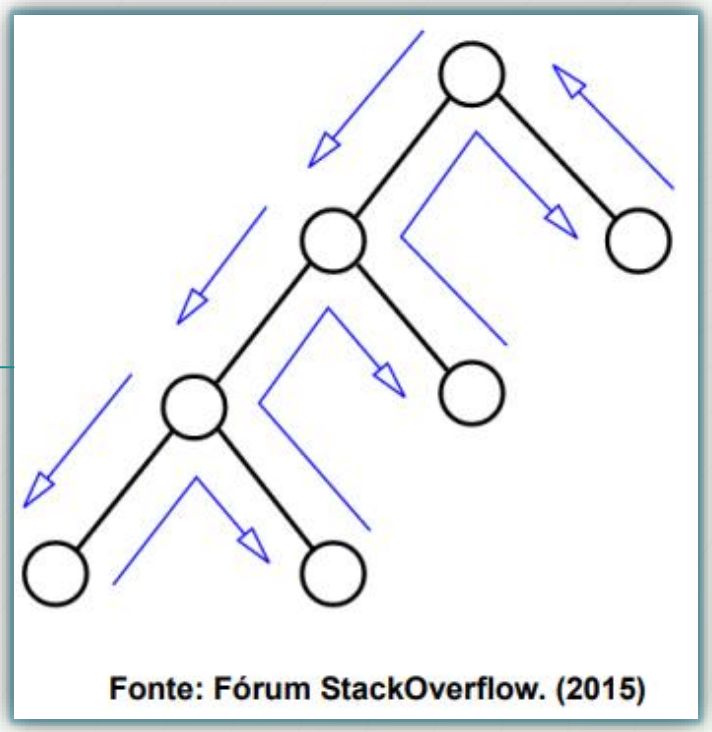
BACKTRACKING - TENTATIVA E ERRO

DAVI VENTURA
EDMILSON LINO
ERIC CASTRO

INTRODUÇÃO



Fonte: Fórum StackOverflow. (2015)

BackTracking é um algoritmo genérico que busca, por força bruta, soluções possíveis para problemas computacionais, em que múltiplas soluções podem ser eliminadas sem serem explicitamente examinadas.

O termo foi criado pelo matemático estadunidense **D. H. Lehmer** na década de **1950**.

Características do BackTracking

- Para cada chamada recursiva existem diversas opções que podem ser seguidas. Ex: Muitos vértices podem ser o próximo;
- Não seguem regra fixa de computação;
- Folhas dessa árvore são de dois tipos:

1. Representam uma possível solução para o problema.
2. Representam um ponto onde o algoritmo não pôde mais ir adiante.

Características do BackTracking

VANTAGEM:

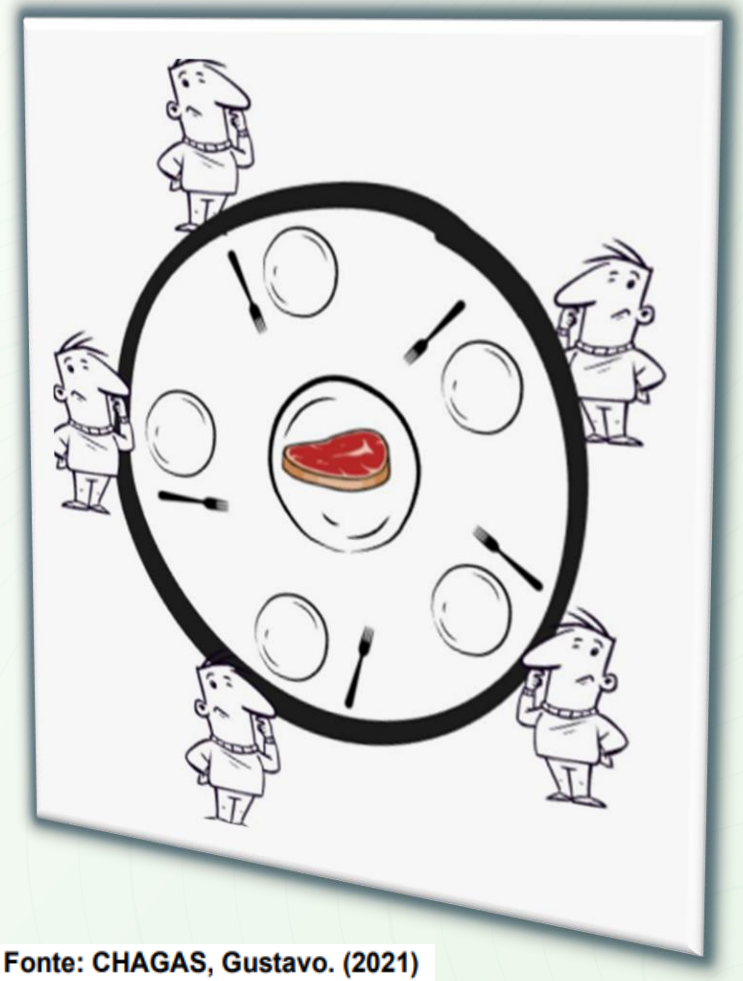
Fácil de implementar um problema que de outra forma seria muito mais complexo de se resolver. Porém essa técnica torna possível a resolução de muitos problemas NP-difícil com instâncias grandes em um tempo aceitável.

