

# Jogo Troca-Cocos dos Macaquinhos

Artur Wiederkehr\*

Escola Politécnica – PUCRS

27 de março de 2023

## Resumo

Este artigo discorre sobre o Trabalho 1 da disciplina Algoritmos e Estruturas de Dados 2 que consiste em auxiliar um antropólogo que observou um jogo de bingo primitivo realizado entre macaquinhos que transferem cocos de acordo com a paridade (ou disparidade) de pedras dentro de cada coco.

Com arquivos que replicam os cenários iniciais, somado ao conhecimento das regras, é possível criar um algoritmo que simula o jogo e nos fornece o resultado informando quem venceu e com quantos cocos. Cada cenário possui mais participantes e mais rodadas do que o anterior, apesar do aumento do número de operações, é apresentada uma solução plausível e averiguada sua eficácia e alternativas, junto de um breve histórico do processo de resolução.

## Introdução

Um antropólogo e explorador da selva observou, durante 6 meses, o comportamento de macaquinhos em seu habitat natural e registrou detalhadamente as regras de um jogo coletivo praticado entre eles, juntos de alguns cenários destes jogos. O objetivo é apresentar quais macaquinhos venceram a competição em cada cenário seguindo as regras do jogo, que são:

1. Antes do jogo há uma etapa de preparação: cada macaquinho enche quantos cocos quiser com quantas pedrinhas quiser.
2. Cada macaquinho sabe o destinatário para seus cocos com número par de pedrinhas e para seus cocos com número ímpar de pedrinhas.
3. Os destinatários e o número de pedrinhas são inalteráveis durante o jogo.
4. No início de cada rodada, o primeiro macaquinho distribui todos os seus cocos e assim sucessivamente até o último, encerrando a rodada. Um coco pode ser transferido múltiplas vezes na mesma rodada.
5. No final de um número determinado de jogadas, o macaquinho com maior número de cocos é o campeão do jogo.

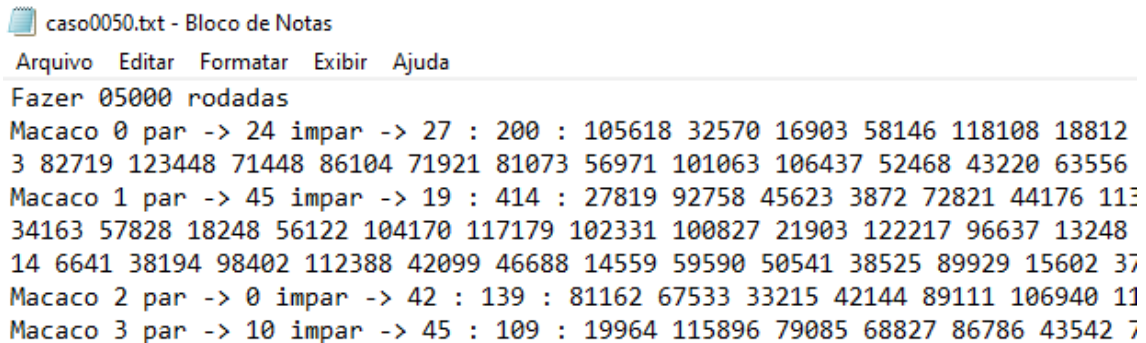
O esforçado antropólogo disponibilizou alguns arquivos do tipo texto (.txt) com os cenários iniciais de cada jogo, estes são os dados a serem executados pelo código para que a informação seja extraída. Antes de pensar em códigos e algoritmos, devemos ter uma boa interpretação sobre o problema e a abordagem aplicada sobre ele, uma má interpretação poderia resultar em um erro grosseiro ou, na melhor das hipóteses, um resultado correto com muito esforço desnecessário e recursos desperdiçados. Apresenta-se a seguir a interpretação do cenário, partes importantes do código, resultados e conclusões obtidas.

