeiro chefe da primeira geração possuia, e esse numero deve ser dividido a todos os filhos deles e dos filhos deles,etc, até o último filho da última geração, em cada geração os filhos de cada chefe da tribo também conquistaram terras para si, onde depois que o chefe falecia, era divida as terras a cada filho, sendo então somado ao total de terras do filho. A ideia do programa é pegar esses pergaminhos, le-los e depois apresentar quem foi o ultimo descendente da tribo e quantas terras ele possuiu no final.

Construção do Programa

Existem 4 Classes que interagem entre si para fazer o programa funcionar:

1. Guerreiro.java = Esta Classe serve para Construir um objeto com as informações dos Guerreiros retirados dos arquivos de teste

2. Connection.java = Esta Classe recebe duas Strings e organiza. elas para ser facil de fazer as conexões dos nomes para ser chamado ao graphviz.

3. Graph.java = Esta Classe serve para poder construir o arquivo para ser lido pelo graphviz, onde ele necessita da classe Connection para poder organizar o nome da seguinte forma:

$$Objetopai- > ObjetoFilho$$

4. Gerenciador.java = Esta Classe é a principal classe do trabalho, onde ele lê a informação do Arquivo escolhido e calcula o tempo de execução e separa as informações de cada linha e armazena na Classe Guerreiro.

[*]gabriel.stundner@acad.pucrs.br
[†]gustavo.vidaletti@acad.pucrs.br

5. main.java = Esta Classe que será utilizada para compilar e testar nosso programa, ela é bem simples porque ela utiliza a Classe do Gerenciador para fazer tudo que é preciso.

Classe Main

A classe Main inicia um novo Objeto da Classe Gerenciador, onde recebe como Parâmetro o somente o nome do Caso Simplificado, como abaixo:

$$NOMEDOARQUIVO : casoMC4a.txt$$

$$NOMEPASSADOCOMOPARAMETRO : MC4a$$

Assim somente é pego o nome principal do arquivo e o resto(caso e .txt) é depois adicionado na Classe Gerenciador.

Classe Gerenciador

A principal Classe do projeto, onde tudo que é necessario é feito.

Após chamado a Classe Gerenciador na Main pelo construtor principal da Classe:

```
public Gerenciador(String caso) this.carregarTeste(caso);
```

dentro desse mesmo Construtor é chamado o Método *carregarTeste(String caso)* com a String colocada como Parâmetro la na Classe Main Este Método faz os Seguintes passos:

1. Pega o Caminho para o arquivo entrado como uma String, colocando as partes necessarias para poder chamar o arquivo armazenado dentro do Diretorio geral do programa, como no codigo abaixo:

```
Path path2 = Paths.get("Barbaros/casos/caso"+ caso
+".txt");
```

2. Agora iremos fazer um **Try/Catch**, onde na estrutura do Try iremos verificar se esta em UTF8 e subdividir a cada Espaço que aparece no arquivo, assim podemos pegar separadamente cada informação dentro do Arquivo. Dentro desse Try também será lido cada linha e feito a conta do tempo que ira levar para ler o arquivo.

```
try (Scanner sc = new Scanner(Files.newBufferedReader(
        path2, Charset.forName("utf8"))).useDelimiter("[\n]")) {

    while (sc.hasNext()){
        this.linhas.add(sc.nextLine());
    }
    long ti = System.currentTimeMillis();
    this.runTeste();
    long tf = System.currentTimeMillis();
    System.out.println("Tempo do teste: " + 1.0 * ((tf - ti) / 1000)
    System.out.println("Gerando digraph");
    this.diGraph(caso);

} catch (IOException e) {
    System.err.println("Erro de arquivos!");
} catch (NumberFormatException number){
    System.err.println("Erro de formatação de números!");
}
```

3. Neste mesmo try, temos a chamada do Método *diGraph()*, que serve para construir o arquivo depois no graphviz, onde chama a Classe Graph, que será explicada mais tarde, o Método em Si na Classe Gerenciador é como abaixo:

```
public void diGraph(String caso){
    Graph graph = new Graph();
    for(String linha: this.linhas){
        if(!linha.matches("(\\w+(\\s)\\w+(\\s)\\w+)")){
            continue;
        }
        String[] linhaSeparada = linha.split("\\s+");
        graph.createConnection(linhaSeparada[0],linhaSeparada[1]);
    }
    graph.geraArquivo(caso);

}
```

4. Após lido as Informacoes do Arquivo, iremos trabalhar com elas, mas antes disso iremos criar duas variaveis para nos auxiliar nesse trabalho, a primeira dela é o inteiro **maiorGeracao** e o

Arraylist dos Guerreiros pertencentes a essa geração, chamado **pertencentes**

5. Criamos então o Metodo void chamado **runTeste()** onde irá pegar cada informação do arquivo lido da seguinte forma:

   (a) iremos pegar a primeira linha do arquivo e armazenar o total de terras do primeiro guerreiro e o armazenar em uma Variavel e depois remove-la da leitura

   (b) iremos inicializar o ArrayList que iremos armazenar cada filho daquela geração.

