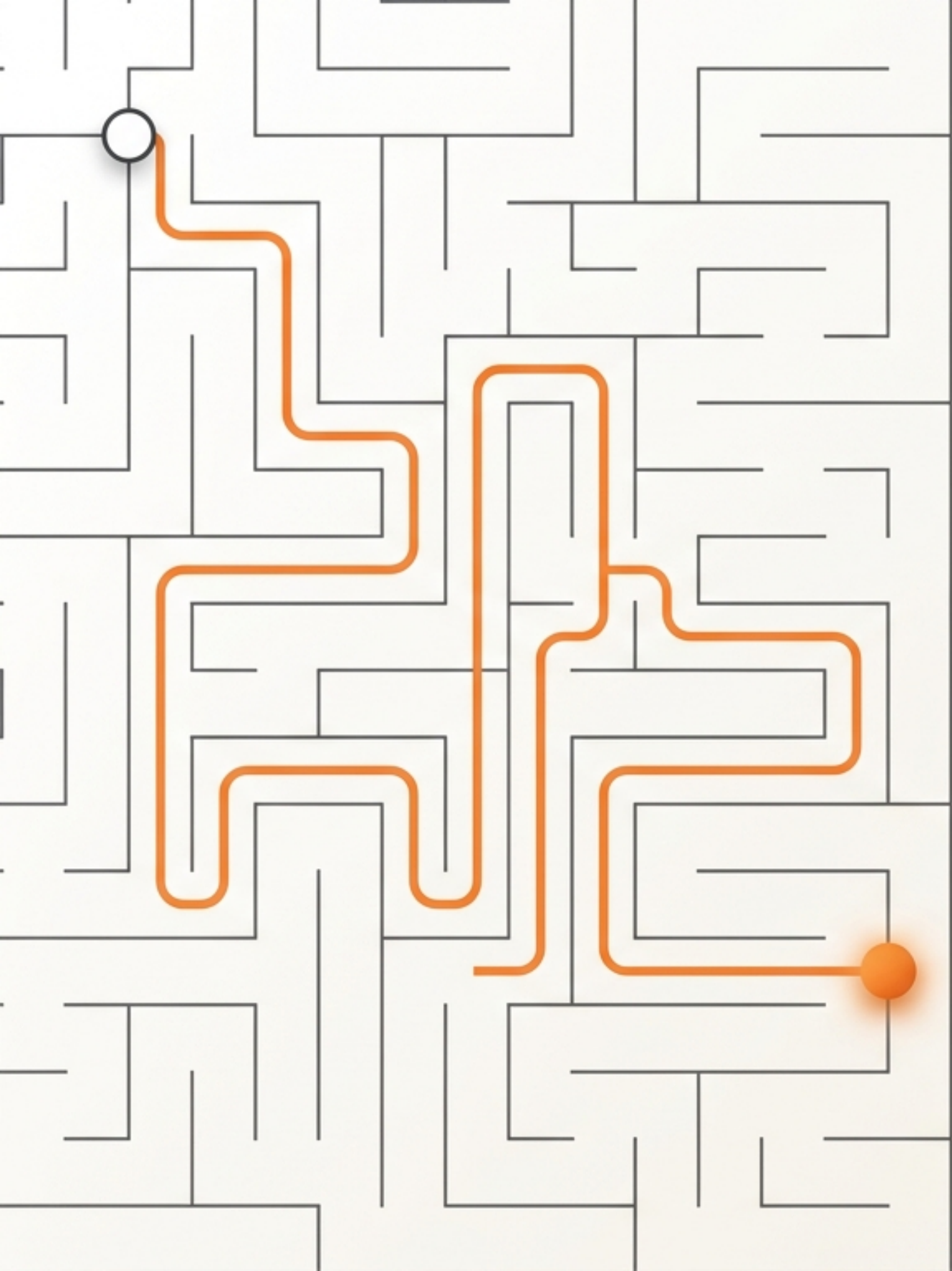
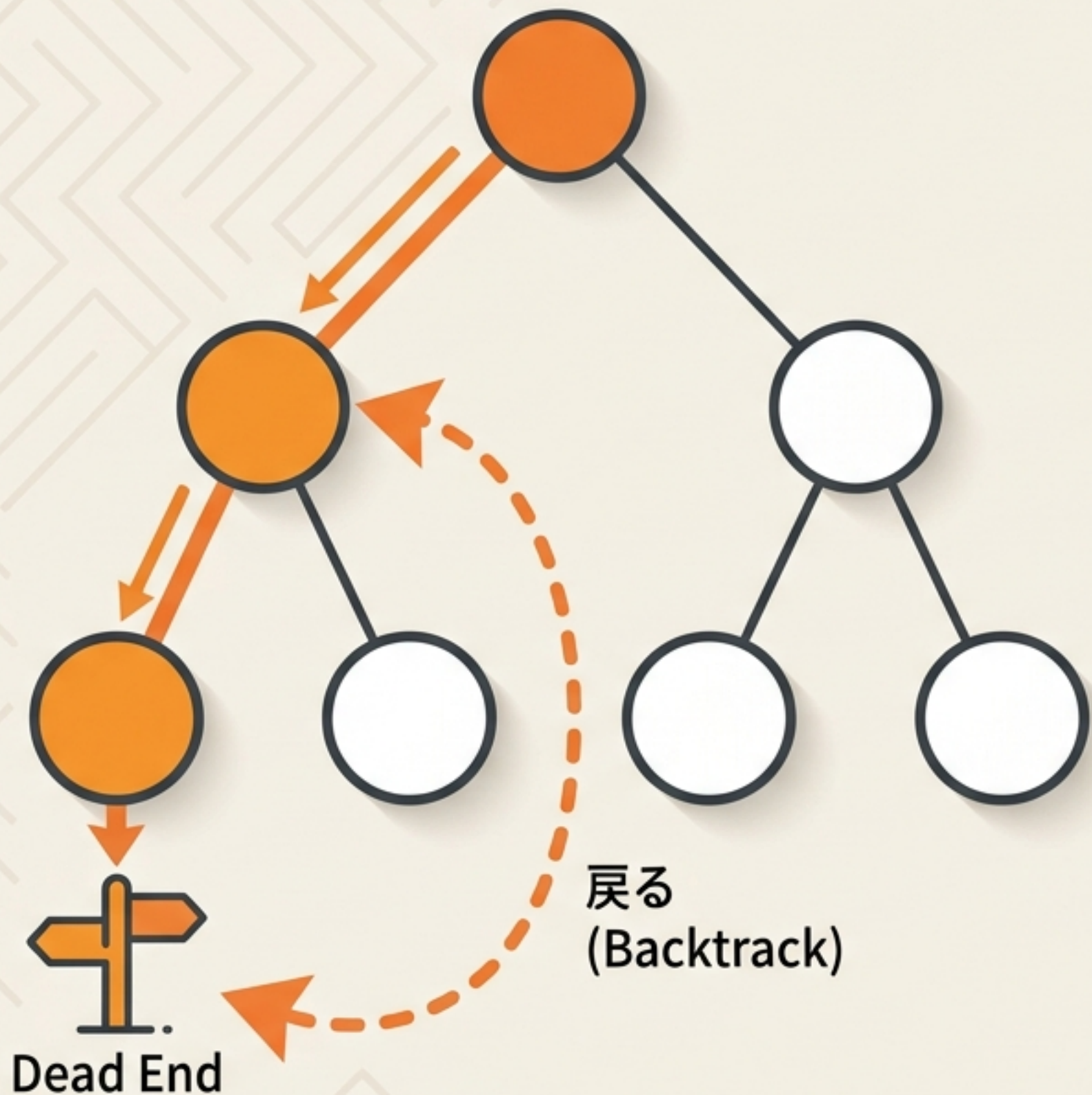




