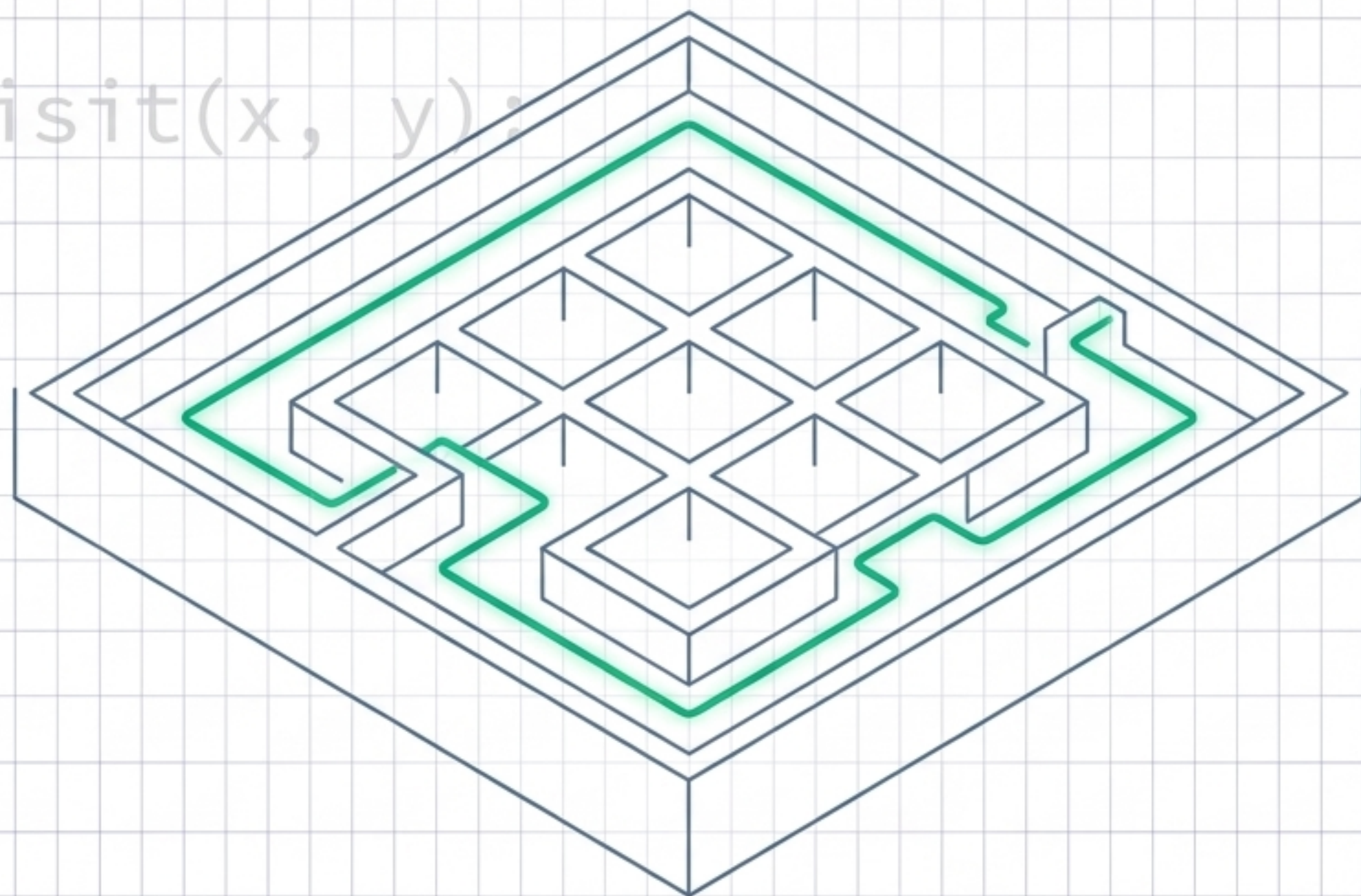


迷路の探索とバックトラッキングの極意

令和4年秋期 応用情報技術者試験 問3 — Pythonコードで解き明かす深さ優先探索(DFS)の全貌

```
def visit(x, y):
```



5×5の空間に潜む、アルゴリズムの挑戦

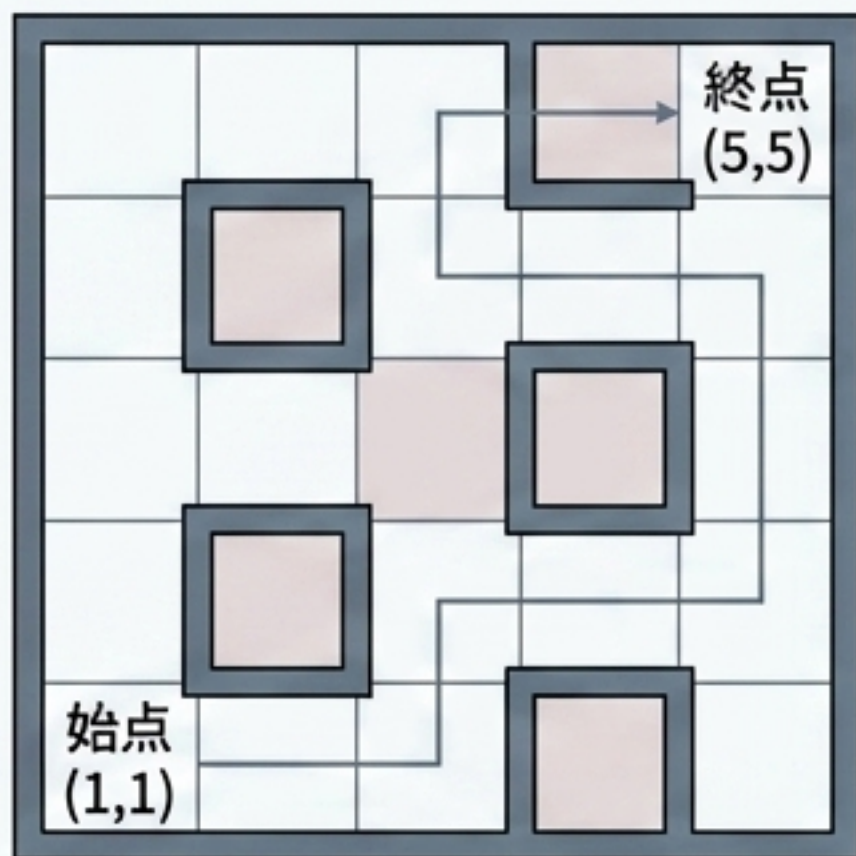


図1：解が1つの迷路

