# Algoritmos e Estruturas de Dados II

# Trabalho 1 (Os Macaquinhos)

**Lara Alves Kunrath Padilha, Leonardo Garcia Forquim Bertinatto**

PUCRS

## 1. Introdução

Um antropólogo e explorador passou um tempo estudando um grupo de macaquinhos, e descobriu que eles possuem um jogo de troca de cocos com pedrinhas dentro, em que o vencedor terá uma semana de tratamentos especiais dentro do grupo. A partir disso, esse antropólogo passou juntar informações a respeito das partidas do jogo, mas as vezes, por serem muitas informações, ele não sabia prever o macaquinho vencedor ou até mesmo desistia de seguir com a pesquisa sobre o comportamento desse grupo de macaquinhos. Então nos foi pedido um simulador de jogada, que fosse capaz de ler as informações sobre uma partida e a partir disso prever qual seria o macaquinho vencedor.

## 2. O problema

A partir da história do antropólogo, nossos problemas eram os seguintes:

- Ler arquivos grandes com diferentes informações;

- Gerenciar essas informações;

- Calcular o vencedor do jogo.

## 3. Processo de solução

O processo de solução inicial possuía diferentes ArrayLists que armazenavam todas as informações fornecidas nos casos de teste, mas ao final disso vimos que o código acabava demorando muito tempo para concluir a execução, podendo levar horas. Com isso, se tornou um quesito para nós melhorar a eficiência do código, visando economizar o tempo do antropólogo ao tentar simular um jogo.

Para isso, optamos por diminuir a quantidade de informação que estávamos salvando, e levamos em consideração apenas o número de rodadas pedido (salvo como contador de partidas), o número de cocos pares e impares de cada macacos e quais eram os destinatários dos respectivos cocos, estes últimos foram salvos em diferentes vetores na classe “Jogo”. Então, tratamos apenas de relacionar os cocos de um macaquinho com o seus respectivos destinatários pares e impares, o tratamento foi realizado na classe “Macaquinho”.

Agora para descobrimos o vencedor, foi feita uma variação da estrutura de busca Bubble Sort, para que encontrássemos o índice do macaco vencedor salvo no ArrayList de macaquinhos, comparando a quantidade de cocos entre eles.

Por fim, para obtermos uma análise mais profunda dos algoritmos utilizados, foi adicionado um contador de tempo, visando saber o tempo de execução de cada caso de

teste fornecido de acordo com os diferentes algoritmos que estávamos testando, afim de tentar otimizar o máximo possível o tempo de execução

## 4. Evidências de que o problema foi resolvido

Os resultados esperados eram os seguintes:

Com 50 macacos | macaco vencedor é o 14 com 1847 cocos

Com 100 macacos | macaco vencedor é o 88 com 16327 cocos

Com 200 macacos | macaco vencedor é o 9 com 17043 cocos

Com 400 macacos | macaco vencedor é o 109 com 402270 cocos

Com 600 macacos | macaco vencedor é o 49 com 307830 cocos

Com 800 macacos | macaco vencedor é o 33 com 626323 cocos

Com 1000 macacos | macaco vencedor é o 77 com 333394 cocos

Com 2000 macacos | macaco vencedor é o 539 com 4844003 cocos

Os resultados obtidos foram os seguintes:

**Caso de teste 1**

```
Digite o nome completo do arquivo .txt que você deseja testar: 0-0050macacos.txt
Arquivo testado: 0-0050macacos.txt

Jogando...

 - O macaquinho vencedor é o número 14, com 1847 cocos
 - O jogo foi executado em 221 milisegundos
```

**Caso de teste 2**

```
Digite o nome completo do arquivo .txt que você deseja testar: 0-0100macacos.txt
Arquivo testado: 0-0100macacos.txt

Jogando...

 - O macaquinho vencedor é o número 88, com 16327 cocos
 - O jogo foi executado em 315 milisegundos
```

**Caso de teste 3**

```
Digite o nome completo do arquivo .txt que você deseja testar: 0-0200macacos.txt
Arquivo testado: 0-0200macacos.txt

Jogando...

 - O macaquinho vencedor é o número 9, com 17043 cocos
 - O jogo foi executado em 307 milisegundos
```

**Caso de teste 4**

```
Digite o nome completo do arquivo .txt que você deseja testar: 0-0400macacos.txt
Arquivo testado: 0-0400macacos.txt

Jogando...

 - O macaquinho vencedor é o número 109, com 402270 cocos
 - O jogo foi executado em 594 milisegundos
```

**Caso de teste 5**

```
Digite o nome completo do arquivo .txt que você deseja testar: 0-0600macacos.txt
Arquivo testado: 0-0600macacos.txt

Jogando...

 - O macaquinho vencedor é o número 49, com 307830 cocos
 - O jogo foi executado em 1,1 segundos
```

**Caso de teste 6**

```
Digite o nome completo do arquivo .txt que você deseja testar: 0-0800macacos.txt
Arquivo testado: 0-0800macacos.txt

Jogando...

 - O macaquinho vencedor é o número 33, com 626323 cocos
 - O jogo foi executado em 1,8 segundos
```

**Caso de teste 7**

```
Digite o nome completo do arquivo .txt que você deseja testar: 0-1000macacos.txt
Arquivo testado: 0-1000macacos.txt

Jogando...

 - O macaquinho vencedor é o número 77, com 333394 cocos
 - O jogo foi executado em 2,5 segundos
```

**Caso de teste 8**

```
Digite o nome completo do arquivo .txt que você deseja testar: 0-2000macacos.txt
Arquivo testado: 0-2000macacos.txt

Jogando...

 - O macaquinho vencedor é o número 539, com 4844003 côcos
 - O jogo foi executado em 7,2 segundos
```

## 5. Conclusão

Na resolução do problema proposto, nos deparamos primeiramente com a definição de como iríamos armazenar as informações e fazer com que elas estivessem ligadas ao componente principal que é o macaquinho. Para isso, foram utilizadas estruturas de dados ArrayList e vetores. A construção do código é simples, mas descobrimos que diferentes formas de resolver o problema podem afetar a eficiência da execução, podendo levar horas para executar casos de teste muito grandes, a partir disso nos focamos em otimizar a execução para todos os casos de teste, e descobrimos que

não era preciso salvar todas as informações como, quantidade de macaquinhos, quantidade de cocos de cada macaquinho, quantidade de pedrinha de cada macaquinho, etc, ao invés disso, salvamos apenas a quantidade cocos impares e pares, e qual o destinatário de cada um, diminuindo assim a quantidade de informações que precisavam ser processadas e calculadas, otimizando o tempo de execução.

Por fim, mesmo que a construção da atividade fosse simples, serviu para pensarmos em diferentes formas de melhorar um código, mesmo que o quesito eficiência não seja avaliado, nos sentimos desafiados a entregar algo melhor que o pedido.