DESVANTAGENS:

A não ser que se programe restrições *(constraints)*, tenderão à explosão combinatória. Além disso, a quantidade de memória requerida para um programa de backtracking cresce exponencialmente com o tamanho do problema.

O PROBLEMA

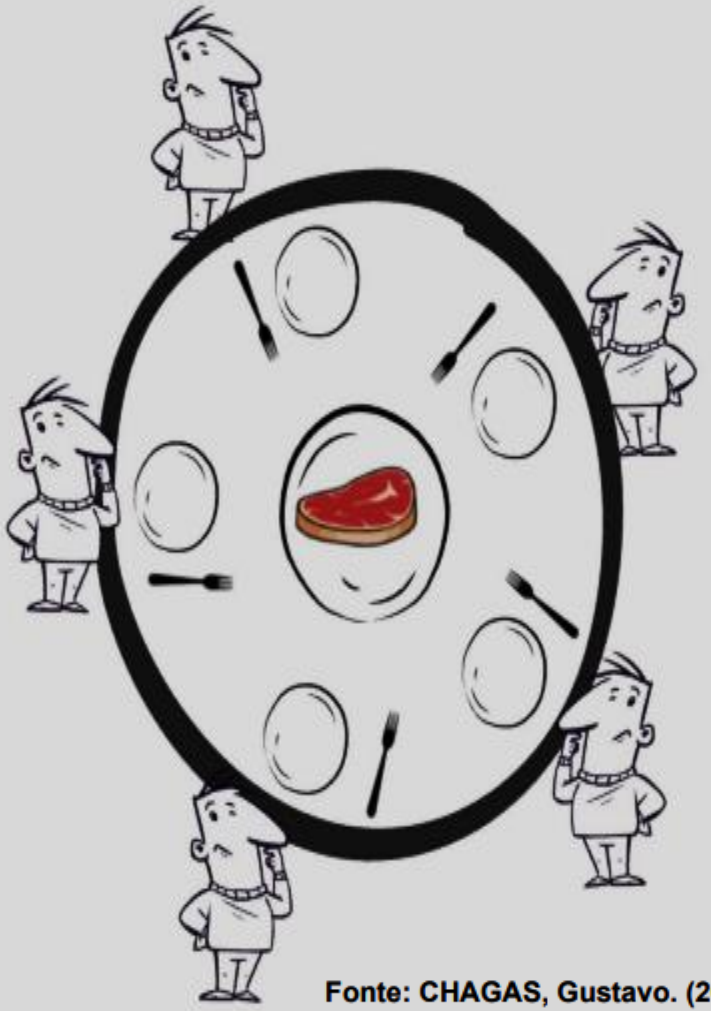
JANTAR DOS FILÓSOFOS



Fonte: CHAGAS, Gustavo. (2021)

Problema de sincronização, proposto pelo cientista da computação **Edsger W. Dijkstra** em **1965.**