深さ優先探索 (DFS) 入門

行き止まりまでとことん進み、ダメなら戻る。
アルゴリズムの動きを直感的に理解する。

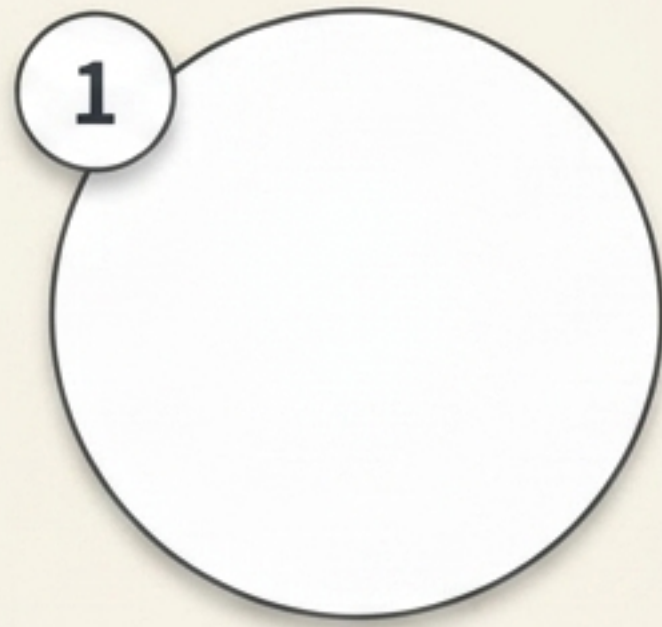


DFSとは何か？

- グラフや木構造を探索するための最も基本的な手法の一つ。
- ルールの核心:
 1. いけるとところまで「深く」潜り続ける。
 2. 行き止まりにぶつかったら、別の道がある分岐点まで「戻る（バックトラック）」。
- 迷路の中でペンを持って、壁にぶつかるまで一筆書きで進むようなプロセス。

探索のルール：状態の3分類

DFSは、各ノードに「色」を塗ることで迷子になるのを防ぎます。



White (未訪問)

まだ足を踏み入れていない
未知のノード。



Grey (探索中)

