Jogo: Troca de Cocos entre Macacos

Mateus F. Poletto
Algoritmos e Estruturas de Dados 2 - P. João B. Oliveira
Escola Politécnica - PUCRS
09 de abril de 2023

RESUMO

Irá ser comentado neste artigo a resolução do Trabalho Avaliativo 1 da cadeira Algoritmos e Estruturas de Dados 2. O trabalho era relacionado a um suposto jogo antigo que é utilizado por macacos para se divertirem em que os mesmos se trocavam cocos até que a quantidade de rodadas acabasse, um jogo de azar e tanto.

Para realizar a simulação desse jogo antigo, foi explicado as regras do jogo e passado possíveis relatórios de dados de início do jogo. Com isso, temos um ponto de partida para começar elaborar um algoritmo que nos ajuda entender melhor o jogo e definir com antecedência qual será o macaco vencedor ou até mesmo qual foi o macaco vencedor.

Entretanto, cada cenário novo apresentado possuía maior quantidade de dados e também uma maior quantidade de rodadas. Isso acabou proporcionando que o algoritmo utilizado fosse repensado mais de uma vez, portanto, veremos passo a passo o algoritmo, e os dados que podemos observar.

Introdução

O projeto foi apresentado para nós como o objetivo de desenvolver um algoritmo que fosse possível interpretar os dados referente ao estudo de um jogo de macacos que foi analisado por um especialista da área. A missão foi captar esses dados e conforme o comportamento dos macacos analisados, criar um algoritmo capaz de simular o jogo e apresentar o macaco vencedor.

Aqui seguem as regras do jogo:

-Existem muitos macaquinhos;

-Cada macaquinho enche seus cocos com pedrinhas, quantos cocos quiser e com quantas pedrinhas quiser;

-Cada macaquinho sabe para quem tem que mandar um COCO com nfimero par de pedrinhas;

-E é Claro que também sabe para quem mandar 0s Cocos com pedrinhas ímpares;

-Uma rodada começa com o macaquinho 0 distribuindo todos os seus cocos, depois 0 macaquinho 1, depois o 2 e assim por diante até o último macaquinho. Um coco pode fazer várias
viagens entre macaquinhos em uma só rodada.

-Depois de fazer um número enorme de rodadas, o macaquinho que tiver mais cocos é eleito o campeão do bando por uma semana, recebendo atenções especiais.

Os dados referente aos jogos foram encaminhados em formato “txt” no seguinte padrão:

Fazer 100000 rodadas
Macaco 0 par -> 4 impar —> 3 : 11 : 178 84 1 111 159 22 54 132 201 51 44
Macaco 1 par -> 0 impar —> 5 : 9 : 80 82 10 83 98 31 56 84 53
Macaco 2 par -> 3 impar —> 4 : 7 : 65 194 35 132 191 202 62
Macaco 3 par -> 0 impar -> 4 : 3 : 121 10 162
Macaco 4 par -> 0 impar -> 5 : 5 : 16 110 125 113 35
Macaco 5 par -> 2 impar -> 0 : 8 : 120 25 20 134 166 100 157 159

A primeira linha apresenta a quantidade de rodadas daquele jogo. O restante das linhas possuem na seguinte sequência, primeiro o id do macaco, o segundo número a aparecer na linha representa o macaco de destino para os cocos com quantidade par, o terceiro número representa o macaco de destino para os cocos com quantidade ímpar. Já o 4 número apresenta a quantidade de cocos presentes naquele macaco, e os próximos números representam um coco cada, sendo que cada valor é a quantidade de pedrinhas existentes naquele coco.

Solução

A linguagem escolhida foi Java. Para começarmos a solucionar o problema primeiro precisamos resgatar os dados presentes nos arquivos “txt” enviados que possuem as informações necessárias para rodar o jogo. Porém como podemos armazenar os macacos de maneira que sua consulta não se torne demorada ao escalarmos mais macacos e rodadas no programa. No início a ideia era utilizar um ArrayList para pesquisa, iria funcionar, porém como a busca de seus dados ocorre percorrendo sempre todo o vetor, o algoritmo se tornaria muito lento. A solução encontrada foi armazenar os macacos em um HashMap, no qual a chave de pesquisa do macaco é seu id e a resposta do cálculo da posição Hash é um objeto

macaco. Um objeto macaco é composto de 5 atributos inteiros, seu id, macacoPar(id), macacoImpar(id), numCocosPar e numCocosImpar.