➔ 5 Filósofos
➔ 5 Garfos
➔ Come quando estiver com 2 garfos
➔ Após comer devolve os garfo



Fonte: CHAGAS, Gustavo. (2021)

```
PS D:\Backup HD\Repositórios Git + VS\Jantar-dos-Filosofos>  d:; cd 'd:\Backup HD\Repositórios Git + VS\Jantar-dos-Filosofos'; & 'C:\Program Files\Java\jdk-16.0.2\bin\java.exe' '-agentlib
ort=dt_socket,server=n,suspend=y,address=localhost:49447' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\User\AppData\Roaming\Code\User\workspaceStorage\4540e194bf7477731661f83
dhat.java\jdt_ws\Jantar-dos-Filosofos_a6c5683\bin' 'JantarDosFilosofos'
```

O ALGORÍTMO E SUA COMPLEXIDADE

Os threads em Java são gerenciados pela JVM (Máquina Virtual Java), Isso quer dizer que o escalonador pode pará-las, dar espaço e tempo para outra thread ser executada.

Teremos um **“for”** na classe principal que iniciará um thread onde há um **“While”** na classe Filosofo.java, com isso podemos concluir que sua complexidade é **N^2.**

- Filósofo é um thread filha da classe Thread
- Semáforo representa o garfo
- Wait e Signal contra o deadlock

Jantar dos Filosofos > JantarDosFilosofos.java > ...

JantarDosFilosofos.java

```java
You, 2 minutes ago | 2 authors (WarLore and others)
import java.util.concurrent.Semaphore;

You, 2 minutes ago | 2 authors (WarLore and others)
public class JantarDosFilosofos {
    private static final int N = 5;

    Run | Debug
    public static void main(String[] args) {
        // Cada garfo será um semaforo
        Semaphore[] garfo = new Semaphore[N];

        for (int i = 0; i < N; i++) {
            garfo[i] = new Semaphore(permits: 1);
        }

        // Cria os filosofos e inicia cada um executando a sua thread
        Filosofo[] filosofos = new Filosofo[5];

        for (int i = 0; i < N; i++) {
            filosofos[i] = new Filosofo(i, garfo[i], garfo[(i + 1) % N]);
            new Thread(filosofos[i]).start();
        }
    }
}
```

filosofos.java

```java
import java.util.Random;
import java.util.concurrent.Semaphore;

WarLore, 1 hour ago | 2 authors (WarLore and others)
public class Filosofo implements Runnable {
    // Usado para definir por quanto tempo um filosofo pensa antes de comer e quanto
    // tempo ele fica comendo.
    private Random number = new Random();
    private int qtdComida = 10;

    // Utilizado para finalizar a execução do thread
    private boolean threadEstado;

    private final int id;
    private final Semaphore garfo_esquerdo;
    private final Semaphore garfo_direito;

    public Filosofo(int id, Semaphore garfo_esquerdo, Semaphore garfo_direito) {
        this.id = id;
        this.garfo_esquerdo = garfo_esquerdo;
        this.garfo_direito = garfo_direito;
        this.threadEstado = true;
    }

    // Ciclo infinito de cada filosofo, executado em threads separados
    @Override
    public void run() {
        try {
            while (threadEstado) {
                pensar();
                pegarGarfo_esquerdo();
                pegarGarfo_direito();
                comer();
                devolverGarfo();
            }
        } catch (InterruptedException e) {
            System.out.println("Filosofo " + id + "foi interrempido.\n");
        }
    }

    // Modelo de pensamento. Define um tempo aleatorio para o filosofo pensar
    private void pensar() throws InterruptedException {
        System.out.println("Filosofo " + id + " está pensando\n");
        // Tempo de execução
        Thread.sleep(number.nextInt(bound: 10));
    }

    // Analisa a quantidade de permissões para poder pegar o garfo esquerdo
    private void pegarGarfo_esquerdo() throws InterruptedException {
        if (garfo_esquerdo.availablePermits() == 0) {
            System.out.println("Filosofo " + id + " está ESPERANDO o garfo esquerdo.\n");
        }

        garfo_esquerdo.acquire();
        System.out.println("Filosofo " + id + " está SEGURANDO o garfo esquerdo\n");
    }

    // Analisa a quantidade de permissões para poder pegar o garfo direito
    private void pegarGarfo_direito() throws InterruptedException {
        if (garfo_direito.availablePermits() == 0) {
            System.out.println("Filosofo " + id + " está ESPERANDO o garfo direito.\n");
        }

        garfo_direito.acquire();
        System.out.println("Filosofo " + id + " está SEGURANDO o garfo direito\n");
    }

    // Define um tempo aleatorio para o filosofo comer
    private void comer() throws InterruptedException {
        if (this.qtdComida > 0) {
            System.out.println("Filoso " + id + " está COMENDO");
            this.qtdComida--;
            Thread.sleep(number.nextInt(bound: 10));
        } else {
            System.out.println(x: "A comida acabou\n");
            threadEstado = false;
            System.exit(status: 0);
```

