G05IART0.02 - Inteligência Artificial
Trabalhos 1 e 2 - Busca (não informada e informada) e 8 Rainhas (Hill Climbing)
Datas de Entrega: Trabalho 1 — 21/10/2025; Trabalho 2 — 30/10/2025

Objetivos Gerais

- Consolidar conceitos de **busca em espaço de estados**, incluindo *buscas não informadas* (por exemplo, BFS/DFS/Custo Uniforme) e *buscas informadas* (**Busca Gulosa** e **A***).
- Aplicar **métricas de desempenho** (tempo, memória, completude e optimalidade) em cenários controlados.
- Explorar **otimização local** através do problema das **8 Rainhas** com *Hill Climbing*, comparando variações (movimentos laterais, reinícios aleatórios, etc.).

Trabalho 1 — Busca no Labirinto (Não Informada e Informada)

Descrição. Implementar e comparar dois algoritmos de busca não informada (ex.: BFS, DFS, Custo Uniforme) e dois algoritmos de busca informada (**Gulosa** e **A***) aplicados ao problema do labirinto da Figura 1. A atividade pode ser individual ou em dupla.

Figura 1. Labirinto para testes

1. Implementação

- Linguagem livre (recomenda-se Python ou C++). Disponibilizamos *códigos-base* mínimos no **??** .
- As **funções de busca (BFS/DFS/Custo Uniforme/Gulosa/A*) não estão implementadas aqui**; os alunos devem implementá-las seguindo o pseudocódigo visto em aula.
- Representação de *estado*: posição no grid (linha, coluna).
  Ações: mover N/S/L/O.
  Transição: aplicar a ação e validar limites/obstáculos.
  Custo: custo unitário por passo (ou tabela de custos, se desejado).
  Teste de objetivo: posição == meta.
  Heurística (para busca informada): exemplos sugeridos no **??** .

Formato dos dados (labirinto.txt)

Arquivo texto com grade retangular; caracteres permitidos: `S` (início), `G` (objetivo), `#` (parede), `.` (livre). Uma linha por linha do grid.

```
S....
.###.
..#..
.###.
....G
```

Listing 1. Exemplo de arquivo data/labirinto.txt

2. Medições de Desempenho

- **Tempo de Execução**: tempo até encontrar a solução.
- **Memória (definição para este trabalho)**: medir como o **máximo** de elementos simultaneamente mantidos nas estruturas de busca (*fronteira* — fila/pilha/heap — *mais* conjunto/mapa de explorados/-visitados), independentemente de hardware.
- **Completude**: o algoritmo encontra solução quando ela existe?
- **Optimalidade**: a solução tem custo mínimo?
- **Nós gerados/expandidos**: relatar contagens para comparação.

3. Análise dos Resultados

- Compare tempo, memória, completude e optimalidade entre (i) dois não informados e (ii) Gulosa vs A*.
- Discuta o impacto da heurística (admissibilidade/consistência) e dos parâmetros do problema (tamanho do labirinto, densidade de paredes).

4. Relatório

- Descrever algoritmos e estruturas de dados.
- Apresentar tabelas/gráficos das métricas.
- Análise comparativa e conclusões.
- Código-fonte e instruções de execução (README). Sugestão de estrutura em **??** .

Prazo e Entrega

Entrega do Trabalho 1: 15/10/2025. Submeter em repositório (GitHub/GitLab) + PDF do relatório no SIGAA/Google Classroom (a confirmar em sala).

Formato de submissão:

- **Repositório** contendo código e `README.md` com: como executar, parâmetros usados e como **reproduzir as tabelas/gráficos** do relatório.
- **PDF** nomeado como `trabalho1_<equipe>.pdf` (máximo de X páginas).
- **Dependências**: incluir `requirements.txt` (ou instruções equivalentes) com versões fixadas quando aplicável.
- Informar tempo de execução e máquina *apenas como referência*; a avaliação compara as **métricas estruturais** (nós expandidos, custo, completude/optimalidade).

Trabalho 2 — 8 Rainhas com Hill Climbing

Descrição. Resolver o problema das 8 Rainhas por *Hill Climbing* (subida de encosta), comparando pelo menos duas variações: (i) **movimentos laterais permitidos** com limite, (ii) **reinícios aleatórios** (Random-Restart). Opcional: *Simulated Annealing* para contraste.

1. Implementação

- Representação recomendada: vetor de tamanho 8, onde o índice é a coluna e o valor é a linha da rainha.
- Função de avaliação: número de pares de rainhas em conflito (ou seu negativo como *fitness*).
- Operador de vizinhança: mover uma rainha em sua coluna para outra linha.
- Critérios de parada: solução sem conflitos ou limite de iterações/reinícios.

