Trabalho 1 - A Tribo Bárbara

Gabriel Bonatto Justo[1]

[1]Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

gabriel.justo@acad.pucrs.br

Resumo. *O objetivo deste trabalho é entender o comportamento de uma tribo bárbara. E através de pergaminhos deixados por eles, identificar o guerreiro que possuía maior quantidade de terras e qual o número de terras ele possuía. Os algoritmos desenvolvidos para decifrar os pergaminhos foram comparados e avaliados quanto aos seus tempos de execução.*

1. Introdução

Antropólogos necessitam de ajuda para entender o comportamento de uma tribo bárbara. Através de pergaminhos encontrados foram descobertos dois costumes dessa tribo. O primeiro costume diz que as terras de um guerreiro são distribuídas igualmente entre os filhos quando ele morre, o segundo diz que um filho pode acumular terras durante sua vida.

Este trabalho tem como objetivo decifrar os pergaminhos bárbaros e descobrir qual o guerreiro da última geração de bárbaros possui a maior quantidade de terras e quantas terras ele possuía. Para resolver este problema foram desenvolvidas três soluções. A primeira é explicada da Seção 2.1, a segunda na Seção 2.2 e a terceira na Seção 2.3, a Seção 3, expõe os resultados encontrados, e por fim a Seção 4 apresenta a conclusão a partir dos resultados obtidos.

2. Soluções

Considerando os dados de entrada fornecidos e os resultados esperados, o problema foi dividido em três partes. A primeira parte lê o arquivo de entrada e as informações de cada bárbaro são armazenadas em dois dicionários, um com todos os bárbaros que são pais de outro bárbaro chamado *pais*, e outro com todos os bárbaros que são filhos de outro bárbaro chamado *filhos*, de forma que o dicionário *pais* terá um bárbaro a mais, já que o primeiro guerreiro da tribo não tem registros de antecessores. A segunda etapa é responsável por encontrar o *primeiroGuerreiro*, para isso, cada elemento do dicionário *pais* é procurado no dicionário *filhos*, o elemento que não for encontrado no dicionário *filhos* é o *primeiroGuerreiro*, como mostra o Algoritmo 1. A terceira etapa é diferente para cada solução, elas serão apresentadas nas Seções abaixo.

Algoritmo 1: Encontrar raiz

```
1 início
2 |  para cadap ∈ pais faça
3 |  |  if !filhos.contém(p) then
4 |  |  |  retorna p
5 |  |  end
6 |  fim
7 fim
```

2.1. Primeira Solução

A primeira solução utiliza um dicionário chamado *bárbaros*, no qual a chave é o nome no bárbaro e o valor são os dados dele (número de terras e o pai) e uma *árvore* genérica. Essa solução inicia armazenando os dados de cada guerreiro no dicionário *bárbaros*.

Para formar a árvore genealógica da tribo, inicialmente o *primeiroGuerreiro*, identificado na primeira etapa, é adicionado na *árvore*, então é analisado cada elemento presente no dicionário *filhos*, se o pai do elemento analisado está na *árvore*, então ele é adicionado na *árvore* e removido do dicionário *filhos*, esse processo se repete até que o tamanho do dicionário *filhos* seja 0. Esse processo é demonstrado no Algoritmo 2.

Algoritmo 2: Inserção de bárbaros na árvore

```
1  início
2      árvore.adiciona(primeroGuerreiro);
3      repita
4          para cadaf ∈ fihos faça
5              if árvore.contém(bárbaros.get(f).pai) then
6                  árvore.adiciona(bárbaros.get(f));
7                  filhos.remove(f);
8              end
9          fim
10     até filhos.tamanho! = 0;
11 fim
```

Porém, apesar de remover um filho do dicionário toda vez que o insere na *árvore*, o dicionário *bárbaros* será percorrido diversas vezes. O que se torna um problema, pois os bárbaros tinham tribos que duraram muitas gerações, e o tempo para armazenar os guerreiros dessas tribos na *árvore* é muito alto, como mostra a Tabela 2.