---

\*artur.wiederkehr@edu.pucrs.br

# Resolução

Inicia-se pela entrada de dados, todos os arquivos apresentam os dados no mesmo formato, logo deve-se ler estes arquivos, selecionar as informações necessárias, executar o jogo seguindo as regras e informar o campeão. Abaixo uma imagem de uma parte de um dos arquivos:



```
caso0050.txt - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
Fazer 05000 rodadas
Macaco 0 par -> 24 impar -> 27 : 200 : 105618 32570 16903 58146 118108 18812
3 82719 123448 71448 86104 71921 81073 56971 101063 106437 52468 43220 63556
Macaco 1 par -> 45 impar -> 19 : 414 : 27819 92758 45623 3872 72821 44176 113
34163 57828 18248 56122 104170 117179 102331 100827 21903 122217 96637 13248
14 6641 38194 98402 112388 42099 46688 14559 59590 50541 38525 89929 15602 37
Macaco 2 par -> 0 impar -> 42 : 139 : 81162 67533 33215 42144 89111 106940 11
Macaco 3 par -> 10 impar -> 45 : 109 : 19964 115896 79085 68827 86786 43542 7
```

Deve-se extrair o elemento central da primeira linha para ser o número de rodadas e, após cada linha, seguir o padrão para extrair as informações necessárias, por exemplo: o macaquinho 0 envia seus cocos com número par de pedrinhas para o macaquinho número 24, os cocos com o número ímpar de pedrinhas para o 27 e possui 200 cocos no início do jogo. Há 5 fatores importantes observados por macaquinho: destinatário par/ímpar, quantidade de cocos par/ímpar e a quantidade inicial de cocos.

Uma vez compreendido o cenário e o comportamento dos dados, define-se uma estratégia: para lidar com estes e manipulá-los, pode-se criar uma lista de macaquinhos e consultar os seus valores para realizar as transferências de cocos. A lista guardará todas as informações dos macaquinhos que também serão em formato de lista, resumindo: trabalharemos com uma lista de listas. Como adicionaremos os macaquinhos na ordem, o índice deles é, convenientemente, o mesmo da lista principal.

Por praticidade e facilidade na manipulação de dados, optou-se pela linguagem de programação Python. Com o uso da função *with open()* [1] (abre o arquivo e o retorna como objeto) e a função *ReadLines()* [2] (retorna as linhas do arquivo em formato de lista) é possível que leiamos os cenários. Imprime-se o nome do arquivo por questão de organização.

Através da observação realizada anteriormente, identifica-se qual a posição que a informação do nosso interesse estará, entretanto, cada item da lista é um valor do tipo *String* e buscar pelo índice retornará apenas um caractere, portanto usa-se a função *split()* [3] que divide os elementos de acordo com o parâmetro. Com o uso de um laço *for each*, passa-se por todos os elementos da lista de frases e transforma todas as frases em listas de palavras.

Usa-se o trecho de código que confeccionamos até então, faz-se um arquivo copiado do “caso0050.txt”, mas com apenas os 4 primeiros macaquinhos para a realização de um teste:

```
for frase in f.readlines():
```

```
    palavra = frase.split(" ")
```

```
    if ('Macaco' in palavra):
```

```
        direcaoPar = palavra[4]
```

```
        direcaoImpar = palavra[7]
```

```
        numCocos = int(palavra[9])
```

```

numCocosPar = 0
for i in range(11, numCocos + 11):
    numero = int(palavra[i])
    print(numero)
    if numero % 2 == 0:
        numCocosPar = numCocosPar + 1
print("Macaco " + palavra[1])
print("Número de cocos par: " + str(numCocosPar))
print("Número de cocos ímpar: " + str(numCocos - numCocosPar))
print("Número de cocos total: " + str(numCocos))
elif rodadasTotal == 0:
    rodadasTotal = int(palavra[1])
    print("Rodadas: " + str(rodadasTotal))

```

#### Saída:

Executando arquivo: casoTeste.txt

Rodadas: 5000

105618

32570

[...]

28826

Macaco 3

Número de cocos par: 62

Número de cocos ímpar: 47

Número de cocos total: 109

Busca-se os resultados de cada macaquinho no terminal e reserva-os para comparação, copia-se os números que representam os cocos, cola-os no *excel* no formato de coluna e cria-se uma tabela, utiliza-se a função *MOD()* para saber a paridade/disparidade de cada coco e por fim usa-se *CONT.NÚM()* e *SOMA()* para sabermos, respectivamente, quantos cocos são no total e quantos destes são ímpares, consequentemente descobre-se os pares. Os resultados convergem, significa que estamos no caminho certo.

