Os Macaquinhos

Felipe Gattelli Gauer

18 de abril de 2023

Resumo

Neste trabalho é tratado um sobre um jogo inventado por uma tribo de macacos com um intelecto acima da média dos primatas. Essa tribo sabe diferenciar se certa quantidades de pedrinhas são par ou ímpares, tendo isso em vista, criaram um jogo. O jogo nada mais é que, por exemplo, um macaco A, tem uma certa quantidade de cocôs com n números de pedrinhas dentro deles, assim todos cocôs pares dará para o macaco B e impares para macaco C e, depois disso, o macaco B dá todos seus cocôs pares para o macaco F e ímpares para macaco A e assim o jogo vai continuando. Para a resolução desse problema, foi feito um algoritmo na linguagem java utilizando orientação por objetos, o pacote "java.io" e "java.util".

1 Introdução

O problema retratado é de uma tribo de macacos encontrado pelo seu primo antropólogo e explorador explorador de selva, pelo seu primeiro avistamento com essa tribo, ele percebe que os macaquinhos são mais espertos que um macaco normal. Ele concluiu isso pelo fato deles saberem a diferença entre números pares e ímpares, isso os ajudou a criar um jogo para passar o tempo.

O jogo criado é versão primitiva de bingo, na qual eles enchem certas quantidades de cocôs com n pedrinhas cada cocô, em seguida, na primeira rodada um primeiro macaco vai pegar todos seus cocôs com número de pedrinhas pares e distribuir para um determinado macaco e fará a mesma coisa com os cocôs com número de pedrinhas ímpares, só que distribuindo para outro macaco. Antes do jogo começar de fato, é decidido quantas rodadas ocorrerão, o vencedor será o macaco que tiver mais cocôs quando acabarem as rodadas.

Para resolver esse problema, foi dado oito arquivos de texto contendo x número de macacos e y número de rodadas, para isso deve-se lê-los e interpretá-los. A primeira linha diz quantas rodadas o jogo terá, da segunda linha em diante é a informação sobre o macaco. Primeiro vem a escrita "Macaco", em seguida, o número do macaco "0", por exemplo, depois "par -¿" e o número do macaco que vai receber os cocos, "impar -¿" e o número do macaco que vai receber os cocos ímpares ": " o número de cocos que ele começa ": " e a quantidade de pedrinhas dentro de cada cocô, como é mostrado na figura 1.

Para ler o arquivo foi feito um reader que devolve um array de String, que foi interpretado mais tarde no código, transformando o conteúdo dele no objeto Macaco, e colando numa variável int o número de rodadas que terá o jogo. Assim, utilizando um for, podemos fazer o macacos fazerem as trocas pelo número de rodadas lidas no arquivo.

```
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
```

Figura 1: Exemplos de leitura do programa.

2 Solução

2.1 Primeira Solução

Depois de considerar o problema, pode-se perceber que a solução mais simples do mesmo é criar uma classe de “Macaco”, nela contendo o “id”, que nada mais é o número próprio de cada macaco, uma variável int “par” e int “impar”, para distribuir, no futuro, os cocôs pares e ímpares respectivamente. Por fim criar um ArrayList de int, chamado “cocos”, para armazenar os cocôs contendo o número de pedrinhas de cada cocô nas posições do array e para saber o total é só chamar um método já implementado no ArrayList, que seria “cocos.size()”.

Antes mesmo de fazer o código, já percebi que essa não era uma ideia tão boa assim, pois cada rodada eu teria que verificar todas as posições do array de cada macaco e ver se é par ou ímpar, então excluí essa ideia e fui para a próxima.

2.2 Segunda Solução

Ainda antes mesmo de fazer o código, pensei em deixar tudo igual a primeira solução, porém, ao invés de fazer um vetor de cocôs fazer dois, um chamado de “cocosPar” e outro “cocosImpares” para não necessitar rodar o vetor toda vez para saber se o cocô é par ou ímpar.

