FlyFood

José Roberto Oliveira de Araújo Filho
Projetos Interdisciplinares para Sistemas de Informação II - 2022.1
BSI - UFRPE

Formulação do problema

- **Entrada:** Matriz *nxm* com os pontos de origem/retorno e entrega.
- **Pontos:** O conjunto **P** = {p1, p2, …, pn} 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 (Dxy), onde Dxy = *D(px,py), que é a distância de px para py.*
- **Fórmula:** $\min \sum Dxy$

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.

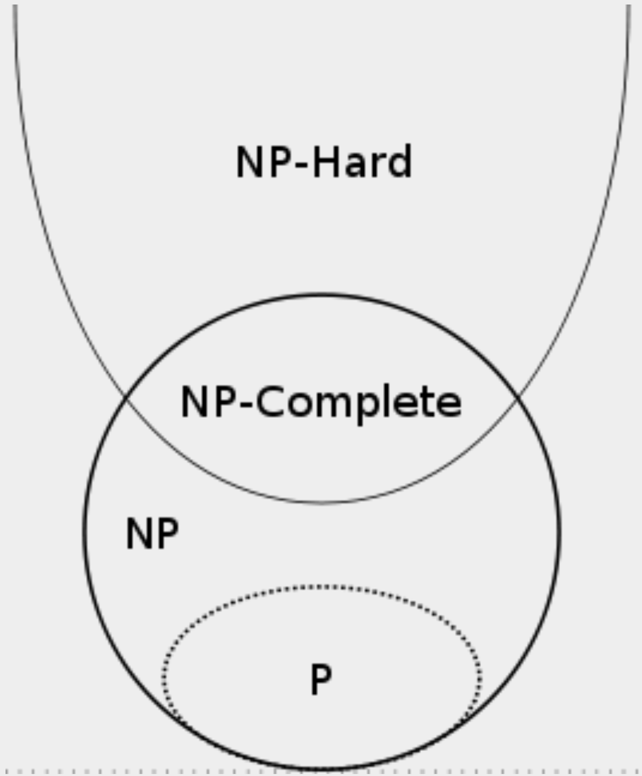
Classes de problemas

- 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
abrir 'arquivo.txt' no modo leitura
        ler primeira linha
                quantidade_linhas ← primeiro elemento da linha lida
                quantidade_colunas ← terceiro elemento da linha lida
        fim-lerprimeiralinha
        matriz_lida ← listavazia
        para cada_linha em ler linhas restantes faça
                matriz_lida.append(lista(cada_linha.split()))
        fimpara
        para cada_elemento em matriz_lida faça
                se cada_elemento[-1] == '\n' então
                        cada_elemento ← cada_elemento[1:-1]
                fimse
        fimpara
        retorne quantidade_linhas, quantidade_colunas, matriz_lida
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
para i de 0 até linhas faça
    para j de 0 até colunas faça
        se matriz[i][j] é letra então
            pontos.append(matriz[i][j])
            pontos_e_coordenadas[matriz[i][j]] = (i, j)
        fimse
    fimpara
fimpara
pontos.remover('R')
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
se len(pontos) == 1 então
    retorne [pontos]
combinações ← listavazia
para i de 0 até len(pontos) faça
    pontos_não_visitados ← pontos[0:i] + pontos[i+1:]
    para cada_ponto em função_3 (pontos_não_visitados) faça
            combinações.append([pontos[i]] + ponto)
    fimpara
fimpara
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
inicio
  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

<u>função</u>

função_5 (caminhos : lista, coords : dicionário)

