Busca em Grafo

Algoritmos e Estrutura de Dados III

Gabriel Pereira Soares
Gustavo Fernandez Pascoaleto
Jorran Luka Andrade dos Santos
Letícia Freitas de Oliveira
Lucas Pessoa Oliveira Alves

Universidade Federal de Alfenas, Março de 2024

Conteúdo

Tópicos

Introdução

- Labirintos são problemas interessantes que podem ser resolvidos usando algoritmos de busca em grafos.
- Duas tecnicas de busca de grafo foram utilizadas:
  - A busca em largura (BFS)
  - A busca em profundidade (DFS)

Tópicos

Tipos de Busca em Grafo

- Busca em Largura (BFS):
  - A busca em largura explora todos os vértices do grafo em largura, garantindo que o caminho encontrado seja o mais curto possível.
- Busca em profundidade (DFS):
  - A busca em profundidade explora um ramo do grafo até atingir uma folha antes de retroceder. Essa abordagem é útil em situações onde o objetivo é encontrar qualquer caminho válido, independentemente de sua extensão.

Tópicos

Explicação do Problema

- Cada labirinto é representado por uma matriz, onde cada célula pode conter diferentes elementos:
  - E' para entrada.
  - 'S' para saída.
  - 'X' para parede (obstáculo).
  - '0' para espaço vazio (caminho possível).
- O objetivo é encontrar um caminho válido, que não passe por paredes ('X'), da entrada ('E') até a saída ('S').

Tópicos

Algoritmos Implementados

- Constroi Grafo.
  - A função constroiGrafo tem como objetivo construir um grafo a partir de um labirinto representado por uma matriz de caracteres. Esse grafo será utilizado posteriormente para realizar buscas.

Algoritmos Implementados

```
// Constroi o grafo
void constroiGrafo(char maze[MAX][MAX], bool grafo[MAX * MAX][MAX * MAX])
{
    for (int x = 0; x < MAX; x++)
    {
        for (int y = 0; y < MAX; y++)
        {
            if (ehValido(x, y, maze))
            {
                int u = getIndex(x, y);
                // Conecta u com seus vizinhos validos no grafo
                if (ehValido(x + 1, y, maze))
                    grafo[u][getIndex(x + 1, y)] = true;
                if (ehValido(x - 1, y, maze))
                    grafo[u][getIndex(x - 1, y)] = true;
                if (ehValido(x, y + 1, maze))
                    grafo[u][getIndex(x, y + 1)] = true;
                if (ehValido(x, y - 1, maze))
                    grafo[u][getIndex(x, y - 1)] = true;
            }
        }
    }
}
```

Algoritmos Implementados

- Busca em Largura (BFS)
  - A (BFS) explora todos os vértices do grafo em largura, começando pelo vértice inicial e expandindo para os vértices vizinhos antes de avançar para os vértices mais distantes.

Algoritmos Implementados

```
bool BFS(char maze[MAX][MAX], Ponto comeco, Ponto final, int parent[MAX * MAX])
{
    bool visited[MAX * MAX] = {false};
    queue<int> q;                                   // criacao da fila de visitados
    int comecoIndex = getIndex(comeco.x, comeco.y); // descobrindo onde esta o E e o S do labirinto
    int finalIndex = getIndex(final.x, final.y);
    q.push(comecoIndex);
    visited[comecoIndex] = true;
    parent[comecoIndex] = -1;

    bool grafo[MAX * MAX][MAX * MAX] = {false};
    constroiGrafo(maze, grafo);

    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        int ux = u / MAX;
        int uy = u % MAX;

        if (u == finalIndex)
            return true; // Encontrou o fim

        for (int vx = 0; vx < MAX; vx++)
        {
            for (int vy = 0; vy < MAX; vy++)
            {
                int v = getIndex(vx, vy);
                if (grafo[u][v] && !visited[v])
                {
                    q.push(v);
                    visited[v] = true;
                    parent[v] = u;
                }
            }
        }
    }
    return false;
```

Algoritmos Implementados

- Busca em Profundidade (DFS)
  - A (DFS) procura um caminho até encontrar um beco sem saída, retrocede e tenta outro caminho, continuando até encontrar a saída ou esgotar todas as possibilidades.

