Macaquinhos

Ana Carolina de Oliveira Xavier
a.xavier004@edu.pucrs.br

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Algoritmos e Estruturas de Dados II
Professor João B. Oliveira
18 de abril de 2023

Resumo. *Este relatório descreve duas soluções para satisfazer a primeira atividade proposta na disciplina de Algoritmos e Estruturas de Dados II. O objetivo do desafio é descobrir qual o macaquinho vencedor em um semelhante jogo de bingo entre macaquinhos. Além de apresentar as duas soluções para o desafio, é proposto uma análise da complexidade de cada algoritmo desenvolvido, e ao fim do documento, é apresentado os resultados adquiridos.*

1. Introdução

Na disciplina de Algoritmos e Estruturas de Dados II foi proposto um desafio para ser desenvolvido pelos alunos da turma. Este desafio consiste em simular um jogo semelhante ao "bingo", onde é realizado um número específico de rodadas, e a cada rodada é realizada uma alternância dos cocos entre todos os macaquinhos participantes do jogo. Ao fim da execução é apresentado o macaquinho ganhador, ou seja, aquele que possuir mais cocos no fim das rodadas.

No objetivo de realizar o desafio corretamente, precisaremos considerar alguns detalhes:

- Existem vários macaquinhos jogando.
- A quantidade de rodadas é necessariamente grande.
- Cada macaquinho preenche seus coquinhos com quantas pedrinhas quiser.
- Cada macaquinho sabe para quem enviar seus cocos com número de pedrinhas par, e com número de pedrinhas ímpar.
- As rodadas iniciam com um macaquinho distribuindo todos os seus coquinhos, isso respectivamente, ou seja, primeiro o macaquinho 0 distribui seus coquinhos, depois o macaquinho 1, em seguida o maquinho 2...
- Todas as informações estão armazenadas em documentos de texto (.txt) e para realizar a simulação corretamente é necessário que o programa leia o documento.

Os arquivos de texto que armazenam as informações sobre as rodadas do jogo possuem o seguinte padrão:

```
Fazer 100000 rodadas

Macaquinho 0 par -> 3 impar -> 1 :  4 :  9 5 2 6
```

No exemplo acima podemos observar como os arquivos de teste de caso são organizados. Lendo as informações podemos compreender alguns conceitos: Na primeira linha do arquivo é informado quantas rodadas este teste terá; a partir da segunda linha é apresentado as informações sobre os macaquinhos que participam do jogo. A cada

linha contendo um macaquinho: É informado a identificação do atual macaquinho (“`Macaquinho 0`”), a identificação do macaquinho que irá enviar os cocos caso o número de pedrinhas seja par (“`par -> 3`”), a identificação do macaquinho que irá enviar os cocos caso o número de pedrinhas seja ímpar (“`ímpar -> 1`”), a quantidade de cocos que o macaquinho atual possui (“`: 4 :`”), e a quantidade de pedrinhas que há em cada um desses cocos (“`9 5 2 6`”). Podem existir várias linhas contendo macacos.

Para solucionar o desafio, foram desenvolvidos dois algoritmos. O primeiro algoritmo (tópico 2) satisfaz o problema: este encontra e apresenta o macaquinho vencedor, entretanto sua complexidade é maior e seu tempo de execução consequentemente é maior. Já no segundo algoritmo (tópico 3), durante seu desenvolvimento, o objetivo era satisfazer o desafio e conjuntamente conseguir maior eficiência, menor tempo de execução e menor complexidade do algoritmo. Os resultados em relação ao macaquinho vencedor foi o mesmo para ambos os algoritmos, entretanto a diferença entre os tempos de execução foi consideravelmente melhor, esses resultados são apresentados no tópico 4.

2. Algoritmo número 1

Nesta solução foram desenvolvidos duas classes: a classe “Macaquinho” e a classe “JogoMacaquinhos”.

2.1. Classe “Macaquinho”

A classe “Macaquinho” foi criada no objetivo de criar vários macaquinhos distintos e futuramente poder acessá-los e modificá-los. Esta classe possuí cinco atributos, sendo dois destes do tipo ArrayList.

