

# Relatório Técnico – Desafio 2: Cálculo de Distâncias Entre Cidades do Mundo

Grupo F - Lucas Grohe e Carlos.

## Descrição do Problema

O desafio proposto consiste no desenvolvimento de uma aplicação em Java capaz de calcular a **maior distância geográfica entre duas cidades do mundo**, considerando a curvatura da Terra. Para isso, utiliza-se a **Equação de Haverseno**, amplamente empregada em sistemas de geolocalização.

O programa deve ler dados de cidades a partir de um arquivo CSV, recebido via **redirecionamento de entrada padrão**, no formato:

continente;país;cidade;latitude;longitude;população

Além disso, a aplicação permite a utilização de **filtros opcionais**, definidos por argumentos de linha de comando, para restringir o conjunto de cidades analisadas por continente, país ou população.

Ao final da execução, o programa exibe quais são as duas cidades mais distantes entre si, bem como a distância calculada em quilômetros. Caso não existam cidades suficientes após a aplicação dos filtros, uma mensagem de erro apropriada é exibida.

---

## Regras e Funcionalidades Implementadas

A aplicação implementa corretamente todos os requisitos do Desafio 2, incluindo:

- Leitura de dados via `stdin` utilizando `BufferedReader`;
- Armazenamento das informações das cidades em listas paralelas;
- Aplicação de filtros opcionais:
  - `C <continente>` – filtra por continente;
  - `P <país>` – filtra por país;
  - `+ <população>` – considera apenas cidades com população maior ou igual ao valor informado;
  - `- <população>` – considera apenas cidades com população menor ou igual ao valor informado;
- Cálculo da distância entre todas as combinações possíveis de cidades filtradas;
- Determinação do maior valor de distância encontrado;

- Tratamento de erros para casos onde não há cidades suficientes para comparação.
- 

## Instruções de Execução

### 3.1 Compilação

```
javac Distancias.java
```

### 3.2 Execução sem filtros

```
java Distancias < cidades.csv
```

### 3.3 Execução com filtros

```
java Distancias C Europa < cidades.csv  
java Distancias P Brazil < cidades.csv  
java Distancias + 1000000 < cidades.csv  
java Distancias - 500000 < cidades.csv
```

**Observação:** Falando especificamente da nossa versão do código, **os filtros podem ser combinados**, e apenas as cidades que atenderem a todos os critérios serão consideradas.

## EXEMPLO DE EXECUÇÃO REAL:

```
lucas.grohe@serverdoin:~$ java Distancias < Dados.csv
```

```
Maior distância: Machachi (Ecuador) <-> Pekanbaru (Indonesia) = 20035.78 km
```

```
lucas.grohe@serverdoin:~$ java Distancias P Brazil < Dados.csv
```

```
Maior distância: Boa Vista (Brazil) <-> Pelotas (Brazil) = 3950.25 km
```

```
lucas.grohe@serverdoin:~$ java Distancias C África < Dados.csv
```

```
Maior distância: Gueznaia (Morocco) <-> Tôlanaro (Madagascar) = 8754.42 km
```

```
lucas.grohe@serverdoin:~$ java Distancias P Brazil + 1000000 < Dados.csv
```

```
Maior distância: Fortaleza (Brazil) <-> Porto Alegre (Brazil) = 3219.28 km
```

---

# Representação Algorítmica (Pseudocódigo)

## Variáveis

```
args: vetor de texto

checkingContinent, checkingCountry: lógico
checkingMoreThanPop, checkingLessThanPop: lógico

paramContinent, paramCountry: texto
paramMoreThanPop, paramLessThanPop: inteiro

continents, countries, cities: lista de texto
lats, lons: lista de real
pops: lista de inteiro

linha: texto
colunas: vetor de texto

maiorDistancia: real
indexCity1, indexCity2: inteiro

i, j: inteiro
dist: real
```

