FATEC OURINHOS

Trabalho 1 - IA

Alunos 2° ADS AMS:
{ Daniela Campos Rodrigues, Lucas Pontes Soares, Pedro Ramos Lima }

RESOLUÇÃO DE LABIRINTO

Ourinhos/SP
2025

TURMA	ADS-AMS			**DISCIPLINA**	IAAM
PERÍODO	VESPERTINO	**ANO**	2º	**PROFESSOR**	VINICIUS GODOY
REF	T1				
ALUNOS	Daniela Campos Rodrigues Lucas Pontes Soares Pedro Ramos de Lima				
RA	Daniela - 0211432412002 Lucas - 0211432412005 Pedro Ramos - 0211432412014				

Exercício: Aplicação de Técnicas de Inteligência Computacional para Resolução de Problemas Reais

Objetivo do Exercício:

Neste exercício, você deverá identificar um problema real que possa ser resolvido utilizando técnicas de Inteligência Computacional, como Computação Evolucionária, Agentes Inteligentes, Sistemas Especialistas ou Métodos de Busca Cega/Gulosa. O exercício será implementado no Google Colab utilizando Python.

Técnicas a Serem Utilizadas:

Escolha uma das seguintes técnicas para resolver o problema que você selecionou:

- **Computação Evolucionária**: Utilize um Algoritmo Genético ou outros métodos evolutivos para encontrar uma solução para o seu problema.
- **Agentes Inteligentes**: Implemente um ou mais agentes que interajam entre si para resolver o problema de forma distribuída.
- **Sistemas Especialistas**: Crie um sistema baseado em regras para resolver o problema de forma especializada.
- **Métodos de Busca Cega/Gulosa**: Aplique uma busca cega (como busca em profundidade ou largura) ou uma busca gulosa para otimizar a solução do problema.

Passos a Seguir:

- **Escolha do Problema Real**:
  - Identifique um problema real que pode ser resolvido utilizando uma das técnicas acima. O problema pode ser de qualquer área, como logística, otimização, previsão, entre outros. Este problema deve ser relevante e passível de ser abordado com a técnica escolhida.
- **Desenvolvimento no Google Colab**:
  - Implemente a solução para o problema escolhido no Google Colab utilizando Python. Certifique-se de incluir:
    - A preparação dos dados (se necessário),
    - A implementação do algoritmo escolhido,
    - A visualização dos resultados de forma clara (gráficos, tabelas, etc.),
    - A análise de desempenho do algoritmo (tempo de execução, qualidade da solução encontrada).
- **Preenchimento do Documento Word**:
  - Após a implementação, você deverá preencher um documento Word com as seguintes seções:

- **Contextualizar o Problema**:

A resolução de problemas em ambientes desconhecidos é uma tarefa desafiadora, comum em diversos domínios da ciência computacional e da engenharia. Um exemplo clássico dessa complexidade pode ser ilustrado por um labirinto, onde um agente precisa encontrar um caminho até o seu destino sem informações prévias sobre a configuração do espaço. No contexto desse trabalho, o problema escolhido envolve a geração de um labirinto aleatório, no qual a tarefa é encontrar uma solução para navegar desde o ponto de partida até o ponto de chegada, ou destino.

Neste cenário, a busca pela solução é realizada através de uma busca cega, uma técnica que simula a exploração do espaço sem qualquer conhecimento adicional sobre o ambiente, caracterizando a solução como uma sequência de tentativas de movimentos até encontrar o destino. A relevância deste problema se reflete na necessidade de soluções adaptativas e eficientes para ambientes dinâmicos e desconhecidos, áreas aplicáveis à robótica, inteligência artificial e sistemas de navegação autônoma. A abordagem escolhida, portanto, visa testar a capacidade de algoritmos de buscar caminhos em espaços desconhecidos de maneira sistemática e sem heurísticas adicionais.

Caso esse tipo de problema não seja adequadamente resolvido, as consequências podem incluir falhas em sistemas de navegação autônoma, como robôs móveis e veículos autônomos, que podem ficar presos ou seguir rotas ineficientes.

