# EP 1: Braço Robótico

**Kauê Sales, Mayara Rosa e Pedro Paulo**

# Modelagem de estados e ações

## Estados

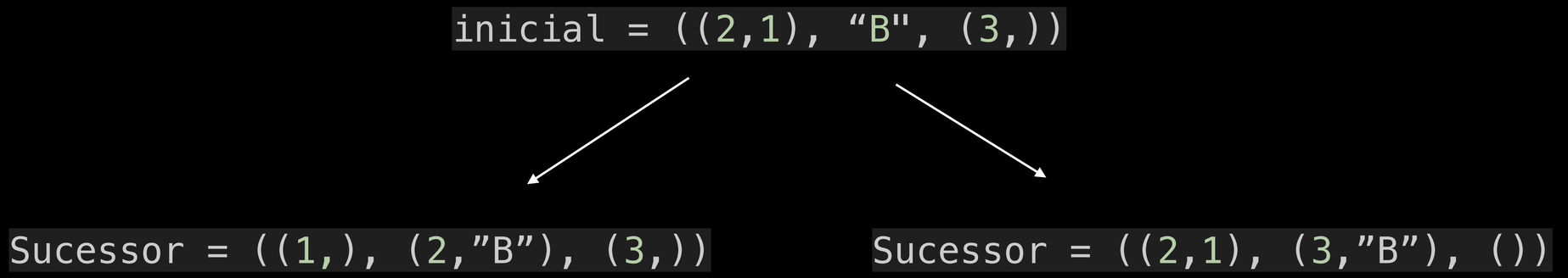
```
inicial = ((1,2), “B", (3,), ())
```

- Cada pilha é uma tupla que representa as caixas e os seus pesos em kg
- Exemplo:  estado inicial posição 0 com uma caixa de 1kg e uma 2kg em cima, na pilha 2 o braço, na pilha 3 uma caixa com 3kg e uma pilha vazia
- Estado objetivo: mesmo formato, porém com a disposição final desejada, exemplo:

```
Final = ((3,2,1), “B", (), ())
```

# Função sucessora e as ações possiveis

- gerar_sucessores é a função responsável por gerar novos estados, movendo as caixas entre as pilhas
- Exemplo: estado inicial -> mover topo da pilha 1 para a pilha 2 ou mover topo da pilha 3 para a pilha 2
- A função sucessora retorna o **novo estado**, a **ação** e o **custo** (se houver).



# Checagem de objetivo

- A função ler_configuracao é responsável tanto por ler o arquivo de entrada e construir o estado inicial, quanto o estado objetivo do problema
- Estado objetivo é construído com base nas caixas disponíveis do estado inicial, elas são ordenadas e distribuídas entre as posições.
- O problema é resolvido quando o estado atual for igual ao estado objetivo gerado automaticamente.

```
inicial = ((50,40), (30,), “B”, (20,10), (15, 5), (25,))
```

```
objetivo = ((50,40,30,25), (20, 15, 10, 5,), “B”, (), (), ())
```

# Função heurística

- A função heurística estima o custo restante para atingir o resultado final
- Se a distância for de uma casa, o custo é 1; para distâncias maiores, o fator é 0,75 por casa
- Adiciona-se um custo extra baseado no peso da caixa (cada 10kg acrescenta 1 de energia)
- Exemplo: Se uma caixa de 20kg estiver na posição errada e a posição correta estiver duas colunas à direita:

**Custo de movimentação:** 2 colunas -> 2 * 0.75 = 1.5

**Custo pelo peso:** 20kg / 10 = 2kg

**Custo total:** 1.5 + 2 = 3.5

# Teoria: Dijkstra

- A partir do estado inicial, para cada ação Dijkstra calcula o custo acumulado sobre os movimentos a serem realizados e o peso das caixas.
- O algoritmo explora todos os estados possíveis em ordem de menor custo acumulado.
- Para cada estado explorado, há a atualização dos custos. Para cada caminho explorado, devemos verificar se o custo dele é menor do que qualquer outro caminho explorado até agora.
- Dijkstra **garante** a solução ótima, mas passa por todos os estados, podendo ter uma performance pior do que a do A*.

# Exemplo: Djikstra

## Estado Inicial

- 50 40 30 25
- 20 15 10 5
- B
- 0 0 0 0
- 0 0 0 0
- 0 0 0 0

## Passos

- E 1 P 30 2
- E 1 S 30 5
- D 3 P 10 6
- E 3 S 10 15
- D 3 P 20 6
- E 2 S 20 10
- D 3 P 5 6
- E 1 S 5 5
- D 1 P 15 2
- E 3 S 15 15
- E 1 P 10 2
- D 1 S 10 5
- D 2 P 5 4
- E 2 S 5 10
- D 4 P 25 8
- E 5 S 25 25

**Estados visitados: 24968**

# Teoria: Ganancioso

- A partir do estado inicial, o algoritmo sempre escolhe a ação com o menor custo imediato, sem levar em consideração o efeito a longo prazo.
- O algoritmo escolhe o estado sucessor e repete até atingir o estado objetivo ou ficar preso num estado não ótimo.
- Esse algoritmo pode ser rápido, mas não garante a melhor solução possível, podendo ficar preso em uma solução subótima ou nem atingir o objetivo.

