AED2 - Algoritmos e Estr. de Dados II

Aula 15: Busca em Largura (BFS)

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

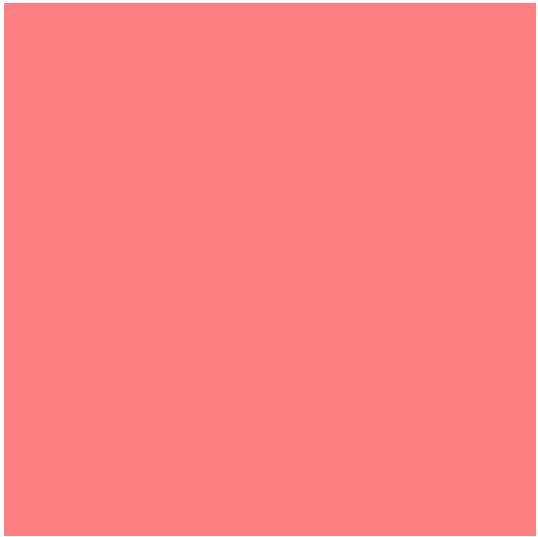
CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Fevereiro de 2026

Sumário

1. Explorando em Camadas

BFS explora em **níveis** (dist. 0, 1, 2...). Garante **caminho mínimo** em arestas para grafos não ponderados.



2. O Algoritmo

Utilizamos uma **Fila (Queue)** para gerenciar a ordem de visita (FIFO - First In, First Out).
Estados dos Vértices:

- **Não Visitado:** Ainda não descoberto.
- **Visitado/Na Fila:** Descoberto, mas seus vizinhos ainda não foram processados.
- **Processado:** Já saiu da fila e seus vizinhos já foram checados.

Passo a Passo: 1. Inicialize um vetor `visited[]` como false e `dist[]` como ∞. 2. Coloque o nó inicial S na Fila, marque `visited[S] = true` e `dist[S] = 0`. 3. Enquanto a Fila não estiver vazia:

- Remova o elemento u da frente.
- Para cada vizinho v de u:
- Se v **não** foi visitado:
- Marque `visited[v] = true`.
- Defina `dist[v] = dist[u] + 1`.
- Adicione v na Fila.

3. Implementação em Java

Assumindo que temos a lista de adjacência `List<List<Integer>> adj`.

```java
import java.util.*;

public class BFS {

    public static void bfs(int start, int V, List<List<Integer>> adj
        ) {
        // Estruturas Auxiliares
        boolean[] visited = new boolean[V];
        int[] dist = new int[V];
        int[] parent = new int[V]; // Para reconstruir o caminho

        Arrays.fill(dist, -1); // -1 indica inalcançável
        Arrays.fill(parent, -1);

        // Configuração Inicial
        Queue<Integer> queue = new LinkedList<>();
        queue.add(start);
```

4. BFS em Grids (Matrizes)

Muitos problemas (ex: labirintos) são dados como uma matriz, onde cada célula `(r, c)` é um vértice e vizinhos são `(r+1, c)`, `(r-1, c)`, etc. Não precisamos criar uma Lista de Adjacência explícita. Usamos a matriz diretamente e vetores de direção.

```
// Direcoes: Cima, Direita, Baixo, Esquerda
int[] dr = {-1, 0, 1, 0};
int[] dc = {0, 1, 0, -1};

// Na BFS:
while(!queue.isEmpty()) {
    Point p = queue.poll();

    for(int i=0; i<4; i++) {
        int nr = p.r + dr[i];
        int nc = p.c + dc[i];

        if(isValid(nr, nc) && !visited[nr][nc]) {
            visited[nr][nc] = true;
```

5. Multi-Source BFS

E se quisermos saber a qual a distância da "saída mais próxima" num labirinto com múltiplas saídas? **Truque:** Coloque **todas** as saídas na fila inicial com distância 0. A BFS vai expandir todas as "frentes de onda" simultaneamente. O primeiro a chegar num vértice define sua distância mínima para *qualquer* saída. **Aplicações Típicas:** 1. Menor caminho em labirintos/mapas sem peso. 2. Checar conectividade (componentes conexos). 3. Web Crawlers (Explorar links nível a nível). 4. Algoritmo de fluxo Edmonds-Karp (usa BFS para achar caminho aumentante).