PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
CURSO DE CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL
DISCIPLINA DE ALGORITIMOS E ESTRUTURAS DE DADOS II
PROFESSOR JOÃO BATISTA DE SOUZA DE OLIVEIRA
AUTOR: MATHEUS MAIA DA SILVA

Resumo

Este artigo tem o objetivo de resolver um problema de estudo antropológico sobre o comportamento social de um grupo de macacos, que foi feito selecionando como objeto de estudo uma atividade entre os primatas que funcionava de forma semelhante a um jogo, na qual possuía regras e um vencedor. O problema que iremos tratar aqui, portanto, se dá em solucionar o macaco vencedor com base em todas as observações feitas pelos antropólogos pesquisadores utilizando a linguagem de programação Java. Durante o desenvolvimento deste artigo serão apresentadas uma hipótese de solução e uma solução real. Por fim, serão apresentados todos os resultados obtidos com o programa criado.

Introdução

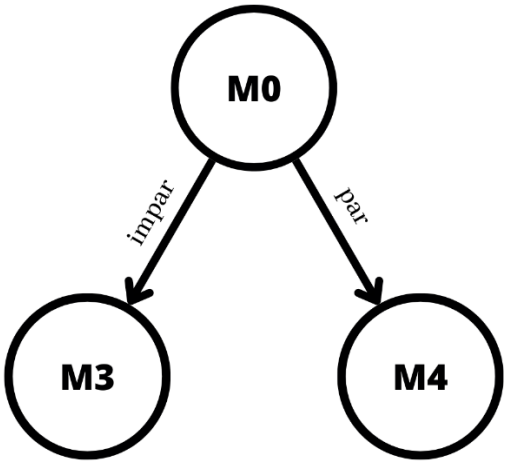
Em um estudo antropológico sobre o dia a dia de um grupo de macacos, foi notada uma atividade coletiva que definia o líder do grupo na semana. Essa atividade consistia em cada macaco reunir uma determinada quantidade de cocos, de tal forma que cada coco contivesse uma quantidade qualquer de pedras, podendo ser *par* ou *ímpar*. Com isso, cada macaco teria a tarefa de entregar os seus cocos para outros dois macacos fixos: com quantidades pares para um macaco *X* e com quantidades ímpares para um macaco *Y*. Isto é feito por cada macaco e realizado em uma quantidade definida de rodadas. Após todas as rodadas serem concluídas, a atividade termina e o macaco com o maior número de pedras é o vencedor.

Para realizar tal estudo, os pesquisadores enumeraram cada macaco e anotaram para quais outros dois ele entregaria os seus cocos. A fim de simplificar a notação, resumiremos o macaco a que estamos nos referindo por Mn, tal que *n* é o número do macaco em questão. Portanto, M2 é a abreviação de Macaco 2. Abaixo é dada uma anotação para tomarmos como exemplo:

Macaco 0 par -> 4 impar -> 3 : 11 : 178 84 1 111 159 22 54 132 201 51 44

Esta anotação nos informa que M0 entrega os seus cocos de quantidade *par* para M4 e os de quantidade *ímpar* para M3. Em seguida, é dada a quantidade total de cocos que ele possui (11) seguido de uma lista com a quantidade de pedras que cada um destes cocos carrega (178, 84, 1, ..., 44). Assim, tem-se que dos 11 cocos que o M0 possui, 6 deles carregam uma quantidade *par* de pedras e 5 carregam uma quantidade *ímpar*. Esta lógica se aplica a cada macaco do grupo, que irá receber um conjunto de cocos e, após agrupar com os seus, dividirá este conjunto e

enviará para outros dois fixos. Podemos visualizar melhor a distribuição de entregas de cocos com o esquema:



A partir do contexto dado acima, o problema a ser solucionado se dá no cálculo do macaco vencedor, visto que vários jogos foram observados e que cada um envolvia diferentes quantidades de participantes e com diferentes rodadas de distribuição. Porém, considerando que as anotações de distribuição foram feitas após todos os macacos já terem sido enumerados e que este cálculo será feito por um algoritmo, surge a necessidade de sistematizar a resolução para que ela possa ser feita de forma eficiente.