## Início

```
checkingContinent = falso
checkingCountry = falso
checkingMoreThanPop = falso
checkingLessThanPop = falso

// Leitura das flags
i = 0
Enquanto i < tamanho(args) faça
    Se i + 1 < tamanho(args) então
        Se args[i] = "C" então
            checkingContinent = verdadeiro
            paramContinent = args[i + 1]
        Senão se args[i] = "P" então
            checkingCountry = verdadeiro
            paramCountry = args[i + 1]
        Senão se args[i] = "+" então
            checkingMoreThanPop = verdadeiro
            paramMoreThanPop = inteiro(args[i + 1])
        Senão se args[i] = "-" então
            checkingLessThanPop = verdadeiro
            paramLessThanPop = inteiro(args[i + 1])
        Senão
            Escreva "[ERRO] Flag desconhecida"
        FimSe
        i = i + 2
    Senão
        Escreva "[ERRO] Flag sem parâmetro"
        i = i + 1
FimSe
```

FimEnquanto

// Inicialização das listas

continents = lista vazia

countries = lista vazia

cities = lista vazia

lats = lista vazia

lons = lista vazia

pops = lista vazia

// Leitura do CSV

Enquanto houver linha para ler faça

  Leia linha

  colunas = dividir(linha, ";")

  Se tamanho(colunas)  $\neq$  6 então

    continue

  FimSe

  continent = colunas[0]

  country = colunas[1]

  city = colunas[2]

  lat = real(colunas[3])

  lon = real(colunas[4])

  pop = inteiro(colunas[5])

  // Aplicação dos filtros

  Se checkingContinent e continent  $\neq$  paramContinent então continue

  Se checkingCountry e country  $\neq$  paramCountry então continue

  Se checkingMoreThanPop e pop < paramMoreThanPop então continue

  Se checkingLessThanPop e pop > paramLessThanPop então continue

  Adicione continent em continents

  Adicione country em countries

  Adicione city em cities

  Adicione lat em lats

  Adicione lon em lons

  Adicione pop em pops

FimEnquanto

// Cálculo da maior distância

maiorDistancia = 0

indexCity1 = -1

indexCity2 = -1

Para i de 0 até tamanho(cities) - 1 faça

  Para j de i + 1 até tamanho(cities) - 1 faça

    dist = distanciaViaHaverseno(

      lats[i], lons[i],

      lats[j], lons[j])

    Se dist > maiorDistancia então

      maiorDistancia = dist

      indexCity1 = i

      indexCity2 = j

```

        FimSe
    FimPara
FimPara

// Saída final
Se indexCity1 ≠ -1 então
    Escreva "Maior distância: ",
           cities[indexCity1], " (" , countries[indexCity1], ") <-> ",
           cities[indexCity2], " (" , countries[indexCity2], ") = ",
           maiorDistancia, " km"
Senão
    Escreva "[ERRO] Nenhuma cidade encontrada após aplicação dos filtros."
FimSe
Fim

Função distanciaViaHaverseno(lat1, lon1, lat2, lon2): real
Variáveis
    RAO_TERRESTRE: real
    lat1Rad, lon1Rad, lat2Rad, lon2Rad: real

Início
    RAO_TERRESTRE = 6378.13

    lat1Rad = radianos(lat1)
    lon1Rad = radianos(lon1)
    lat2Rad = radianos(lat2)
    lon2Rad = radianos(lon2)

    Retorne 2 * RAO_TERRESTRE *
        arcseno( raiz(
            sen²((lat2Rad - lat1Rad)/2) +
            cos(lat1Rad) * cos(lat2Rad) *
            sen²((lon2Rad - lon1Rad)/2)
        ))
FimFunção

```

---

## Decisões de Projeto e Comentários Importantes

- **Uso de listas paralelas:** Optou-se pelo uso de `ArrayList` separados para cada atributo (cidade, país, latitude, longitude etc.) por simplicidade e clareza, seguindo os princípios de programação estruturada abordados na disciplina.
- **Equação de Haverseno:** A fórmula foi implementada em um método separado (`distanciaViaHaverseno`), promovendo reutilização de código e melhor organização.
- **Tratamento de erros:** Foram implementadas validações para flags inválidas, parâmetros ausentes e casos onde o conjunto filtrado não contém cidades suficientes para formar um par.

---

## Conclusão

A solução demonstra o uso correto de estruturas de repetição, leitura de dados via redirecionamento, validação de entradas, aplicação de filtros e implementação de um algoritmo matemático real (Equação de Haverseno).

O projeto contribuiu para o aprofundamento do pensamento computacional, organização algorítmica e prática com a linguagem Java, consolidando os conteúdos abordados ao longo da disciplina.