Macaco 0	Macaco 1	Macaco 2	Macaco 3
Número de cocos par: 93	Número de cocos par: 207	Número de cocos par: 59	Número de cocos par: 62
Número de cocos ímpar: 107	Número de cocos ímpar: 207	Número de cocos ímpar: 80	Número de cocos ímpar: 47
Número de cocos total: 200	Número de cocos total: 414	Número de cocos total: 139	Número de cocos total: 109

Macaco 0	Resto	Macaco 1	Resto	Macaco 2	Resto	Macaco 3	Resto
14899	1	75802	0	57198	0	28826	0
105156	0	77721	1	11661	1	109	47
102765	1	5182	0	67351	1		
28256	0	98426	0	139	80		
107429	1	70011	1				
200	107	56701	1				
		114117	1				
		414	207				

(Linhas foram ocultadas com propósito de mostrar os resultados das colunas na mesma imagem)

Com os dados prontos, necessita-se executar a ação das rodadas, então faz-se um laço *for* para elas e, por se tratar de uma lista de macaquinhos, deve-se fazer um *for* encadeado para que, a cada rodada, a lista seja percorrida e as transferências realizadas. Dentro deste *for* encadeado faz-se duas verificações: se a quantidade de cocos com número par de pedrinhas do macaquinho é diferente de zero, então realiza-se a operação de transferir, repete-se essa lógica para os cocos com número ímpar de pedrinhas, estas duas verificações tornam o algoritmo mais eficiente pois economiza tempo de execução e operações desnecessárias. O código mostrado a seguir é responsável pela execução das rodadas do jogo, todavia não informa o campeão.

```
for i in range(0, rodadasTotal):
    for j in range(0, len(listaMacacos)):
        if listaMacacos[j][2] != 0:
            direcaoPar = int(listaMacacos[j][0])
            listaMacacos[direcaoPar][2] += listaMacacos[j][2]
            listaMacacos[j][2] = 0
        if listaMacacos[j][3] != 0:
            direcaoImpar = int(listaMacacos[j][1])
            listaMacacos[direcaoImpar][3] += listaMacacos[j][3]
            listaMacacos[j][3] = 0
```

Testar a execução deste trecho manualmente seria muito trabalhoso se feito até com o caso com menor número de macaquinhos e rodadas, por isto teremos que nos contentar em fazer um teste com baixíssimo número de operações e acompanharmos as variáveis durante a execução, usando um *for* para imprimir-las antes de cada rodada e outro para após a última rodada. Tem-se a entrada:

Fazer 00002 rodadas

Macaco 0 par -> 4 ímpar -> 1 : 3 : 1 2 3

Macaco 1 par -> 2 ímpar -> 3 : 3 : 1 2 5

Macaco 2 par -> 1 ímpar -> 0 : 3 : 1 2 4

Macaco 3 par -> 2 impar -> 4 : 3 : 1 1 3

Macaco 4 par -> 3 impar -> 0 : 2 : 2 2

**Execução manual (rodada 0 concluída):**

Macaco 0 par -> 4 impar -> 1 : 3 : 1 1 1 1 3 1 5 1 3

Macaco 1 par -> 2 impar -> 3 : 3 : 2 4 2

Macaco 2 par -> 1 impar -> 0 : 3 :

Macaco 3 par -> 2 impar -> 4 : 3 : 2 2 2

Macaco 4 par -> 3 impar -> 0 : 2 :

**Execução manual (rodada 1 concluída):**

Macaco 0 par -> 4 impar -> 1 : 3 : 1 1 1 1 3 1 5 1 3

Macaco 1 par -> 2 impar -> 3 : 3 : 2 4 2

Macaco 2 par -> 1 impar -> 0 : 3 : 2 2 2

Macaco 3 par -> 2 impar -> 4 : 3 :

Macaco 4 par -> 3 impar -> 0 : 2 :

**Saída (dividida exclusivamente no documento para propósito de visualização):**

Rodada: 0	Pares	Ímpares	Rodada: 0	Pares	Ímpares	Rodada: 1	Pares	Ímpares
Macaco 0:	1	2	Macaco 0:	0	8	Macaco 0:	0	8
Macaco 1:	1	2	Macaco 1:	3	0	Macaco 1:	3	0
Macaco 2:	2	1	Macaco 2:	0	0	Macaco 2:	3	0
Macaco 3:	0	3	Macaco 3:	3	0	Macaco 3:	0	0
Macaco 4:	2	0	Macaco 4:	0	0	Macaco 4:	0	0

A execução do teste manual é uma tentativa de prever o comportamento do código em casos com maior número de operações, a princípio as transferências foram executadas corretamente.