Como veremos a seguir, das duas soluções pensadas, ambas possuíam o princípio de ler o arquivo e organizar as anotações de macacos em uma lista de objetos em que cada objeto é um macaco com todas as suas informações e todos os métodos necessários para lidar com elas. Dois métodos importantes de serem mencionados são os de entregar as quantidades de cocos (seja as pares ou ímpares) e as de receber os cocos de outros macacos. Ou seja, cada vez que um macaco entrega uma quantidade para outro, a sua quantidade total é subtraída da que será entregue, enquanto a quantidade total do macaco que recebe é somada com a recebida. Assim podemos garantir que todas as informações são modeladas a todo momento.

Solução descartada

A primeira solução pensada, mas não implementada, seria realizar a distribuição de cocos partindo da primeira anotação feita e continuando a partir dos macacos que recebessem. Por exemplo, se M0 entrega para M3 e M4, então após o algoritmo retirar a quantidade de cocos de M0 e entregar aos outros dois, o mesmo processo seria feito para cada um deles. Em termos de

estrutura de dados teríamos uma organização e distribuição feita em uma Árvore Binária. Porém, observando as anotações, não é difícil perceber alguns problemas nesta implementação. Abaixo uma amostra um pouco maior das notas de exemplo:

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

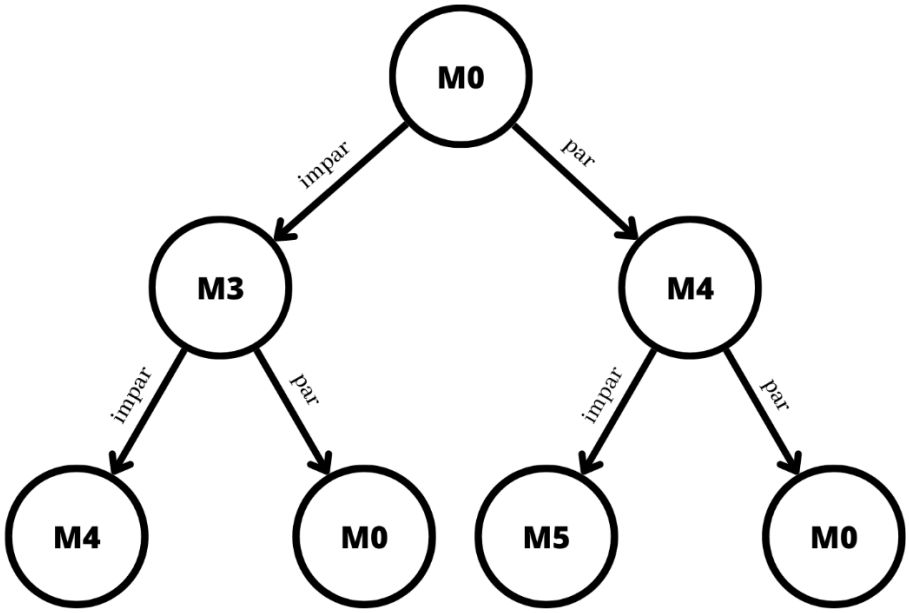
...

Macaco 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

Como podemos ver, seguindo a ideia inicial, após o M0 distribuir seus cocos a próxima distribuição seria feita com M4, mas este está a um indeterminado número de linhas abaixo de M0. Com uma lista de milhares de objetos macaco, podendo o primeiro enviar a outro de mil posições a frente e um outro de outras duas mil atrás, teríamos um algoritmo que faria muitas voltas sobre a estrutura, implicando em um maior tempo de execução pela alta quantidade de passos a serem dados. Podemos ver, com a figura abaixo, como os primeiros níveis da árvore seriam estruturados.

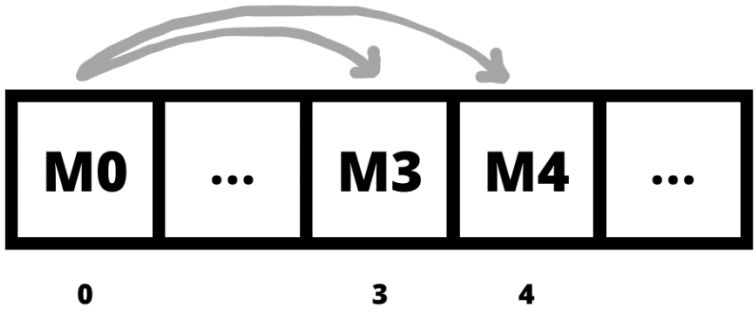


Outros dois problemas a serem considerados estaria na própria montagem da árvore, que por si só também exigiria que o algoritmo desse outras tantas voltas, e, por fim, no próprio cálculo do

vencedor que a priori deveria ser feito após todas as distribuições, aumentando ainda mais a complexidade final do algoritmo e implicando no seu tempo de execução.