   (c) iremos separar as informações da linha e iremos utilizar cada pedaço dela como ja foi dito como era a construção de cada linha, como abaixo

   ```
   nomepai nomeFilho numeroTerrasFilho
   ```

   (d) iremos armazenar a primeira parte da linha cortada em pedaços em um novo objeto Guerreiro, que chamaremos de Guerreiro pai, onde iremos procurar cada filho que este nome possui como Filho, com o Método *procurarFilhoPorNome(String nome)*

   (e) iremos pegar o numero de terras(que é a terceira parte da linha) e transformaremos a String em um inteiro e iremos armazenar dentro variavel **terras**.

   (f) pegamos a Segunda parte da linha, que é o nome do filho do pai(nome da primeira parte da linha) e iremos criar um Objeto para ele e iremos verificar se esse filho possui filhos

   (g) apos feito isso, temos que orgnizar a ordem de gerações, com um if/else para verificar cada geração e coloca-los em ordem, se o filho for de uma geracao maior, a maior geracao sera atualizada e criamos um novo ArrayList para adicionar os pertencentes daquela geracao, senao somente iremos adicionar o filho para os pertencentes de sua geração atual.

   (h) Abaixo uma foto do Método Citado

```java
public void runTeste(){
    String primeiraLinha = this.linhas.remove(0);
    int terrasIniciais = Integer.parseInt(primeiraLinha);

    this.pertencentes = new ArrayList<>();

    for(String linha: this.linhas){
        String[] linhaSeparada = linha.split("\\s+");

        if(this.raiz == null) {
            this.raiz = new Guerreiro(null,linhaSeparada[0],terrasIniciais);
        }

        Guerreiro pai = raiz.procurarFilhoPorNome(linhaSeparada[0]);
        int terras = Integer.parseInt(linhaSeparada[2]);
        Guerreiro filho = pai.procurarSomenteFilhoDestePorNome(linhaSeparada[1]);
        if(filho == null){
            filho = new Guerreiro(pai,linhaSeparada[1],terras);
            if(filho.getGeracao() > maiorGeracao){
                this.maiorGeracao = filho.getGeracao();
                pertencentes = new ArrayList<>();
            }
            if(filho.getGeracao() == maiorGeracao){
                pertencentes.add(filho);
            }

        }
    }
```

(i) agora iremos pegar o pai raiz e iremos utilizar um método da Classe Guerreiros que ira pegar o numero de terras do pai e ira dividir o valor de terras entre seus filhos, que será somado ao numero de terras de cada filho, este Método será explicado depois na Classe Guerreiro

(j) Iremos imprimir a maior geração do Arquivo, onde esse valor foi sempre atualizado na variavel **maiorGeracao**.

```
System.out.println("Maior geracao:  "+ maiorGeracao);
```

(k) Iremos imprimit tambem na tela o numero total de guerreiros que existem nessa Geração, como abaixo:

```
System.out.println("Numero de guerreiros nela "+
pertencentes.size());
```

(l) Agora iremos verificar qual o Guerreiro que possui mais terras dessa Geração, onde ele ira verificar o numero de terras de todos os guerreiros da geracao e armazenar separadamente quem possui mais terras para imprimir na tela, Após isso, somente iremos imprimir na tela o nome e o numero de terras

(m) a foto abaixo ira mostrar o codigo como explicado acima:

```
raiz.dividirTerrasEntreFilhosRecursivamente();
System.out.println("Maior geracao: " + maiorGeracao);
System.out.println("Numero de guerreiros nela " + pertencentes.size());
if(pertencentes.size() == 0) return;
Guerreiro maior = pertencentes.get(0);
for(Guerreiro g: pertencentes){
    if(g.getTerras() > maior.getTerras()){
        maior = g;
    }
}
System.out.println("Maior guerreiro: " + maior.getNome() + ", com " + maior.getTerras() + " terras");
```

Classe Guerreiro

Esta Classe é onde é construído e faz manutenção dos objetos Guerreiro

As informações necessárias para fazer um Objeto do tipo Guerreiro são:

(a) Objeto Guerreiro Pai

(b) String do nome do Guerreiro

(c) numero inteiro de Terras

(d) Lista de Objetos Guerreiro com os Filhos desse Guerrieiro

(e) Inteiro com a Geração do Filho, começando no 1

Possui um contrutor que orienta as informações entradas e atualiza a geração se o pai não for nulo, como mostrado na imagem abaixo:

```java
public Guerreiro(Guerreiro pai, String nome, int terras){
    this.pai = pai;
    this.nome = nome;
    this.terras = terras;
    this.filhos = new ArrayList<>();
    if(this.pai != null){
        this.geracao = pai.getGeracao() + 1;
        this.pai.pushFilhos(this);
    }
}
```

Como em toda Classe Orientada a Objetos possui os Metodos Getters e Setters para podermos mexer nas informações armazenadas:

```java
public Guerreiro getPai() {
    return pai;
}

public void setPai(Guerreiro pai) {
    this.pai = pai;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public int getTerras() {
    return terras;
}

public void setTerras(int terras) {
    this.terras = terras;
}

public void aumentarTerras(int terrasNovas){
    this.terras += terrasNovas;
}

public List<Guerreiro> getFilhos() {
    return filhos;
}
```

nesses Getters e Setters básicos tem um extra que aumenta o numero de terras, que ira servir para somar com o numero de terras do pai que será somado junto com o numero total de terras do filho

Abaixo temos os Métodos onde adicionamos os filhos e organizamos a Geração do Guerreiro

```java
public void pushFilhos(Guerreiro... filhos){
    this.filhos.addAll(Arrays.asList(filhos));
}

public int getGeracao() {
    return geracao;
}

public void setGeracao(int geracao) {
    this.geracao = geracao;
}
```

Temos o Método falado anteriormente, que divide o numero de terras pelos filhos de forma recursiva, onde se o Guerreiro não tiver filhos, ele retorna com nada extra e se tiver iremos dividir o numero de terras pelo numero de filhos que este Guerreiro possui e aumentamos o numero de terras de cada filho

```java
public void dividirTerrasEntreFilhosRecursivamente(){
    if(filhos.size() == 0) return;
    int terrasPorFilho = (int) this.terras / this.filhos.size();
    for(Guerreiro filho: this.filhos){
        filho.aumentarTerras(terrasPorFilho);
        filho.dividirTerrasEntreFilhosRecursivamente();
    }
}
```

Possuimos também um Método para procurar o filho somente pelo nome, com isso podemos procurar quais filhos esse guerreiro possui mais Fácil

```java
public Guerreiro procurarSomenteFIlhoDestePorNome(String nome){
    for(Guerreiro filho: this.filhos){
        if(filho.getNome().equals(nome)){
            return filho;
        }
    }
    return null;
}
```

Possui tambem outro Método para procurar o filho pelo nome, mas diferente do Método anterior este procura de forma recursiva e procura o Objeto Guerreiro do Filho especifico, onde no outro só volta uma String do nome do Filho

```java
public Guerreiro procurarFilhoPorNome(String nome){
    if(this.getNome().equals(nome)){
        return this;
    }
    for(Guerreiro filho: this.filhos){
        Guerreiro aux = filho.procurarFilhoPorNome(nome);
        if(aux != null){
            return aux;
        }
    }
    return null;
}
```

Classe Connection

Esta Classe serve para organizar os nomes do Guerreiros para poder usar o Graphviz

É uma Classe Auxiliar bem Simples, ela recebe duas Strings e transforma em um formato que o Graphviz pode ler, como mostrado abaixo:

```
public class Connection {

    private String start;
    private String end;

    public Connection(String start, String end) {
        this.start = start;
        this.end = end;
    }

    public String getConnection(){
        return start + "->" + end + ";";
    }
}
```

Classe Graph

Esta Classe serve para podermos transformar nossas informações para o Graphviz

Graphviz é uma ferramenta para podermos construir Árvores e outras estruturas de forma bem rápida e Fácil, ela possui uma estrutura bem simples, onde esta nossa Classe serve para podermos construir o Árquivo para podermos colocar nesse programa

Nossa Classe possui Métodos bem simples

(a) Método Construtor = Constroi a Estrutura Básica que deve existir para depois colocar os elementos Necessários

```
public Graph(){
    inicio = "digraph G{";
    fim = "}";
    lista = new ArrayList<>();
    lista.add(inicio);
}
```