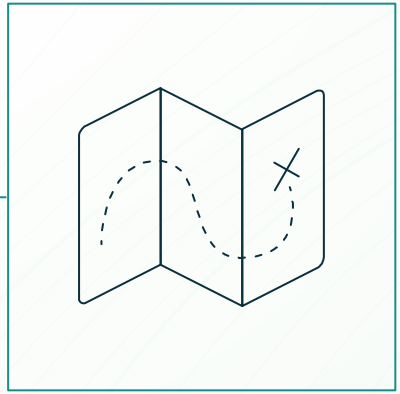
```
81      // Liberar os garfos para outro filosofo poder pegar
82      private void devolverGarfo() {
83          garfo_esquerdo.release();
84          garfo_direito.release();
85          System.out.println("Filosofo " + id + " SOLTOU os garfos\n");
86      }
87  }
```

Exemplo de Execução:

```
Filosofo 1 está pensando

Filosofo 4 está pensando

Filosofo 2 está pensando

Filosofo 3 está pensando

Filosofo 0 está pensando

Filosofo 3 está SEGURANDO o garfo esquerdo

Filosofo 3 está SEGURANDO o garfo direito

Filoso 3 está COMENDO
Filosofo 1 está SEGURANDO o garfo esquerdo

Filosofo 2 está SEGURANDO o garfo esquerdo

Filosofo 1 está ESPERANDO o garfo direito.

Filosofo 2 está ESPERANDO o garfo direito.

Filosofo 3 SOLTOU os garfos

Filosofo 3 está pensando

Filosofo 2 está SEGURANDO o garfo direito

Filoso 2 está COMENDO
Filosofo 0 está SEGURANDO o garfo esquerdo

Filosofo 0 está ESPERANDO o garfo direito.

Filosofo 4 está SEGURANDO o garfo esquerdo

Filosofo 4 está ESPERANDO o garfo direito.

Filosofo 3 está ESPERANDO o garfo esquerdo.

Filoso 1 está COMENDO
Filosofo 3 está SEGURANDO o garfo esquerdo

Filosofo 2 está pensando

Filosofo 3 está ESPERANDO o garfo direito.

Filosofo 2 está ESPERANDO o garfo esquerdo.

Filosofo 1 SOLTOU os garfos

Filosofo 2 está SEGURANDO o garfo esquerdo

Filosofo 2 está ESPERANDO o garfo direito.

Filosofo 0 está SEGURANDO o garfo direito

Filosofo 1 está pensando

Filoso 0 está COMENDO
Filosofo 1 está ESPERANDO o garfo esquerdo.

Filosofo 4 está SEGURANDO o garfo direito

Filoso 4 está COMENDO
Filosofo 0 SOLTOU os garfos

Filosofo 1 está SEGURANDO o garfo esquerdo

Filosofo 0 está pensando

Filosofo 1 está ESPERANDO o garfo direito.

Filosofo 4 SOLTOU os garfos

Filosofo 3 está SEGURANDO o garfo direito

Filosofo 4 está pensando

A comida acabou
```

CONCLUSÃO

Como o nome diz, retornar pelo caminho, a recursividade promove a elegância e a fácil implementação desse algoritmo, porque o vetor de novos candidatos é alocado com um procedimento recursivo.



Obrigado!

Alguma pergunta?