到達したが、そこから繋がる
先の探索がまだ終わってい
ないノード。

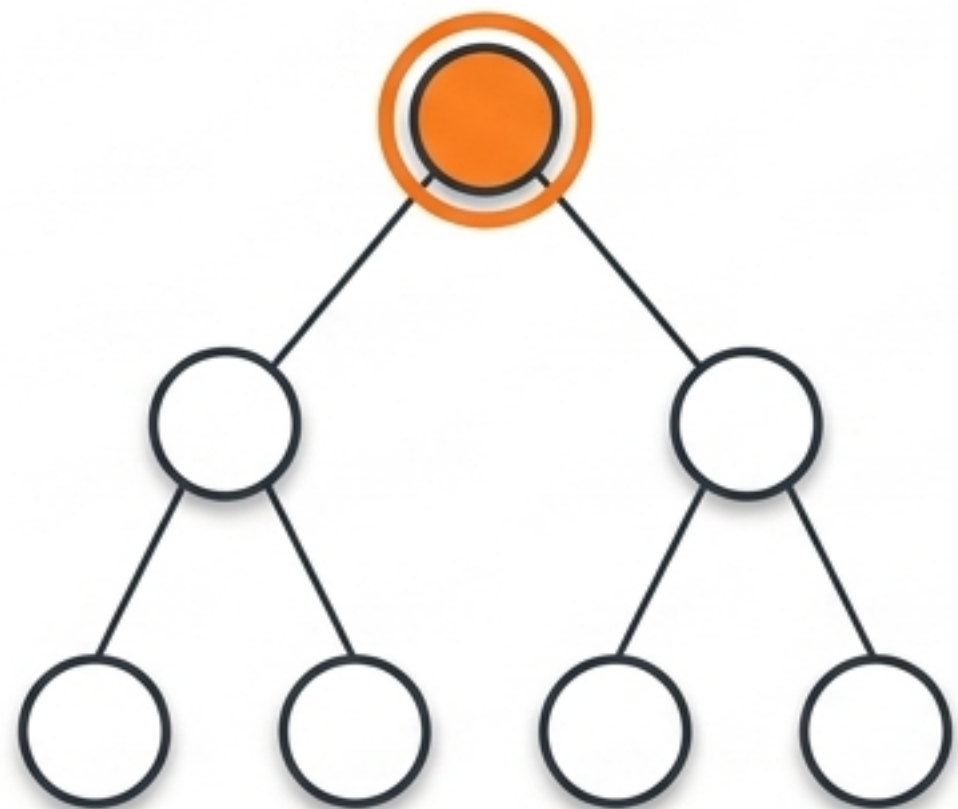


Black (完了)

