Algoritmos e Estruturas de Dados II

Capítulo da Aula 16: Busca em Profundidade (DFS)

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br
CEFET-MG - Campus Timóteo

Fevereiro de 2026

1. O Labirinto do Minotauro

A DFS funciona como alguém explorando um labirinto com um novelo de lã (como Teseu). 1. Siga um caminho até onde der. 2. Se chegar num beco sem saída, volte (**Backtracking**) até a última bifurcação. 3. Tente outro caminho.

Diferente da BFS (que usa Fila/Níveis), a DFS usa uma **Pilha (Stack)** (geralmente a pilha de recursão do próprio sistema).

2. Algoritmo Recursivo

A implementação recursiva é elegante e curta.

Código Java

```
// Variáveis Globais (na classe da Solução)
boolean[] visited;
List<List<Integer>> adj;

public void dfs(int u) {
    // 1. Marca como visitado
    visited[u] = true;
    System.out.print(u + " "); // Pré-ordem

    // 2. Explora vizinhos
    for (int v : adj.get(u)) {
        if (!visited[v]) {
            dfs(v); // Chamada Recursiva
        }
    }
}
```

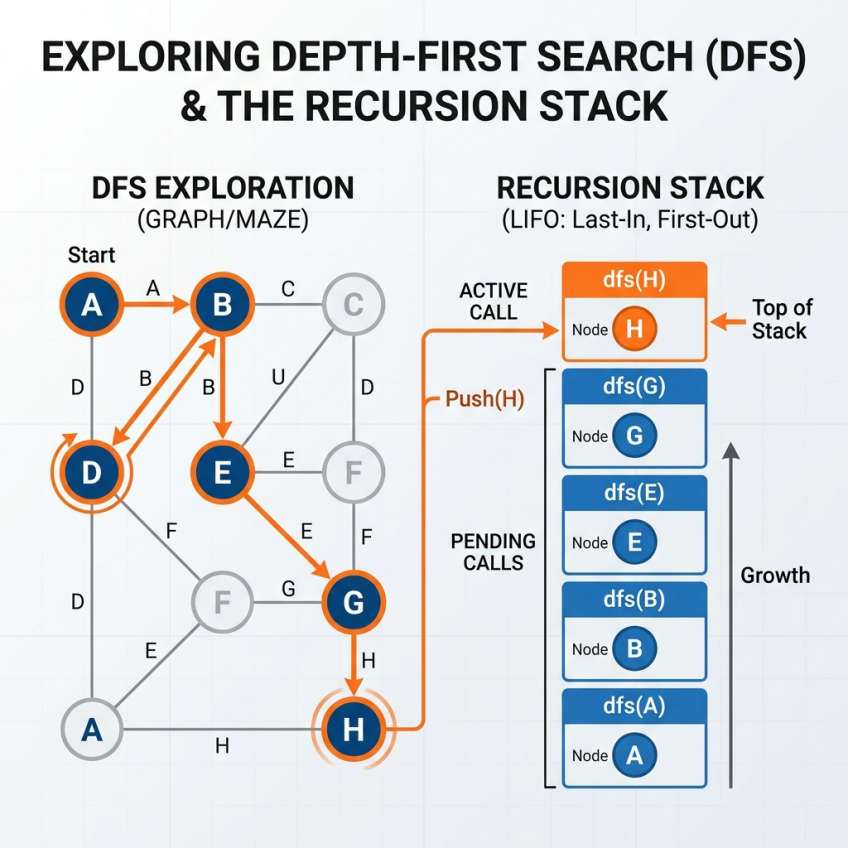

Figure 1: DFS usa pilha (ou recursão): desce até o fim de um caminho e volta (backtracking).

O Perigo do Stack Overflow

Se o grafo for uma "linha" de 10^5 nós ($1 \to 2 \to 3 \cdots \to 10^5$), teremos 10^5 chamadas recursivas empilhadas.

- O limite padrão da Stack do Java/C++ pode estourar (StackOverflowError).
- Em competições, C++ geralmente aguenta, mas Java pode precisar de configuração (`-Xss`).

3. Classificação de Arestas e Ciclos

Para tirar o máximo proveito da DFS, classificamos os vértices em 3 cores (estados): 1. **BRANCO (0):** Não visitado. 2. **CINZA (1):** Em processamento (está na pilha de recursão). 3. **PRETO (2):** Finalizado (já exploramos todos os descendentes).

Detecção de Ciclos

- **Grafo Não-Direcionado:** Se virmos uma aresta para um nó **CINZA** ou **PRETO** que **não** seja o nosso pai imediato, achamos um ciclo.

- **Grafo Direcionado:** Só existe ciclo se encontrarmos uma aresta para um nó **CINZA** (Back Edge). Se for PRETO, é apenas um cruzamento (Cross Edge) ou avanço (Forward Edge), não ciclo.