Voltando a leitura do arquivo, após conseguir definir o tipo de estrutura que armazenaria os macacos foi iniciado os trabalhos na leitura do arquivo(a lista de cocos não é salva, apenas a quantidade total de cocos pares e ímpares),o código se apresenta similarmente a esse:

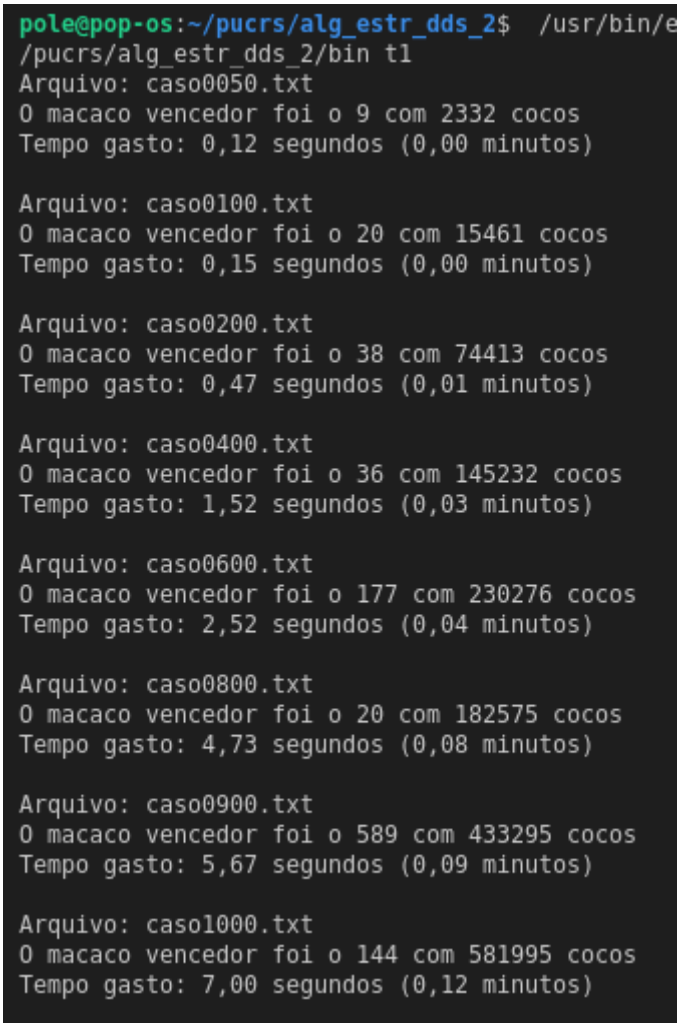
```
função leitura(fileName) {
  macacos <- mapa vazio
  rodadas <- 0
  tente fazer o seguinte {
     br <- crie um BufferedReader para ler o arquivo fileName
     line <- leia a primeira linha do arquivo e pegue o número de rodadas
        rodadas <- converta line para inteiro e pegue o segundo elemento (após o espaço)
    enquanto houver mais linhas em br faça {
       line <- leia a próxima linha
       partes <- divide line por " : "

       infoMacaco <- divida partes[0] por " "
       id <- converta o segundo elemento de infoMacaco[2] para inteiro
       par <- converta o quinto elemento de infoMacaco[5] para inteiro
       impar <- converta o oitavo elemento de infoMacaco[8] para inteiro

       cocosStr <- divida partes[2] por " "
       numCocosPar <- 0
       numCocosImpar <- 0

       para cada coco em cocosStr faça {
          pedrinhas <- converta coco para inteiro
          se pedrinhas for par então {
             numCocosPar <- numCocosPar + 1
          } senão {
             numCocosImpar <- numCocosImpar + 1
          }
       }

             macaco <- crie um objeto macaco com os valores de id, par, impar, numCocosPar e numCocosImpar
       adicione macaco ao mapa macacos com a chave id
    }
} capture qualquer exceção e imprima o erro
retorne um novo objeto resultArquivo com os valores de rodadas e macacos
}
```

Após termos realizado a leitura do arquivo de forma correta criamos então a classe referente ao jogo e criamos uma instância desse jogo passando como parâmetro as rodadas e o HashMap de macacos e armazenamos esses valores. Após ter feito isso, iniciamos o jogo chamando uma função denominada de jogar. Essa função retorna um objeto macaco que é o macaco vencedor do jogo.

A função jogar é bem simples, ela percorre toda a lista de macacos passando um macaco de cada vez. No macaco da vez, ela pega o id dos macacos pares e ímpares e passa como chave para buscar no HashMap. Ao retornar a instância desses dois macacos, eles são passados como atributos da função transferirCocos presente na classe macaco, presente no macaco principal do laço naquela iteração e então é transferido a quantidade de cocos pares e ímpares os macacos que foram atribuídos a função. Aqui se apresenta o pseudo-código:

```
jogar() {
  para o número total de rodadas {
    para macacoAtual em todos os macacos {
      macacoPar <- busca e retorna o macacoPar conforme o atributo em
      macacoAtual
      macacoÍmpar <- busca e retorna o macacoÍmpar conforme o atributo em
                     macacoAtual

      chama função transferirCocos(macacoPar, macacoÍmpar) existente no
      macacoAtual
    }
  }

  macaco vencedor <- nulo
  para macacoAtual em todos os macacos {
    se (macaco vencedor <- nulo ou número de cocos do macacoAtual for maior que
        do vencedor) então{
      vencedor <- macaco
    }
  }
retorna macaco vencedor
}

transferirCocos(macacoPar, macacoÍmpar){
  numCocosPar do macacoPar recebe o numCocosPar do macaco da iteração
  numCocosPar do macaco da iteração é zerado

  numCocosÍmpar do macacoÍmpar recebe o numCocosÍmpar do macaco da
  iteração
  numCocosÍmpar do macaco da iteração é zerado
}
```

