Relatório do Algoritmo do

Bingo de Macacos

Lucas Candemil Chagas

Politécnica – PUCRS

27 de março de 2023

Resumo

Neste relatório trataremos a resolução de um problema dentro de uma situação de bingo de macacos envolvendo cocos com pedras, estes animais devem dar seus cocos para dois macacos especificados nos casos de teste, fornecido pelo professor, em rodadas. Com isto em mente o problema a ser resolvido é descobrir quem destes macacos, depois do término do bingo, será o chefe do bando sendo este o que tem mais cocos dentre todos, e quantos cocos ele possui no final do jogo.

Para abordar o problema de encontrar o macaco com mais cocos de todos, pensamos em fazer funções *loops*, uma que fazia as rodadas e a outra que encontrava o chefe do bando depois do bingo. Primeiramente fizemos o algoritmo que fazia as rodadas, onde fazemos somas para mudar a variável que armazena o número de quantidade de cocos que os dois macacos, que recebem os cocos de um macaco, tinham anteriormente e por último criamos o programa que identificava qual deles, no final do jogo, tinha conseguido mais cocos. O resultado da solução obtido por meio destes métodos foi quase que instantâneo, o que surpreende neste resultado é o número de cocos que o chefe fica no final, alguns ultrapassando os cinco dígitos.

Introdução

O trabalho tem seu contexto em volta de um bingo de macacos onde em cada rodada um macaco entrega seus cocos pares e ímpares para dois outros macacos, um

recebendo com pares e o outro com ímpares. Depois de todas as rodadas serem feitas, quem desses animais tiver o maior número de cocos será o novo chefe do bando.

O enunciado do problema foi imposto pelo professor de Algoritmo e Estrutura de Dados 2 e inicialmente não sabíamos nada sobre a solução, por exemplo se alguém já havia resolvido e se havia soluções parecidas com essa que poderia vir a nos ajudar posteriormente na resolução do algoritmo. A única dificuldade que tivemos no início foi como diminuir drasticamente o tempo de execução da solução, a única forma que resolveu nosso problema foi refazer o algoritmo inteiro, assim tivemos que fazer preparos anteriores a criação do programa e esse em vários passos:

1. Inicialmente fizemos um método que lê o arquivo dos casos e que permitiu analisarmos cada linha do texto;
2. Extraímos todas as informações necessárias de cada linha de todos os arquivos de caso para assim fazer a solução, como a quantidade de rodadas a serem feitas, o tamanho do vetor de cocos de cada macaco, o vetor de cocos destes animais, e os índices para achar os macacos, que pegarão os cocos pares e ímpares do macaco da linha, no vetor;
3. Montamos um método em que transforma o vetor de cocos que inicialmente é de *Strings* e transformamos ele em vetor de Inteiros
4. Criamos uma classe Macaco em que dentro existe informações importantes de cada macaco, como por exemplo: os índices dos macacos que receberão os cocos pares e ímpares deste e o vetor dos cocos com pedras;
5. Armazenamos todos os macacos com suas informações em um vetor;
6. Armazenamos em variáveis as quantidades de cocos pares e ímpares de cada macaco;
7. Fizemos um *loop* em que neste ocorrerá as rodadas em que haverá as somas das quantidades de cocos pares e ímpares dos macacos com o que receberam de cocos pares e ímpares de um macaco;
8. No final fizemos um método também *loop* em que encontre o macaco com mais cocos entre todos;

Desenvolvimento para a solução deste problema

Para solucionar este problema e montar o algoritmo de resolução começamos a desenvolver primeiramente a leitura do arquivo do caso de teste que esse tem que ser capaz de extrair os dados de cada linha. Como usamos a linguagem de programação Java, usamos um dos *layouts* padrões em que dentro desse irá os códigos de armazenamento das informações dos textos, sendo esse fornecido em uma aula de Programação Orientada a Objeto na PUCRS.

```java
public boolean readFile(String nomeArq) {
    Path path1 = Paths.get(nomeArq);
    try (BufferedReader reader = Files.newBufferedReader(path1, Charset.forName(charsetName:"utf8"))) {
        String line = null;
        while ((line = reader.readLine()) != null) {
            //inserir codigos de manipulacao de linha
        }
    }
    catch (IOException x) {
        System.err.format(format:"Erro de E/S: %s%n", x);
    }
    return true;
}
```

Figura 1 – Layout em Java de leitura de arquivo;

Antes de extrairmos as informações dos textos, necessitamos criar uma classe Macacos para quando for pegar os dados possamos atribui-los aos macacos. Nessa classe montamos seu construtor, em que esse pega quatro inteiros, um que é o índice, do *LinkedList<Macacos>* que foi criado na classe principal, e que dentro será armazenado todos os macacos com seus dados, que aponta para o local onde está o macaco que receberá os cocos com quantidade de pedras pares, outro índice que aponta para o macaco com cocos ímpares e as últimas duas variáveis são quantos cocos pares e ímpares o macaco tem consigo. Sendo seu pseudocódigo:

Classe Macacos

Inteiro indiceP, indiceImp, qCocosPar, qCocosImpar

Método Macacos(Inteiro indicePar, Inteiro indiceImpar, Inteiro quatidadeCocosPares, Inteiro quatidadeCocosImpares)

indiceP = indicePar

indiceImp = indiceImpar

quatidadeCocosPar = qCocosPar

quatidadeCocosImpar = qCocosImpar

Fim do Método

Fim da Classe

Também dentro dessa classe teremos sete métodos em que estes são feitos para posterior mente chamar os dados de cada animal. E esses são: para pegar os índices dos macacos que receberam os cocos pares e ímpares, dois que pegam os conteúdos da variável qCocosPar e qCocosImpar (getCocosPares(), getCocosImpares()) e mais dois que os atualizam para outras quantidades (setCocosPares(novaQuantidade), setCocosImpares(novaQuantidade)).

