

# FlyFood

**José Roberto Oliveira de Araújo Filho**

**Projetos Interdisciplinares para Sistemas de Informação II - 2022.1**

**BSI - UFRPE**

# Formulação do problema

- **Entrada:** Matriz  $n \times m$  com os pontos de origem/retorno e entrega.
- **Pontos:** O conjunto  $\mathbf{P} = \{p_1, p_2, \dots, p_n\}$  representa o conjunto com  $n$  pontos  $p$  presentes na matriz de entrada.
- **Coordenadas:** As coordenadas de cada ponto são dadas pela sua posição  $ij$  na matriz.
- **Distância:** A distância entre cada ponto é dada por  $(D_{xy})$ , onde  $D_{xy} = D(p_x, p_y)$ , que é a distância de  $p_x$  para  $p_y$ .
- **Fórmula:**  $\min \sum D_{xy}$

# Análise de algoritmos

- Usada para presumir os recursos necessários para que um determinado algoritmo seja executado, como memória e tempo, por exemplo.
- A eficiência de um algoritmo se dá pela quantidade de passos que ele executa para processar uma entrada de tamanho  $n$  (CORMEN, 2012).
- A análise assintótica nos diz qual a ordem de crescimento do tempo de execução de um algoritmo quando a entrada é muito grande (BRUNET, 2019).
- A notação  $O$  (Big O notation) define o limite superior do crescimento do tempo de execução de um algoritmo no pior caso.
- O algoritmo mais eficiente é aquele que tem o menor Big O em comparação com outros algoritmos que buscam resolver o mesmo problema.

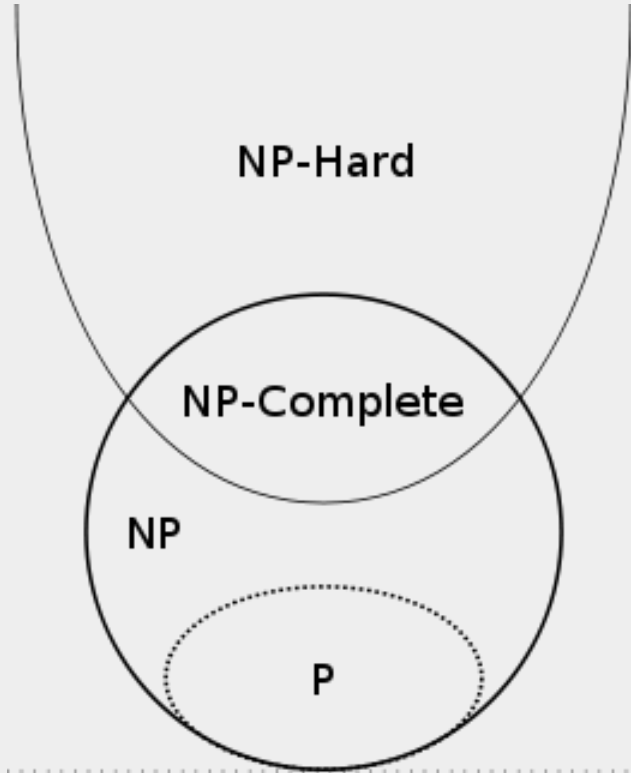
# Classes de problemas

- A **classe P** compreende os problemas que podem ser resolvidos em tempo polinomial.
- A **classe NP** consiste nos problemas que são verificáveis em tempo polinomial, ou seja, dada uma solução, conseguimos verificar em tempo polinomial se esta solução é válida ou não.
- O problema é **NP-Completo** se ele também pertencer à classe **NP**.
  - Se existir um algoritmo que resolva um problema NP-Completo em tempo polinomial, todos os problemas NP também serão resolvidos em tempo polinomial.

# Classes de problemas

- Um problema é **NP-Difícil** se ele for tão difícil quanto um problema NP-Completo.
- A diferença entre um problema NP-Completo e NP-Difícil é que, o problema é NP-Completo se ele fizer parte da classe NP e NP-Difícil; e para o problema ser NP-Difícil ele tem que ser tão difícil quanto um problema NP, mas não necessariamente pertencer à classe NP (ACERVO LIMA, c2022).