- `int id`: armazena a identificação do macaquinho atual.
- `int idPar`: armazena a identificação do macaquinho caso par.
- `int idImpar`: armazena a identificação do macaquinho caso ímpar.
- `List<Integer> cocosPares`: armazena uma lista de cocos com pedrinhas de número par.
- `List<Integer> cocosImpares`: armazena uma lista de cocos com pedrinhas de número ímpar.

O método construtor desta classe irá receber por paramêtro a linha do arquivo contendo as informaçõs sobre o macaquinho, e assim são inicializadas todas os atributos citados acima. Para as listas de cocos, antes de adicionar algum coco, é realizado a verificação se o número de pedrinhas dentro desse coco é par ou ímpar, e assim este coco é adicionado a sua respectiva lista. Abaixo é possível verificar o trecho de código que faz essa verificação:

```
for(int i = 11; i < lines.length; i++){
    int valor = Integer.parseInt(lines[i]);
    if(valor % 2 == 0){
        macaquinhosPares.add(valor);
    }else{
        macaquinhosImpares.add(valor);
    }
}
```

Listing 1. Verificação se caso par ou ímpar em Java

Esta classe possui apenas mais dois métodos:

- `int getId()`: retorna a identificação do macaquinho atual.
- `int getCocosTotal()`: retorna a quantidade de cocos que o macaquinho atual possui (é realizada a soma do tamanho de ambas as listas do macaquinho).

2.2. Classe "JogoMacaquinhos"

O objetivo dessa classe é realizar a leitura do arquivo, a manipulação do jogo, e possuir o método *"main"* (principal) onde o programa é de fato executado. A classe possui dois atributos:

- `int numRoodadas`: armazena o número de rodadas do teste de caso atual; esse atributo é inicializado no momento da leitura do arquivo que identifica na primeira linha a quantidade de rodadas.
- `List<Macaquinho> macaquinhos`: armazena os macaquinhos criados a partir da leitura do arquivo.

Para a execução do jogo foi desenvolvido três métodos:

- `void ler(String file)`: realiza a leitura do arquivo de texto informado por paramêtro com auxílio da classe *"BufferedReader"*, como apresentado abaixo:

```
1 BufferedReader br = new BufferedReader(new FileReader(file)
     );
```

Listing 2. BufferedReader

A leitura da primeira linha do arquivo é realizada separadamente para poder ser realizada a atribuição ao atributo "numRodadas"; assim, a cada nova linha lida é realizada a criação de um novo macaquinho passando por paramêtro a linha récem lida. É adicionado o novo macaquinho na lista de macaquinhos. Abaixo é apresentado alguns trechos para melhor entendimento:

```
1 String[] partes;
2 partes = br.readLine().split(" ");
3 ...
4 Macaquinho macaquinho = new Macaquinho(partes);
5 macaquinhos.add(macaquinho);
```

Listing 3. Trechos do método ler() para melhor entendimento

- `void main(String [] args]`: inicializa a classe "JogoMacaquinhos" e chama pelos métodos "ler()", informando o teste de caso desejado, e "jogar()". Também realiza a contagem de tempo que o algoritmo leva para executar.
- `void jogar()`: este método é o responsável pela jogatina.

No método "jogar()", inicialmente é realizado a verificação se há algum macaquinho na lista de macaquinhos. Em seguida, o jogo funciona a partir de dois laços de repetição: no primeiro laço, é percorrido o número de rodadas do jogo, e no segundo laço é percorrido o tamanho da lista de macaquinhos (assim, acessamos todos os macaquinhos, um por vez). Toda vez que os laços repetirem, temos um macaquinho para aquela rodada. Com o macaquinho da rodada identificado (chamado por "mamaco"), utilizamos duas variáveis auxiliares para armazenarem as identificações dos macaquinhos que devemos mandar os cocos caso par (chamado pro "par") ou caso ímpar (chamado pro "ímpar"). Identificando todos os macaquinhos necessários para a rodada, identificamos o macaquinho par na lista de macaquinhos e atribuímos todos os cocos com pedrinhas