Quando todos os bárbaros estão na *árvore* é necessário calcular o número de terras de cada bárbaro da última geração da tribo, e identificar o que possui a maior quantidade. Para isso, primeiramente são separados todos os guerreiros armazenados na *árvore* que não possuem filhos, esses fazem parte da última geração da tribo.

Para calcular a quantia de terra pertencente a um guerreiro, foi desenvolvido um algoritmo, o qual tem como entrada um *guerreiro*, e a quantidade de *terras* que ele conquistou durante a vida. Se o *guerreiro* não é o *primeiroGuerreiro*, então a função é chamada novamente, agora a entrada *guerreiro* é o pai do guerreiro, e a entrada *terras* é a quantidade de terras deixadas pelo pai do guerreiro mais a quantidade de terras conquistadas pelo guerreiro durante sua vida. Se o *guerreiro* é o *primeiroGuerreiro*, então, a função retorna *terras* mais a quantidade de terras que o *primeiroGuerreiro* deixou para cada filho. Esse processo é mostrado no Algoritmo 3. Esse método para calcular a quantidade de terras é usado também na segunda solução, demonstrada da Secção 2.2.

Algoritmo 3: Calcula

```
Entrada: guerreiro, terras
1 início
2     if guerreiro.éRaiz then
3         retorna terras + nodo.terras/nodo.quantFilhos
4     end
5     retorna Calcula(nodo.pai, (terras +
          (nodo.pai.terras/nodo.pai.quantFilhos))
6 fim
```

2.2. Segunda Solução

A segunda solução utiliza, além do dicionário *bárbaros* e da *árvore*, utilizadas na versão anterior, mais um dicionário chamado *tribo*, o qual tem como chave o nome de um bárbaro e o valor é uma lista encadeada com o nome de seus filhos. Nesse caso para adicionar um bárbaro ao dicionário *tribo*, primeiramente é conferido se o nome de seu pai já foi adicionado anteriormente, se sim, então o nome do bárbaro é adicionado na lista de filhos de seu pai, se não, é criado um novo elemento, no qual o valor é uma lista contendo apenas o nome do bárbaro e a chave é o nome de seu pai.

Para montar a árvore genealógica da tribo, inicialmente o *primeiroGuerreiro* é adicionado na *árvore*. Depois é usado um algoritmo, demonstrado em Algoritmo 4, que tem como entrada o *guerreiro* a ser adicionado. O algoritmo começa adicionando todos os filhos do *guerreiro* na *árvore*, então é executado novamente para cada filho, até que o *guerreiro* não tenha filhos.

Algoritmo 4: Adiciona

```
Entrada: guerreiro
1 início
2     filhos = tribo.get(guerreiro);
3     if filhos.tamanho > 0 then
4         para cadaf ∈ filhos faça
5             árvore.add(f);
6         fim
7         para 0 ≤ i < filhos.tamanho faça
8             Adiciona(tribo.get(guerreiro).get(i));
9         fim
10    end
11 fim
```

2.3. Terceira Solução

Como visto a solução anterior resolve o problema, e apesar de ser mais eficiente que a primeira ainda é muito custosa e tem um tempo de execução muito alto, principalmente quando se trata de uma tribo bárbara com muitas gerações. Para melhorar o desempenho da aplicação foi desenvolvido uma nova solução.

Nessa solução, além do dicionário *bárbaros*, já usado nas versões descritas anteriormente, foi usado também um dicionário idêntico ao usado na segunda solução chamado

tribo . Para um elemento ser adicionado primeiramente é conferido se o nome do pai já está no dicionário *tribo*, se não está, o elemento é adicionado. Se o nome do pai já está no dicionário então o nome do filho é adicionado em sua lista de filhos, como é demonstrado no Algoritmo 5, esse algoritmo é executado na leitura de cada linha do arquivo de entrada, garantindo que todos os bárbaros serão adicionados.