# Classes de problemas



# Metodologia

- **Passos seguidos:**

1. Ler a matriz de entrada.
2. Identificar quais são os pontos de entrega e suas posições na matriz.
3. Gerar todas as permutações possíveis com os pontos de entrega identificados.
4. Calcular a distância total de cada um desses caminhos e encontrar aquele de menor custo.

## Passo 1:

### Função 1: Ler matriz

função

função\_1 () : inteiro, inteiro, lista

var

quantidade\_linhas, quantidade\_colunas : inteiro

matriz\_lida : lista

inicio

```
1: abrir 'arquivo.txt' no modo leitura
2:   ler primeira linha
3:     quantidade_linhas ← primeiro elemento da linha lida
4:     quantidade_colunas ← terceiro elemento da linha lida
5:   fim-lerprimeiralinha
6:   matriz_lida ← listavazia
7:   para cada_linha em ler linhas restantes faça
8:     matriz_lida.append(lista(cada_linha.split()))
9:   fimpara
10:  para cada_elemento em matriz_lida faça
11:    se cada_elemento[-1] == '\n' então
12:      cada_elemento ← cada_elemento[1:-1]
13:    fimse
14:  fimpara
15:  retorne quantidade_linhas, quantidade_colunas, matriz_lida
16: fechar-aquivo
fimfunção
```



## Passo 2:

### Função 2: Identificar os pontos de entrega

#### função

função\_2 (linhas : inteiro, colunas : inteiro, matriz : lista) : lista,  
dicionário

#### var

pontos: lista

pontos\_e\_coordenadas : dicionário

#### inicio

1: **para** i **de** 0 **até** linhas **faça**

2:     **para** j **de** 0 **até** colunas **faça**

3:         **se** matriz[i][j] **é** letra **então**

4:             pontos.**append**(matriz[i][j])

5:             pontos\_e\_coordenadas[matriz[i][j]] = (i, j)

6:         **fimse**

7:     **fimpara**

8: **fimpara**

10: pontos.**remove**('R')

11: **retorne** pontos, pontos\_e\_coordenadas

#### fimfunção

### Passo 3:

#### Função 3: Gerar as permutações

função

função\_3 (pontos : lista) : lista

var

combinações, pontos\_não\_visitados : lista

inicio

1: **se** len(pontos) == 1 **então**

2:     **retorne** [pontos]

3: combinações ← listavazia

4: **para** i **de** 0 **até** len(pontos) **faça**

5:     pontos\_não\_visitados ← pontos[0:i] + pontos[i+1:]

6:     **para** cada\_ponto **em** função\_3 (pontos\_não\_visitados) **faça**

7:         combinações.append([pontos[i]] + ponto)

8:     **fimpara**

9: **fimpara**

10: **retorne** combinações

fimfunção

## Passo 4:

---

**Função 4:** Calcular distância entre dois pontos

---

função

função\_4 (ponto\_a : inteiro, ponto\_b : inteiro) : inteiro

var

distância : inteiro

início

1: distância = abs(ponto\_a[0] - ponto\_b[1]) + abs(ponto\_a[0] -  
ponto\_b[1])

2: **retorne** distância

fimfunção

## Passo 5 (1/2):

**Função 5:** Calcular distância total de cada caminho

função

função\_5 (caminhos : lista, coords : dicionário)

var

custo\_menor\_caminho : inteiro

menor\_caminho : lista

custo\_caminho\_atual : inteiro

inicio

1: custo\_menor\_caminho  $\leftarrow$  999

2: menor\_caminho  $\leftarrow$  listavazia

3: **para** cada\_caminho **em** caminhos **faça**

4:     custo\_caminho\_atual  $\leftarrow$  0

5:     inserir ponto de origem no início do caminho

6:     inserir ponto de retorno no final do caminho

7:     **para** i **de** 0 **até** len(caminho) - 1 **faça**

8:         coord\_ponto\_a  $\leftarrow$  coords[cada\_caminho[i]]