Porém um tempo refletindo, não achei que essa seria a melhor solução possível para esse trabalho, pois ainda achava que teria um jeito mais rápido e prático de resolver o problema.

2.3 Solução Final

Após passar um tempo refletindo como faria para optimizar ao máximo o algoritmo, decidi ler o problema mais algumas vezes. Após a terceira vez lendo, dei-me conta que em nenhum momento dizia ser relevante a quantidade de pedrinhas dentro dos cocôs, falava que o macaquinho tinha apenas conhecimento se era um número par ou ímpar, então fiz meu algoritmo baseado nessa conclusão.

Assim ao invés de fazer um ArrayList de int, decidi fazer apenas duas variáveis int “cocosPar” e “cocosImpar”, que serão os cocôs enviados para os macacos de id armazenadas nas variáveis “par” e “impar”, como mostrado na figura 2.

```
public class Macaco {
    private int id;
    private int par;
    private int impar;
    private int cocosPar;
    private int cocosImpar;
```

Figura 2: Classe Macaco

Em seguida fiz o método construtor e os getter como mostra na figura 3.

Antes de continuar a classe Macaco, fui para o main, comecei implementando um ArrayList do Objeto Macaco, para deixar todos macacos dentro de um vetor e em seguida fiz o reader para ler os dados do arquivo, transformar eles em macaco e adicionar para o vetor criado anteriormente. Também foi criado uma variável int de rodadas, para ser utilizada mais tarde.Como mostrado na figura 4

2.3.1 Reader e criação do macaquinhos

Para fazer o reader, criei um método estático que recebe um nome de arquivo como parâmetro e devolve um ArrayList de String contendo os dados lidos. Nele contendo outro ArrayList de String chamado “recordlist” (que será usado mais além para facilitar a leitura e o armazenamento do método), uma String “delimiter” que já é iniciada com o valor “ “, ou seja, essa variável, mais além, separará as palavras por array separando pelo espaço vazio, uma String “currentLine” para armazenar o conteúdo de uma linha, outro Array de String chamado “arrayToReturn” que será o array de retorno do método

```java
public Macaco(int id, int par, int impar) {
    this.id = id;
    this.par = par;
    this.impar = impar;
}

public int getId() {
    return id;
}

public int getPar() {
    return par;
}

public int getImpar() {
    return impar;
}

public int getCocosPar() {
    return cocosPar;
}

public int getCocosImpar() {
    return cocosImpar;
}
```

Figura 3: Construtor e getters.

```java
ArrayList<Macaco> macacos = new ArrayList<>();
int rodadas = 0;
```

Figura 4: main() inicio

e, por fim, um vetor String[] “line” que será usado juntamente com o “delimiter” para separar o conteúdo da linha.Como mostra na figura 5

```java
public static ArrayList<String> reader(String filePath) {
    List<String> recordList = new ArrayList<String>();
    String delimiter = " ";
    String currentLine;

    ArrayList<String> arrayToReturn = new ArrayList<>();
    String[] line;
```

Figura 5: Início do reader

Após abre-se um try catch, dentro dele cria-se um FileReader “fr” iniciado como new FileReader(filePath), o “filePath” sendo o caminho do arquivo a ser lido e também um BufferedReader “br” iniciando como new BufferedReader(fr), o que será usado para ler as linhas do arquivo. Assim usa-se while((currentLine = br.readLine()) != null), ou seja, o “br” fará a leitura da linha e adicionará o conteúdo para o “currenteLine” até que esse conteúdo seja igual a null. Dentro do while o currentLine é adicionado para o “recordList” através do método “add” presente na classe ArrayList.

