Relatório Técnico: Lógica de Grafos no Algoritmo de Labirinto

Autor: Bruno Andrade, Gabriel Gauterio

Curso: Ciência da Computação - Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

Data: Novembro de 2024

1. Introdução

Este relatório detalha a lógica de grafos utilizada no algoritmo de labirinto para identificar regiões isoladas. O algoritmo foi desenvolvido com base em conceitos de teoria dos grafos, aproveitando o uso de matrizes para representar as conexões e as barreiras do labirinto.

2. Metodologia

2.1 Representação do Grafo

O labirinto é modelado como um grafo onde cada célula da matriz representa um vértice. As conexões entre os vértices são determinadas pela ausência de barreiras nas direções adjacentes (cima, baixo, esquerda e direita). A matriz é utilizada para armazenar as informações de conexões e paredes, representadas por valores hexadecimais. A lógica do grafo permite explorar todas as células conectadas, formando regiões isoladas.

2.2 Algoritmo de Busca em Profundidade (DFS)

O algoritmo de busca em profundidade (DFS) é utilizado para explorar o grafo. A DFS percorre recursivamente os vértices conectados a um vértice inicial, marcando-os como visitados para evitar ciclos. Cada nova chamada da DFS identifica uma nova região isolada do labirinto.

2.3 Lógica de Exploração

A lógica da DFS para exploração de regiões é baseada nos seguintes passos:

1. Iniciar a exploração em uma célula não visitada.
2. Marcar a célula como visitada.
3. Verificar cada direção (cima, direita, baixo, esquerda):
   a. Se não houver parede e a célula adjacente não foi visitada, continuar a DFS.
4. Registrar os vértices explorados como parte da mesma região.

Exemplo de código para a DFS:

```
private void dfs(int i, int j, Map<String, Integer> contador) {
    if (i < 0 || i >= m || j < 0 || j >= n || visitado[i][j] || labirinto[i][j] == '#') {
        return;
    }

    visitado[i][j] = true;
```

```
        char tipoSer = labirinto[i][j];

        if (Character.isUpperCase(tipoSer)) {
            String raca = MAPA_RACAS.get(tipoSer);
            contador.put(raca, contador.getOrDefault(raca, 0) + 1);
        }

        // Movimenta para cima, direita, baixo e esquerda
        if (!temParede(i, j, 'U')) dfs(i - 1, j, contador); // Cima
        if (!temParede(i, j, 'R')) dfs(i, j + 1, contador); // Direita
        if (!temParede(i, j, 'D')) dfs(i + 1, j, contador); // Baixo
        if (!temParede(i, j, 'L')) dfs(i, j - 1, contador); // Esquerda
    }
```

3. Resultados

Os testes demonstraram que o algoritmo é capaz de identificar corretamente as regiões isoladas em labirintos de tamanhos variados. Para cada região, o algoritmo contabiliza os seres presentes, com base no mapeamento de caracteres para raças, permitindo análise mais detalhada das áreas exploradas.

4. Análise de Desempenho

A análise de complexidade do algoritmo é fundamental para entender seu desempenho em diferentes tamanhos de labirinto. A seguir, apresentamos a análise detalhada.

4.1 Complexidade de Tempo

O algoritmo utiliza uma abordagem de busca em profundidade (DFS) para explorar todas as células do labirinto. Cada célula é visitada exatamente uma vez, e em cada visita são verificadas as quatro direções (cima, baixo, esquerda, direita). Portanto, a complexidade de tempo do algoritmo é O(V + E), onde:

- V é o número de vértices (células no labirinto, ou m * n).
- E é o número de arestas (conexões entre células adjacentes, que dependem do número de células sem paredes).

Na prática, para um labirinto representado por uma matriz de m x n, a complexidade é aproximadamente O(m * n), já que o número de arestas é limitado pelas células adjacentes.

4.2 Complexidade de Espaço

O consumo de memória do algoritmo é determinado por três fatores principais:

1. A matriz de entrada, que ocupa O(m * n).
2. A matriz de visitados, que também ocupa O(m * n).
3. A pilha de recursão, cujo tamanho máximo é proporcional à profundidade máxima da DFS. "

Assim, a complexidade de espaço total é O(m * n), dominada pelas matrizes de entrada e visitados.

5. Conclusão

O algoritmo implementado demonstra a eficiência da lógica de grafos para resolver problemas de identificação de regiões isoladas. A análise de complexidade confirma que o algoritmo é adequado para labirintos de tamanhos grandes, sendo eficiente tanto em tempo quanto em consumo de memória. A escolha de DFS, associada à estrutura de matrizes, proporciona clareza e robustez à solução.