de número par do macaquinho atual (mamaco) ao macaquinho identificado como par, e o mesmo processo é realizado com o macaquinho identificado como ímpar. Após a distribuição dos cocos, o macaquinho atual (mamaco) tem seus cocos zerados. Abaixo é possível verificar o trecho de código explicado:

```
for(int i = 0; i < numRodadas; i++){
    for(int j = 0; j < macaquinhos.size(); j++){
        Macaquinho mamaco = macaquinhos.get(j);
        par = mamaco.pares;
        impar = mamaco.impares;

        macaquinhos.get(par).macaquinhosPares.addAll(mamaco.
           macaquinhosPares);
        macaquinhos.get(impar).macaquinhosImpares.addAll(mamaco.
           macaquinhosImpares);

        mamaco.macaquinhosPares.clear();
        mamaco.macaquinhosImpares.clear();
    }
}
```

Listing 4. Jogatina com distribuição de cocos com listas

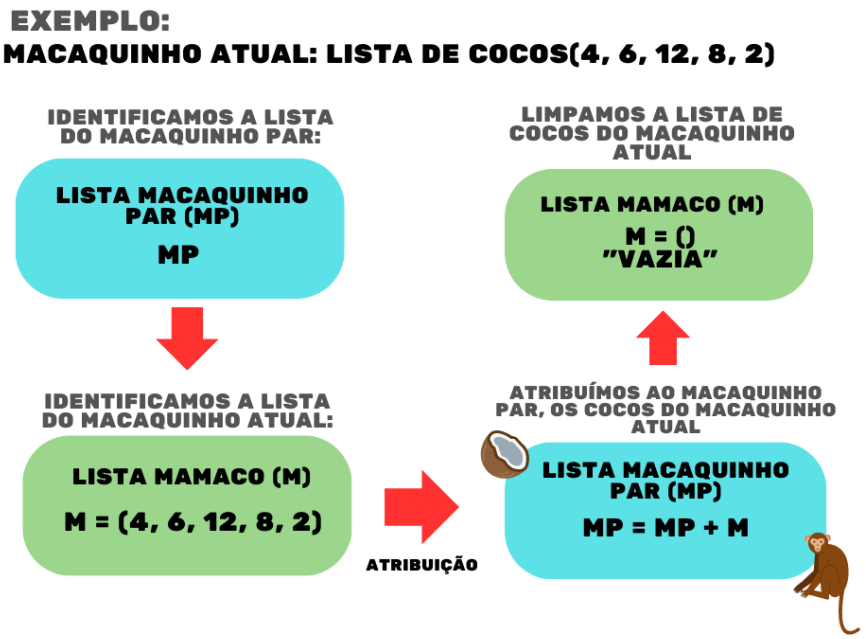

Figure 1. Exemplo de funcionamento da distribuição dos cocos entre os macaquinhos.

A imagem acima exemplifica a funcionalidade da distribuição dos cocos entre os macaquinhos com o sistema de listas. A lista dada de exemplo é uma lista de números pares, ou seja, cocos com pedrinhas de número pares. Ao observar, após a identificação de cada lista, tanto do macaquinho sendo consultado atualmente quanto do macaquinho que iremos enviar os cocos (no caso par), atribuímos os cocos do macaquinho atual ao

macaquinho que devemos enviar os cocos, e após essa distribuição, o macaquinho atual não possuí mais cocos, ou seja, limpamos sua lista de cocos.