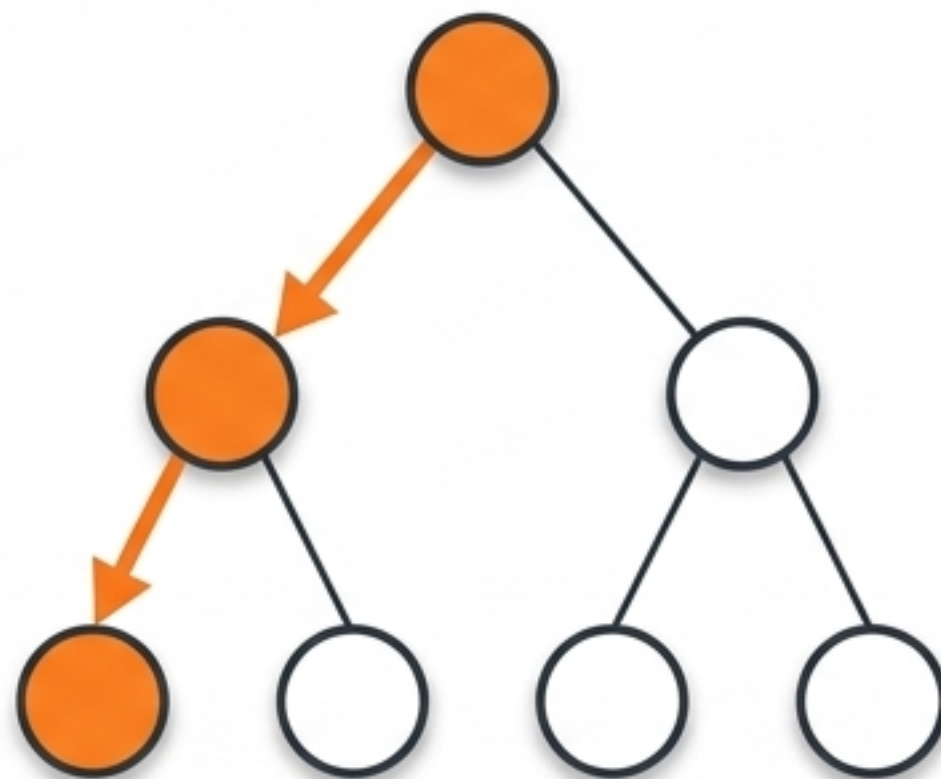
繋がるすべての道の探索が
完了し、「もう用済み」と
なったノード。

DFSの動きを追ってみよう（潜るフェーズ）

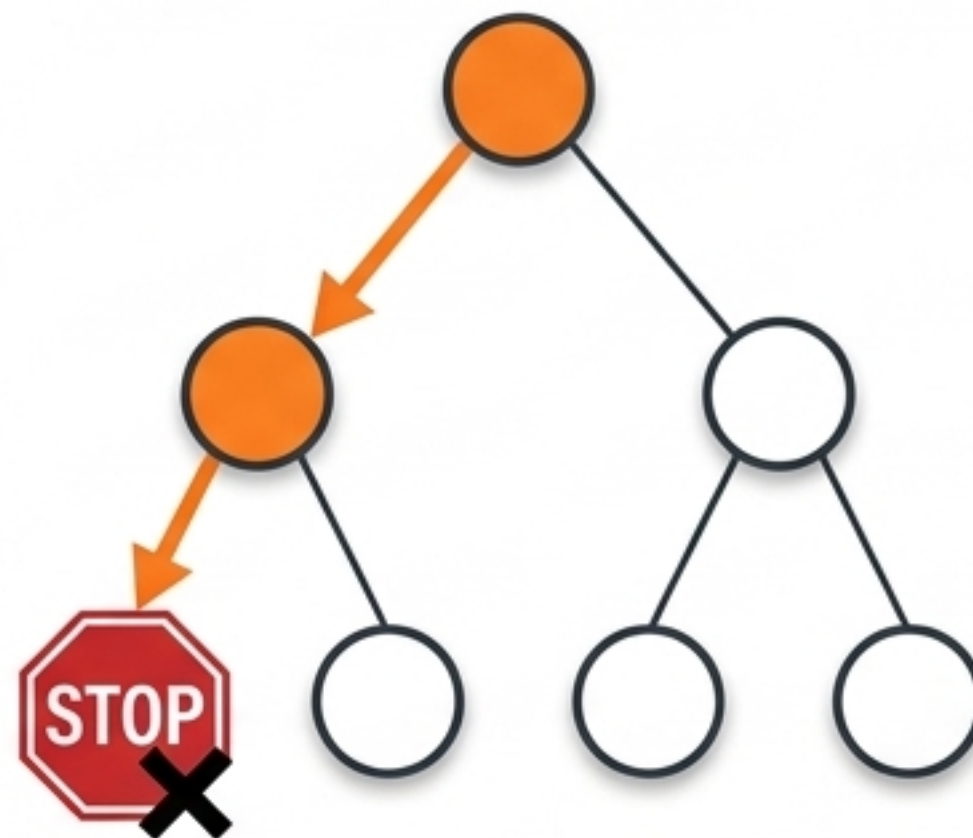
- 未訪問（White）のノードを見つけたら、とにかくそこへ進み、自分を探索中（Grey）にする。
- 分岐があっても、まずは一つのルートを徹底的に深掘りする。