Algortimos Implementados

```
// Função auxiliar para realizar a busca DFS no grafo
bool DFSUtil(int u, bool grafo[MAX * MAX][MAX * MAX], bool visited[], int parent[], int endIndex)
{
    visited[u] = true; // Marca o vértice atual como visitado

    if (u == endIndex)
        return true; // Se alcançou o destino, retorna verdadeiro.

    for (int v = 0; v < MAX * MAX; v++)
    {
        if (grafo[u][v] && !visited[v])
        {
            parent[v] = u;
            if (DFSUtil(v, grafo, visited, parent, endIndex))
                return true;
        }
    }

    return false; // Retorna falso se o destino não pode ser alcançado a partir de u.
}

// Realiza DFS para encontrar um caminho do labirinto
bool DFS(char maze[MAX][MAX], Ponto start, Ponto end, int parent[MAX * MAX])
{
    bool visited[MAX * MAX] = {false};          // Armazena se cada vértice foi visitado
    bool grafo[MAX * MAX][MAX * MAX] = {false}; // Representação do labirinto como um grafo
    constroiGrafo(maze, grafo);
    int startIndex = getIndex(start.x, start.y); // Índice do ponto de início
    int endIndex = getIndex(end.x, end.y);       // Índice do ponto de fim
    parent[startIndex] = -1;                     // O ponto de início não tem pai

    return DFSUtil(startIndex, grafo, visited, parent, endIndex);
}
```

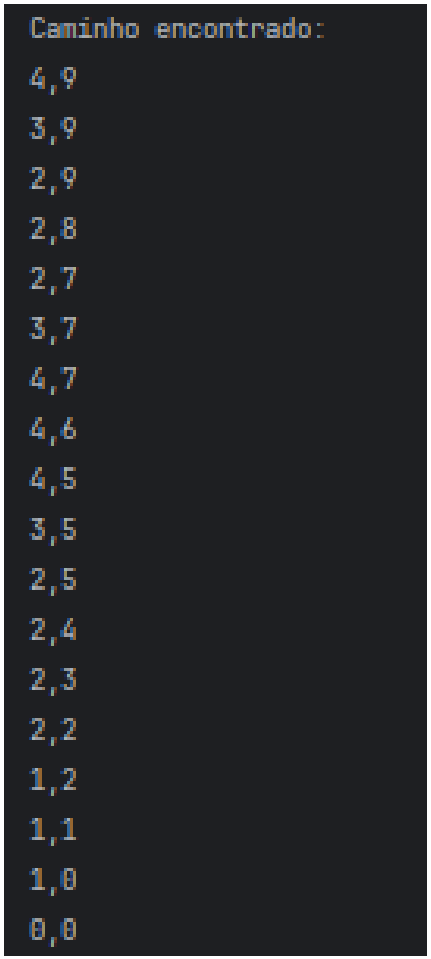
Tópicos

Exemplo

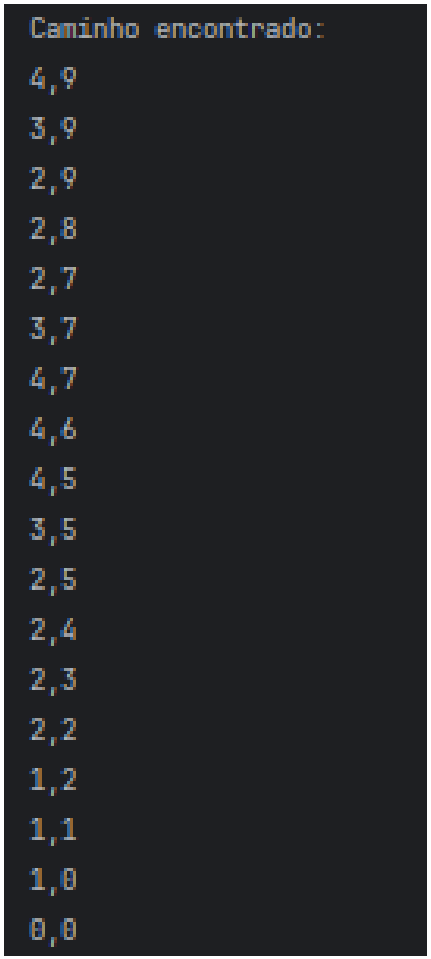
```
char maze2[MAX][MAX] = { // Labirinto escolhido
        {'S','X','X','X','X','X','X','X','X','X'},
        {'0','0','0','X','X','X','X','X','X','X'},
        {'0','X','0','0','0','0','X','0','0','0'},
        {'0','X','0','X','X','0','X','0','X','0'},
        {'0','X','0','X','X','0','0','0','X','E'},
        {'0','X','0','X','X','X','X','X','X','X'},
        {'0','X','0','0','0','0','0','X','X','X'},
        {'0','X','X','X','X','X','0','X','X','X'},
        {'0','0','0','X','X','0','0','0','X','X'},
        {'0','X','X','X','X','0','X','0','X','X'},
};
```

Figura: Entrada do labirinto 2

Exemplo



BFS



DFS

Tópicos

Conclusão

- Os algoritmos de busca em grafos são ferramentas poderosas na resolução de problemas de caminho em labirintos e em uma variedade de outros problemas.
- Além disso, a modelagem adequada do labirinto como uma matriz de caracteres e a utilização de técnicas de busca apropriadas são cruciais para a eficácia e eficiência da solução.