



RELATÓRIO: Os robôs dançam quadrilha

INTEGRANTES:

Larissa Oliveira da Silva
Maria Eduarda Schöler

Porto Alegre, 2025

1. INTRODUÇÃO

Neste trabalho, exploramos um problema interessante envolvendo robôs que dançam em uma sequência determinada por uma receita de permutação. A ideia é descobrir quantas rodadas serão necessárias até que os robôs voltem à formação inicial, criando um ciclo.

Para resolver isso, simulamos os movimentos dos robôs e aplicamos conceitos matemáticos para identificar quando esse ciclo vai se repetir. A modelagem se concentra em entender as permutações e como detectar a repetição de maneira eficiente, permitindo resolver o problema de forma otimizada.

Inicialmente nos foi apresentado o seguinte enunciado:

Os robôs dançam quadrilha

Viajando pelo espaço você encontra um planeta habitado por robôs. O planeta é pacífico, agradável e os robôs têm uma vida tranquila. Por isso mesmo, os robôs têm vários feriados e festividades para comemorar.

Nos feriados os robôs dançam uma espécie de quadrilha de forma muito organizada: eles formam uma longa linha de robôs numerados de 0 a $n - 1$ e trocam de posição de acordo com uma “receita” de dança produzida pelo robô-mestre.

O plano é que eles troquem de posição usando a receita muitas e muitas vezes. (Na verdade, é para ser um número absurdo de vezes, pois os robôs têm muito tempo livre). A dança termina quando uma linha de robôs repetir alguma linha que já tenha sido dançada, porque neste caso os robôs sabem que entraram em um ciclo e se aborrecem. Ao mesmo tempo, isso introduz um elemento de surpresa na dança, pois nunca sabemos quando ela vai terminar. Se chegar a terminar.

Já que você está de visita, é gentilmente convidado pelo robô-mestre para avaliar algumas das receitas de dança propostas para descobrir quantas rodadas vão ser necessárias até que uma linha de dançarinos se repita. As receitas funcionam assim:

- *Elas iniciam com o número n de robôs previstos para esta receita;*
- *Em seguida vêm os números de 0 a $n - 1$ em alguma ordem. Por exemplo, uma receita para 7 robôs pode ser*

7

5 6 0 4 2 3 1

Esta linha de números significa que a cada rodada o robô que estiver na quinta posição vai parar em primeiro lugar na próxima rodada, o robô que estava na sexta vai para o segundo lugar, o robô que estava na posição 0 vai para o terceiro lugar, e assim por diante. A dança repete sempre os mesmos movimentos.

0 1 2 3 4 5 6

5 6 0 4 2 3 1

3 1 5 2 0 4 6

4 6 3 0 5 2 1

2 1 4 5 3 0 6

0 6 2 3 4 5 1

5 1 0 4 2 3 6

3 6 5 2 0 4 1

4 1 3 0 5 2 6

...

Então, se a receita acima for usada com uma linha de robôs iniciada com números de 0 a $n - 1$, os robôs começam seus movimentos assim:

Com estas informações sua missão é simples: você deve ajudar o robô-mestre, que tem várias receitas de dança diferentes e quer saber quantas rodadas cada uma delas vai durar até uma repetição.