Para a verificação de qual macaquinho é o vencedor, ao terminar a distribuição de todos os cocos, percorremos a lista de macaquinhos e verificamos o total de cocos que cada macaquinho possuí. Se o macaquinho X possuir mais cocos que o macaquinho Y, então ele é o novo campeão e a maior quantidade de cocos que temos é o tanto que ele possui, assim como apresentado no trecho de código 5.

```
int maxCocos = 0;
Macaquinho campeao = null;
for (Macaquinho macaquinhu : macaquinhos) {
    int cocos = macaquinhu.getCocosTotal();
    if (cocos != 0 && cocos > maxCocos) {
        maxCocos = cocos;
        campeao = macaquinhu;
    }
}
```

Listing 5. Verificando o vencedor algoritmo 1

3. Algoritmo número 2

O segundo algoritmo foi desenvolvido no objetivo de diminuir o tempo de execução do programa (apresentados no tópico 4, em "resultados"). Este algoritmo é uma versão do primeiro algoritmo (apresentado no tópico 2) porém com algumas modificações. Esta solução também foi desenvolvida a partir de duas classes: "Macaquinho" e "JogoMacaquinhos".

3.1. Classe "Macaquinho"

A classe "Macaquinho", após modificações, continua possuindo 5 atributos, porém possui mais métodos. O método construtor da classe agora recebe por parâmetro as informações já filtradas para as atribuições dos atributos, ou seja, no método "ler()" modificado (apresentado no subtópico 3.2) identificamos o macaquinho atual, o macaquinho par ou ímpar, e realizamos a verificação se o número de pedrinhas dos cocos é par ou ímpar, e só então criamos os macaquinhos com as informações já filtradas.

- `int id`: armazena a identificação do macaquinho atual.
- `int idPar`: armazena a identificação do macaquinho caso par.
- `int idImpar`: armazena a identificação do macaquinho caso ímpar.
- `int cocosPares`: armazena a contagem de cocos com pedrinhas pares.
- `int cocosImpares`: armazena a contagem de cocos com pedrinhas ímpares.

Agora, os atributos "cocosPares" e "cocosImpares" deixaram de ser listas, e são atributos que armazenam um número inteiro. Portando, na classe "Macaquinho" foi desenvolvidos alguns métodos diferentes do algoritmo 1 para a manipulação desses atributos:

- `int getPares()`: retorna o valor do atributo "cocosPares", no caso, a quantidade de cocos com pedrinhas de número par.
- `int getImpares()`: retorna o valor do atributo "cocosImpares", no caso, a quantidade de cocos com pedrinhas de número ímpar.

- `int getCocosTotal()`: retorna o total de cocos do macaquinho (é realizado a soma da quantidade de cocos com pedrinhas par com a quantidade de cocos com pedrinhas ímpar).
- `void setQtdPares()`: modifica a quantidade de cocos com pedrinhas de número par.
- `void setQtdImpares()`: modifica a quantidade de cocos com pedrinhas de número ímpar.
- `void clear()`: modifica os atributos "cocosPares" e "cocosImpares" para 0.

3.2. Classe "JogoMacaquinho"

Nesta classe, possuímos os mesmos métodos e atributos que o primeiro algoritmo, entretanto há algumas modificações nos métodos "ler()" e no método "jogar()".

No método "ler()", ao invés de apenas ser realizado a leitura da linha e ser criado um macaquinho passando por paramêtro essa linha, agora todos os filtros para adquirir as informações já são realizadas dentro desse método. Ou seja, a cada nova linha lida, é atribuído a variáveis auxiliares as informações de identificação dos macaquinhos e realizada a verificação se o número de pedrinhas de cada coco é par ou ímpar. Nesta versão, a cada número de pedrinhas par ou ímpar encontrado, "1" será somado a uma variável contador que armazena os cocos com número par ou os cocos com número ímpar de pedrinhas. No trecho de código abaixo apresenta a funcionalidade:

```
1  while (partes != null && partes.length > 0) {
2      String linha = br.readLine();
3      int id = Integer.parseInt(partes[1]);
4      int pares = Integer.parseInt(partes[4]);
5      int impares = Integer.parseInt(partes[7]);
6
7      int qtdPares = 0;
8      int qtdImpares = 0;
9      for(int i = 11; i < partes.length; i++){
10         int valor = Integer.parseInt(partes[i]);
11         if(valor % 2 == 0) {
12             qtdPares++;
13         }else{
14             qtdImpares++;
15         }
16     }
17     Macaquinho macaquinho = new Macaquinho(id, pares, impares,
           qtdPares, qtdImpares);
18     macaquinhos.add(macaquinho);
19 }
```

Listing 6. Leitura do arquivo e criação dos macaquinhos

Após realizada todas as atribuições, é criado um novo macaquinho com as informações adquiridas e esse macaquinho é adicionado à lista de macaquinhos.