# Exemplo: Greedy

## Estado Inicial

- 50 40 30 25
- 20 15 10 5
- B
- 0 0 0 0
- 0 0 0 0
- 0 0 0 0

## Passos

- D 3 P 25 6
- E 5 S 25 25
- D 4 P 5 8
- E 3 S 5 15
- D 3 P 15 6
- E 4 S 15 20
- D 1 P 5 2
- D 2 S 5 10
- E 2 P 30 4
- D 3 S 30 15
- E 4 P 15 8

- D 1 S 15 5
- D 2 P 5 4
- E 3 S 5 15
- D 3 P 10 6
- E 2 S 10 10
- E 1 P 5 2
- D 1 S 5 5
- E 1 P 25 2
- D 4 S 25 20
- E 1 P 20 2
- E 3 S 20 15
- D 4 P 25 8
- E 1 S 25 5
- E 3 P 20 6
- D 3 S 20 15
- D 1 P 30 2
- E 4 S 30 20
- D 3 P 20 6
- D 1 S 20 5
- E 1 P 25 2
- E 3 S 25 15
- D 4 P 20 8
- E 1 S 20 5
- E 2 P 5 4
- D 3 S 5 15
- E 3 P 10 6

- D 1 S 15 5
- D 2 P 5 4
- E 3 S 5 15
- D 3 P 10 6
- E 2 S 10 10
- E 1 P 5 2
- D 1 S 5 5
- E 1 P 25 2
- D 4 S 25 20
- E 1 P 20 2
- E 3 S 20 15
- D 4 P 25 8
- E 1 S 25 5
- E 3 P 20 6
- D 3 S 20 15
- D 1 P 30 2
- E 4 S 30 20
- D 3 P 20 6
- D 1 S 20 5
- E 1 P 25 2
- E 3 S 25 15
- D 4 P 20 8
- E 1 S 20 5
- E 2 P 5 4
- D 3 S 5 15
- E 3 P 10 6

- D 2 S 10 10
- D 1 P 5 2
- E 3 S 5 15
- D 2 P 10 4
- D 1 S 10 5
- E 1 P 20 2
- D 2 S 20 10
- E 1 P 10 2
- E 1 S 10 5
- D 2 P 20 4
- E 1 S 20 5
- E 3 P 5 6
- D 2 S 5 10
- E 2 P 15 4
- D 3 S 15 15
- E 1 P 5 2
- E 2 S 5 10
- D 2 P 10 4
- D 2 S 10 10
- E 1 P 15 2
- E 1 S 15 5
- D 2 P 10 4
- E 2 S 10 10
- E 2 P 5 4
- D 2 S 5 10

- D 2 S 5 10
- D 1 P 20 2
- E 3 S 20 15
- D 2 P 5 4
- E 2 S 5 10
- D 2 P 10 4
- D 1 S 10 5
- E 1 P 15 2
- D 2 S 15 10
- E 1 P 10 2
- E 1 S 10 5
- D 2 P 15 4
- E 1 S 15 5
- E 3 P 5 6
- D 2 S 5 10
- D 1 P 15 2
- E 3 S 15 15
- D 2 P 5 4
- D 1 S 5 5
- E 1 P 10 2
- E 2 S 10 10
- D 3 P 5 6
- E 3 S 5 15

**Estados visitados: 5121**

# Teoria: A*

- A partir do estado inicial o A* utiliza uma função f(n) que combina
  - g(n): O custo do caminho acumulado até o momento (similar ao Dijkstra)
  - h(n): A função heurística que estima o custo restante baseado no peso e distância das caixas da posição ideal
- Para a expansão dos estados, A* escolhe o sucessor que minimiza o custo f(n) = g(n) + h(n)
  - O algoritmo termina quando encontra o caminho de menor custo e é o estado final.

# Exemplo: Busca A*

## Estado Inicial

- 50 40 30 25
- 20 15 10 5
- B
- 0 0 0 0
- 0 0 0 0
- 0 0 0 0

## Passos

- E 1 P 30 2
- E 1 S 30 5
- D 3 P 10 6
- E 3 S 10 15
- D 3 P 20 6
- E 2 S 20 10
- D 3 P 5 6
- E 1 S 5 5
- D 1 P 15 2
- E 3 S 15 15
- E 1 P 10 2
- D 1 S 10 5
- D 4 P 25 8
- E 5 S 25 25
- D 3 P 5 6
- E 2 S 5 10

**Estados visitados: 4922**

# Análise

- Estados visitados
  - A*: 4922
  - Djikstra: 24968
  - Greedy: 5121

Djkistra foi muito mais custoso que os outros dois algoritmos pois busca caminhos mais curtos explorando todos os possíveis nós, já o A* usa uma heurística para reduzir o número de estados visitados.

Enquanto o A* e Greedy podem ser mais eficientes, Djikstra garante o caminho mais curto

# Conclusão

- A*: Equilibra bem o número de estados visitados com o número de movimentos do braço mecânico.
- Greedy: Prioriza soluções imediatas. Uma solução é uma solução, mesmo que seja a pior.
- Djikstra: completo e ótimo, porém muito custoso, o que não é prático para problemas maiores ou reais.

Se o objetivo for otimizar a solução e o tempo de execução, A* parece ser a melhor escolha.