Após realizar o jogo é imprimido o macaco vencedor com a quantidade de cocos que ele acaba e o tempo que demorou para ser realizado o jogo.
Aqui vai o print do resultado ao rodar o algoritmo:



Ao ser feita a análise dos dados presentes no resultado, os dados que foram apresentados para rodar o algoritmo e o algoritmo em si, percebemos que a complexidade do programa pode ser resumida a O(M * R), pois o tempo de execução do programa é altamente afetado conforme M = quantidade de macacos e R = número de rodadas.

Observações

No meio do percurso de desenvolvimento do programa foram utilizadas algumas lógicas que conforme o passar da criação do algoritmo acabaram por se tornar menos ineficientes, aqui vão algumas:

- Utilizar uma lista só para guardar todos os cocos e seus valores em cada macaco, o que ocasionava em mais trabalho para procurar cocos pares e ímpares e transferir os cocos pares e ímpares.
- Utilizar duas listas de cocos, uma par e uma ímpar em cada macaco para guardar os cocos pares e cocos ímpares. Porém ainda tomava muito tempo em transferir todos os cocos presentes na lista para o próximo macaco, portanto no final foi utilizado apenas a quantidade existente de cocos pares e ímpares que são são obtidos ainda na leitura, não são armazenadas listas de cocos, o que ocasiona em menos memória ocupada e menos serviço de processamento ao realizar o jogo.
- Utilizar arraylist para guardar os macacos acabou se tornando uma má idéia pois sempre que ocorria o jogo era necessário procurar os macacos percorrendo toda a lista existente, o que conforme escala os dados acaba se tornando inviável para a execução do algoritmo.

Conclusão

O programa está realizando o esperado ao nos apontar o macaco vencedor e sua quantidade final de cocos após finalizar o jogo. O tempo de execução após as várias execuções acabou por ficar razoavelmente bom em comparação às execuções iniciais que levavam até 30 minutos para serem executadas.

Escolhas como manter na execução no programa apenas o necessário (um exemplo é não manter listas de cocos) e utilizar cálculos de Hash para armazenar os macacos tornaram-se ótimas escolhas, sendo mais econômicas e rápidas.

A principal conclusão que pode se tirar nesse trabalho é que, quanto maior a quantidade de dados ou fluxo de dados a ser utilizado pelo algoritmo, as escolhas para construção do programa acabam se tornando muito diferentes, principalmente pois afetaram o custo futuro ou até mesmo imediato do programa.