A parte final do algoritmo passa pela lista de macaquinhos uma última vez para descobrir quem é o campeão. Para sabermos qual o tempo de execução importa-se a biblioteca *time*, armazena o valor da função *time.time()* no início da execução do algoritmo e subtrai pela mesma função ao final do programa, [4] desta forma teremos o tempo de execução em segundos, como demonstrado a seguir.

```
for i in range(0, len(listaMacacos)):
    if listaMacacos[i][2] + listaMacacos[i][3] > qtdCampeao:
        qtdCampeao = listaMacacos[i][2] + listaMacacos[i][3]
        numCampeao = i

print("O macaco campeão é o de número " + str(numCampeao) + "! " + "Com um total de " + str(qtdCampeao) + " cocos!")

segundos = (time.time() - tempoInicio)

print(str(int(segundos / 60)) + " minutos e " + str(segundos % 60) + " segundos.")
```

### Saída (Considerando todo o algoritmo):

Executando arquivo: caso1000.txt

O macaco campeão é o de número 144! Com um total de 581995 cocos!

0 minutos e 19.6239652633667 segundos.

Em seguida, o quadro de resultados da execução de cada caso:

Nome do Documento	Número de Rodadas	Tempo de Execução Médio (s)	Número do Campeão	Quantidade de Cocos do Campeão
caso0050.txt	5000	0,07	9	2332
caso0100.txt	10000	0.27	20	15461
caso0200.txt	20000	0,90	38	74413
caso0400.txt	40000	3,50	36	145232
caso0600.txt	60000	7,15	177	230276
caso0800.txt	80000	12,72	20	182575
caso0900.txt	90000	15,35	589	433295
caso1000.txt	100000	19.62	144	581995

(O número de macaquinhos é igual ao número presente no nome do documento)

## Conclusão

A confecção da resolução foi um processo constituído passo a passo que ocasionou em um algoritmo simples e eficaz, suas partes foram testadas conforme o avanço de seu desenvolvimento e apresentam um resultado satisfatório além da fácil manutenibilidade.

Existem algumas outras abordagens que não foram testadas: possivelmente a programação paralela tornaria o algoritmo mais eficiente, um processador transfere os cocos com paridade de pedrinhas e outro com disparidade, por exemplo; um algoritmo mais complexo que observa o destino de um grupo de cocos: se A passa para o B e B passa para o C, então A e B podem mandar diretamente para C, exceto se C vier antes de B para não desrespeitar a ordem.

## Breve Histórico de Erros

Inicialmente o programa seria escrito em *Java* para a utilização de listas encadeadas de forma que bastaria ter 2 listas, alterar as referências e comparar o tamanho no final, entretanto esta ideia foi descartada, pois notou-se que bastava saber a quantidade de cocos pares/ímpares.

A função *split()* foi empregada de forma inocente, sem entender detalhadamente sua operação, sabia-se que ela retornaria apenas uma palavra caso fosse seguida de um índice. Entretanto se um segundo parâmetro não for informado (que é um limitador), a função retorna uma lista com todos os elementos, ela era convocada múltiplas vezes por linha e sempre apenas um dado era extraído, a execução do “caso1000.txt” demorou mais de 40 minutos na primeira tentativa.

Utilizou-se o conceito de orientação a objetos para ter mais controle e segurança sobre os dados com a utilização de construtores, *getters* e *setters*. Infelizmente esta ideia aumentou consideravelmente o tempo de execução sem prover benefícios que a justificasse.

## Referências

- [1] Documentação Python, Função ReadLines (não foi possível marcar o local exato):  
<https://docs.python.org/pt-br/3/tutorial/inputoutput.html>
- [2] Documentação Python, Função With Open:  
<https://docs.python.org/pt-br/3/library/functions.html#open>
- [3] Documentação Python, Função Split:  
<https://docs.python.org/pt-br/3.8/library/stdtypes.html#str.split>
- [4] Documentação Python, Função Time:  
<https://docs.python.org/3/library/time.html#time.time>