

水面に広がる波紋のように

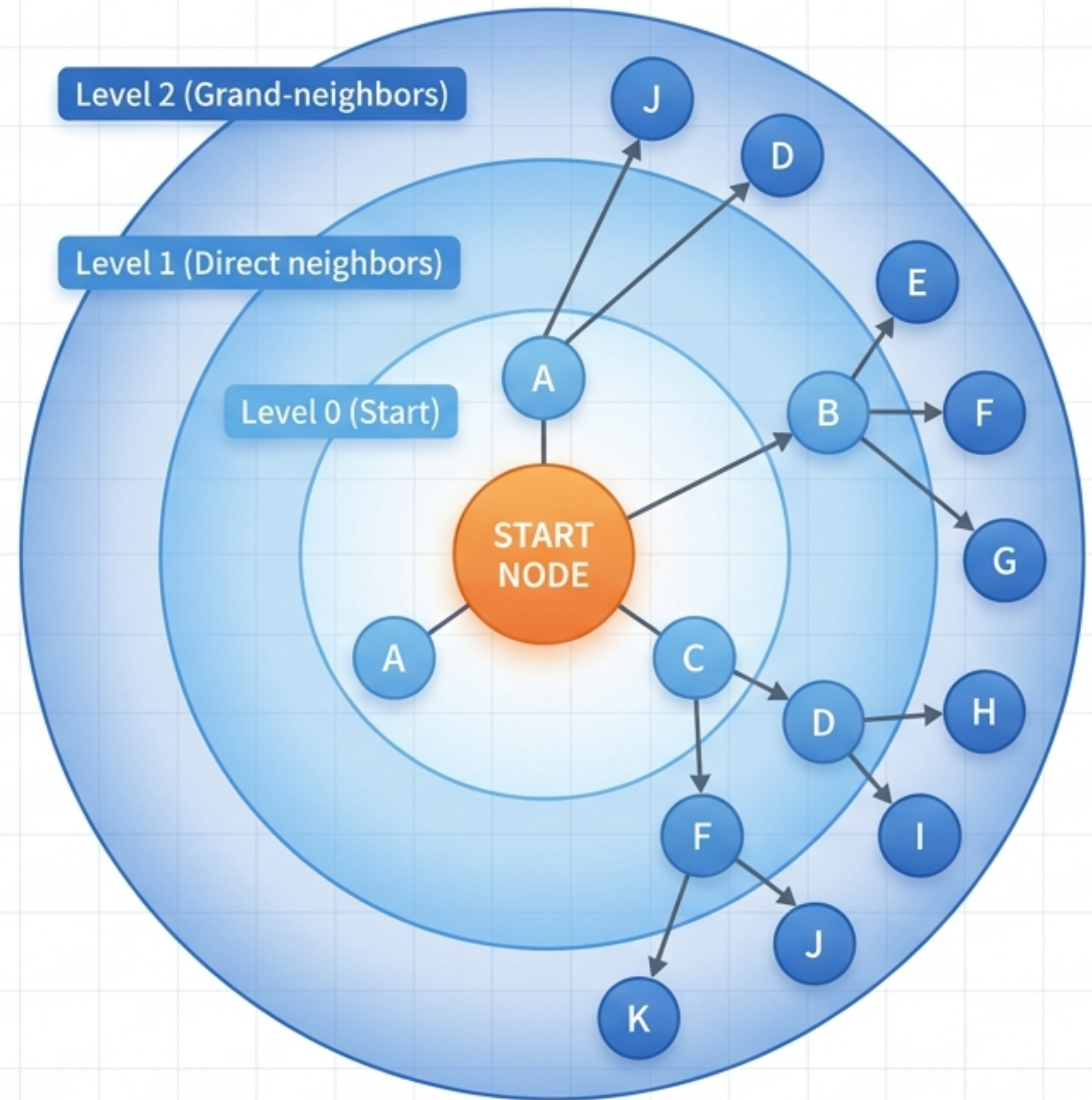
幅優先探索（BFS）を直感で完全に理解する



近いところから、 面で塗りつぶす。

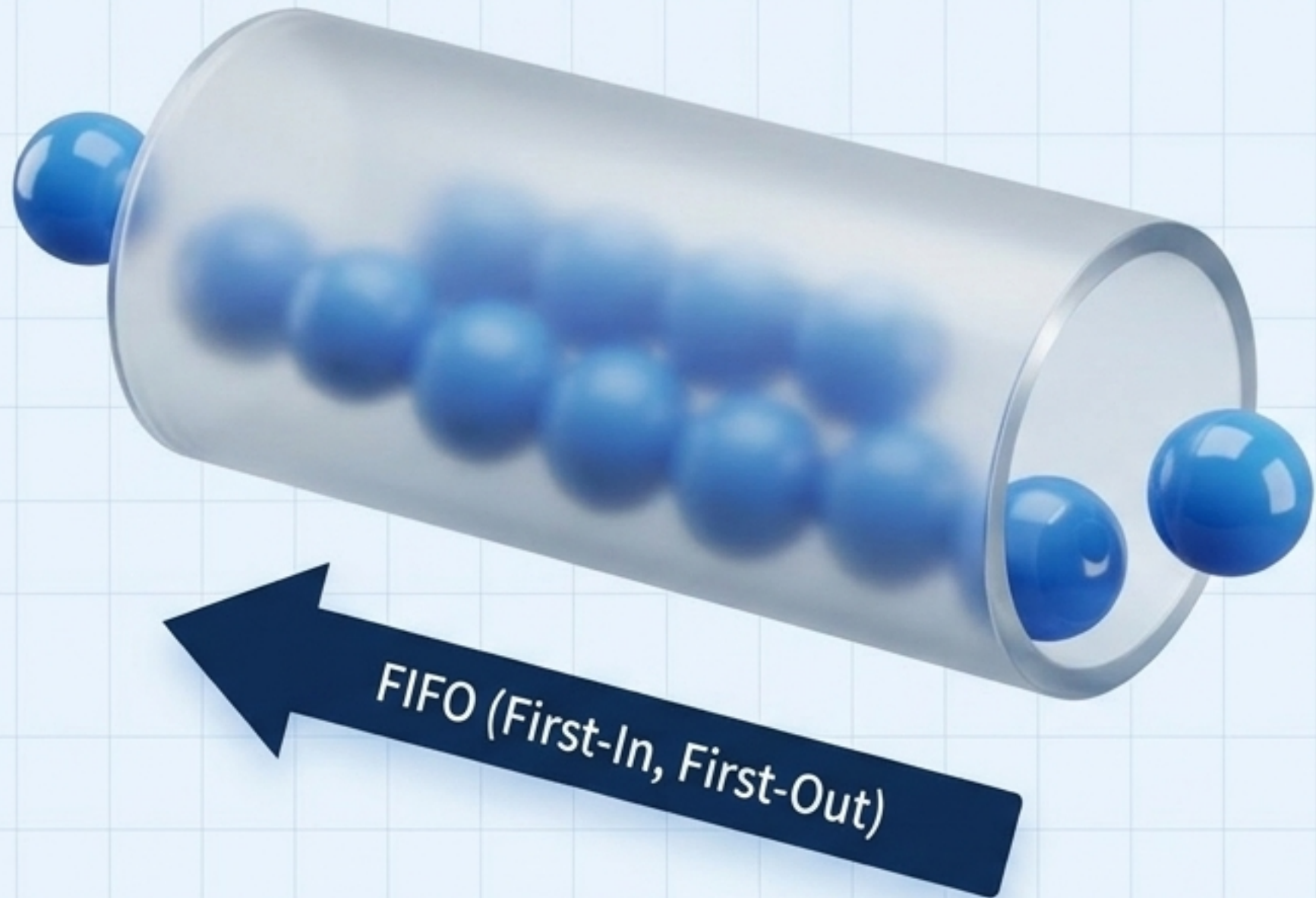
幅優先探索（BFS）の本質は「横型」の探索です。開始点から近いノード（節点）を優先的に調べます。水面に石を投げ入れたとき、波紋が同心な円状に一層ずつ広がっていくように、探索の「層」を外側へと拡大していきます。

距離が近いノード群から順に探索が完了するため、エッジの重みが等しいグラフでは「最短経路」を確実に見つけることができます。



波紋を動かすエンジン： 「キュー」

BFSのメカニズムを駆動するのは、「キュー (Queue)」というデータ構造です。



ルール: 先入れ先出し (FIFO: First-In, First-Out)



