Problema do Labirinto
Busca em Grafos

Gabriel P. Soares[1], **Lucas P. O. Alves**[1], **Leticia F. Oliveira**[1],
Gustavo F. Pascoaleto[1], **Jorran L. A. Santos**[1]

[1] Instituto de Ciências Exatas – Ciência da Computação
Universidade Federal de Alfenas (UNIFAL)
Março de 2024

1. Introdução

Neste trabalho, iremos apresentar o algoritmo para resolução do problema do labirinto. O problema do labirinto constitui uma questão central em diversas áreas da ciência da computação, englobando aspectos da teoria dos grafos e algoritmos de busca. Define-se como a tarefa de identificar um percurso que, partindo de um ponto de entrada especificado, atinja um ponto de saída em um ambiente estruturado como um labirinto. Este ambiente é caracterizado por um conjunto de caminhos possíveis intercalados por barreiras, exigindo a navegação através de um espaço delimitado por restrições físicas ou lógicas.

A representação deste problema frequentemente recorre à utilização de estruturas de dados como grafos ou grades, nas quais os nós simbolizam posições possíveis dentro do labirinto e as arestas indicam permissões de deslocamento entre estas posições. As obstruções são representadas pela ausência de arestas conectivas ou pela designação de nós como intransitáveis. Assim configurado, o desafio se articula em torno da implementação de algoritmos de busca capazes de descobrir um trajeto que ligue o início ao fim, cumprindo as condições impostas pelo labirinto.

Vários algoritmos têm sido empregados na resolução do problema do labirinto, destacando-se pela eficiência ou pela adequação a contextos específicos. A Busca em Largura (BFS) e a Busca em Profundidade (DFS) sobressaem pela simplicidade e eficácia, com o BFS assegurando a identificação do caminho mais curto em labirintos sem ponderação nas conexões, e o DFS investigando as rotas de maneira intensiva, ainda que sem garantir a eficiência ótima. Métodos como o algoritmo de Dijkstra e o A*(A-estrela) apresentam alternativas para cenários onde os deslocamentos acarretam custos diferenciados, visando a otimização do percurso.

O problema do labirinto encontra aplicações práticas significativas. Na robótica, serve como base para a navegação autônoma em ambientes complexos ou mutáveis; nos jogos eletrônicos, contribui para o desenvolvimento de desafios de navegação e para a inteligência artificial de entidades virtuais; e nas redes de computadores, inspira algoritmos dedicados ao roteamento de dados. A investigação e solução do problema do labirinto promovem uma compreensão aprofundada de conceitos fundamentais em algoritmos de busca e estruturas de dados, além de fomentar o desenvolvimento de habilidades críticas para a resolução de problemas em um leque amplo de aplicações computacionais.

2. Estrutura de Dados

Estrutura Ponto

A estrutura Ponto é definida para representar uma posição dentro do labirinto, contendo coordenadas x e y que indicam a localização específica em uma grade bidimensional. Ela facilita a manipulação de localizações dentro do labirinto, permitindo uma representação dos pontos de entrada e saída, bem como das posições intermediárias exploradas pelos algoritmos de busca.

Matriz Bidimensional (char maze[MAX][MAX])

O labirinto é representado como uma matriz bidimensional de caracteres, onde cada célula pode conter um valor que indica um espaço livre ('0'), uma barreira ('X'), o ponto de entrada ('E') ou o ponto de saída ('S'). Esta representação permite uma visualização simplificada do ambiente de navegação, além de facilitar a verificação de movimentos válidos e a implementação de algoritmos de busca.

Grafo (bool grafo[MAX * MAX][MAX * MAX])

Para a representação do labirinto como um grafo, é utilizada uma matriz de adjacência, na qual cada célula acessível do labirinto é tratada como um vértice. A matriz de adjacência grafo é uma matriz bidimensional de valores booleanos, onde a presença de um valor verdadeiro (true) na posição [u][v] indica uma aresta direta entre os vértices u e v, ou seja, uma possibilidade de movimento direto entre as respectivas células no labirinto. Esta estrutura permite a aplicação eficaz de algoritmos de busca em grafos, como DFS e BFS, para encontrar um caminho entre o ponto de entrada e o ponto de saída.

Fila (queue int q)

Foi utilizada para armazenar os índices dos vértices que ainda devem ser explorados, garantindo que a busca se desenvolva de maneira sistemática e em camadas, o que é fundamental para a identificação do caminho mais curto no labirinto.

Vetor de Visitados (bool visited[MAX * MAX])

O vetor de visitados foi utilizado para marcar os vértices que já foram explorados pelos algoritmos de busca, evitando que um mesmo vértice seja processado mais de uma vez.

Vetor de Pais (int parent[MAX * MAX])

O vetor de pais armazena, para cada vértice, o índice do vértice a partir do qual ele foi alcançado durante a busca. Este mecanismo facilita a reconstrução do caminho encontrado ao final dos algoritmos de busca, permitindo que se trace o percurso de volta do ponto de saída ao ponto de entrada, ao seguir a cadeia de predecessores registrados.

3. Algoritmos Utilizados

Algoritmo de Busca em Largura (BFS)

O algoritmo de Busca em Largura é utilizado para explorar o grafo camada por camada, partindo do vértice de origem. Ele emprega uma fila para gerenciar os vértices que precisam ser explorados, garantindo que os vértices sejam visitados na ordem em que foram descobertos. Isso é particularmente útil para encontrar o caminho mais curto em um grafo não ponderado, como é o caso do labirinto representado neste código.

Complexidade do BFS:

A complexidade temporal do BFS é O(V+E), onde V é o número de vértices e E é o número de arestas no grafo. Isso se deve ao fato de cada vértice ser enfileirado e desenfileirado exatamente uma vez, e cada aresta é explorada no processo. A complexidade espacial, por sua vez, é O(V), devido à necessidade de armazenar um vetor de visitados e a fila de vértices a serem explorados.

Algoritmo de Busca em Profundidade (DFS)

O algoritmo de Busca em Profundidade explora o grafo seguindo um caminho até o final antes de retroceder e tentar outra direção. É uma estratégia recursiva que se aprofunda tanto quanto possível ao longo de cada ramo antes de voltar atrás.

Complexidade do DFS:

A complexidade temporal do DFS também é O(V+E) para um grafo representado por uma lista de adjacências, o que é comparável ao BFS. No entanto, a representação do labirinto como uma matriz de adjacência implica que cada vértice é verificado contra todos os outros vértices para identificar arestas existentes. A complexidade espacial do DFS é O(V) devido à necessidade de armazenar um vetor de visitados, além do espaço necessário para a pilha de chamadas de função devido à natureza recursiva do algoritmo.