Algoritmos e Estruturas de Dados II

Capítulo da Aula 15: Busca em Largura (BFS)

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

Fevereiro de 2026

1. A Topologia das Ondas: Explorando Margens de Fronteira

A Busca em Largura (BFS - *Breadth-First Search*) renega as varreduras logísticas em abismos estritos verticais profundos, adotando um comportamento fundamentalmente tático de cerco geográfico. Sua essência operacional replica fisicamente uma vibração isotrópica na água cruzada: as frentes de onda (as margens dinâmicas do processamento) deslocam-se de forma massiva e simétrica preenchendo todos os horizontes ativos simultâneos do raio transversal.

Ela esmaga e desvenda o Grafo escalonando rigorosamente o ambiente base em estritos **Níveis Concêntricos de Expansão (Camadas)**, aterrissando as sondas de sondagem originadas do núcleo do epicentro isolado absoluto S:

1. **Círculo Focal Zero ($d = 0$):** A ignição exclusiva isolada restrita alojada internamente núcleo origem.
2. **Frente de Margem Primária ($d = 1$):** A absorção logística esmagadora simultânea transversal de absolutamente todos os entes colaterais (vizinhos) blindados puros cravados do contato de ramificação direta primária.
3. **Frente de Choque Secundária ($d = 2$):** O engate cego da esteira expansiva engolindo maciçamente e arrastando todos os contatos adjacentes limpos isolados dos operários vizinhos processados primários, transbordando o efeito cruzado varrendo redes cruas progressivamente sem distinção.

Cimentado em cima dessa propriedade estrutural e analítica inviolável na absorção limpa simétrica cravada integral dos raios limites adjacentes absolutos fechados atômicos idênticos unificadores isolados iterativamente, a máquina base BFS confessa abertamente o maior e absoluto prêmio resolutivo logístico base que detém o poder esmagador de invariavelmente rastrear impiedoso o isolado tráfego definitivo restrito da **Camada de Caminho Mínimo** purista em métrica limpa (saltos absolutos numéricos enxutos restritíssimos em contagem de pontes atreladas e restritas puras limitantes cravadas absolutas limpas de arestas puras limitantes restritas cruas isoladas vazias cruas) dissecando ativamente toda limitação cega complexa de grafos soltos massivos que preterem ou carecem uniformes massivos (**grafos não-ponderados**).

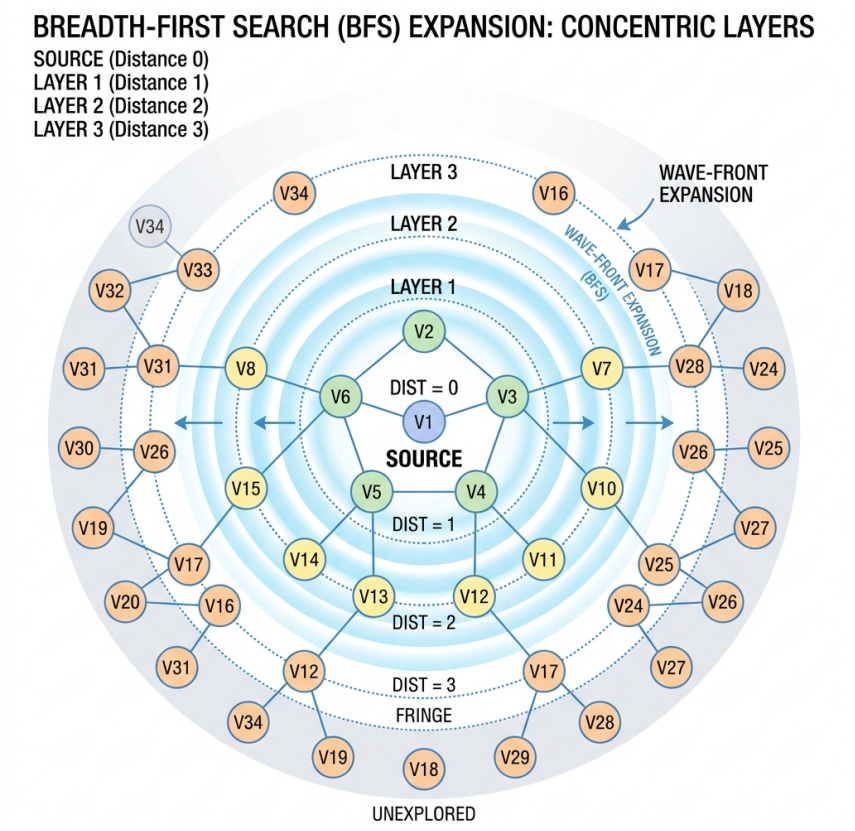

Figure 1: BFS explora em camadas: distância 0, depois 1, depois 2...