Agora para pegar as informações dentro da função *while* da figura 1, criamos uma função *if-else* que quando for a primeira linha de todos os textos, onde está o número de rodadas que será feito, peguemos as rodadas e as guardamos em uma variável, e dentro do *else* codificamos a criação de três vetores de *Strings*, uma *LinkedList<Integer>*, uma

função recursiva e da variável do tipo Macaco. Dentro de um dos vetores tem a divisão da linha de acordo com " : " assim separando em 3 *Strings*, a primeira seria a *String* que contém os dados do número do macaco, que não será usado, e dos índices, da *LinkedList<Macacos>*, dos macacos que receberão os cocos pares e impares, assim o segundo vetor seria apenas esses índices que para isso retiramos todas as letras por meio de *regex*, a segunda *String* seria a quantidade de cocos do macaco e a terceira *String* é o vetor de cocos em forma que para resolver isso e transforma-lo inicialmente em vetor de *Strings*, tivemos que fazer a mesma coisa feita do primeiro vetor só que agora será de acordo com os espaços, sendo esse o nosso último vetor, a *LinkedList<Integer>* seria o segundo vetor transformado em uma *LinkedList<Integer>*, que para isso criamos um método que retorna uma *LinkedList<Integer>*, essa transformação é feita por função recursiva transformando os conteúdos do vetor de *Strings* em inteiros e os transferindo para uma nova *LinkedList*. Também temos uma variável que armazena o número de rodadas. Por último temos uma função "for" aonde nessa conta quantos cocos pares cada macaco possui e a criação da variável do tipo Macaco onde irá armazenar a quantidade de cocos pares e ímpares, e os índices dos macacos que receberão os cocos pares e ímpares.

O algoritmo que faz todas as rodadas do bingo é muito simples, para fazê-lo criamos inicialmente uma função recursiva que representas as rodadas, e dentro dela outra função recursiva, em que dentro dessa fizemos as somas dos cocos ímpares e pares dos macacos que receberam cocos pares e ímpares com esses cocos, assim no final da recursão zeramos os cocos do macaco atual e atualizamos valores de quantidades de cocos pares e impares dos outros macacos. Sendo o pseudocódigo desse algoritmo:

```
Para cada rodada
    Para cada macaco na LinkedList<Macacos>
        //A quantidade de cocos pares que o macaco recebedor terá
        int somaDosCocosPares = macaco.getCocosPares + cocos do recebedor de cocos pares

        //A quantidade de cocos ímpares que o macaco recebedor terá
        int somaDosCocosImpares = macaco.getCocosImpares + cocos do recebedor de cocos ímpares

        setCocosPares(somaDosCocosPares) do macaco recebedor dos cocos pares
        setCocosImpares(somaDosCocosImpares) do macaco recebedor dos cocos ímpares

        macaco. setCocosPares(zero)
        macaco. setCocosImpares(zero)
    Fim do para-cada
Fim do para-cada
```

Para o algoritmo que depois de todas as rodadas ele verifica qual dos macacos tem mais cocos e armazena seu número, fizemos outra função recursiva em que passa por todos os macacos da lista, armazena a quantidade de cocos, sendo o primeiro macaco inicialmente se tornando o chefe, e se mais para frente da função houver um outro com maior número de cocos, o chefe passa o título para esse, assim por diante até acabar a recursão. Para melhor compreensão do algoritmo este é o pseudocódigo:

```
Inteiro indiceChefe = zero
Inteiro quantidadeCocosChefe = zero
Para cada macaco em lista de macacos
        Inteiro cocosTotal = número de cocos do macaco
        Se cocosTotal > quantidadeCocosChefe então
                indiceChefe = indiceMacacoAtual
                quantidadeCocosChefe = cocosTotal
        Fim-se
        indiceMacacoAtual = ele + 1
Fim para-cada
```

Resultados

Ao terminar de executar os algoritmos acima em linguagem Java, nossos resultados foram estes abaixo:

Caso 50 – Macaco chefe foi o número 9 com 2332 cocos;

Caso 100 – Macaco chefe foi o número 20 com 15461 cocos;

Caso 200 – Macaco chefe foi o número 38 com 74413 cocos;

Caso 400 – Macaco chefe foi o número 36 com 145232 cocos;

Caso 600 – Macaco chefe foi o número 177 com 230276 cocos;

Caso 800 – Macaco chefe foi o número 20 com 182575 cocos;

Caso 900 – Macaco chefe foi o número 589 com 433295 cocos;

Caso 1000 – Macaco chefe foi o número 144 com 581995 cocos;

Conclusão

Os resultados obtidos na solução comprovam que a abordagem tomada para a solução do algoritmo tornou-se muito eficiente, especialmente para os casos de teste maiores, porque no começo do trabalho havíamos programado um outro programa mais complexo, ineficiente para o que havia sido imposto para solucionarmos e extremamente lento quando chegava nos casos maiores, e com mais volume de macacos. O algoritmo de achar o macaco com mais cocos no final do jogo foi muito simples de ser feito, também sendo o jeito mais eficiente que encontramos.

Embora soubermos que exista soluções melhores para algumas coisas no algoritmo, como o meio de armazenar os dados dos macacos, que poderia ser por matrizes, entretanto não temos certeza se realmente isso iria deixar mais rápido ou eficiente nosso programa, por isso optamos pelo que foi dito anteriormente. Entretanto é importante salientar que a busca de melhorias para a otimização do programa pode proporcionar oportunidades futuras em que permitirá o aprimoramento do algoritmo.