Quando o while parar de ser executado fiz um for (int i = 0; i ¡ recordList.size(); i++), dentro o “line” recebe o valor de “recordList.get(i).split(delimiter)” ,ou seja, cada fez que o “i” mudar de valor ele está indo para uma próxima linha, fazendo um split recebe-se uma String[] como retorno, que nada mais seria que a linha quebrada conforme o que tem no “delimiter”. Ainda dentro do for, foi posto outro for for (int j = 0; j¡ line.length; j++) para o for rodar o tamanho do veto de String[] “line” e dentro, adiciona o conteúdo da linha para o array de retorno. Rodando tudo isso devolve-se o array de retorno. Como mostrado na figura 6

```java
try {
    FileReader fr = new FileReader(filePath);
    BufferedReader br = new BufferedReader(fr);

    while ((currentLine = br.readLine()) != null) {
        recordList.add(currentLine);
    }

    for (int i = 0; i < recordList.size(); i++) {
        line = recordList.get(i).split(delimiter);

        for (int j = 0; j < line.length; j++) {
            arrayToReturn.add(line[j]);
        }
    }
    br.close();

    return arrayToReturn;

} catch (Exception e) {
    System.out.println(x:"Error");
    return null;
}
```

Figura 6: Reader

Voltando para o main, peguei o vetor que será retornado pelo método “reader” e fiz um for para transformar a informação em macacos.Inicia-se com um if, se não for “Macaco”, sabe-se que se trata da primeira linha do programa que seria quantas rodadas tem, então pego “x+1” que é a posição que as rodadas estão no array de dados e faço um “x+=3” logo em seguida para já ir para a linha de baixo que nessa posição x é igual a “Macaco”.

```java
for (int x = 0; x < data.size();) {
    if (data.get(x).equals(anObject:"Macaco")) {
        x += 4;
        int par = Integer.parseInt(data.get(x));
        x += 3;
        int impar = Integer.parseInt(data.get(x));
        Macaco aux = new Macaco(macacos.size(),par, impar);
        macacos.add(aux);
        x += 2;
        int qntcoco = Integer.parseInt(data.get(x));
        x += 2;
        for (int i = 0; i < qntcoco; i++) {
            int coco = Integer.parseInt(data.get(x));

            if (coco % 2 == 0)
                aux.addCocosPar(cocos:1);
            else
                aux.addCocosImpar(cocos:1);
            x++;
        }
    } else {
        rodadas = Integer.parseInt(data.get(x + 1));
        x += 3;
    }
}
```

Figura 7: Transformando os dados em macacos

Quando a String no dado for igual a ”Macaco”, já avanço 4 posições (fazendo ”x+4”) para pegar qual macaco, o que está sendo adicionado, precisa dar os cocôs pares. Então, ja pulo 3 posições (fazendo ”x+3”) para pegar agora o macaco que receberá os cocôs ímpares.

Após ser concluído essa etapa, crio um new Macaco adicionando as informações lidas e, para o id, pego o tamanho do array, pois se ele for vazio o macaco tera id igual a zero, o próximo igual a 1 e assim por diante.

Agora, já tendo um array com todos os macacos, suas quantidades de cocôs pares e ímpares e para quem cada macaco vai distribuir seus cocôs, o que resta era fazer o jogo.

2.3.2 O jogo

Para fazer o jogo, utilizei o método mais simples, inicie com um for que acaba quando "i"for maior ou igual ao número de rodadas, ou seja, aquela variável que foi pega antes será usada agora para ver quantas rodadas terão o jogo e, em cada rodada, os macacos darão seus cocôs pares e ímpares para os macacos já definidos antes do jogo começar.

Dentro desse for, foi necessário fazer outro for para que, cada vez que desse uma rodada, todos os macacos trocassem seus cocôs, ele roda até o "j"ser maior ou igual o tamanho do array de macacos. Dentro deste cria-se um macaco "m"que pega no array a posição j, em seguida cria-se um macaco "aux"que no primeiro instante é pego no array de macaco o macaco par do macaco m, pelo método "m.getPar()".