- **Objetivo**:

Este trabalho tem como objetivo principal a implementação de um algoritmo de busca cega para resolver o problema de navegação em um labirinto aleatório. O algoritmo desenvolvido utilizará a técnica de busca cega, caracterizada pela exploração de todos os possíveis caminhos do ambiente até que o ponto de destino seja alcançado. A busca cega por largura foi escolhida justamente por sua simplicidade e pela capacidade de explorar completamente o espaço de soluções, sem a necessidade de informações ou heurísticas sobre o labirinto.

O ponto inicial é representado por I e o ponto final por F. A solução proposta visa a implementação do algoritmo de forma eficiente, que, após a execução, permita uma visualização clara do caminho percorrido. A análise da performance do algoritmo será realizada, destacando aspectos como o tempo de execução e a eficácia da técnica aplicada para resolver o problema proposto.

- **Justificativa**:

O problema escolhido foi o de resolver um labirinto, ou seja, encontrar um caminho do ponto de partida até o destino em um ambiente desconhecido. Esse tipo de problema é comumente encontrado em diversas áreas da Inteligência Artificial (IA), especialmente no desenvolvimento de algoritmos de busca. Labirintos são um excelente modelo para aplicar técnicas de busca, pois representam um espaço de estados bem definido, composto por caminhos e obstáculos, tornando-os um bom ponto de partida para explorar métodos de navegação computacional.

Foi utilizado a técnica dos métodos de busca cega, pois como é um labirinto, é adequada para ambientes onde não temos informações adicionais, como distâncias entre pontos ou custos dos caminhos, como heurística por exemplo. Especificadamente utilizamos a busca em largura, na qual o algoritmo é simples, eficiente e ideal para problemas de busca em grafos, onde o objetivo é explorar todos os caminhos possíveis de forma sistemática. Vantagens:

- Garante encontrar a solução;
- Exploração nivelada, onde cada nível é totalmente explorado antes de passar para o próximo;
- Explorando nível por nível (em camadas);

- Ideal quando o caminho mais curto é importante.

Em relação a outras abordagens:

- Memória e tempo: Embora a busca em largura seja eficiente, ela pode consumir bastante memória, pois mantém todos os caminhos possíveis na memória, o que pode ser um problema em labirintos grandes;
- Prevenção de loops: A busca por largura não fica "presa" em loops infinitos, algo que pode acontecer com a busca por profundidade, que explora um único caminho até o fim antes de voltar atrás;
- Simplicidade: Não exige a heurística, informações externas, dicas, que é justamente a ideia do labirinto desconhecido.

Metodologia:

A metodologia aplicada para resolver o problema de navegação no labirinto envolveu as seguintes etapas:

1. Pesquisamos sobre geração de labirintos;
2. Criamos um pseudocódigo (uma representação textual e simplificada do algoritmo) a partir do que aprendemos, para termos uma ideia geral do que precisamos antes de programar;
3. No ambiente Google Colab. Primeiramente fizemos a geração do labirinto aleatórios com coordenadas de inicio e fim fixas, mas os caminhos sempre aleatórios e o tamanho configuravél. A visualização do labirinto foi feita diretamente no console, utilizando o caractere # para representar paredes e o espaço em branco (' ') para representar caminhos;
4. No labirinto, usamos a técnica de gerar todo o labirinto primeiro, preenchido tudo com paredes '#'. Em seguida, utilizamos uma técnica de escavação baseada em busca em profundidade, usando uma pilha para "cavar" os caminhos (' ') partindo de um ponto inicial. Essa abordagem garante que o labirinto seja aleatório, mas sempre possível de ser resolvido;

