Documentação do Projeto 6: Banco de Conhecimentos (Sistema Especialista)

1. Visão Geral do Projeto

Este projeto é uma implementação de um **Banco de Conhecimentos (Knowledge Base)**, também conhecido como um **Sistema Especialista** simples. O objetivo é simular um agente inteligente (`SmartInvestor`) capaz de tomar decisões de investimento.

O agente utiliza um conjunto de **fatos** conhecidos sobre o mercado (ex: "a taxa Selic está alta") e um conjunto de **regras** (ex: "SE a Selic está alta, ENTÃO invista em Tesouro Direto") para inferir logicamente novas conclusões e responder a consultas.

Este projeto demonstra os conceitos fundamentais de agentes baseados em lógica e representação de conhecimento.

2. Como Foi Desenvolvido

O projeto foi desenvolvido inteiramente em **Python**, sem a necessidade de bibliotecas externas, focando na implementação pura da lógica de inferência.

- **Linguagem:** Python.
- **Estrutura Principal:** Uma classe `SmartInvestor` que encapsula toda a lógica.
- **Base de Conhecimentos (KB):** É composta por dois elementos principais:
    1. `self.facts`: Um `set` (conjunto) que armazena todos os fatos conhecidos como tuplas (ex: `('selic', 'alta')`). Usar um `set` garante que não haja fatos duplicados e a verificação seja rápida.
    2. `self.rules`: Uma `list` (lista) que armazena as regras de inferência. Cada regra é uma tupla contendo `(premissas, conclusão)`.
- **Interface:** A interação é feita via console. O código demonstra como adicionar fatos (`add_fact`), adicionar regras (`add_rule`) e fazer consultas (`ask`).

3. Algoritmo Utilizado

Encadeamento Para Frente (Forward Chaining)

O motor de inferência deste sistema especialista utiliza o algoritmo de **Encadeamento Para Frente (Forward Chaining)**, implementado no método `infer()`.

Este algoritmo funciona da seguinte maneira:

1. **Início:** O agente começa com um conjunto inicial de `fatos` na sua base de conhecimentos.
2. **Iteração de Regras:** O método `infer()` entra em um loop. A cada iteração, ele varre todas as `regras` cadastradas.
3. **Verificação de Premissas:** Para cada regra, o algoritmo verifica se *todas* as suas premissas (`premise`) estão presentes no conjunto `self.facts`.
4. **Disparo da Regra (Inferência):** Se todas as premissas de uma regra são verdadeiras (estão nos `fatos`) e a conclusão (`conclusion`) ainda não é um fato conhecido, a regra "dispara".
5. **Adição de Novo Fato:** A `conclusion` da regra disparada é adicionada ao `self.facts` como um novo fato.
6. **Repetição:** O loop continua (`while new_inferences:`) enquanto novas inferências forem feitas. Quando uma varredura completa das regras não gerar nenhum novo fato, o processo para.

Processo de Consulta (`ask`)

O método `ask(query)` combina a consulta com a inferência:

1. Ele primeiro verifica se o `query` (a pergunta, ex: "investimento_tesouro_direto") já está em `self.facts`.
2. Se não estiver, ele chama `self.infer()` para executar o encadeamento para frente e derivar todos os fatos possíveis com base no conhecimento atual.
3. Após a inferência, ele verifica novamente se o `query` foi adicionado aos `fatos`.
4. Finalmente, retorna `True` ou `False`.

```
❯ python3 -m pip freeze > requirements.txt
❯ /home/gabriel/inteligencia-artificial/env/bin/python /home/gabriel/inteligencia-artificial/6_banco_conhecimentos/b.py
Investir em Tesouro Direto? True
{'investimento_tesouro_direto', ('selic', 'alta'), 'investimento_petroleo_estrangeiro', ('petroleo', 'alto'), 'investimento_PETR4', 'investiment
o_tesouro_ipca', 'investimento_ativo_dolar', ('dolar', 'alto'), ('inflacao', 'alta')}
Investir em Tesouro IPCA? True
Investir em PETR4? True
Investir em Ativo Dolar? True
Investir em Petroleo Estrangeiro? True
Investir em bitcoin? False
Fato bitcoin alto adicionado.
Investir em bitcoin? True
Fato selic alta removido.
Investir em Tesouro Direto? False
```

Figura 01: Visão geral da saída