2. Métricas

- **Tempo** e **número de reinícios** até encontrar solução.
- **Taxa de sucesso** (proporção de execuções que encontram solução em até K reinícios).

3. Relatório

- Descrever variações implementadas, parâmetros e resultados (médias, desvios, gráficos).
- Discussão sobre platôs, máximos locais e efeito de reinícios aleatórios.

Prazo e Entrega

Entrega do Trabalho 2: 29/10/2025. Mesmo formato do Trabalho 1.
Formato de submissão:

- **Repositório** com código e `README.md` descrevendo variações (laterais/reinícios), parâmetros e como **reproduzir** taxa de sucesso/tempo.
- **PDF** nomeado como `trabalho2_<equipe>.pdf` (máximo de X páginas).
- **Dependências**: `requirements.txt` (ou equivalente) com versões fixadas.

Critérios de Avaliação (ambos os trabalhos)

- **Correção e aderência (40%)**: implementação segue o pseudocódigo; algoritmos retornam soluções corretas; em T1, validade de custos/objetivos; em T2, implementação das variações (laterais/reinícios) conforme descritas.
- **Análise e relatório (30%)**: qualidade das métricas e comparações; discussões sobre completude/optimalidade; em T1, impacto da heurística (admissível/consistente); em T2, taxa de sucesso, efeito de platôs e reinícios.
- **Qualidade de código (20%)**: clareza, modularização, comentários, testes básicos; organização do repositório.
- **Reprodutibilidade (5%)**: existência de `README.md`, scripts/comandos, `requirements.txt` e possibilidade de replicar números do relatório.
- **Conformidade de entrega (5%)**: prazos, nomeação de arquivos, formato solicitado e declaração de uso de IA e créditos/autoria.

Estrutura sugerida de repositório

```
ia-trabalhos/
|-- trabalho1/
|   |-- src/
|   |   |-- maze.py
|   |   |-- search.py
|   |   `-- heuristics.py
|   |-- data/
|   |   `-- labirinto.txt
|   |-- tests/
|   |-- README.md
|   `-- relatorio.pdf
`-- trabalho2/
```

```
    |-- src/
    |   |-- eight_queens.py
    |   `-- hill_climbing.py
    |-- tests/
    |-- README.md
    `-- relatorio.pdf
```

Listing 2. Estrutura de pastas

Códigos-base mínimos (Python)

Exemplos de Representação (sem implementação de busca)

Trabalho 1 — Labirinto

```python
# maze_representation.py
from typing import List, Tuple

Grid = List[List[str]]
Pos = Tuple[int, int]

class Maze:
    Representa o espaço de estados para o problema do labirinto.
    Convenções:
      - 'S' = início, 'G' = objetivo, '#' = parede, '.' = célula livre.
      - Ações: mover para N, S, O, L (4-direções). Expandir para 8-direções se
    desejado.

    def __init__(self, grid: Grid):
        self.grid = grid
        self.H = len(grid)
        self.W = len(grid[0]) if self.H > 0 else 0
        self.start = self._find('S')
        self.goal = self._find('G')

    def _find(self, ch: str) -> Pos:
        for r in range(self.H):
            for c in range(self.W):
                if self.grid[r][c] == ch:
                    return (r, c)
        raise ValueError(fCaractere '{ch}' não encontrado no grid)

    def in_bounds(self, p: Pos) -> bool:
        r, c = p
        return 0 <= r < self.H and 0 <= c < self.W

    def passable(self, p: Pos) -> bool:
        r, c = p
        return self.grid[r][c] != '#'

    def actions(self, p: Pos):
        Retorna a lista de ações válidas em p. Ex.: ['N','S','O','L']
        acts = []
        r, c = p
        candidates = {
            'N': (r-1, c),
            'S': (r+1, c),
            'O': (r, c-1),
            'L': (r, c+1),
        }
        for a, q in candidates.items():
```

```
            if self.in_bounds(q) and self.passable(q):
                acts.append(a)
        return acts

    def result(self, p: Pos, a: str) -> Pos:
        Função de transição T(p,a).
        r, c = p
        delta = {'N':(-1,0), 'S':(1,0), 'O':(0,-1), 'L':(0,1)}
        dr, dc = delta[a]
        q = (r+dr, c+dc)
        if not (self.in_bounds(q) and self.passable(q)):
            raise ValueError(Ação inválida em p)
        return q

    def step_cost(self, p: Pos, a: str, q: Pos) -> float:
        Custo de passo. Por padrão, custo unitário.
        return 1.0

    def goal_test(self, p: Pos) -> bool:
        return p == self.goal

# Exemplo de uso (apenas representação; sem chamada de BFS/A*/etc.)
if __name__ == __main__:
    grid = [
        list(S....),
        list(.###.),
        list(..#..),
        list(.###.),
        list(....G),
    ]
    mz = Maze(grid)
    s = mz.start
    print(Ações válidas em S:, mz.actions(s))
```