動き: 列に並んだ順番通りに処理される、レジ待ちの列と同じ仕組み。

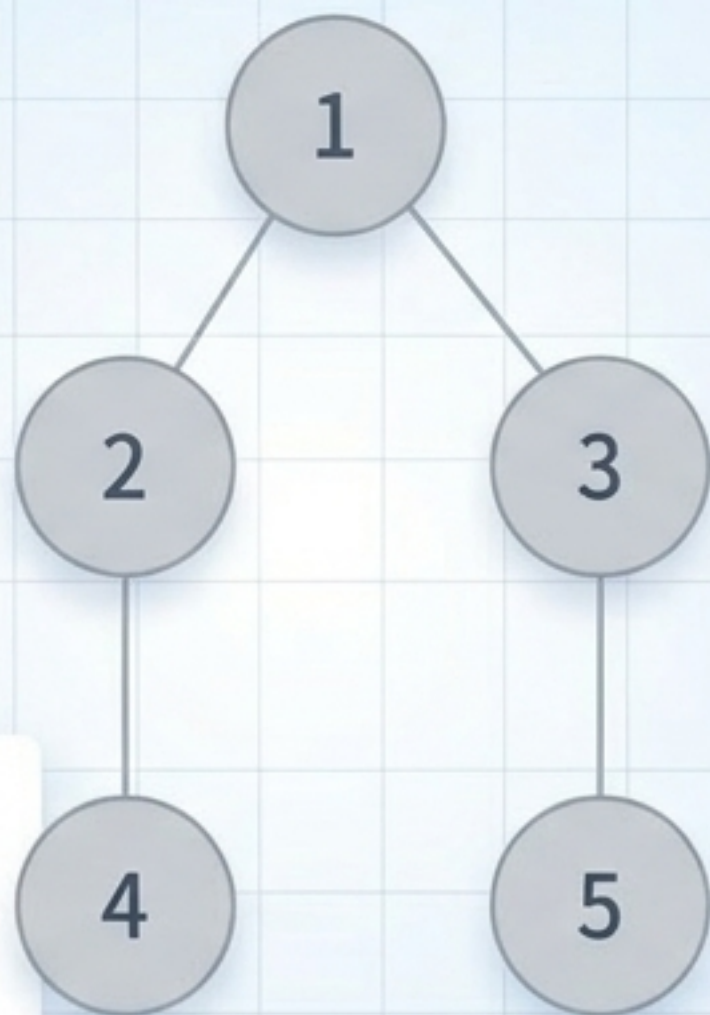


役割: 先に見つけた「近いノード」をキューに入れ、順番に取り出すことで、波紋のような階層ごとの探索を強制します。

探索ダッシュボード：3つの状態の連動

アルゴリズムが動くとき、頭の中では「グラフ」「キュー」「訪問済みリスト」の3つが連動して変化します。
このダッシュボードで、状態遷移をコマ送りで追跡しましょう。

Left Pane



Color Legend

- 未訪問 (Unvisited)
- 探索中/キュー内 (Queued)
- 訪問済み (Visited)

キュー (Queue)