2. MODELAGEM DO PROBLEMA

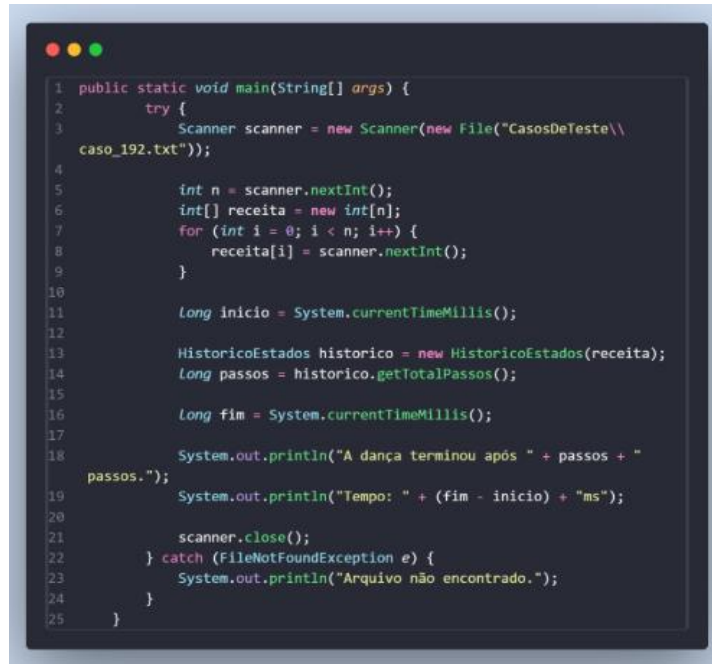
O problema foi modelado utilizando um vetor de inteiros que representa a sequência de posições dos robôs em uma coreografia de dança. Essa coreografia é definida por uma permutação, também chamada de “receita”, que indica para onde cada robô deve ir a cada rodada. O objetivo principal é determinar quantas rodadas são necessárias até que os robôs voltem à configuração original.

A modelagem se baseia nos seguintes elementos:

- Vetor de receita: Indica o novo posicionamento de cada robô. Por exemplo, $\text{receita}[3] = 1$ significa que o robô da posição 3 vai para a posição 1.
- Simulação das rodadas: A cada rodada, o vetor de posições é atualizado conforme a receita, e esse processo se repete até que a sequência volte à forma inicial.
- Detecção de ciclos: Cada robô segue um ciclo de movimentação, e o número total de passos será o mínimo múltiplo comum (MMC) entre todos esses ciclos.
- Cálculo de MMC e MDC: Utiliza-se o algoritmo de Euclides para calcular o máximo divisor comum (MDC), necessário para obter o MMC.
- Visitação e marcação de ciclos: Um vetor booleano indica se a posição de um robô já foi analisada, evitando repetições desnecessárias.

3. PROCESSO DE SOLUÇÃO E EXEMPLOS

Para resolver o problema, primeiro lemos os dados do arquivo de entrada — que trazem a quantidade de pessoas (ou robôs) e o “passo” que cada uma deve dar na dança. Esse vetor é interpretado como uma coreografia circular, em que cada pessoa aponta para a próxima.



```
1 public static void main(String[] args) {
2     try {
3         Scanner scanner = new Scanner(new File("CasosDeFeste\\
4         caso_192.txt"));
5
6         int n = scanner.nextInt();
7         int[] receita = new int[n];
8         for (int i = 0; i < n; i++) {
9             receita[i] = scanner.nextInt();
10        }
11
12        Long inicio = System.currentTimeMillis();
13
14        HistoricoEstados historico = new HistoricoEstados(receita);
15        Long passos = historico.getTotalPassos();
16
17        Long fim = System.currentTimeMillis();
18
19        System.out.println("A dança terminou após " + passos + "
20        passos.");
21        System.out.println("Tempo: " + (fim - inicio) + "ms");
22
23        scanner.close();
24    } catch (FileNotFoundException e) {
25        System.out.println("Arquivo não encontrado.");
26    }
27 }
```

Figura 1: classe Main

Com essas informações, criamos uma instância da classe `HistoricoEstados`, que analisa os caminhos que se formam a partir desses passos. A lógica usada é a de ciclos: seguimos o caminho indicado por cada pessoa até voltarmos à posição inicial, identificando assim ciclos individuais.