Algoritmo 5: Adiciona

```
   Entrada: pai,filho
1  início
2  |  if tribo.contémChave(pai) then
3  |  |  pai.filhos.add(filho);
4  |  end
5  |  else
6  |  |  tribo.put(pai, filho);
7  |  end
8  fim
```

Para calcular o número de terras de cada bárbaro e identificar o guerreiro da última geração da tribo que possui a maior quantidade de terras,foi desenvolvido um método que recebe como entrada o elemento pelo qual deve-se começar o calculo, nesse caso o *primeiroGuerreiro*, então o número de terras do bárbaro é dividido pelo número de filhos que ele possui, e esse valor é somado ao número de terras de cada filho e o método é chamado novamente para cada filho com o filho sendo o novo elemento a ser calculado.

Com isso sabemos o número de terras de cada bárbaro, então para saber qual o guerreiro com mais terras e necessário separar os bárbaros correspondentes a última geração da tribo, que são todos os bárbaros que não possuem filhos. Após separar toda a última geração de bárbaros, é identificado o bárbaro com o maior número de terras, então a função retorna seu nome e a quantidade de terras que ele possui.

3. Resultados

Os algoritmos descritos nas Seções anteriores foram implementados na linguagem java, em todos os casos foram encontrados os mesmos resultados, que são descritos na Tabela 1.

Para cada caso de teste foram realizadas cinco repetições usando cada algoritmo. Os dados de tempo de execução de cada execução foram armazenados e posteriormente foi feito uma média com os tempos de execução. Os resultados são mostrados na Tabela 2, Tabela 3 e Tabela 4.

4. Conclusão

Este trabalho apresentou três soluções para decifrar os pergaminhos bárbaros. Os tempos de execução de cada versão, para cada caso de teste foram comparados, e apesar de todas as soluções resolverem o problema para todos os casos de teste, as duas primeiras versões tiveram um tempo de execução muito alto, principalmente para tribos com muitas gerações. Já a terceira solução se destaca nesse aspecto com tempos de execução não passando dos 2 segundos enquanto para o mesmo caso a primeira versão demorou mais de 1 hora e segunda aproximadamente 20 minutos para decifrar o pergaminho.

Tabela 1. Resultados encontrados

Caso de teste	Guerreiro	Quantidade de Terras
CasoJB4a	Rimeemicpox	25307.0
CasoJB4b	Nepulacritrikix	12629.9
CasoJB8a	Cristiumimonlax	21531.0
CasoJB8b	Stropunngemonstax	16457.4166
CasoJB10a	Nepstitrigax	13964.85
CasoJB10b	Delrenvax	11624.842
CasoJB12a	Gruvervix	12579.923
CasoJB12b	Prepeeblepscapox	14002.888
CasoJB14a	Carmanclox	12181.941
CasoJB14b	Gruumimontex	11857.975

Tabela 2. Tempos de execução para primeira solução

Caso de teste	Tempo de execução(em segundos)
CasoJB4a	0,008
CasoJB4b	0,008
CasoJB8a	0,024
CasoJB8b	0,299
CasoJB10a	61,024
CasoJB10b	1,275
CasoJB12a	117,24
CasoJB12b	95,129
CasoJB14a	5283,441
CasoJB14b	2415,46

Tabela 3. Tempos de execução para segunda solução

Caso de teste	Tempo de execução(em segundos)
CasoJB4a	0,08
CasoJB4b	0,08
CasoJB8a	0,024
CasoJB8b	0,154
CasoJB10a	9,048
CasoJB10b	0,265
CasoJB12a	11,279
CasoJB12b	10,06
CasoJB14a	1194,9
CasoJB14b	680,112

Tabela 4. Tempos de execução para terceira solução

Caso de teste	Tempo de execução(em segundos)
CasoJB4a	0,007
CasoJB4b	0,006
CasoJB8a	0,011
CasoJB8b	0,046
CasoJB10a	0,175
CasoJB10b	0,07
CasoJB12a	0,202
CasoJB12b	0,178
CasoJB14a	1,234
CasoJB14b	0,94