Prints do Código:

```java
import java.util.Map;

public class t1 {
    Run | Debug
    public static void main(String[] args) {

        String arquivos[] = {"caso0050.txt", "caso0100.txt", "caso0200.txt", "caso0400.txt", "caso0600.txt", "caso0800.txt", "caso0900.txt", "caso1000.txt"};
        for (String fileName : arquivos){
        long startTime = System.nanoTime();
        resultArquivo result = new readArquivo().leitura(fileName);
        int rodadas = result.getRodadas();
        Map<Integer, macaco> macacos = result.getMacacos();

        game jogo = new game(rodadas, macacos);

        macaco vencedor = jogo.jogar();

        long endTime = System.nanoTime();

        double durationSeconds = (endTime - startTime) / 1000000000.0;
        double durationMinutes = durationSeconds / 60.0;
        System.out.println("Arquivo: " + fileName);
        System.out.println("O macaco vencedor foi o " + vencedor.getId() + " com " + vencedor.getNumeroDeCocos() + " cocos");
        System.out.printf(format:"Tempo gasto: %.2f segundos (%.2f minutos)%n", durationSeconds, durationMinutes);
        System.out.println();
        }
        // System.out.println("Tempo de execução: " + durationSeconds + " segundos");
        // System.out.println("Tempo de execução: " + durationMinutes + " minutos");
    }
}
```

```java

public class game {

    private int rodadas;
    private Map<Integer, macaco> macacos;

    public game(int rodadas, Map<Integer, macaco> macacos) {
        this.rodadas = rodadas;
        this.macacos = macacos;
    }

    public macaco jogar() {
        for (int r = 0; r < rodadas; r++) {
            for (macaco macacoAtual : macacos.values()) {
                macaco macacoPar = macacos.get(macacoAtual.macacoPar);
                macaco macacoImpar = macacos.get(macacoAtual.macacoImpar);

                macacoAtual.transferirCocos(macacoPar, macacoImpar);
            }
        }

        macaco vencedor = null;
        for (macaco macaco : macacos.values()) {
            if (vencedor == null || macaco.getNumeroDeCocos() > vencedor.getNumeroDeCocos()) {
                vencedor = macaco;
            }
        }

        return vencedor;
    }
```

```java

public class readArquivo {

    public resultArquivo leitura(String fileName) {
        Map<Integer, macaco> macacos = new HashMap<Integer, macaco>();
        int rodadas = 0;
        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String line;

            // Ler a primeira linha do arquivo, que contém o número de rodadas
            rodadas = Integer.parseInt(br.readLine().split(regex:" ")[1]);

            // Ler as linhas restantes com informações dos macacos
            while ((line = br.readLine()) != null) {
                String[] partes = line.split(regex:" : ");

                // Obter informações sobre o macaco
                String[] infoMacaco = partes[0].split(regex:" ");
                int id = Integer.parseInt(infoMacaco[1]);
                int par = Integer.parseInt(infoMacaco[4]);
                int impar = Integer.parseInt(infoMacaco[7]);

                // Obter a quantidade de cocos e as pedrinhas em cada coco
                //int numCocos = Integer.parseInt(partes[1]);
                String[] cocosStr = partes[2].split(regex:" ");
                // List<Integer> cocosPar = new ArrayList<>();
                // List<Integer> cocosImpar = new ArrayList<>();
                int numCocosPar = 0;
                int numCocosImpar = 0;

                for (String coco : cocosStr) {
                    int pedrinhas = Integer.parseInt(coco);
                    if (pedrinhas % 2 == 0) {
                        // cocosPar.add(pedrinhas);
                        numCocosPar++;

                    } else {
                        // cocosImpar.add(pedrinhas);
                        numCocosImpar++;
                    }
                }

                // Criar um objeto Macaco e armazená-lo no HashMap
                macacos.put(id, new macaco(id, par, impar, numCocosPar, numCocosImpar));
            }

        } catch (IOException e) {
            e.printStackTrace();
        }

        return new resultArquivo(rodadas, macacos);
    }
}
```

```java
public class macaco {

    public int id;
    public int macacoPar;
    public int macacoImpar;
    // List<Integer> cocosPar;
    // List<Integer> cocosImpar;
    public int numCocosPar;
    public int numCocosImpar;

    // public macaco(int id, int par, int impar, List<Integer> cocosPar, List<Integer> cocosImpar) {
    //     this.id = id;
    //     this.macacoPar = par;
    //     this.macacoImpar = impar;
    //     this.cocosPar = cocosPar;
    //     this.cocosImpar = cocosImpar;
    // }

    public macaco(int id, int par, int impar, int numCocosPar, int numCocosImpar) {
        this.id = id;
        this.macacoPar = par;
        this.macacoImpar = impar;
        this.numCocosPar = numCocosPar;
        this.numCocosImpar = numCocosImpar;
    }

    // public int getNumeroDeCocos() {
    //     return cocosPar.size() + cocosImpar.size();
    // }

    public int getNumeroDeCocos() {
        return numCocosPar + numCocosImpar;
    }

    // public void transferirCocos(macaco macacoPar, macaco macacoImpar) {
    //     // Transfere todos os cocos pares
    //     macacoPar.cocosPar.addAll(this.cocosPar);
    //     this.cocosPar.clear();

    //     // Transfere todos os cocos impares
    //     macacoImpar.cocosImpar.addAll(this.cocosImpar);
    //     this.cocosImpar.clear();
    // }

    public void transferirCocos(macaco macacoPar, macaco macacoImpar) {
        // Transfere todos os cocos pares
        macacoPar.numCocosPar += this.numCocosPar;
        this.numCocosPar = 0;

        // Transfere todos os cocos impares
        macacoImpar.numCocosImpar += this.numCocosImpar;
        this.numCocosImpar = 0;

    }

```

```java
public class resultArquivo {

    private int rodadas;
    private Map<Integer, macaco> macacos;

    public resultArquivo(int rodadas, Map<Integer, macaco> macacos) {
        this.rodadas = rodadas;
        this.macacos = macacos;
    }


    public int getRodadas() {
        return rodadas;
    }

    public Map<Integer, macaco> getMacacos() {
        return macacos;
    }
}
```

Referências e Links Úteis

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html

https://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html

https://acervolima.com/arraylist-vs-hashmap-em-java/

https://github.com/mati-fp/t1_alest2