2. O Motor FIFO de Escalonamento

A engenharia estrutural que orquestra a inércia e represa a fúria das ondas progressivas espaciais exige sob pressão estrita um container limpo inflexível e imparcial. Abandonamos neste passo a recursão solta abismal do hardware estático de Pilhas e embutimos a garantia mecânica passiva fria da enforcada **Fila Clássica Bruta de Escalonamento (Queue)** no fluxo linear limpo (*FIFO - First In, First Out*): as frentes pioneiras ditam incondicionalmente a lei absoluta prioritária base restrita limpa transversal fechada limpa transversal absoluta temporal das limpezas estruturais subsequentes.

A Decomposição da Vida Logística dos Vértices Limitados:

- **1. O Silêncio Vazio Cru (Não Visitado isolado):** O ramo passivo desconhecido boia ainda virgem indetectável invisível nos abismos mortos limpos da esteira de varredura ativa estático da malha cega nativa inativa absoluta nativa purista morta pura inexplorada inativa cega estrita.
- **2. O Encarceramento Dinâmico Estreito Cruzado Estrito Primário Cego (Margem Descoberta Cega na Fila):** O ramo central cego base atômico acatou submerso fisicamente ao toque violento letal da onda de rastreio limitante da fronteira estrita impiedosa! A identificação abstrata pura inativa cega restrita do contato imediato aciona imediatamente engates cravados de salvaguarda vetorial passiva pura cega enjaulando imediatamente de foma ativa englobando sem restrições cruas limpas pas-

sivas. Apenas enfileirado para os procedimentos formais limitantes no aguardo passivo limite atômico morto cruzado puro morto fechado limiar estéril estéril processual.

- **3. Purificação Logística Completa Bruta Cruzada Atômica (O Processo Final Cru Limpo Limiar):** O topo da esteira e o encabeçador limpo final executam a expurgação final violenta isolada. Arrancamos o topo limpo da fila base, mapeamos e rasgamos todos os parentes e vínculos em volta e, na câmara mortuária limitante limpa despachante do loop contínuo, sepultamos o nó isolado cego garantindo processamento logístico pleno isolado final da rede infinita cruzada!

O Passo a Passo Analítico: 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`.

```
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);
        visited[start] = true;
        dist[start] = 0;
```