No método "jogar()" o funcionamento da distribuição de cocos é semelhante ao algoritmo 1, porém também houveram alterações. Após verificar se há macaquinhos na lista de macaquinhos, iniciamos as rodadas executando o laço de repetição com o número de rodadas e o laço de repetição com o número de macaquinhos presentes na

lista. Para cada macaquinho da lista (j), identificamos o macaquinho para quem deve-se distribuir os cocos caso par (par) ou caso ímpar (ímpar). Com os macaquinhos identificados, é realizada a criação de duas variáveis auxiliares, do tipo inteiro, que armazenam a quantidade de cocos com pedrinhas de número par e cocos com pedrinhas de número ímpar. E assim, com o auxílio dos métodos "set" da classe "Macaquinho" apresentados anteriormente, modificamos o atributo "cocosPares" do macaquinho par e o atributo "cocosImpares" do macaquinho ímpar, consequentemente modificando o valor total dos cocos destes macaquinhos. Ao fim da distribuição, os cocos do macaquinho atual (j) são zerados. No trecho de código abaixo é apresentado o funcionamento da distribuição de cocos modificado:

```
for(int i = 0; i < numRodadas; i++){
    for(int j = 0; j < macaquinhos.size(); j++){
        par = macaquinhos.get(j).pares;
        impar = macaquinhos.get(j).impares;

        int auxP = macaquinhos.get(j).getPares();
        int auxI = macaquinhos.get(j).getImpares();

        macaquinhos.get(par).setQtdPares(auxP);
        macaquinhos.get(impar).setQtdImpares(auxI);

        macaquinhos.get(j).clear();
    }
}
```

Listing 7. Distribuição de cocos com variáveis tipo inteiro

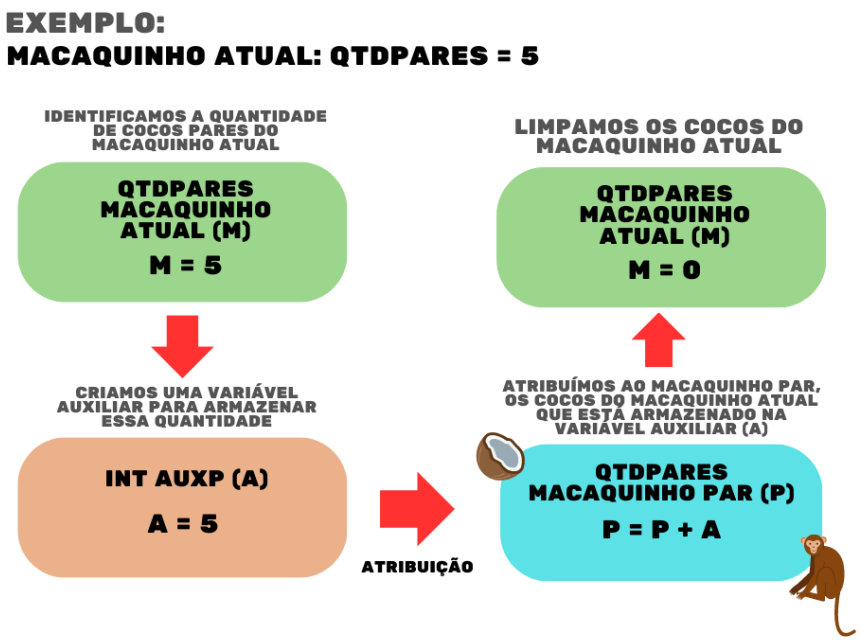

Figure 2. Exemplo de funcionamento da distribuição dos cocos entre os macaquinhos modificado.