<u>var</u>

custo_menor_caminho : inteiro

menor_caminho : lista

custo_caminho_atual : inteiro

Passo 5 (2/2):

```
inicio
custo_menor_caminho ← 999
menor_caminho ← listavazia
para cada_caminho em caminhos faça
    custo_caminho_atual ← 0
    inserir ponto de origem no início do caminho
    inserir ponto de retorno no final do caminho
    para i de 0 até len(caminho) - 1 faça
        coord_ponto_a ← coords[cada_caminho[i]]
        coord_ponto_b ← coords[cada_caminho[i + 1]]
        custo_caminho_atual ← função_4(coord_ponto_a, coord_ponto_b)
    fimpara
    se custo_caminho_atual < custo_menor_caminho então
        custor_menor_caminho ← custo_caminho_atual
        menor_caminho ← caminho
    fimse
fimpara
escreva(menor_caminho)
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 ← função_1 ()
  2: pontos_de_entrega, coordenadas ← função_2 (linhas, colunas,
matriz)
  3: rotas ← 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

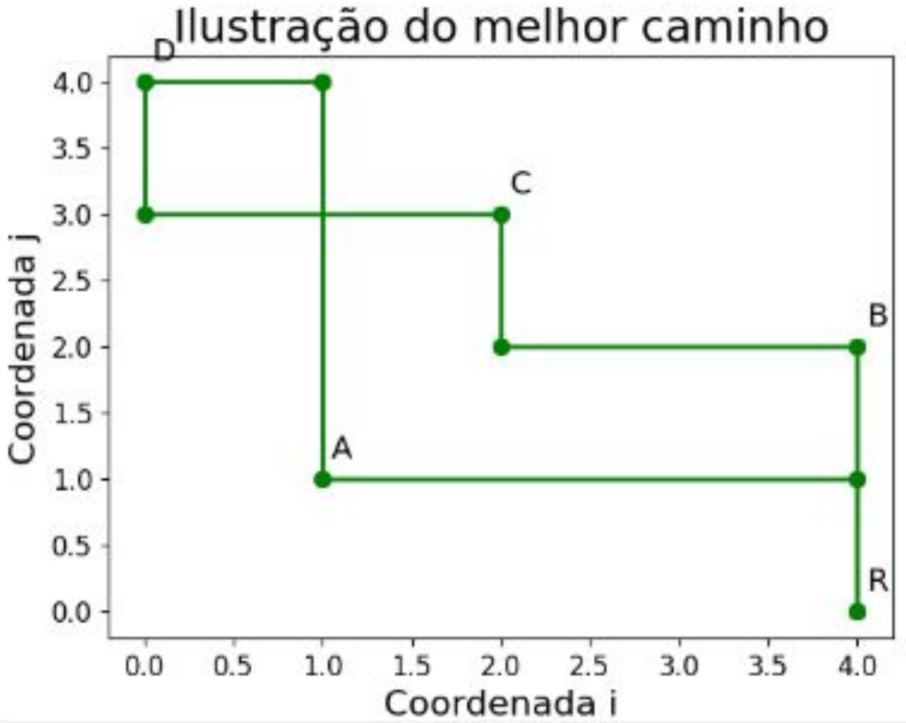
```
6 6
A 0 0 0 0 0
0 0 0 0 B 0
0 C 0 D 0 0
0 0 0 0 G 0
0 E 0 0 0 H
R 0 0 F 0 0
```

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
```