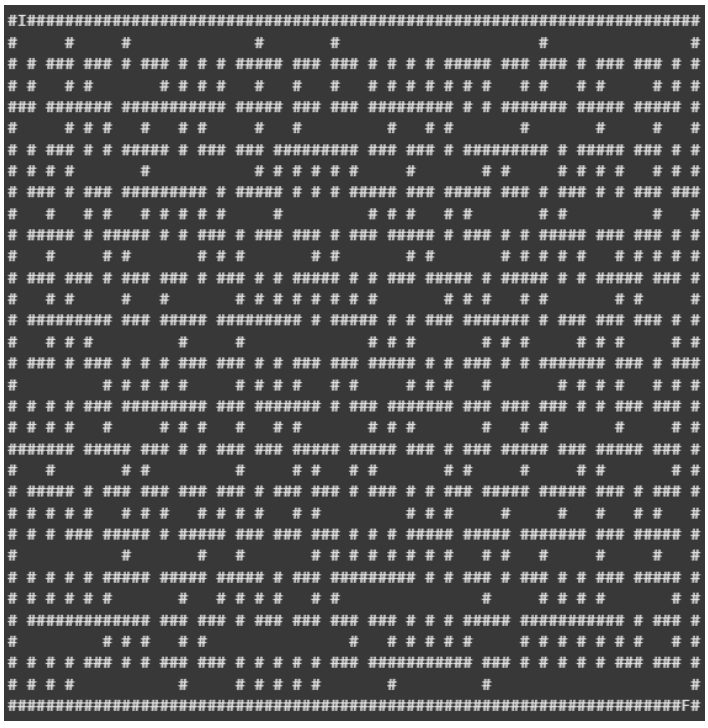

Figura 1: Exemplo visual do labirinto gerado

5. Com o labirinto gerado, aplicamos o algoritmo de busca cega em largura (BFS - Breadth-First Search). Esse algoritmo utiliza uma fila para explorar o labirinto, visitando todos os nós (células) de uma profundidade n antes de prosseguir para os de profundidade n+1, e armazenando os nós já visitados;
6. A cada passo, imprimimos o labirinto no console para visualizar o progresso de busca, com um pequeno atraso (delay) permitindo visualizar a progressão da busca em tempo real, o que facilita a compreensão do funcionamento do algoritmo;