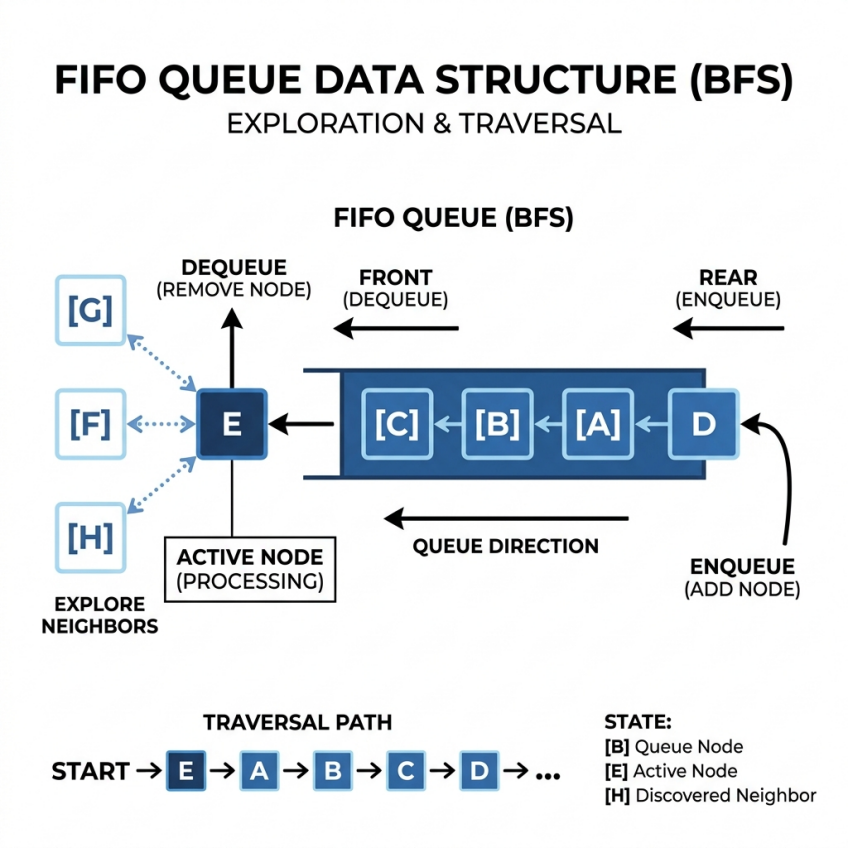

Figure 2: Fila FIFO: o primeiro vértice que entra é o primeiro a ser processado.

```
        System.out.println("Iniciando BFS a partir de " + start);

        while (!queue.isEmpty()) {
            int u = queue.poll(); // Tira da frente
            // System.out.print(u + " "); // Ordem de visita

            for (int v : adj.get(u)) {
                if (!visited[v]) {
                    visited[v] = true;
                    dist[v] = dist[u] + 1;
                    parent[v] = u; // Guardamos de onde viemos
                    queue.add(v);
                }
            }
        }

        // Exemplo: Imprimir distancias
        for(int i=0; i<V; i++) {
            System.out.println("Distancia de " + start + " para " + i + ":
        }
    }
```

```
    // Bonus: Reconstruir o caminho de Start até Target
    public static List<Integer> getPath(int target, int[] parent) {
        List<Integer> path = new ArrayList<>();
        if (parent[target] == -1) return path; // Sem caminho

        for (int v = target; v != -1; v = parent[v]) {
            path.add(v);
        }
        Collections.reverse(path); // O caminho foi montado de trás pra fre
        return path;
    }
}
```

Complexidade

- **Tempo:** $O(V + E)$. Cada vértice entra na fila uma vez e cada aresta é verificada duas vezes (uma por ponta, em grafos não direcionados).
- **Espaço:** $O(V)$ para a fila e vetores auxiliares.

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.

```
// Direçoes: 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;
            queue.add(new Point(nr, nc));
        }
    }
}
```