Listing 3. Representação de labirinto e operações do problema (sem algoritmos de busca)

```
# heuristics_examples.py
from typing import Tuple
Pos = Tuple[int,int]

# Exemplos de assinaturas. Implementar conforme o pseudocódigo/discussão em aula.

def h_manhattan(a: Pos, b: Pos) -> float:
    Retorna a distância Manhattan entre a e b (admissível em grid 4-dir).
    raise NotImplementedError


def h_euclidiana(a: Pos, b: Pos) -> float:
    Distância Euclidiana (pode ser usada conforme o modelo do problema).
    raise NotImplementedError
```

Listing 4. Esqueleto de heurísticas (preencher pelos alunos)

Trabalho 2 — 8 Rainhas

```
# eight_queens_representation.py
# Representação: board[c] = linha (0..7) da rainha na coluna c (0..7)
from typing import List, Iterable, Tuple

Board = List[int]
Move = Tuple[int,int]  # (coluna, nova_linha)

```

```python
N = 8

def initial_board() -> Board:
    Pode-se iniciar aleatoriamente ou determinístico. Deixar livre para o aluno.
    Exemplo (a preencher pelos alunos): retornar uma lista de tamanho 8.

    raise NotImplementedError

def conflicts(board: Board) -> int:
    Função de avaliação: número de pares de rainhas em conflito.
    Implementar conforme pseudocódigo.

    raise NotImplementedError

def neighbors(board: Board) -> Iterable[Move]:
    Gera movimentos de vizinhança: mover a rainha de uma coluna para outra linha.
    Retorna pares (coluna, nova_linha) válidos.

    raise NotImplementedError

def apply(board: Board, mv: Move) -> Board:
    Retorna um novo tabuleiro após aplicar o movimento mv.
    c, r = mv
    newb = board.copy()
    newb[c] = r
    return newb

# Observação: a lógica de Hill Climbing (escolha do melhor vizinho,
# movimentos laterais, reinícios) NÃO está aqui; os alunos a implementarão.
```

Listing 5. Representação do estado e vizinhança (sem algoritmo de subida de encosta)

Observações finais

- É permitido usar bibliotecas padrão, mas **não** usar bibliotecas que resolvam diretamente os problemas (ex.: solvers prontos).
- Fixe a semente pseudoaleatória quando necessário (ex.: `random.seed(42)`) para reprodutibilidade em testes.
- Entregas em dupla devem incluir declaração de coautoria e divisão de tarefas.
- **Política de uso de IA**: permitido para revisão textual/comentários de código. **Vedado** para gerar a implementação dos algoritmos. Declare qualquer uso de IA no final do relatório (ferramenta e finalidade).
- **Créditos e autoria**: inclua uma seção no relatório detalhando *quem fez o quê* (por pessoa), recursos externos consultados (artigos, repositórios) e eventuais contribuições de terceiros (com permissão e citação).
- **Reprodutibilidade**: inclua `requirements.txt` (ou equivalente), versões fixadas quando possível, scripts/comandos de execução e instruções para **reproduzir as tabelas e gráficos** do relatório.

Créditos e Declaração de Autoria (modelo para o relatório)

Inclua, ao final do relatório, uma seção com o seguinte modelo:

- **Autores e papéis**: Nome A (implementação A, experimento X, escrita seções 1–2); Nome B (implementação B, análise Y, escrita seções 3–4).
- **Uso de IA**: ferramenta(s) utilizada(s) (ex.: ChatGPT) *apenas* para revisão textual/comentários. Nenhuma parte de código de algoritmos foi gerada por IA.
- **Recursos externos**: listar artigos, livros, repositórios consultados, com links/citações.
- **Declaração**: confirmamos que o código entregue foi desenvolvido pela equipe, respeitando as políticas da disciplina.

Prazo geral

Para referência no cronograma da disciplina: **Trabalho 1 — 21/10/2025**; **Trabalho 2 — 30/10/2025**.