(b) Método getDigraph = Serve para pegarmos a informação construida sem erro de código

```
public ArrayList<String> getDiGraph(){
    ArrayList<String> temp = this.lista;
    temp.add(fim);
    return temp;
}
```

(c) Método createConnection = é o método que utilizamos para podermos criar as conexões entre pai e filho, onde iremos chamar a Classe **Connection** já apresentada para construirmos a estrutura do Graphviz

```
public void createConnection(String start, String end){
    Connection c = new Connection(start,end);
    this.lista.add(c.getConnection());
}
```

(d) Método gerarArquivo = É o método que irá pegar todas as informações e irá construir um arquivo **.gv** para ser lido no programa graphviz, onde este programa será armazenado em um Diretorio novo que será criado

```java
public void geraArquivo(String caso){
    File pastaItens = new File("Barbaros/Graph");
    if(!pastaItens.exists())pastaItens.mkdir();
    Path path1 = Paths.get("Barbaros/Graph/Caso" + caso + ".gv");
    try (PrintWriter arquivo = new PrintWriter(Files.newBufferedWriter(path1, Charset.forName("ut
        for(String s: getDiGraph()){
            arquivo.println(s);
        }
    } catch (IOException e) {
        System.err.println("Erro ao gerar o arquivo!");
    }
}
```

Resultados

Agora iremos apresentar todos os Resultados que foram testados a partir dos programas passados pelo professor

Caso MC4a.txt

Resposta:

```
Maior geracao: 4
Numero de guerreiros nela 11
Maior guerreiro: Criplivalmax, com 6228 terras
Tempo do teste: 0.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Caso MC4b.txt

Resposta:

```
Maior geracao: 6
Numero de guerreiros nela 6
Maior guerreiro: Racverdricynax, com 8013 terras
Tempo do teste: 0.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Caso MC8a.txt

Resposta:

```
Maior geracao: 12
Numero de guerreiros nela 35
Maior guerreiro: Flertolox, com 9295 terras
Tempo do teste: 0.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Caso MC8b.txt

Resposta:

```
Maior geracao: 13
Numero de guerreiros nela 9
Maior guerreiro: Pridenstitrotix, com 7876 terras
Tempo do teste: 0.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Caso MC10a.txt

Resposta:

```
Maior geracao: 23
Numero de guerreiros nela 6
Maior guerreiro: Cripliplinix, com 6827 terras
Tempo do teste: 3.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Caso MC10b.txt

Resposta:

```
Maior geracao: 17
Numero de guerreiros nela 17
Maior guerreiro: Retmanmantix, com 9187 terras
Tempo do teste: 0.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Caso MC12a.txt

Resposta:

```
Maior geracao: 19
Numero de guerreiros nela 4
Maior guerreiro: Pridentrolax, com 6850 terras
Tempo do teste: 5.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Caso MC12b.txt

Resposta:

```
Maior geracao: 18
Numero de guerreiros nela 16
Maior guerreiro: Grurenflimnox, com 7909 terras
Tempo do teste: 0.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Caso MC14a.txt

Resposta:

```
Maior geracao: 24
Numero de guerreiros nela 8
Maior guerreiro: Bercrabarstax, com 7086 terras
Tempo do teste: 67.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Caso MC14b.txt

Resposta:

```
Maior geracao: 21
Numero de guerreiros nela 11
Maior guerreiro: Nepmiccinpax, com 7176 terras
Tempo do teste: 19.0 segundos
Gerando digraph
```

Arquivo Graphviz Criado Esta Armazenado Junto com Este Relatório

Conclusões

Todos os Testes foram feitos em um Sansung intel Core i7 com placa de video Nvidia Geforce 720, o que ajudou no desempenho dos Algoritmos, onde temos que:

MELHOR CASO: Seria o Caso MC10b, porque possui 17 gerações de Guerreiros e foi Concluido em 0 segundos, não sabemos dizer em milisegundos, mas pelo numero de gerações e o tamanho do arquivo .txt dizemos que este foi o melhor caso adiquirido

PIOR CASO: Sem duvida o pior caso foi o MC14a, onde era de se esperar por um arquivo de 61812 linhas, mas o desempenho no meu computador mesmo assim foi de 67 segundos, mostrando que para um arquivo tão grande assim nosso algoritmo não seria eficiente, já que não usamos nenhuma Estrutura de Dados muito eficiente, fomos mais no touch que usando alguma estrutura mais eficiente que estudamos.

Então este foi o nosso trabalho, onde construimos uma estrutura bem simples mas não muito eficiente quando envolve arquivos muito grandes, mas funciona mesmo assim e entrega o que foi prometido, adquirindo uma estrutura descente.