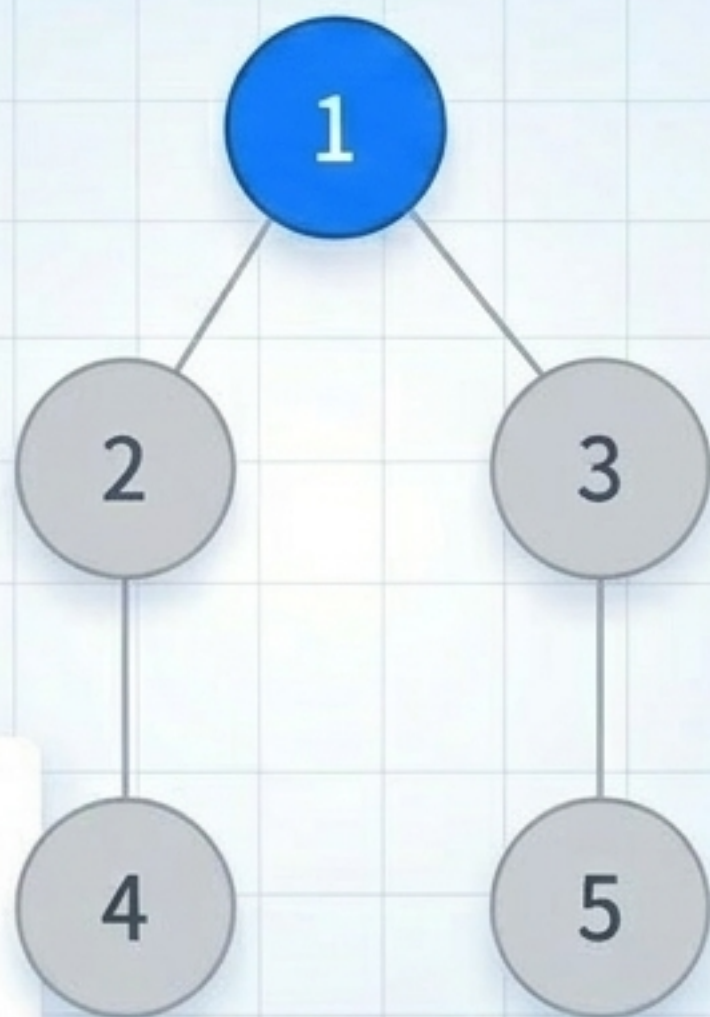
訪問済みリスト (Visited)



Step 1: スタート地点をキューに入れる

1. 根 (Root) である Node 1 をスタート地点に設定。
2. Node 1 をキューに追加。
3. 同時に、同じ場所を二度調べないよう Node 1 を「訪問済み」にマーク。

Left Pane



Color Legend

- 未訪問 (Unvisited)
- 探索中/キュー内 (Queued)
- 訪問済み (Visited)

キュー (Queue)



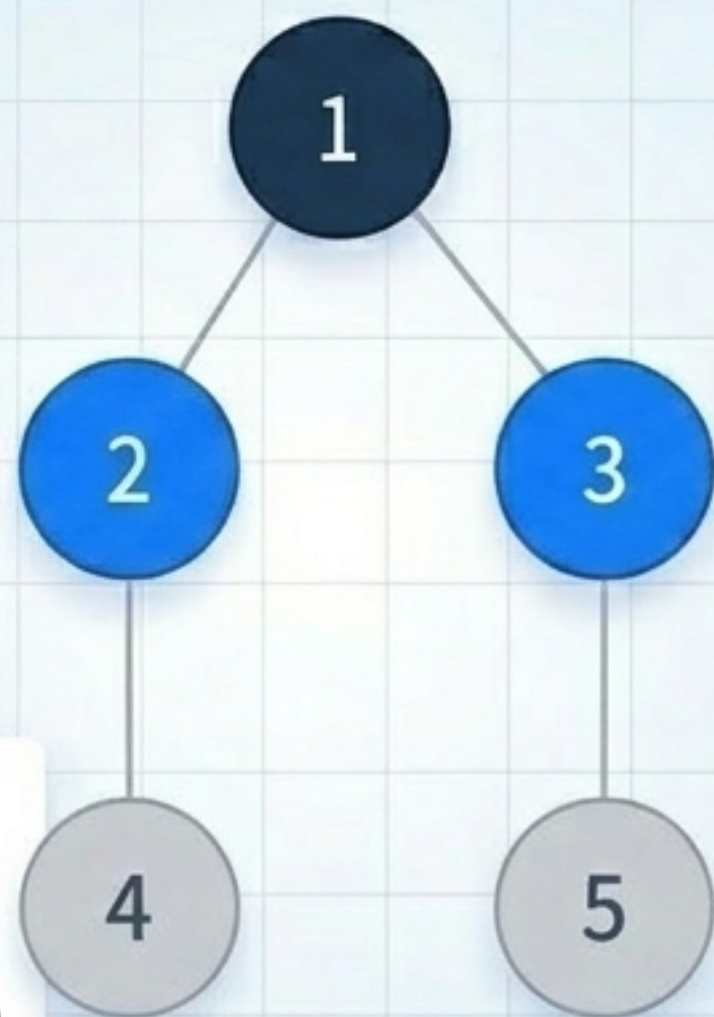
訪問済みリスト (Visited)

