Jogo do Coco

Gustavo W. M. Silva

16 de abril de 2023

Resumo

Esse artigo descreve alternativas de soluções, feita em Java, para o primeiro problema proposto na disciplina de Algoritmo e Estrutura de Dados 2 da PUC-RS, uma simulação de jogo entre macacos, envolve encher cocos com pedrinhas e distribuí-los com base no número de pedrinhas para outro jogador correspondente. O vencedor é aquele que conseguir o maior numero de cocos, se tornando assim, o campeão do bando por uma semana. O artigo apresenta variações de soluções para o problema, incluindo uma análise de sua eficiência, além dos resultados das simulações, que envolveram oito casos diferentes.

1 Introdução

A partir do relato do antropólogo descrito no trabalho[1], é possível entender que os macacos possuem conhecimento matemático básico para diferenciar números pares de ímpares. Eles desenvolveram um jogo em que cada macaco é identificado por um número (2), eles enchem cada coco com o numero de pedrinhas que desejarem (6) e sabe para quem enviar seus cocos com números pares (3) e ímpares (4). O jogo tem uma quantidade de rodadas estabelecida (1) e cada macaco possui uma quantidade de cocos com pedrinhas de números variados (5). O padrão para jogar consiste em cada macaco, em ordem numérica, enviar seus cocos para os macacos correspondentes, com base no número de pedrinhas ser par ou ímpar, assim depois que terminar a última rodada, quem tiver mais cocos é o campeão.

Listing 1: Estrutura do Arquivo

```
Linha 1:
 Fazer 05000 (1) rodadas

A partir da Linha 2:
 Macaco 0 (2) par -> 24 (3) impar -> 27 (4) : 200 (5) : 105618 32570 16903 ...(6)
 Macaco 1 (2) par -> 45 (3) impar -> 19 (4) : 414 (5) : 27819 92758 45623 ...(6)
 Macaco 2 (2) par -> 0 (3} impar -> 42(4) : 139 (5) : 81162 67533 33215 ...(6)
                                        .
                                        .
                                        .
 Macaco 49 (2) par -> 1 (3) impar -> 5 (4) : 385 (5) : 88667 27434 39460 ...(6)
```

Seguindo essas regras, analisaremos três possíveis soluções, bem como suas características, optando por uma solução mais eficiente. Em seguida, os resultados obtidos serão apresentados, bem como as conclusões obtidas.

2 Soluções - Leitura de Arquivo

2.1 List Vs BufferedReader

Para a leitura de arquivo foram testados dois meios diferentes, o primeiro foi utilizando a classe List, que apresentou uma performance inferior ao segundo caso, no qual foi utilizado a classe BufferedReader.

Para ler as linhas desse arquivo de texto e armazena-las em uma lista foi empregado a expressão Files.lines(path).collect(Collectors.toList()), no qual Files.lines(path) retorna as linhas do arquivo especificados pelo path, e o collect(Collectors.toList()) coleta os elementos e os armazena em uma lista.

Listing 2: Código List

```
List <String > lines = Files.lines(path).collect(Collectors.toList());
for (String line : lines)
{
  [] linha = line.split("\\s+");
  if (line.startsWith("Macaco")) {
 .
 .
 .
} else {
    rodadas = Integer.parseInt(linha[1]);
    }
}
```

Esta implementação apresentou menos eficiência em tempo de execução do que a feita com BufferreadReader. Vale salientar que o List teve algumas limitações que não foram exploradas para serem resolvidas, como não sendo delimitado o tamanho do Array de Macacos e a eliminação da verificação se a linha corresponde aos macacos ou à quantidade de rodadas.

No caso do BufferedReader, utilizou-se o método Files.newBufferedReader(Path path, Charset cs) que recebe como a parâmetro instância da classe Path, o qual contem a localização do arquivo, e um objeto Charset representando a codificação de caracteres do arquivo. Retornando um objeto BufferedReader que faz o processamento das linhas do arquivo, uma de cada vez. Para uma maior segurança foi utilizado try-with-resources, nos permitindo a garantia de que sera fechado corretamente, mesmo ocorrendo alguma exceção.

