**Resumo**

Para simplificar, resumiremos o macaco a que estamos nos referindo por Mn, tal que *n* é o número do macaco em questão. Por exemplo, M2 é a abreviação de Macaco 2.

Duas soluções utilizando estruturas de dados distintas foram pensadas para resolver o problema. A primeira, que foi descartada, utilizaria uma Árvore Binária para fazer a distribuição de cocos, e a segunda, que foi a utilizada na versão final, utiliza ArrayList.

Para simular um macaco na atividade, foi criado uma classe que permitisse implementar objetos do tipo Macaco e que se comportasse da mesma forma que eles faziam na atividade: recebendo e entregando cocos de e para outros macacos. Com isto foi criada uma ArrayList de objetos para simular a distribuição tal como foi anotada, ou seja, para toda entrega feita por *x(i)*, onde *i* é a posição do macaco na lista, havia outros dois y e *z* para receber esta entrega

O que o algoritmo faz: lê o arquivo, cria uma lista com ArrayList com objetos macaco e faz a distribuição.

, e considerando que o primeiro passo seria implementar uma estrutura de dados Lista que organizasse as informações da mesma forma que foram anotadas (para que o algoritmo percorresse o arquivo uma única vez),

**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 quantidade ímpares para um macaco *Y*. Isto é feito por cada macaco e realizado em uma quantidade 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. Um exemplo de anotação é dado abaixo:

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

Isso os informa que o M0 entrega os seus cocos de quantidade *par* para o M4 e os de quantidade *ímpar* para o M3 3. 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, enviará dividirá este conjunto e enviará para outros dois fixos. Podemos visualizar melhor a distribuição de entregas de cocos com o esquema abaixo:

Diagrama

Descrição gerada automaticamente

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 do problema para que ele possa ser solucionado de forma eficiente.

**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 macacos, fazer o mesmo processo para cada um deles. Em termos de estrutura de dados teríamos esta 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 determinado 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 dado. Podemos ver, com a figura abaixo, como os primeiros níveis da árvore seriam estruturados.

Diagrama

Descrição gerada automaticamente

Outros dois problemas a serem considerados estaria na própria montagem da árvore, que por si só também exigiria que o algoritmo desse outras tantas 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.

Calendário

Descrição gerada automaticamente

A imagem acima mostra o quão simples é a distribuição de cocos utilizando ArrayList.

Resultados

Conclusões