☒ Node 1

Step 2: 先頭を取り出し、隣接ノードをキューへ（深さ1）

1. キューの先頭から Node 1 を取り出す（調査開始）。
2. Node 1 に隣接する未訪問ノード Node 2, Node 3 を見つける。
3. Node 2, Node 3 をキューの末尾に追加し、訪問済みにする。

Left Pane



Color Legend

- 未訪問 (Unvisited)
- 探索中/キュー内 (Queued)
- 訪問済み (Visited)

キュー (Queue)



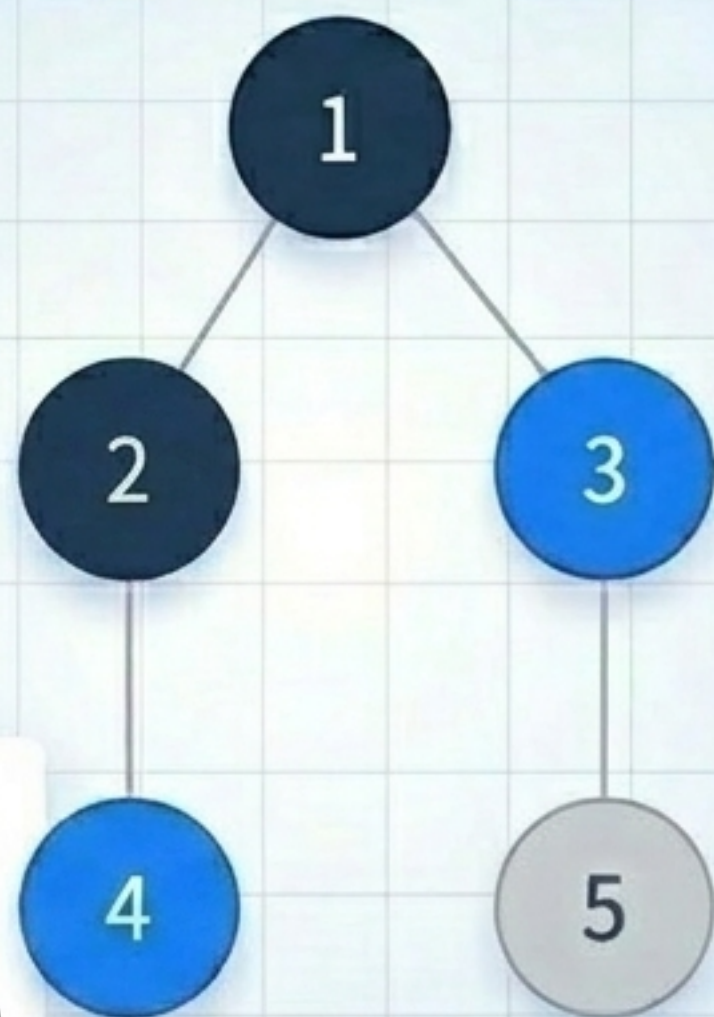
訪問済みリスト (Visited)

- ☒ Node 1
- ☒ Node 2
- ☒ Node 3

Step 3: 次の先頭を取り出す (深さ2への波及)

1. キューの先頭から Node 2 を取り出す。
 2. Node 2 に隣接する未訪問ノード Node 4 をキューに追加し、訪問済みにする。
- ※注意: 次に処理されるのはNode 4ではなく、前から並んでいる Node 3 です。これが波紋状に広がる理由です。

Left Pane



Color Legend

- 未訪問 (Unvisited)
- 探索中/キュー内 (Queued)
- 訪問済み (Visited)

キュー (Queue)