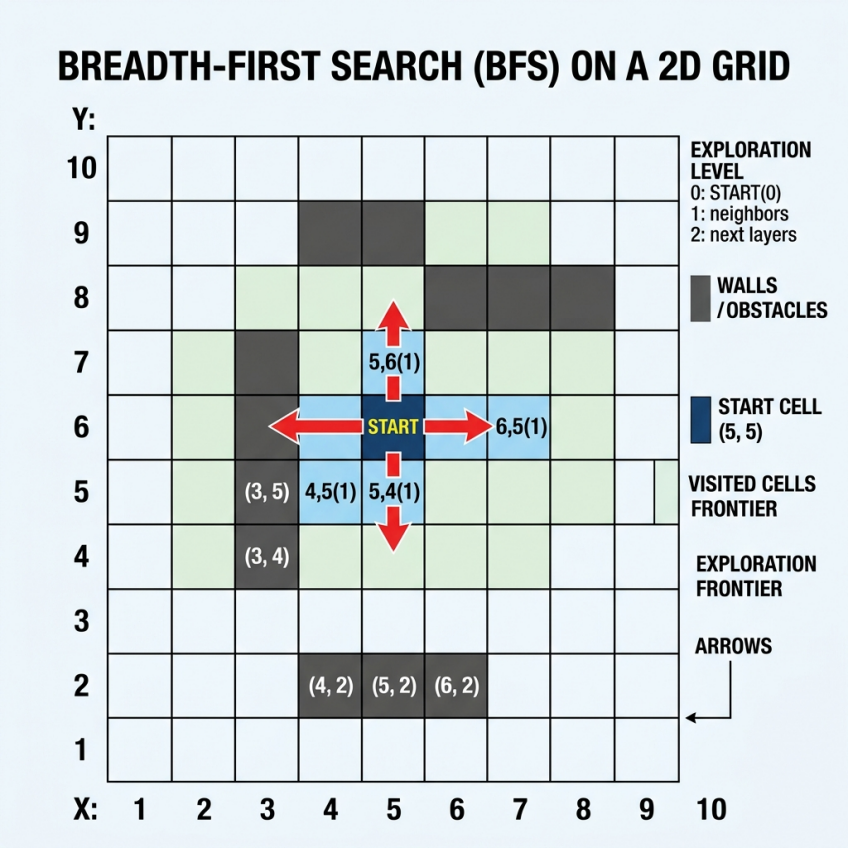

Figure 3: BFS em grid: vizinhos são as quatro (ou oito) células adjacentes.

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).

• Teoria

A BFS garante encontrar o caminho com menor número de arestas de s para qualquer vértice alcançável. Isso ocorre porque ela explora vértices em ordem crescente de distância de s, garantindo que quando um vértice é visitado pela primeira vez, já encontramos o caminho mínimo até ele.

6. Demonstração Passo a Passo

Considere o grafo não direcionado:

```
0 -- 1 -- 2
```

```
|     |     |
3 -- 4 -- 5
```

BFS a partir do vértice 0:

Iteração	Fila	Vértice Processado	Distâncias Atualizadas
Inicial	[0]	-	dist[0]=0
1	[1,3]	0	dist[1]=1, dist[3]=1
2	[3,2,4]	1	dist[2]=2, dist[4]=2
3	[2,4]	3	(nenhuma, 4 já visitado)
4	[4,5]	2	dist[5]=3
5	[5]	4	(nenhuma)
6	[]	5	-

Resultado: distâncias de 0 para todos os vértices: [0, 1, 2, 1, 2, 3].

7. Aplicações em Engenharia de Computação

- **Roteamento em Redes:** BFS encontra o caminho com menor número de saltos (hops) entre roteadores.
- **Sistemas de Navegação:** Em mapas onde todas as estradas têm o mesmo custo, BFS encontra o caminho com menor número de curvas.
- **Análise de Redes Sociais:** Encontrar o grau de separação entre duas pessoas (número de conexões intermediárias).
- **Algoritmos de Fluxo:** Edmonds-Karp usa BFS para encontrar caminhos aumentantes em grafos de fluxo.

8. Exercícios

8.1. Exercícios Conceituais

1 Por que BFS garante encontrar o caminho mínimo em grafos não ponderados? Use argumentos sobre a ordem de exploração dos vértices.

2 Compare BFS com DFS em termos de estrutura de dados usada, ordem de visita e aplicações típicas.

3 Explique o conceito de Multi-Source BFS. Dê um exemplo prático onde essa técnica é útil.

8.2. Exercícios Analíticos

1 Simule BFS passo a passo no seguinte grafo direcionado a partir do vértice 0:

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

Mostre a fila em cada iteração e as distâncias calculadas.

2 Para um grafo em grade (grid) $N \times N$, calcule a complexidade de BFS para encontrar o caminho mínimo entre duas células. Considere que cada célula tem no máximo 4 vizinhos.

3 Analise o espaço usado por BFS em termos de V e E. Compare com DFS recursiva e DFS iterativa.

8.3. Exercícios de Programação

1 Implemente BFS completa em Java com:

- Cálculo de distâncias mínimas.
- Reconstrução de caminhos usando array de pais.
- Detecção de componentes conexos.
- Suporte a grafos direcionados e não direcionados.

2 Resolva problemas de labirinto usando BFS:

- Dado um grid com células livres e paredes, encontre o caminho mínimo da entrada à saída.
- Visualize o caminho encontrado.

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

- **Beecrowd 1194:** Pre, In e Post.
- Problemas de menor caminho em grafos não ponderados.
- Problemas de conectividade e componentes conexos.