9:         coord\_ponto\_b  $\leftarrow$  coords[cada\_caminho[i + 1]]

10:         custo\_caminho\_atual      $\leftarrow$      função\_4(coord\_ponto\_a,  
coord\_ponto\_b)

11:     **fimpara**

## Passo 5 (2/2):

inicio

```
1: custo_menor_caminho ← 999
2: menor_caminho ← listavazia
3: para cada_caminho em caminhos faça
4:   custo_caminho_atual ← 0
5:   inserir ponto de origem no início do caminho
6:   inserir ponto de retorno no final do caminho
7:   para i de 0 até len(caminho) - 1 faça
8:     coord_ponto_a ← coords[cada_caminho[i]]
9:     coord_ponto_b ← coords[cada_caminho[i + 1]]
10:    custo_caminho_atual ← função_4(coord_ponto_a,
coord_ponto_b)
11:   fimpara
12:   se custo_caminho_atual < custo_menor_caminho então
13:     custo_menor_caminho ← custo_caminho_atual
14:     menor_caminho ← caminho
15:   fimse
16: fimpara
17: escreva(menor_caminho)
18: escreva(custo_menor_caminho)
fimfunção
```

## Função principal:

**Função main:** Chamar as demais funções

---

função

função\_main ()

var

linhas, colunas, matriz, pontos\_de\_entrega, rotas : lista  
coordenadas : dicionário

inicio

1: linhas, colunas, matriz  $\leftarrow$  função\_1 ()

2: pontos\_de\_entrega, coordenadas  $\leftarrow$  função\_2 (linhas, colunas,  
matriz)

3: rotas  $\leftarrow$  função\_3 (pontos\_de\_entrega)

4: função\_5 (rotas)

fimfunção

# Experimentos

**Matriz 1**

5	5				
0	0	0	0	D	
0	A	0	0	0	
0	0	0	C	0	
0	0	0	0	0	
R	0	B	0	0	

**Matriz 2**

5	5				
0	0	0	0	D	
A	0	0	F	0	
0	0	0	0	C	
0	E	0	0	0	
R	0	B	0	0	

**Matriz 3**

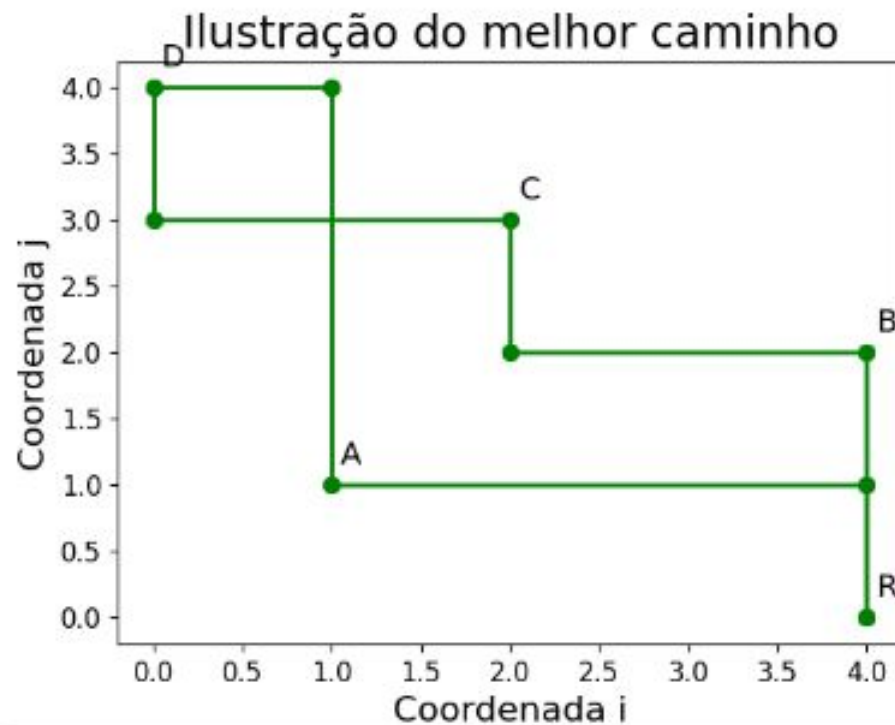
6	6				
A	0	0	0	0	0
0	0	0	0	B	0
0	C	0	D	0	0
0	0	0	0	G	0
0	E	0	0	0	H
R	0	0	F	0	0