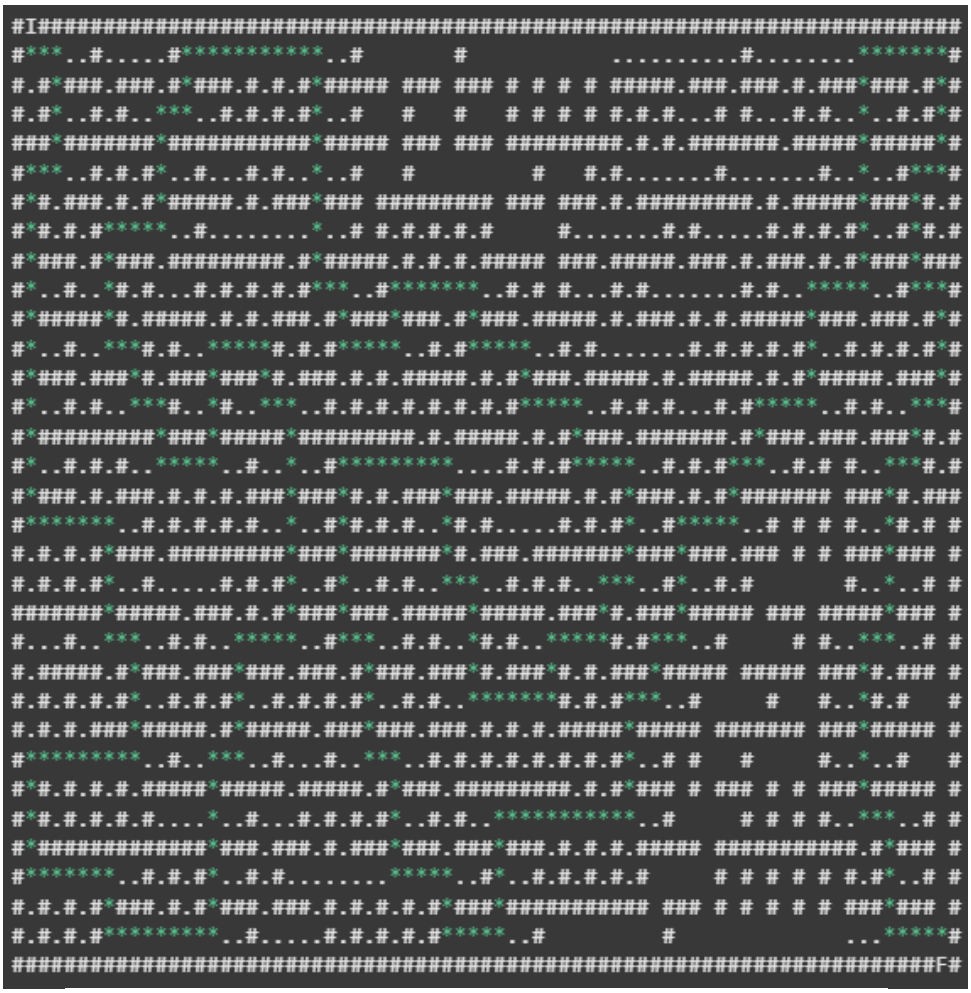

Figura 2: Exemplo visual do labirinto sendo resolvindo
("#" - Paredes)
("." – caminhos explorados)
("" – caminho final realizado)*

7. Também realizamos uma versão da execução sem o atraso e sem a impressão a cada passo, para medir o desempenho real da busca em termos de tempo de execução;
8. Mantivemos o código organizado em funções específicas e bem comentado, seguindo boas práticas de programação para facilitar o entendimento e a manutenção.

- **Hipóteses**:

Partimos da hipótese de que, em todos os labirintos gerados aleatoriamente, sempre existirá pelo menos um caminho viável que conecte o ponto de início ao ponto de destino. Para garantir essa condição, utilizamos uma técnica de geração que assegura que o labirinto seja solucionável. Além disso, o tamanho do labirinto é flexível e pode ser definido pelo usuário no momento da execução, permitindo testes em diferentes escalas.

Embora a busca em largura (BFS) seja eficiente e garanta encontrar a solução mais curta, ela pode consumir muita memória e tempo de processamento em labirintos muito grandes, uma vez que explora amplamente todos os caminhos possíveis até atingir o objetivo. Com base nisso, levantamos a hipótese de que técnicas mais avançadas, como a busca bidirecional ou buscas informadas com heurísticas (como A*), poderiam otimizar significativamente o desempenho, tanto em termos de tempo quanto de uso de recursos computacionais.

Outra hipótese considerada é que a fixação dos pontos de início e fim que pode limitar a variabilidade e a dificuldade do problema. Ao tornar essas posições aleatórias a cada execução, o desafio se tornaria mais dinâmico, além de permitir uma avaliação mais ampla da robustez dos algoritmos implementados.

Por fim, ampliando o escopo do problema, consideramos que a inclusão de fatores como custos de movimentação, tempo de execução, limites de recursos, ou penalidades para determinados caminhos permitiria a aplicação de algoritmos com heurísticas, capazes de encontrar não apenas um caminho possível, mas o melhor caminho com base em critérios definidos. Essa evolução abriria espaço para resolver problemas mais próximos do mundo real, como rotas em mapas, planejamento de trajetos logísticos ou movimentação autônoma de robôs em ambientes com restrições.

- **Envio do Exercício**:
  - O código implementado no Google Colab deve ser enviado em formato de link (notebook).
  - O documento Word preenchido também deve ser enviado como parte do exercício.

Avaliação (Pesos):

A avaliação do exercício será realizada com base nos seguintes critérios, com a distribuição de pesos conforme detalhado abaixo:

1. **A clareza na escolha do problema e na aplicação da técnica (25%)**
   - Avalia a relevância do problema escolhido e a adequação da técnica utilizada para resolvê-lo.
2. **A implementação correta do algoritmo no Google Colab (35%)**
   - Avalia a qualidade da implementação no Google Colab, incluindo a preparação dos dados, a correta aplicação do algoritmo e a análise de resultados.
3. **A qualidade da análise dos resultados e a apresentação dos mesmos (20%)**
   - Avalia a clareza e profundidade na análise dos resultados obtidos, bem como a apresentação visual dos mesmos (gráficos, tabelas, etc.).
4. **A justificativa e a profundidade na explicação das etapas do exercício no documento Word (20%)**
   - Avalia a profundidade e clareza das explicações no documento Word, incluindo a contextualização do problema, o objetivo, a justificativa da escolha da técnica e a metodologia seguida.