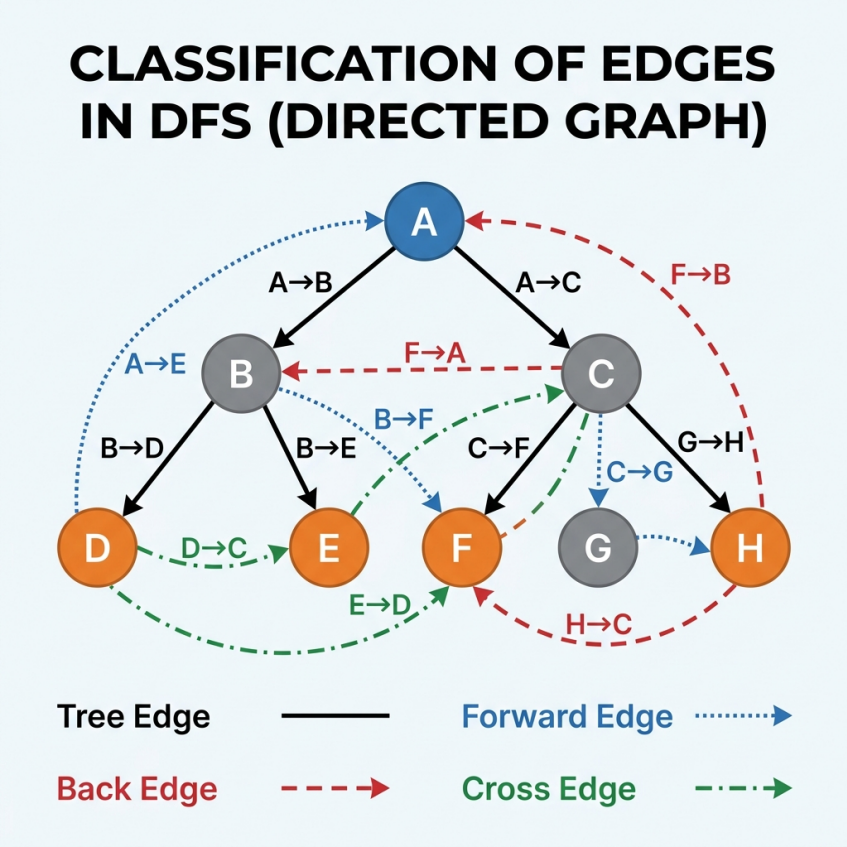

Figure 2: Classificação de arestas: tree, back, forward, cross (em digrafos).

Implementação de Detecção de Ciclo (Direcionado)

```
int[] state; // 0=White, 1=Gray, 2=Black

boolean hasCycle(int u) {
    state[u] = 1; // Pinta de Cinza (Entrando)

    for (int v : adj.get(u)) {
        if (state[v] == 1) {
            return true; // Achou nó cinza: CICLO! (Back Edge)
        }
        if (state[v] == 0) {
            if (hasCycle(v)) return true;
        }
    }
```

```
        state[u] = 2; // Pinta de Preto (Saindo)
        return false;
    }
```

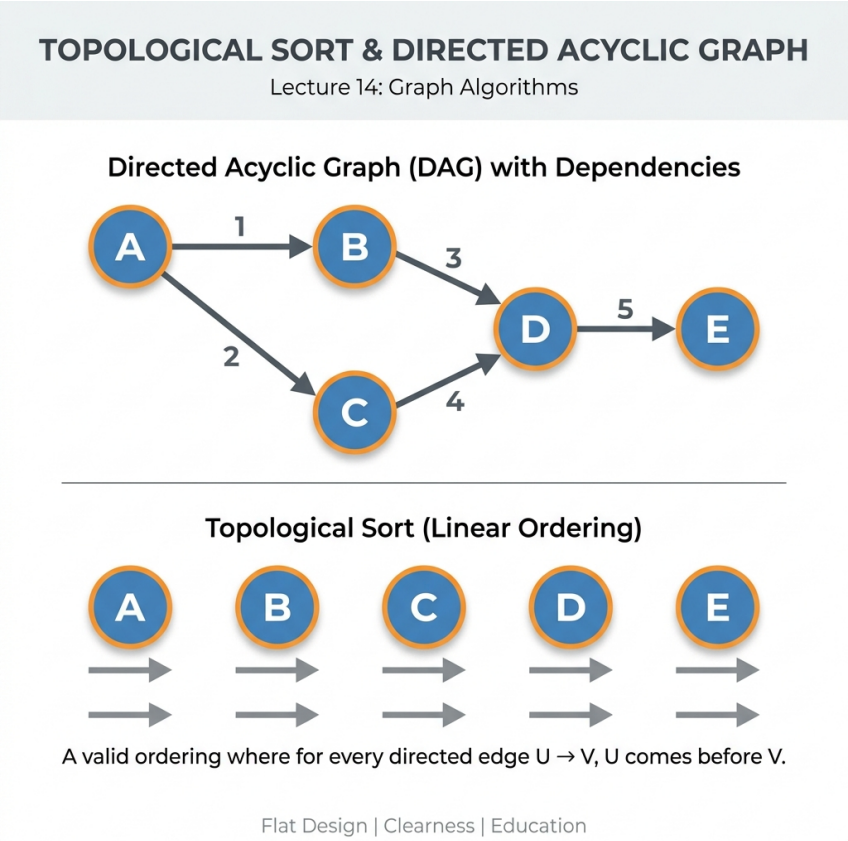

Figure 3: Em um DAG, a ordem em que os nós ficam pretos na DFS dá ordenação topológica.