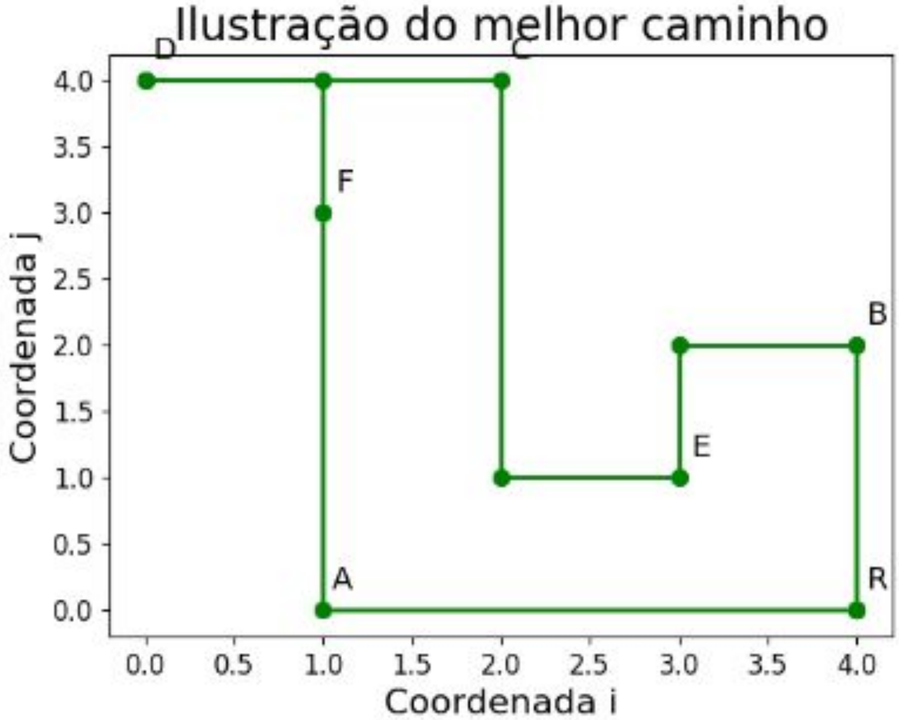
Menor caminho da matriz 1

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



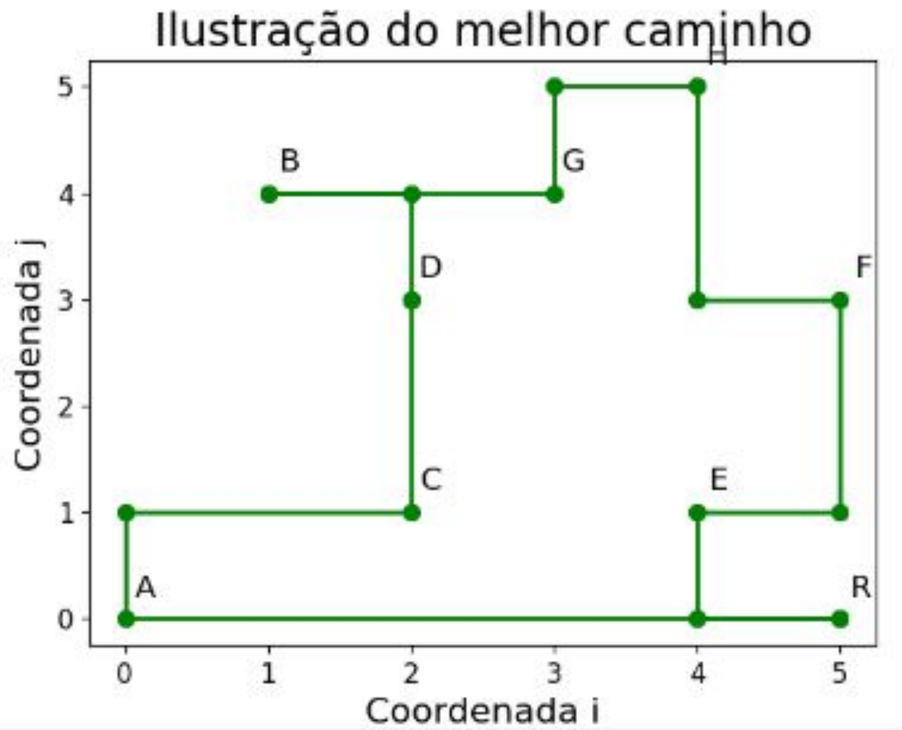
Menor caminho da matriz 2

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



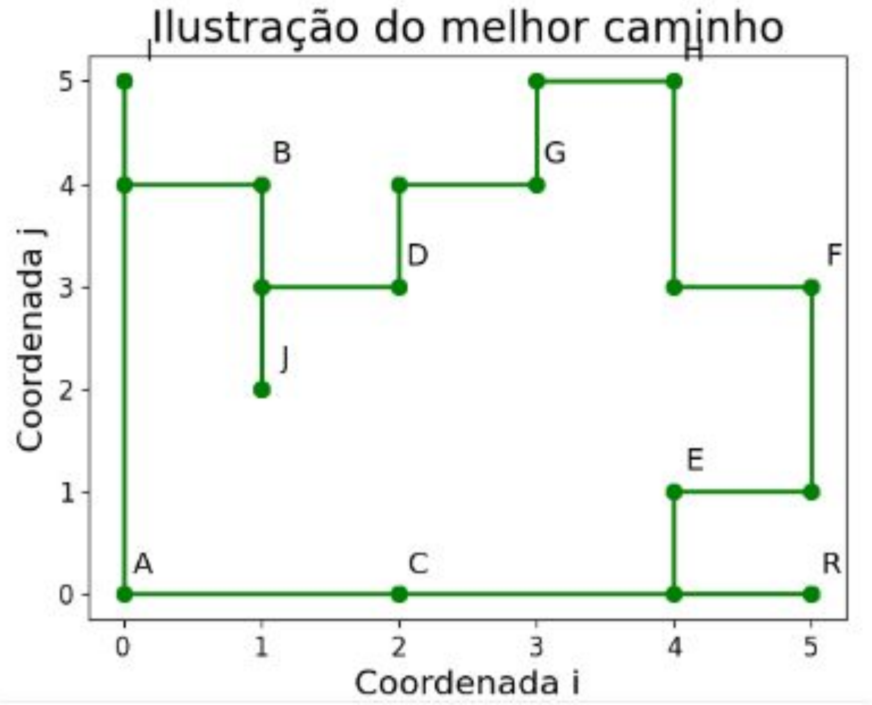
Menor caminho da matriz 3

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



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


Tempo de execução
Tempo (segundos)
80
60
40
20
0
0,008
0,01
0,5
53,24
4
6
8
10
Quantidade de pontos de entrega

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.