Listing 3: Código BufferedReader

```
try(BufferedReader reader=Files.newBufferedReader(Paths.get(nomeArquivo),
Charset.defaultCharset()));
{
 line = reader.readLine();
 rodadas = linha[1];

 while ((line = reader.readLine()) != null) {
 .
 .
 .

 } catch (IOException e)
 { System.err.format("Erro na leitura do arquivo: ", e);}
```

O uso da função readLine() da classe BufferedReader possibilita o acesso do número de rodadas antes de entrar no laço 'while', que percorre as linhas do arquivo referente aos jogares. Isso nos permite calcular o numero de macacos participando, uma vez que o jogo segue um padrão onde o numero de rodadas dividido por 100 corresponde ao numero de jogadores. Com o numero de jogares em mãos, é possível criar um Array de macacos com o tamanho correto. Permitindo assim, que não precise utilizar a verificação da condicional no laço, o que torna o código mais eficiente e legível.

2.2 Estrutura

Foram testadas para guardar as informações dos jogadores algumas estruturas como HashMap, ArrayList e Array que são amplamente usadas na programação. HashMap apresentou o pior caso, como esperado, sendo que não precisa se importar com a ordem - já que o arquivo esta organizado - além de ocupar mais memória.

O ArrayList foi implementado quando havia à necessidade de se importar com o redimensionamento da capacidade, como isso foi resolvido com a eliminação da necessidade da verificação no laço e a possibilidade de prever a quantidade de macacos através do numero de rodadas, possibilitou utilizar o Array, que representa uma melhor performance. Sendo que os elementos do Array são armazenados em uma posição contígua de memória, é mais rápido acessar elementos em um Array do que em um ArrayList, que precisa procurar o elemento desejado na lista vinculada.

2.3 Macaco - separação

Para fazer a distribuição das informações dos macacos, foi preciso criar uma classe Macaco contendo as seguintes informações: o numero do macaco, para quem enviar os cocos Pares e Ímpares, e a quantidade total de cada um.

O laço 'while' percorre todas as linhas do arquivo, utilizando o padrão do arquivo para extrair as informações necessárias de cada linha e os armazenando no Array de Macaco.

Listing 4: Código Distribuição

```
//Os numeros se referem ao exemplo 1
//Linha 2 em diante: Macaco (2) par->(3) impar->(4) : (5) : (6 ate n);

 while (line != null) {

    id = linha[2];
    envP = linha[3];
    envI = linha[4];
    tam = linha[5];
    nPar = 0;
    nImp = 0;

 for (int i = 6; i  < 6 + tam; i++) {
   if (linha[i] %2 = 0)  nPar++;
   else  nImp++;
 }
```

A verificação da quantidades de números de cocos par ou ímpar foram feitas 2 versões: a primeira consistia em pegar o número de pedrinhas inteiro e calcular seu resto da divisão por 2, incrementando os contadores apropriados. A segunda foi uma modificação da primeira, ao invés de pegar o número inteiro pegamos apenas o último carácter para verificar, assim a divisão pode ser feita de forma mais rápida, mesmo tendo que fazer mais operações antes de chegar na divisão, ganhamos performance pois a quantidade de pedrinhas em cada coco é muito grande e demoraria mais para pegar o resto da divisão.

Pegamos esses dados e colocamos eles no Array de tipo Macaco, com os dados em mãos podemos começar o jogo.

3 Solução - Jogo

Para o funcionamento do jogo utilizou-se o 'while' para controlar o numero de vezes que o loop 'for' sera executado. A condição é que a variável 'rodadas' deve ser maior que 0. 'rodadas' vai sendo decrementada a cada interação, de forma que o laço 'for' seja executado o número de vezes que 'rodadas' tem inicialmente.

O 'for' é utilizado para percorrer a lista de macacos. Para cada 'macaco' na lista, ele envia os cocos pares do macaco ao seu macaco par correspondente (representada pela posição retornada pelo método getEnvP()) e envia os cocos ímpares ao seu macaco ímpar correspondente (representado pela a posição retornada do método getEnvI()). A quantidade de cocos pares e impares são representado pelos métodos removePar() e removeImp(), respectivamente, que além de retornar o valor, também remove do macaco a quantidade que ele enviou.

Listing 5: Código de Troca

```
 while (rodadas--> 0) {
   for (macaco : listaMacacos) {
     listaMacacos[macaco.getEnvP()].addPar(macaco.removePar());
     listaMacacos[macaco.getEnvI()].addImp(macaco.removeImp());
 }
}
```

Para melhorar o tempo de execução pode-se dividir o número de rodadas por 1000 sendo possível obter o mesmo resultado caso o programa rodasse o número de rodadas total.

