Explicação Linha a Linha do Script `grover_adiabatico_8qubits.py` (Ocean SDK)

Objetivo do script

O script `grover_adiabatico_8qubits.py` implementa, em **simulação clássica**, uma validação do **Hamiltoniano problema** (Ising) cuja energia mínima corresponde ao estado marcado

$$|m\rangle = |00000011\rangle \quad \text{(número } m = 3 \text{ em 8 qubits)}.$$

Ele **não** simula a dinâmica unitária completa de Roland–Cerf; em vez disso, utiliza um *sampler* clássico (recozimento simulado) para encontrar o estado de menor energia no modelo Ising.

1 Código completo

Listing 1: Script `grover_adiabatico_8qubits.py` (versão funcional).

```
"""
grover_adiabatico_8qubits.py
========================================
"Grover adiabtico" (verso de validao do Hamiltoniano final) usando Ocean SDK
em SIMULAO CLSSICA (Simulated Annealing).

O que este script faz:
- Constri um Hamiltoniano Ising simples cujo estado fundamental corresponde
    ao estado-alvo
  (neste exemplo: nmero 3 em 8 qubits -> |00000011>).
- Usa o SimulatedAnnealingSampler (clssico) para encontrar o mnimo de energia
    .
- Converte a melhor amostra de spins para bits e para o nmero inteiro.
- Mostra estatsticas de sucesso (quantas vezes o alvo apareceu em num_reads).

Execuo (PowerShell):
    ./.venv/Scripts/Activate.ps1
    python grover_adiabatico_8qubits.py
"""

from __future__ import annotations

import sys
from collections import Counter
```

```
from dimod import BinaryQuadraticModel
from dwave.samplers import SimulatedAnnealingSampler


def int_to_bits(x: int, nbits: int) -> list[int]:
    """Converte inteiro x para lista de bits (MSB -> LSB) com nbits."""
    if x < 0:
        raise ValueError("x deve ser no-negativo.")
    bits_lsb_first = [(x >> i) & 1 for i in range(nbits)]
    return list(reversed(bits_lsb_first))


def bits_to_int(bits_msb_first: list[int]) -> int:
    """Converte lista de bits (MSB -> LSB) para inteiro."""
    value = 0
    for b in bits_msb_first:
        if b not in (0, 1):
            raise ValueError("bits devem ser 0 ou 1.")
        value = (value << 1) | b
    return value


def bits_to_spins(bits_msb_first: list[int]) -> list[int]:
    """Mapeamento bit -> spin (Ising): 0 -> -1, 1 -> +1."""
    return [-1 if b == 0 else +1 for b in bits_msb_first]


def spins_dict_to_bits(sample: dict[int, int], nbits: int) -> list[int]:
    """Converte amostra {i: spin} em bits (MSB -> LSB) assumindo i=0..nbits-1.
        """
    bits = []
    for i in range(nbits):
        s = sample[i]
        if s not in (-1, +1):
            raise ValueError("spin deve ser -1 ou +1.")
        bits.append(0 if s == -1 else 1)
    return bits


def format_bits(bits: list[int]) -> str:
    """Formata bits como string '0101...'."""
    return "".join(str(b) for b in bits)


N_QUBITS = 8
TARGET_NUMBER = 3
NUM_READS = 1000


print("=== Diagnstico do ambiente ===")
```

```
print("Python executvel :", sys.executable)
print("Verso do Python :", sys.version.split()[0])
print()


target_bits = int_to_bits(TARGET_NUMBER, N_QUBITS)
target_spins = bits_to_spins(target_bits)

print("=== Estado-alvo ===")
print(f"Nmero alvo : {TARGET_NUMBER}")
print(f"Bits (MSB->LSB) : {format_bits(target_bits)} (lista: {target_bits})")
print(f"Spins (Ising) : {target_spins} (0->-1, 1->+1)")
print()


h = {i: -target_spins[i] for i in range(N_QUBITS)}
J = {}

bqm = BinaryQuadraticModel.from_ising(h, J)

print("=== Hamiltoniano Ising (BQM) ===")
print("Campos locais h_i:")
for i in range(N_QUBITS):
    print(f" h[{i}] = {h[i]:+d}")
print("Acoplamentos J_ij: (vazio)")
print()


sampler = SimulatedAnnealingSampler()

print("=== Executando recozimento simulado (clssico) ===")
sampleset = sampler.sample(bqm, num_reads=NUM_READS)


best = sampleset.first
best_sample = best.sample
best_energy = best.energy

best_bits = spins_dict_to_bits(best_sample, N_QUBITS)
best_number = bits_to_int(best_bits)

print("\n=== Melhor soluo ===")
print("Spins encontrados :", best_sample)
print("Energia :", best_energy)
print("Bits encontrados :", format_bits(best_bits), f"(lista: {best_bits})")
print("Nmero encontrado :", best_number)
print()


target_bitstring = format_bits(target_bits)
counts = Counter()
```

```
for row in sampleset.data(fields=["sample", "energy", "num_occurrences"]):
    sample_dict = row.sample
    occ = row.num_occurrences
    bits = spins_dict_to_bits(sample_dict, N_QUBITS)
    counts[format_bits(bits)] += occ

successes = counts[target_bitstring]
success_rate = successes / NUM_READS

print("=== Estatstica ===")
print(f"Total de leituras (num_reads) : {NUM_READS}")
print(f"Ocorrncias do alvo ({target_bitstring}) : {successes}")
print(f"Taxa de sucesso : {success_rate:.3f}")
print()

print("Top-5 resultados mais frequentes (bitstring: contagem):")
for bitstring, cnt in counts.most_common(5):
    print(f" {bitstring}: {cnt}")

print("\nObservao:")
print("- Este script valida a codificao do estado-alvo como mnimo do
    Hamiltoniano final.")
print("- Ele NO simula a dinmica unitria adiabtica completa (RolandCerf)   um
     solver clssico.")
```