Step 1



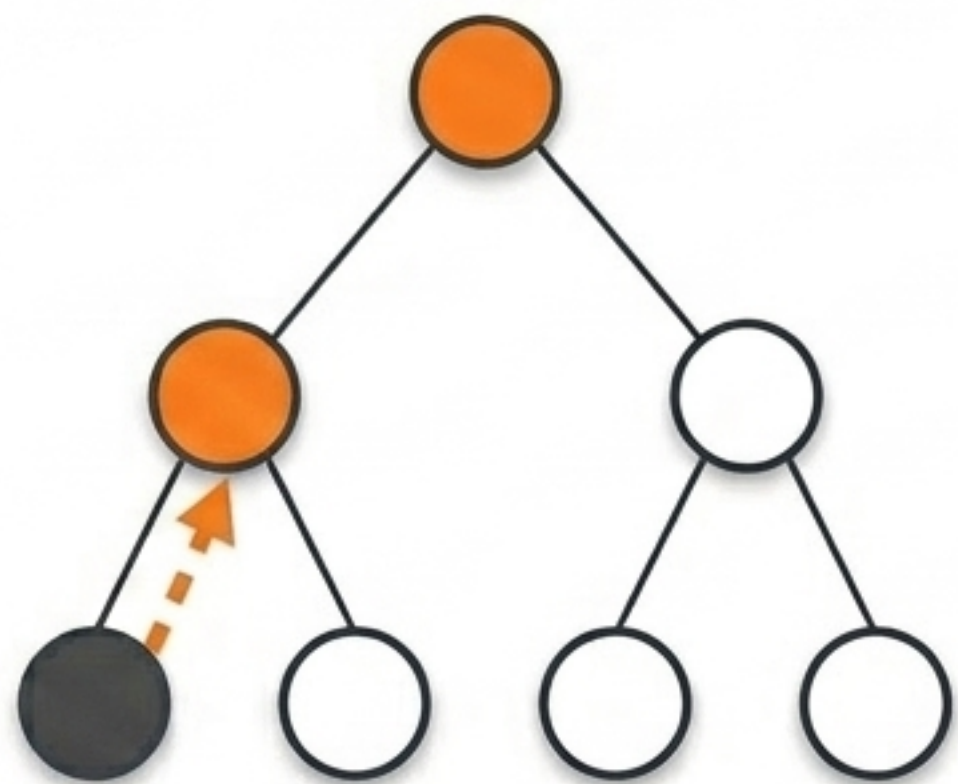
Step 2



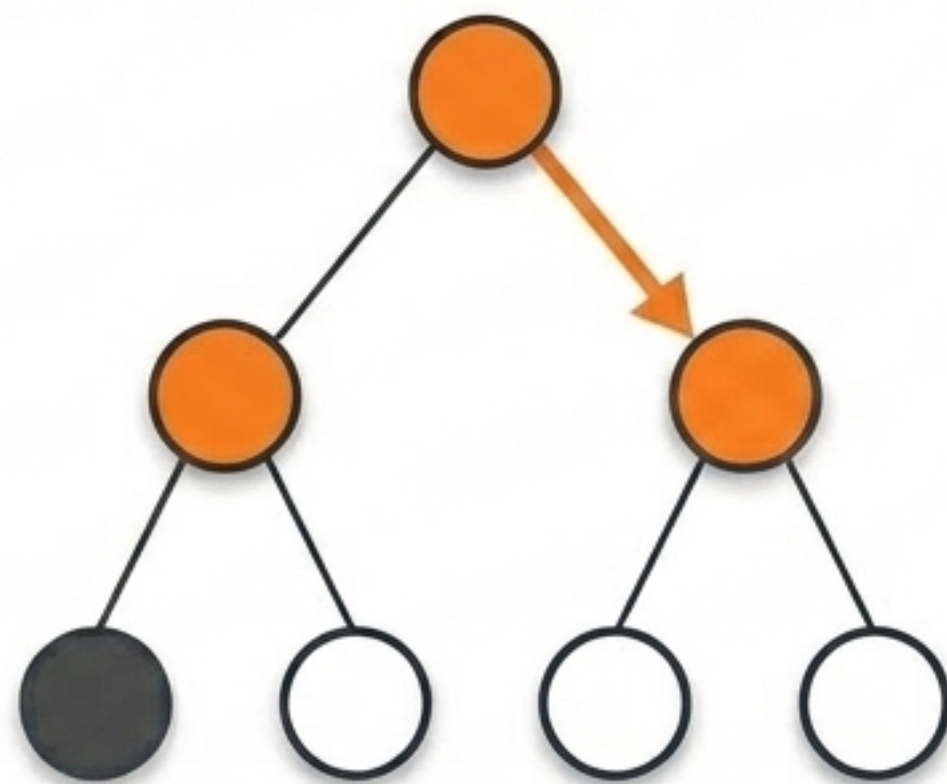
Step 3

DFSの動きを追ってみよう（戻るフェーズ）

- 行き先がなくなったら探索完了（Black）とし、一つ前のノードへ戻る（バックトラック）。
- Blackになったノードは二度と探索しない。これにより無駄な往復を防ぐ。
- 全ノードがBlackになれば探索全体の完了。計算量は $O(n+m)$ （頂点数+辺数）。