**Matriz 4**

6	6				
A	0	0	0	0	I
0	0	J	0	B	0
C	0	0	D	0	0
0	0	0	0	G	0
0	E	0	0	0	H
R	0	0	F	0	0

# Menor caminho da matriz 1

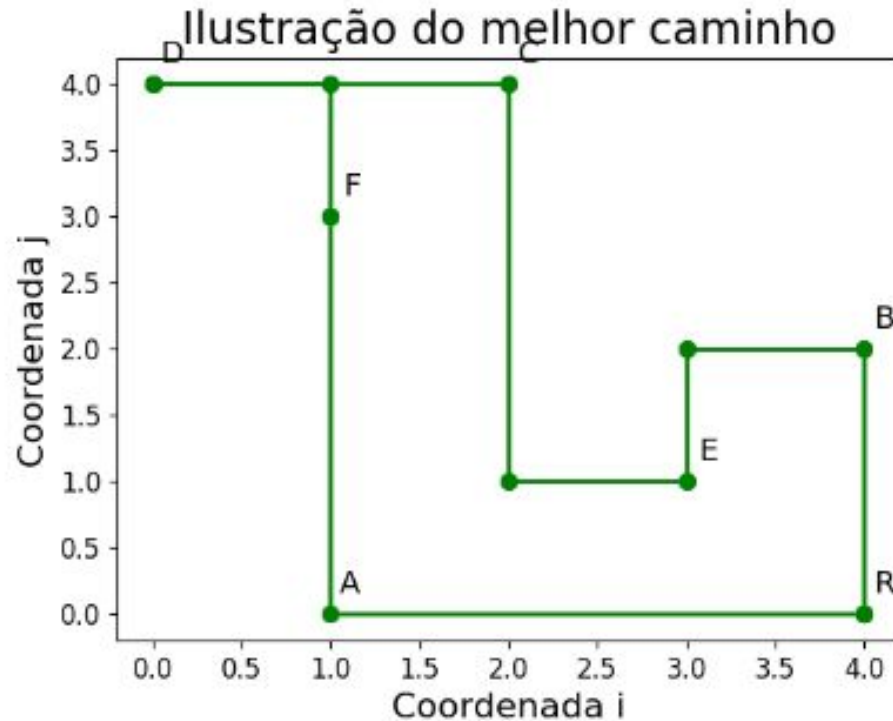
Pontos de entrega: 4; Rota: 'R A D C B R'; Custo: 16