2 Explicação linha a linha (didática)

Nesta seção, descrevemos item a item o papel de cada bloco do script, relacionando-o ao formalismo do modelo Ising.

2.1 Docstring inicial

- O bloco entre aspas triplas `"..."` é um **docstring**. Ele descreve:
  - o objetivo do script (validar o Hamiltoniano problema);
  - a ferramenta usada (Ocean SDK + recozimento simulado);
  - como executar no PowerShell.

2.2 Importações

- `from __future__ import annotations`: permite usar anotações de tipo de forma mais flexível (padrão moderno).
- `import sys`: usado para imprimir o caminho do interpretador Python e sua versão.
- `from collections import Counter`: usado para contar quantas vezes cada resultado (bitstring) aparece.

- `from dimod import BinaryQuadraticModel`: classe que representa um modelo quadrático binário, incluindo o caso Ising.
- `from dwave.samplers import SimulatedAnnealingSampler`: *sampler* clássico (não-quântico) que busca mínimos de energia em modelos Ising/BQM.

2.3 Funções utilitárias

- `int_to_bits(x, nbits)`: converte um inteiro x em uma lista de n bits **na ordem MSB→LSB**. Isso serve para obter $|m\rangle$ na forma binária (ex.: $3 \to 00000011$).
- `bits_to_int(bits)`: converte a lista de bits (MSB→LSB) de volta para um inteiro, para imprimir o resultado final.
- `bits_to_spins(bits)`: aplica o mapeamento Ising

$$0 \mapsto -1, \qquad 1 \mapsto +1.$$

- `spins_dict_to_bits(sample, nbits)`: converte a amostra retornada pelo solver (spins) em bits.
- `format_bits(bits)`: apenas formata a lista em string (`"00000011"`), para facilitar a leitura.

2.4 Parâmetros do problema

- `N_QUBITS = 8`: número de variáveis/spins (analogia com 8 qubits).
- `TARGET_NUMBER = 3`: o elemento marcado, isto é, o alvo da busca.
- `NUM_READS = 1000`: número de execuções independentes do solver (amostras) para estimar a taxa de sucesso.

2.5 Diagnóstico do ambiente

- `sys.executable` confirma **qual Python** está rodando (importante para evitar rodar fora do `.venv`).
- `sys.version` imprime a versão do Python.

2.6 Construção do estado-alvo

- `target_bits = int_to_bits(TARGET_NUMBER, N_QUBITS)` produz:

$$3 \mapsto [0,0,0,0,0,0,1,1] \equiv |00000011\rangle\,.$$

- `target_spins = bits_to_spins(target_bits)` produz:

$$[0,0,0,0,0,0,1,1] \mapsto [-1,-1,-1,-1,-1,-1,+1,+1].$$

2.7 Construção do Hamiltoniano Ising (problema)

O modelo Ising usado pelo Ocean tem energia clássica:

$$E(\boldsymbol{s}) = \sum_i h_i s_i + \sum_{i<j} J_{ij} s_i s_j, \qquad s_i \in \{-1, +1\}.$$

No script:

- `h = {i: -target_spins[i] for i in range(N_QUBITS)}` define:

  $$h_i = -t_i,$$

  onde t_i é o spin-alvo. Assim, a energia mínima ocorre em $\boldsymbol{s} = \boldsymbol{t}$.

- `J = {}` escolhe $J_{ij} = 0$ (sem acoplamentos).

- `bqm = BinaryQuadraticModel.from_ising(h, J)` cria o objeto compatível com os samplers do Ocean.

2.8 Solver (recozimento simulado clássico)

- `sampler = SimulatedAnnealingSampler()` seleciona um algoritmo **clássico** que tenta achar o mínimo de energia.

- `sampleset = sampler.sample(bqm, num_reads=NUM_READS)` executa o solver `NUM_READS` vezes.

2.9 Melhor solução e conversões

- `sampleset.first` retorna a melhor amostra encontrada.

- `best_energy` é a energia associada; no caso ideal:

  $$E_{\min} = \sum_i h_i t_i = \sum_i (-t_i) t_i = \sum_i (-1) = -8.$$

- O script converte spins→bits→inteiro e imprime o número final (esperado: 3).

2.10 Estatística de sucesso

- O `Counter` acumula quantas vezes cada bitstring apareceu no conjunto de amostras.

- `success_rate` estima a fração de execuções em que o solver encontrou exatamente `00000011`.

2.11 Mensagem final

O script termina explicitando a interpretação correta:

- Ele valida a **codificação** do estado marcado como mínimo do Hamiltoniano problema.

- Ele **não** simula a evolução unitária local-adiabática (Roland–Cerf).