Step 4



Step 5

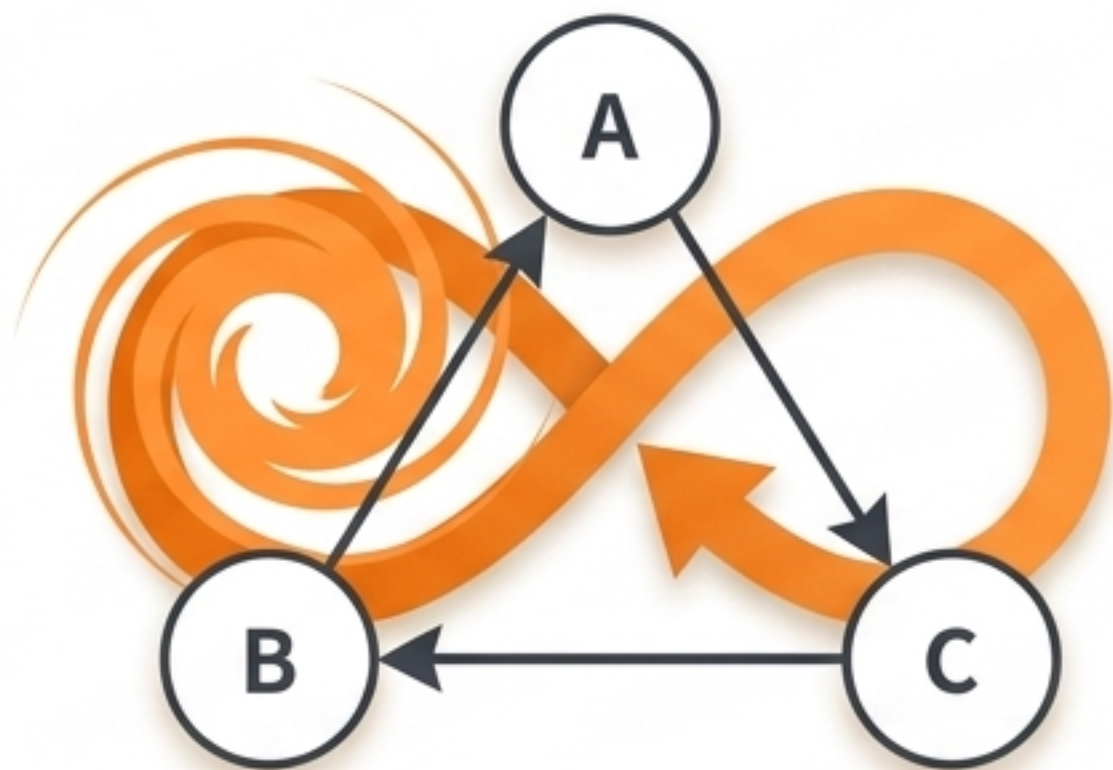


Step 6

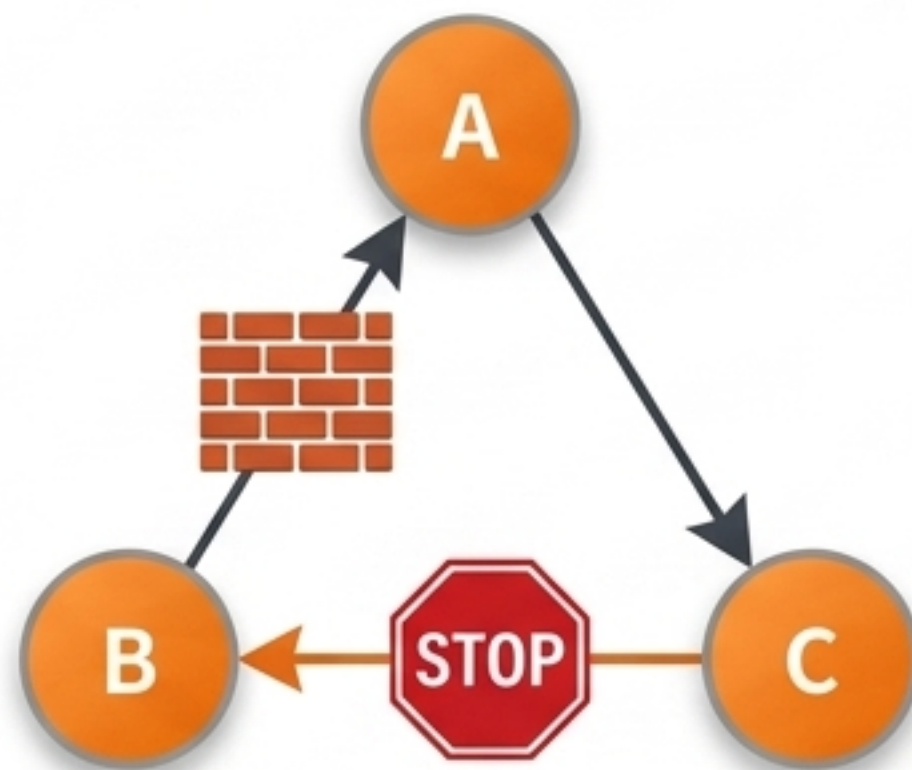
グラフ探索の落とし穴：循環の回避

- 木構造とは異なり、一般的なグラフには「循環（サイクル）」が存在します。
- 探索中にGreyの頂点に遭遇した場合、それは「自分が通ってきた道に戻ってきた」こと（循環）を意味します。
- 状態（色）を確認せずに進むと無限ループに陥ります。GreyやBlackのノードには進まないことが絶対のルールです。

状態管理なし（無限ループ）



状態管理あり（循環の検知）



実装アプローチ1：再帰 (The Recursive Way)

- 関数が「自分自身を呼び出す」ことで、OSのコールスタックを暗黙的に利用します。
- 「戻る」処理を自分で書かなくても、関数が終了すれば自然に一つ前のノードに戻ります。

DFS(u):