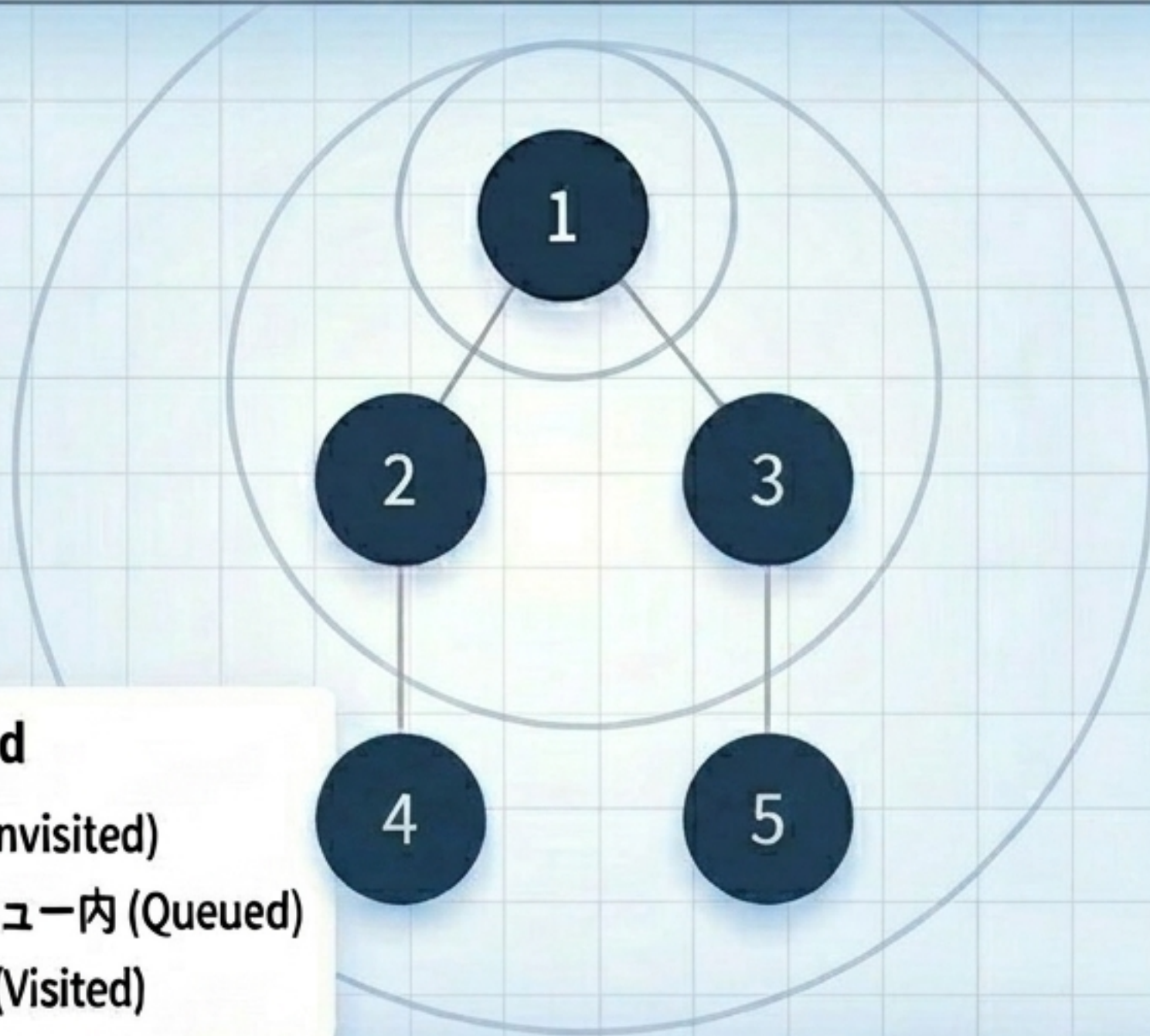
訪問済みリスト (Visited)

- ☒ Node 1
- ☒ Node 2
- ☒ Node 3
- ☒ Node 4

Step 4: 波紋が端まで到達し、探索完了

1. Node 3 が取り出され、隣接する Node 5 がキューに入る。
2. 最後に Node 4, Node 5 が順次取り出されるが、未訪問の隣接ノードはもう存在しない。
3. キューが空になり、アルゴリズムが終了。最短経路が特定されました。

Left Pane



Color Legend

- 未訪問 (Unvisited)
- 探索中/キュー内 (Queued)
- 訪問済み (Visited)

キュー (Queue)