4. DFS Iterativa (Pilha Explícita)

Para evitar Stack Overflow ou ter controle manual, usamos uma `Stack<Integer>`.

```
public void dfsIterative(int start) {
    Stack<Integer> stack = new Stack<>();
    stack.push(start);
    visited[start] = true;

    while (!stack.isEmpty()) {
        int u = stack.pop();
        System.out.print(u + " ");

        for (int v : adj.get(u)) {
            if (!visited[v]) {
                visited[v] = true;
```

```
                    stack.push(v);
                }
            }
        }
}
```

Nota: A ordem de visita pode diferir ligeiramente da versão recursiva dependendo da ordem que os vizinhos são empilhados.

5. Aplicações Típicas

A DFS é mais versátil que a BFS para propriedades estruturais: 1. **Contar Componentes Conexos:** Chame DFS em um loop para cada nó não visitado. 2. **Ordenação Topológica:** Use DFS e guarde os nós numa lista quando eles ficarem PRETOS (pós-ordem reversa). 3. **Componentes Fortemente Conexos (SCC):** Algoritmo de Kosaraju ou Tarjan (baseados em DFS). 4. **Pontes e Articulações:** Algoritmo de Tarjan para grafos não-direcionados. 5. **Resolver Labirintos:** Achar *qualquer* caminho da entrada à saída.

Complexidade

- **Tempo:** $O(V + E)$ (semelhante à BFS).
- **Espaço:** $O(V)$ no pior caso (grafo linha), tanto para recursão quanto pilha explícita.

• Teoria

A DFS explora o grafo seguindo um caminho até o fim antes de retroceder (backtracking). Isso a torna ideal para problemas que requerem exploração completa de subgrafos, como detecção de ciclos, ordenação topológica e componentes fortemente conexos.

6. Classificação de Arestas em DFS

Em uma DFS, as arestas podem ser classificadas em:

- **Tree Edge:** Aresta que leva a um vértice não visitado (parte da árvore DFS).
- **Back Edge:** Aresta que conecta um vértice a um ancestral na árvore DFS (indica ciclo em grafos direcionados).
- **Forward Edge:** Aresta que conecta um vértice a um descendente já processado (apenas em digrafos).
- **Cross Edge:** Aresta entre vértices em subárvores diferentes (apenas em digrafos).

7. Aplicações em Engenharia de Computação

- **Compiladores:** Ordenação topológica de dependências, análise de fluxo de controle.
- **Sistemas de Arquivos:** Detecção de links simbólicos circulares, travessia de diretórios.
- **Redes:** Detecção de loops em topologias de rede, análise de dependências.
- **Algoritmos de Grafos:** Base para algoritmos mais complexos (Kosaraju, Tarjan, detecção de pontes).

8. Exercícios

8.1. Exercícios Conceituais

1 Explique a diferença entre DFS recursiva e DFS iterativa. Quando cada uma é preferível?

2 Descreva como usar DFS para detectar ciclos em grafos direcionados e não direcionados. Por que a abordagem é diferente?

3 Compare DFS com BFS em termos de estrutura de dados, ordem de visita e casos de uso típicos.

8.2. Exercícios Analíticos

1 Simule DFS (recursiva) no seguinte grafo a partir do vértice 0, mostrando a ordem de visita e classificando cada aresta:

```
0 -> 1 -> 2
|    |    |
v    v    v
3 -> 4 -> 5
```

2 Para um grafo com V vértices e E arestas, calcule o número máximo de chamadas recursivas em DFS. Em que tipo de grafo isso ocorre?

3 Dado um grafo direcionado, explique como usar DFS para verificar se é um DAG (Directed Acyclic Graph) e, se for, como obter uma ordenação topológica.

8.3. Exercícios de Programação

1 Implemente DFS completa em Java com:

- Versão recursiva e iterativa.
- Detecção de ciclos (direcionado e não direcionado).
- Classificação de arestas (tree, back, forward, cross).

- Contagem de componentes conexos.
- Ordenação topológica (para DAGs).

2 Resolva problemas de labirinto usando DFS:

- Encontre qualquer caminho da entrada à saída.
- Encontre todos os caminhos possíveis.
- Compare com a solução usando BFS.

3 Resolva problemas de juiz online que usam DFS, como:

- Detecção de ciclos.
- Ordenação topológica.
- Componentes fortemente conexos (usando algoritmo de Kosaraju ou Tarjan).