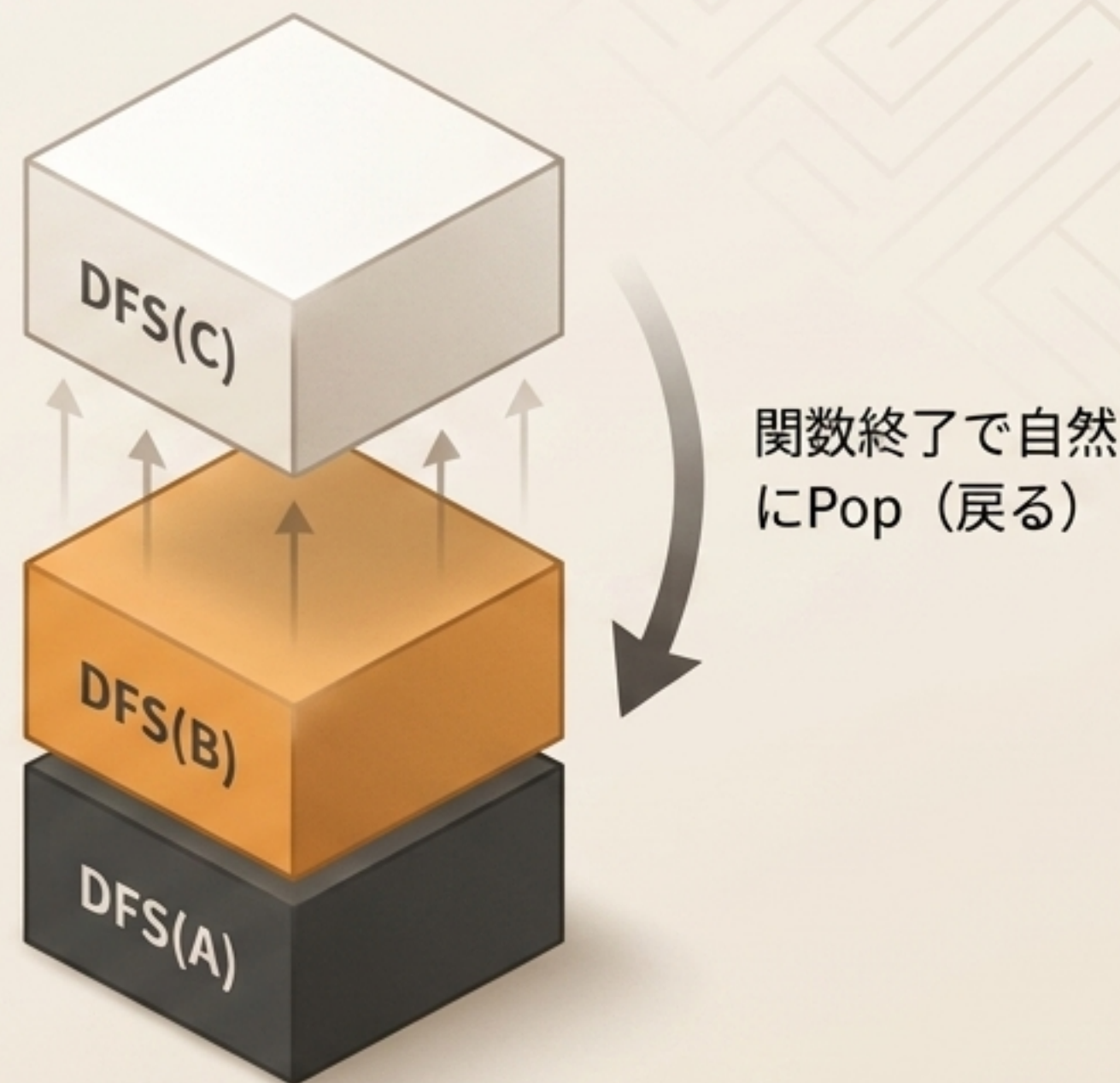
u を **Grey** (探索中) にする

for 各隣接ノード v:

if v が **White** なら:

DFS(v) // 自分自身を呼び出す

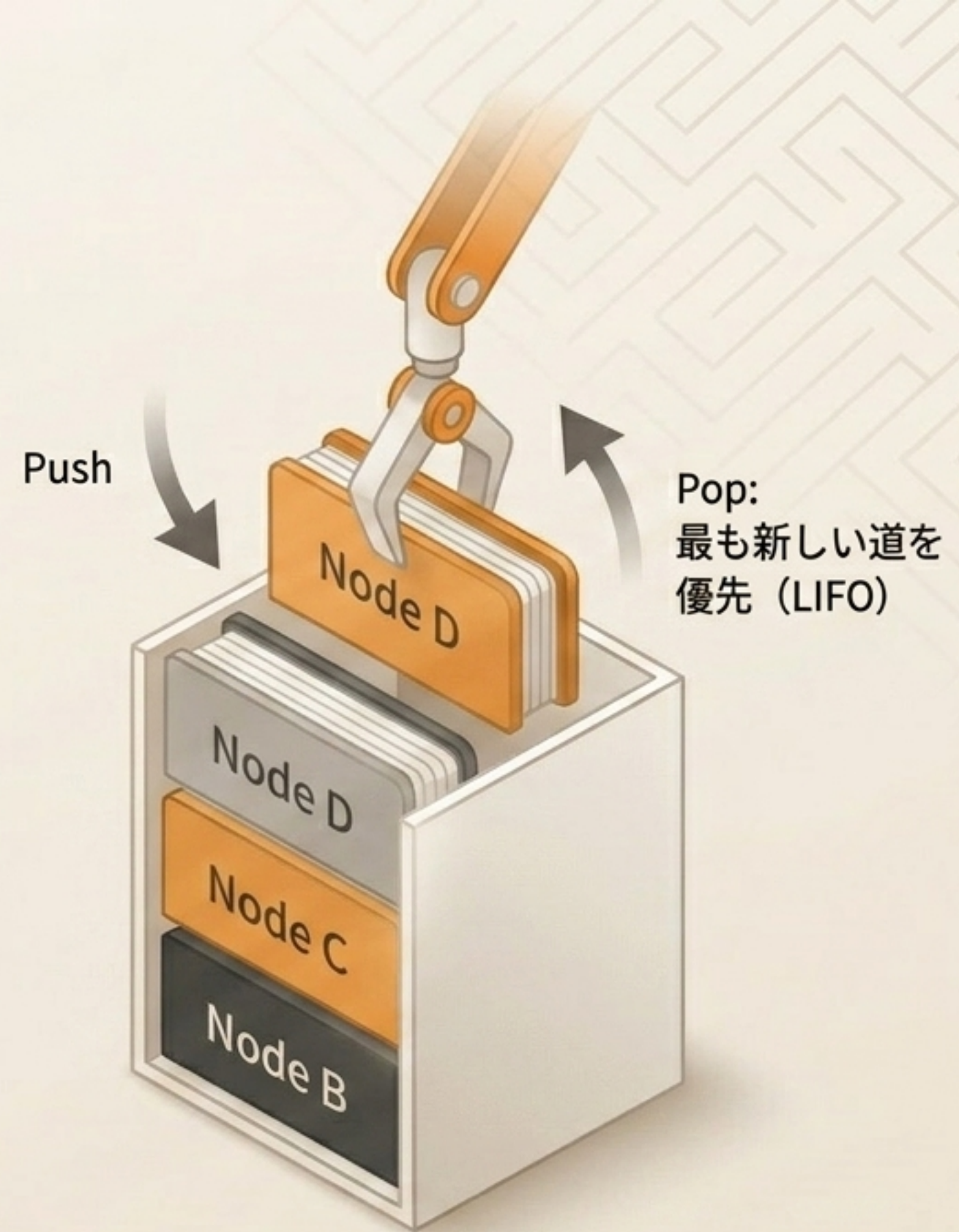
u を **Black** (完了) にする



実装アプローチ2：スタック (The Stack Way)

- 再帰を使わず、明示的なデータ構造「スタック」を用いて次に探索するノードを管理します。
- 後入れ先出し（LIFO）の性質により、追加されたばかりの「最も深い」ノードが常に次に取り出されます。
- キュー（FIFO）を使うと幅優先探索（BFS）になりますが、スタックに変えるだけで深さ優先探索（DFS）になります。

```
DFS(start):  
  Stack S を用意し、startをPush  
  while S が空でない:  
    u = S.Pop()  
    if u が White なら:  
      u を 訪問済みにする  
      for 各隣接ノード v:  
        S.Push(v)  // 次に行く候補を積む
```



比較マトリックス：再帰 vs スタック

	 再帰 (Recursive)	 スタック (Stack)
コードの シンプルさ	 非常にシンプルで直感的。 コード量が少ない。	 やや冗長。スタックの管理 コードが必要。
メモリ領域	コールスタック (OS) を使用。	ヒープメモリ (データ構造) を使用。
リスクと適性	 極めて深いグラフではスタックオーバーフローのリスクあり。	 深いグラフや巨大な木構造でも安全に動作する。

DFSは何に使えるのか？（応用例）



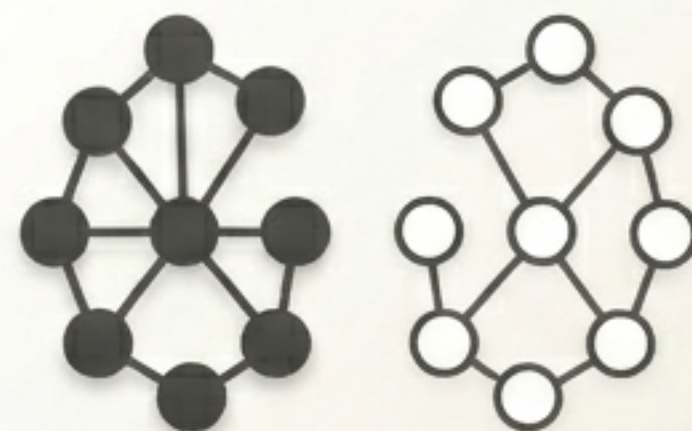
1. 経路探索とパズルの解法

迷路を抜け出す経路や、条件を満たす手順を見つけ出す（到達可能性の確認）。



2. 循環（サイクル）の検出

依存関係のループや、OSのデッドロックを検知する（Greyノードへの再到達で判別）。



3. 連結成分の発見

巨大なネットワークが、いくつかの独立したグループに分かれているかをラベル付けする。

まとめ (Key Takeaways)

1

行けるところまで深く潜る

DFSの基本は「行き止まりまで進み、ダメなら戻る（バックトラック）」。
迷路の壁伝いのような直感的な動き。

2

3つの色で状態を管理する

White（未訪問）、Grey（探索中）、Black（完了）。この状態管理が
無限ループを防ぎ、計算量 $O(n+m)$ の効率的な走査を可能にする。

3

LIFOが深さを生み出す

再帰の「コールスタック」も、明示的な「スタックデータ構造」も、後入れ
先出し（LIFO）の性質が「常に新しい道を優先する」DFSの挙動を支えている。