Com as distribuições feitas podemos verificar qual jogador terminou com o maior numero de cocos, para fazer essa verificação foi desenvolvido duas formas diferentes, a primeira usamos o for each para percorrer toda a lista de macaco e verificar se o macaco atual tem mais cocos que os macacos anteriores, se houver é salvo seu id e seu número de cocos, foi utilizado para o HashMap.

A segunda foi utilizado o método stream() e a função lambda max(). stream() é usado para converter uma lista de macacos em um fluxo de objetos que poder ser processados usando max(). A função lambda: (a,b) ->a.getTam() - bgetTam() é usada como comparador para encontrar o objeto com o maior numero de cocos. E o método orElse(null) é usado para lidar com o caso de a lista estiver vazia.

Um ponto importante é que esse código não verifica se houve empate, pois, foi testado que não haviam empates nos casos apresentados.

4 Resultados

Para a apresentação do resultado e para ver a velocidade de execução do sistema foi utilizado o método System.currentTimeMillis(), pegando a hora do sistema no início e no final do código e fazendo um calculo de subtração de um pelo outro, para ver o tempo de execução. Cada caso de teste foi executado separadamente. Vale ressaltar que o tempo varia em cada execução mas não difere muito um do outro e sendo testado apenas em uma configuração de sistema.

Sistema utilizado para testes:
Nome do dispositivo: Notebook Dell G15
Processador: 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz 2.69 GHz
RAM instalada: 16,0 GB (utilizável: 15,7 GB), 2 slots de 3200 MHz
GPU Laptop: NVIDIA GeForce RTX 3050, Memória dedicada 4,0 GB
Tipo de sistema Sistema operacional de 64 bits, processador baseado em x64
Edição: Windows 11 Home Single Language
Versão: 22H2

Resultados - Casos de Teste		
Casos	Macaco Vencedor	Qtd de Cocos
50	9	2332
100	20	15461
200	38	74413
400	36	145232
600	177	230276
800	20	182575
900	589	433295
1000	144	581995

Variações Tempo de Execução (ms) - sem rodadas/1000			
Casos	Array e BufferedReader	HashMap e BufferedReader	List e ArrayList
50	52	60	56
100	75	99	98
200	124	177	149
400	225	515	290
600	417	948	509
800	614	1518	841
900	690	1768	940
1000	782	2026	1084

Variações Tempo de Execução (ms) - com rodadas/1000			
Casos	Array e BufferedReader	HashMap e BufferedReader	List e ArrayList
50	36	38	43
100	62	65	75
200	105	112	115
400	203	195	219
600	322	334	349
800	440	488	490
900	530	585	642
1000	593	618	678

5 Conclusão

As variações de solução foram surgindo a partir de otimizações que iam sendo feitas no código, contribuindo para um melhor entendimento do problema. A ultima solução se tornou simples, legível e efetivo, embora tenha exigido longos debates e estudos de como melhorar a eficiência do programa, servindo como um bom exemplo da diversidades de formas que é possível desenvolver códigos, apenas mudando algumas linhas.

Esse trabalho foi uma caixinha de surpresa com grandes charadas e pistas mostrando maneiras de como olhar as informações fornecidas. No meio da conclusão estávamos refletindo o que melhoraríamos e testaríamos em uma próxima vez, e nos venho a discussão de algo que tinha nos deixado intrigados no meio dos resultados, que era quantidade de macacos que ficavam sem cocos, então fomos testar a divisão por 100, que já havia sido utilizado em outra otimização, só que percebia-se que havia uma grande margem, então testamos a divisão por 1000 e sai os mesmos resultados. Portanto, podemos tirar como conclusão que a diversos caminhos para serem seguidos.

Acreditamos ter desenvolvido uma solução interessante e rápida, oferecendo resultados satisfatórios para o problema proposto. Mas para projetos futuro gostaria de explorar o uso de programação paralela, acreditando que a divisão das tarefas em mais instância melhoraria o desempenho do programa, e utilizar o fato de grande parte dos macacos ficaram com o numero de cocos zerados para retira-los da lista.

Referências

OLIVEIRA, João Batista Souza de. Os Macaquinhos, 2023.

BAELDUNG, try-with-resources, Acesso: 16 de Abril 2023.
ORACLE, BufferedReader, Acesso: 16 de Abril 2023.
ORACLE, String, Acesso: 16 de Abril 2023.