Tendo o macaco que irá dar os cocos(o "m") e o que irá receber(o "aux"), pode-se fazer a troca pelos métodos "aux.addCocoPar()", que recebe por parâmetro um valor inteiro e adiciona com os cocôs pares do macaco "aux", o valor inteiro que tem que por no parâmetro é pegado através do método "m.zeraCocosPar()", que acaba zerando os cocôs par do macaco "m"e devolvendo o valor que havia antes de ser zerado.

Em seguida foi feito parecido para distribuir os cocôs ímpares, somente mudamos o macaco "aux"para agora ser o macaco impar do "m". É feito isso através do mesmo modo feito para pegar o par, porém se usa o método "m.getImpar()". E para adicionar os cocôs ímpares do "m"para o "aux", utiliza o método "aux.addCocosImpar()"e é passado por parâmetro o método "m.zeraCocosImpar()", que são similares com os métodos utilizados para os cocôs pares, só que as mudanças ocorrem nos "cocosImpar"de cada macaco.

Assim o jogo é finalizado.

```
for (int i = 0; i < rodadas; i++) {
    for (int j = 0; j < macacos.size(); j++) {
        Macaco m = macacos.get(j);
        Macaco aux = macacos.get(m.getPar());
        aux.addCocosPar(m.zeraCocosPar());

        aux = macacos.get(m.getImpar());
        aux.addCocosImpar(m.zeraCocosImpar());

    }
}
```

Figura 8: O jogo.

2.3.3 Quem ganhou

Para fazer esse método de quem ganhou foi criado duas variáveis, uma int "ganhador"que diz o id do macaco que teve mais cocôs e uma int "somaganhador"que é a soma de cocôs ímpares e pares do macaco ganhador, as variáveis são iniciadas com -1"e "0", respectivamente.

Assim é feito um for para cada macaco, tendo um int "soma"que soma os cocôs do macaco que está sendo testado se é o maior e um if. Se a "soma"do macaco for maior que a "somaganhador"a variável "somaganhador"recebe o valor de "soma"e o int "ganhador"recebe "m.getId()"que retorna, como int, o "id"do macaco. E, após a conclusão do for, diz qual macaco que ganhou.

```java
int ganhador = -1;
int somaganhador = 0;
for (Macaco m : macacos) {
    int soma = m.getCocosPar() + m.getCocosImpar();
    if(soma>somaganhador) {
        somaganhador = soma;
        ganhador = m.getId();
    }
}
System.out.println("O macaco ganhador foi o macaco "+ganhador);
```

Figura 9: Caption

3 Resultados

Após a implementação do algoritmo na linguagem java e executar os oito casos, separadamente, o resultado encontrados estão na tabela 1.

Caso	Macaco Ganhador	RunTime
50	9	40ms
100	20	70ms
200	38	151ms
400	36	308ms
600	177	528ms
800	20	785ms
900	589	907ms
1000	144	1110ms

Tabela 1: Resultados por execução

4 Conclusão

As abordagens iniciais de pensamento, embora não tenha sida usadas, ajudaram sim para o desenvolvimento de outras ideias que vieram ser ideias mais prática, pois ajudaram a aumentar o desempenho do algoritmo tornando-o mais eficaz.

A parte ruim dessa ideia que foi implementada, é o fato de o usuário final não ter acesso a quantas pedrinhas havia dentro de cada cocô.

Entretanto, esse algoritmo resolve o problema proposto sim, pelo fato de o único questionamento ser: "Qual macaco ganha o jogo?"E ele responde corretamente essa questão.

Olhando e analisando os resultados recebidos por cada caso, percebe-se que quantos mais macacos e cocos tiver, mais lento será a execução do algoritmo. Embora não apresentado neste artigo, foi feito alguns testes, um para ver se estava adicionando os cocôs corretamente e o outro se o macaco que ganhava era de fato para ele ganhar, os dois testes foram bem sucedidos.

Para finalizar, acredito ter desenvolvido uma solução rápida para o programa proposta, tendo em vista que uma implementação diferente pode chegar a demorar dez vezes mais tempo de execução.

Referências