Depois que todos os ciclos são descobertos, o algoritmo calcula o mínimo múltiplo comum (MMC) entre os tamanhos desses ciclos — já que só quando todos se completam juntos é que a dança retorna ao estado inicial. Isso é feito no método `getTotalPassos()`, e o resultado é impresso na tela.

```

1 public class HistoricoEstados {
2     private int[] receita;
3     private boolean[] visitados;
4     private int quantidadePessoas;
5     private long totalPassos;
6
7     public HistoricoEstados(int[] receita) {
8         this.receita = receita;
9         this.quantidadePessoas = receita.length;
10        this.visitados = new boolean[quantidadePessoas];
11        this.totalPassos = 1;
12        contaEstados();
13    }
14
15    private void contaEstados() {
16        for (int pessoa = 0; pessoa < quantidadePessoas; pessoa++) {
17            if (!visitados[pessoa]) {
18                int tamanhoCiclo = 0;
19                int atual = pessoa;
20
21                while (!visitados[atual]) {
22                    visitados[atual] = true;
23                    atual = receita[atual];
24                    tamanhoCiclo++;
25                }
26
27                totalPassos = calcularMMC(totalPassos, tamanhoCiclo);
28            }
29        }
30    }
31
32    private long calcularMMC(long a, long b) {
33        return a * (b / calcularMDC(a, b));
34    }
35
36    private long calcularMDC(long a, long b) {
37        while (b != 0) {
38            long resto = a % b;
39            a = b;
40            b = resto;
41        }
42        return a;
43    }
44
45    public long getTotalPassos() {
46        return totalPassos;
47    }

```

Figura 2: método HistoricoEstados

O código busca ciclos em uma permutação onde cada pessoa aponta para outra. Cada ciclo se repete após certo número de passos (seu tamanho).

Para que todos os ciclos voltem à posição inicial ao mesmo tempo, é preciso sincronizá-los. O menor número de passos em que isso acontece é dado pelo MMC dos tamanhos dos ciclos.

Por isso o algoritmo acumula o MMC entre todos os ciclos encontrados. Por exemplo, se o vetor for [1, 0, 3, 2], temos dois ciclos de tamanho 2. O MMC entre eles é 2, ou seja, a dança repete a formação original após 2 passos.

4. CASOS DE TESTE

Por meio dos arquivos de texto fornecidos pelo professor, via Moodle, obtivemos os seguintes resultados na execução da nossa aplicação:

Caso de Teste	Passos até a repetição
Caso 12	A dança terminou após 42
Caso 22	A dança terminou após 182
Caso 52	A dança terminou após 38.038
Caso 72	A dança terminou após 1.939.938
Caso 102	A dança terminou após 22.700.678
Caso 132	A dança terminou após 1.744.004.262
Caso 152	A dança terminou após 13.370.699.342
Caso 172	A dança terminou após 283.551.037.770
Caso 182	A dança terminou após 432.788.426.070
Caso 192	A dança terminou após 1.484.147.626.962

```
PS C:\Users\dudas\OneDrive\Area de Trabalho\PUCRS\ALESTII\T1-ALESTII-Robots> c::;
owCodeDetailsInExceptionMessages' '-cp' 'C:\Users\dudas\AppData\Roaming\Code\User\
A dança terminou após 42 passos.
Tempo: 0ms
A dança terminou após 182 passos.
Tempo: 0ms
A dança terminou após 38038 passos.
Tempo: 0ms
A dança terminou após 1939938 passos.
Tempo: 0ms
A dança terminou após 22700678 passos.
Tempo: 0ms
A dança terminou após 1744004262 passos.
Tempo: 0ms
A dança terminou após 13370699342 passos.
Tempo: 0ms
A dança terminou após 283551037770 passos.
Tempo: 0ms
A dança terminou após 432788426070 passos.
Tempo: 0ms
A dança terminou após 1484147626962 passos.
Tempo: 0ms
PS C:\Users\dudas\OneDrive\Área de Trabalho\PUCRS\ALESTII\T1-ALESTII-Robots> █
```

Figura 3: Resultados dos casos de teste

5. CONCLUSÃO

A implementação da solução baseada na decomposição da movimentação em ciclos e uso do mínimo múltiplo comum mostrou-se extremamente eficiente. Mesmo com grandes quantidades de robôs e passos, o tempo de execução se manteve próximo de 0 ms, o que demonstra que a lógica de detecção de ciclos e o uso do algoritmo de Euclides foram apropriados.

Além disso, a estrutura do código permite fácil manutenção e extensão. A separação das responsabilidades em classes distintas (main e HistoricoEstados) facilita a legibilidade e reutilização.