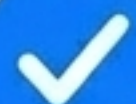
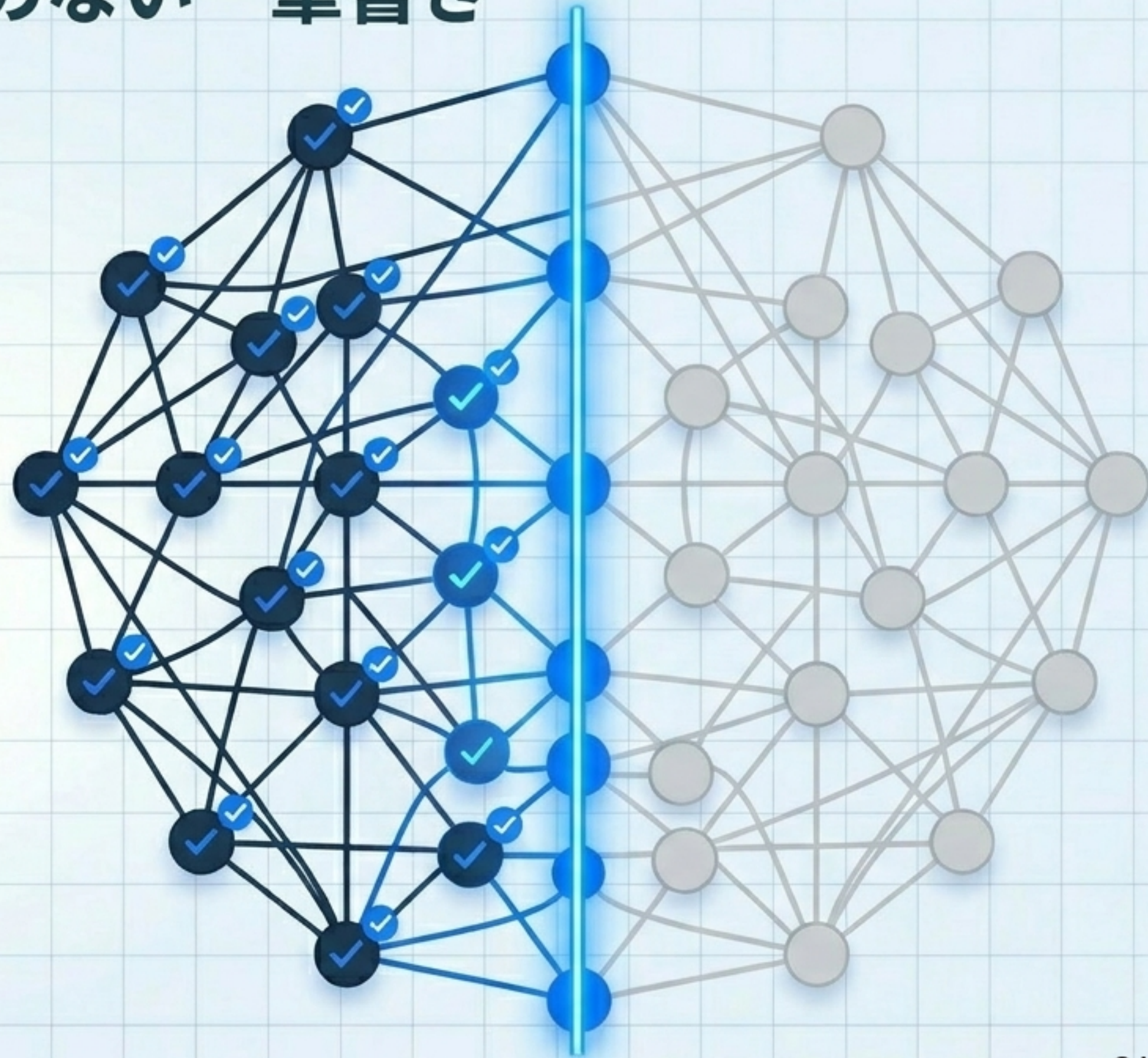
訪問済みリスト (Visited)