A figura 2 busca exemplificar o funcionamento do sistema de distribuição de cocos entre os macaquinhos utilizando variáveis de tipo inteiro. Após identificar a quantidade de cocos que o macaquinho atualmente consultado, atribuímos esse valor em uma variável auxiliar. Esta variável auxiliar será atribuída a quantidade de cocos que o macaquinho par possuí, assim realizando a distribuição. Após a distribuição, o macaquinho atual tem seus cocos "zerados".

Agora, no trecho de código abaixo é possível verificar as modificação do sistema de identificar qual macaquinho vencedor:

```
int maxCocos = 0;
int idCampeao = 0;
for(int i = 0; i < macaquinhos.size(); i++){
    if((macaquinhos.get(i).getCocosTotal()) > maxCocos){
        maxCocos = macaquinhos.get(i).getCocosTotal();
        idCampeao = i;
    }
}
```

Listing 8. Verificando o vencedor algoritmo 2

Para cada macaquinho da lista será verificado se este é o vencedor, ou seja, se possui mais cocos que o anteriormente verificado. Caso sim, identificamos o macaquinho e a quantidade total de cocos que este possuí, este será o macaquinho vencedor e sua quantidade de cocos é a maior.

4. Resultados

Os resultados de qual o macaquinho vencedor foi o mesmo em ambos os algoritmos, entretanto as modificações entre eles são consideráveis ao comparar seu tempo de execução. No algoritmo 1, ao realizar a jogatina do bingo, temos uma manipulação em *arrays* onde é necessário redimensionar todo um novo *array* e calcular o seu tamanho para ter a informação de quantos cocos o macaquinho possuí. Já no algoritmo 2, foi apresentado uma nova maneira de chegar ao mesmo resultado, de maneira semelhante, porém menos custosa. O manuseio de listas custa mais do que o manuseio de variáveis do tipo inteiro, portanto, após iniciar os testes de caso com o algoritmo 1 notou-se que seria muito custoso realizar a jogatina do bingo dessa maneira. Então, foi observado que o número de pedrinhas não seria necessário para nada além de verificar para qual macaquinho seria enviado o coco durante a distribuição, assim, foi realizado a modificação de listas para variáveis de tipo inteiro que serviram como um "contador". Agora, cada macaquinho não armazena duas listas de cocos, e sim dois contadores de cocos, um contando quantos cocos com o número de pedrinhas pares e um contando quantos cocos com o número de pedrinhas ímpares.

Na tabela abaixo é apresentado os resultados, junto ao tempo de execução, do algortimo otimizado:

Teste	Macaquinho vencedor	Número de cocos	Tempo de execução (em ms)
0050	9	2332	60ms
0100	20	15461	89ms
0200	38	74413	155ms
0400	36	145232	321ms
0600	177	230276	458ms
0800	20	182575	685ms
0900	589	433295	840ms
1000	144	581995	1094ms

Tabela de resultados do algoritmo 2

Apenas para realizar uma comparação entre o tempo de execução do algoritmo 1 e o tempo de execução do algoritmo 2, a tabela abaixo apresenta os 5 primeiros testes realizados com o algoritmo 1. Ressaltando que o programa foi executado somente até o teste 0800 utilizando o algoritmo 1, isto devido ao seu longo tempo de execução.

Teste	Macaquinho vencedor	Número de cocos	Tempo de execução (em ms)
0050	9	2332	292ms
0100	20	15461	1718ms
0200	38	74413	8627ms
0400	36	145232	145232ms
0600	177	230276	237455ms
0800	20	182575	692449ms

Tabela de resultados do algoritmo 1

A partir das duas tabelas é possível verificar que o algoritmo 2, com as modificações, melhorou significativamente o tempo de execução. Com a análise dos algoritmos, considerando que o objetivo é unicamente encontrar qual o macaquinho campeão, aquele que contêem mais cocos, podemos concluir que descartar o armazenamento de algumas informações pode ser consideravelmente útil para a eficiência de um algoritmo. Considerar apenas as informações relevantes, e utilizar apenas quando necessário fez com que o mesmo desafio fosse executado com um eficiência ruim e com uma ótima eficiência.