Solução final

Após descartar a solução com Árvore Binária foi levantada uma segunda ideia que, após realizar alguns testes, se demonstrou simples e eficiente. Seguindo a lógica inicial de ler o arquivo, criar os objetos e acrescentá-los em um ArrayList, foi pensado na possibilidade de fazer a distribuição a partir desta própria estrutura, visto que o tempo de busca para um elemento nela é de O(n). Assim, não haveria mais nada a ser feito se não criar a lista, distribuir e calcular.



A imagem acima mostra o quão simples é a distribuição de cocos utilizando ArrayList. Temos, portanto, que cada posição desta lista é um objeto do tipo Macaco. Se cada objeto faz duas distribuições, uma para cocos de quantidades ímpares de pedras e outra para quantidades pares, utilizamos uma função que obtém o objeto da posição que se queira e modificamos os atributos dele com os métodos *getters* e *setters.* Este processo é repetido o tanto de rodadas que se pede em cada arquivo.

Em relação ao código, definimos um loop que irá iterar a quantidade de rodadas definida em cada arquivo e fazendo a instanciação de cada objeto com seus respectivos atributos. De forma mais detalhada, temos que inicialmente, ao entrar na iteração de leitura, o algoritmo extrai o conteúdo da linha que se está e transforma ela em um vetor, extrai seus dados e, por fim, instancia um objeto do com todos os dados obtidos. Num caso de exemplo seria *[[Macaco 3 par -> 0 impar -> 4], [3], [121 10 162]].*

linha = leLinha().separaElementos(linha, “ : ”)

numeroDoMacaco = linha[0].separaElemento(numeroDoMacaco, “ ”)[1]

enviarQtdPar = linha[0].separaElemento(enviarPar, “ ”)[4]

enviarQtdImpar = linha[0].separaElemento(enviarQtdImpar, “ ”)[4]

qtdTotal = linha[1]

vetorPar = linha[2].obterElementosPares()

qtdDeCocosPar = vetorPar.tamanho()

qtdDeCocosImpar = qtdTotal – qtdPar

listaDeMacacos.adicionar(novo Macaco(numeroDoMacaco, enviarQtdPar, enviarQtdImpar, qtdTotal qtdDeCocosPar, qtdDeCocosImpar))

Depois de fazer o total de iterações necessárias é calculado o macaco vencedor considerando que o primeiro da lista é o que possuí a maior quantidade de cocos e comparando com o próximo, se o próximo tiver uma quantidade maior então ele será o novo vencedor. Isto se repetirá em uma iteração sobre todos os objetos.

Resultados

Após a implementação e execução do algoritmo, obtivemos os seguintes resultados.

caso0050.txt

RODADAS = 5000

Vencedor: 9

qtdTotal: 2332

Tempo de execução: 134 milissegundos

caso0100.txt

RODADAS = 10000

Vencedor: 20

qtdTotal: 15461

Tempo de execução: 148 milissegundos

caso0200.txt

RODADAS = 20000

Vencedor: 38

qtdTotal: 74413

Tempo de execução: 281 milissegundos

caso0400.txt

RODADAS = 40000

Vencedor: 36

qtdTotal: 145232

Tempo de execução: 627 milissegundos

caso0600.txt

RODADAS = 60000

Vencedor: 177

qtdTotal: 230276

Tempo de execução: 1380 milissegundos

caso0800.txt

RODADAS = 80000

Vencedor: 20

qtdTotal: 182575

Tempo de execução: 1917 milissegundos

caso0900.txt

RODADAS = 90000

Vencedor: 589

qtdTotal: 433295

Tempo de execução: 2257 milissegundos

caso1000.txt

RODADAS = 100000

Vencedor: 144

qtdTotal: 581995

Tempo de execução: 2719 milissegundos

Conclusões

Podemos concluir que, embora tenha sido utilizada dois tipos de estruturas de dados, uma levantada como hipótese e uma de fato utilizada, a forma de resolução deste problema poderia ser ainda mais variada, utilizando estruturas ainda mais sofisticadas como Grafos, já que, como mostrado na árvore, se um nodo tivesse uma relação mais flexível com seus adjacentes, as informações poderiam ter sido repassadas de forma mais rápida e eficiente. Porém, ainda assim, acreditamos que a forma utilizada foi consistente para o problema.