# Menor caminho da matriz 2

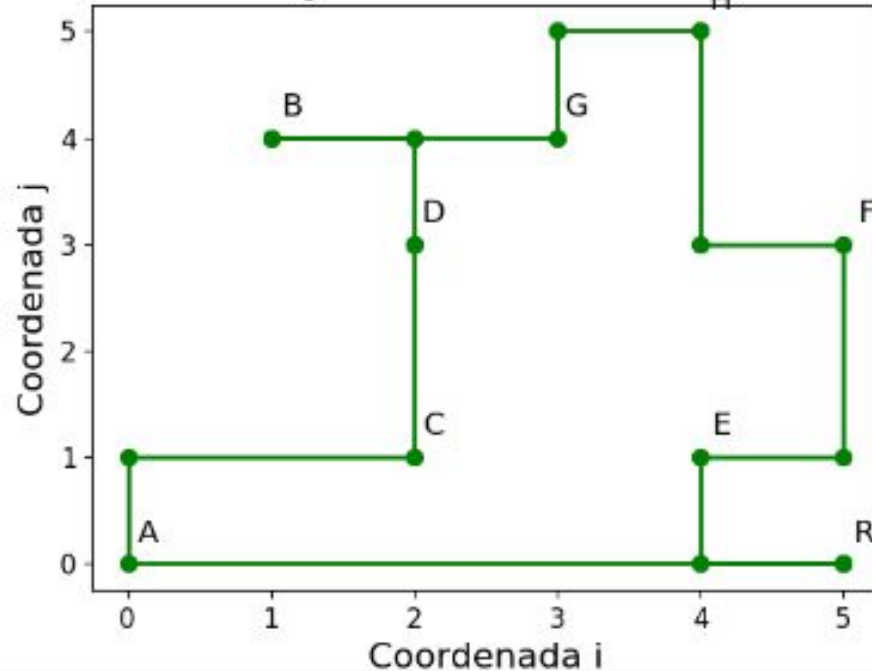
Pontos de entrega: 6; Rota: 'R A F D C E B R'; Custo: 18



# Menor caminho da matriz 3

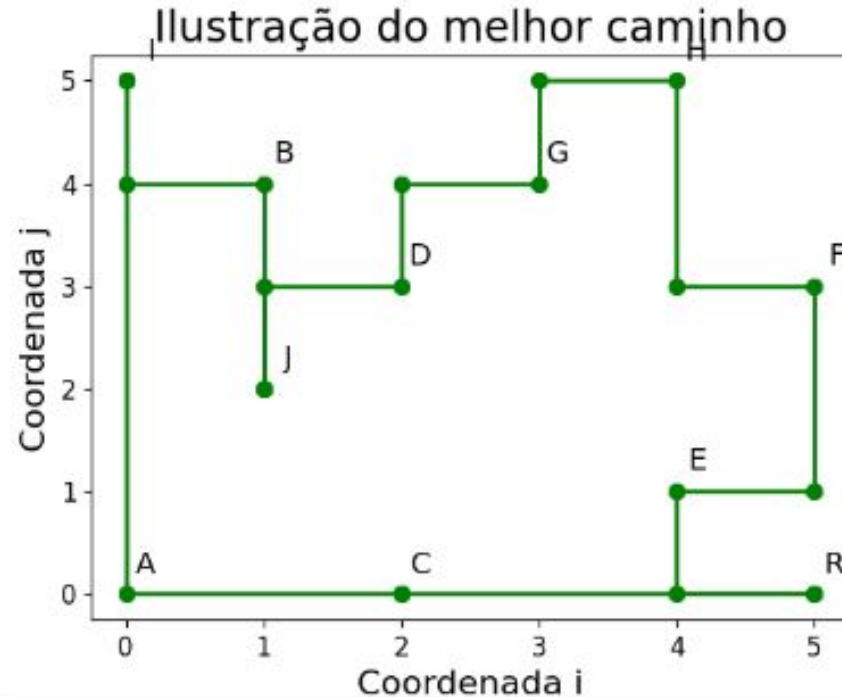
Pontos de entrega: 8; Rota: 'R A C D B G H F E R'; Custo: 24

Ilustração do melhor caminho



# Menor caminho da matriz 4

Pontos de entrega: 10; Rota: 'R C A I B J D G H F E R'; Custo: 28



# Resultados

Ordem da matriz	Pontos de entrega	Custo do caminho	Tempo (segundos)
5x5	4	16 dronômetros	0.008 segundos
5x5	6	18 dronômetros	0.01 segundos
6x6	8	24 dronômetros	0.5 segundos
6x6	10	28 dronômetros	53.24 segundos

# Resultados



# Conclusão

- O algoritmo consegue encontrar o menor caminho;
- Como a complexidade do programa é fatorial, ele se mostrou eficiente apenas para um pequeno número de pontos de entrega;
- O método de força-bruta não é o mais adequado para esse tipo de problema.