- ☒ Node 1
- ☒ Node 2
- ☒ Node 3
- ☒ Node 4
- ☒ Node 5

計算量 $O(n+m)$: 無駄のない一筆書き

アルゴリズムの実行時間は
 $O(n+m)$ と表されます。

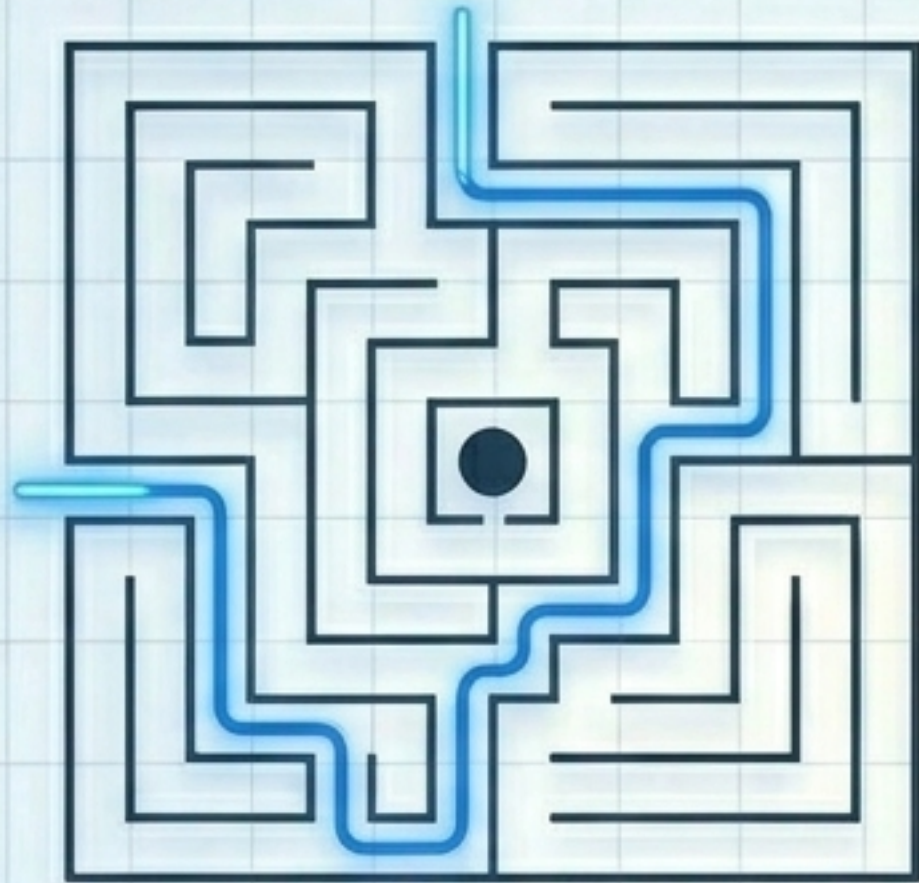
(n =ノード数、 m =エッジ数)



これは単なる数式ではありません。「キューを使って訪問済みを管理することで、すべてのノードを最大1回、すべてのエッジを最大1回しか通過しない」という、アルゴリズムの極めて高い効率性を証明しています。

特性と評価：BFSのトレードオフ

メリット：強み



最短経路の保証

エッジの重みが均等の場合、スタートから各ノードへの最短距離（迷路の最短脱出ルートなど）を確実に見つけ出します。

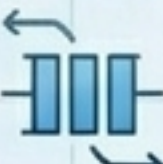
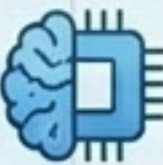
デメリット：弱み



メモリ消費量が多い

波紋の「外周」にあるすべてのノードをキューに記憶し続ける必要があるため、グラフが巨大になるとメモリを激しく消費します。

探索アルゴリズムの全体像：BFS vs DFS

	BFS（幅優先探索） 	DFS（深さ優先探索） 
 探索方向	横型（波紋状に面で広がる）	縦型（行けるところまで深く掘る）
 使用データ構造	キュー（FIFO）	スタック（LIFO / 再帰）
 メモリ消費	多い（探索中の階層を全記憶）	少ない（現在の経路のみ記憶）
 適した用途	最短経路の探索、迷路の最短解答	経路の網羅的探索、パズルの全解探索



迷ったら、近いところから面で探せ。

キュー（FIFO）というシンプルな筒が、「波紋」という確実で美しい探索の広がりを生み出します。
メモリと引き換えに最短経路を約束する、それが幅優先探索（BFS）の本質です。