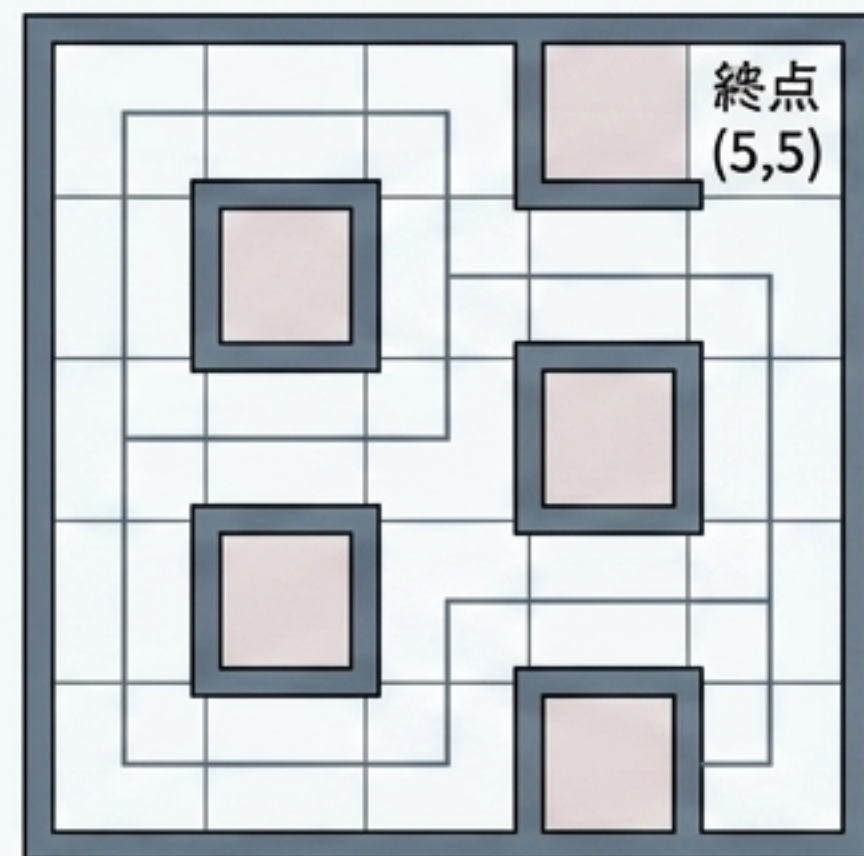
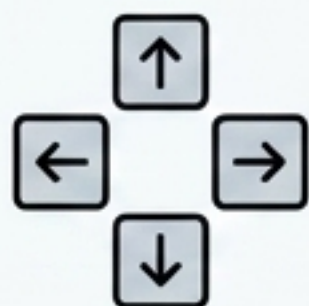
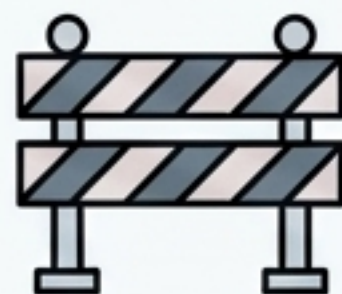


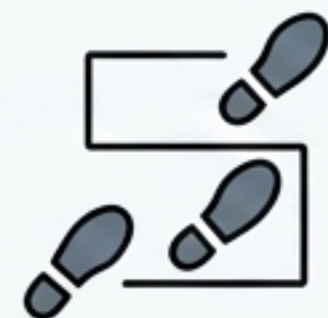
図2：解が複数（20個）ある迷路



1. 上下左右の4方向のみ
移動可能（斜め不可）

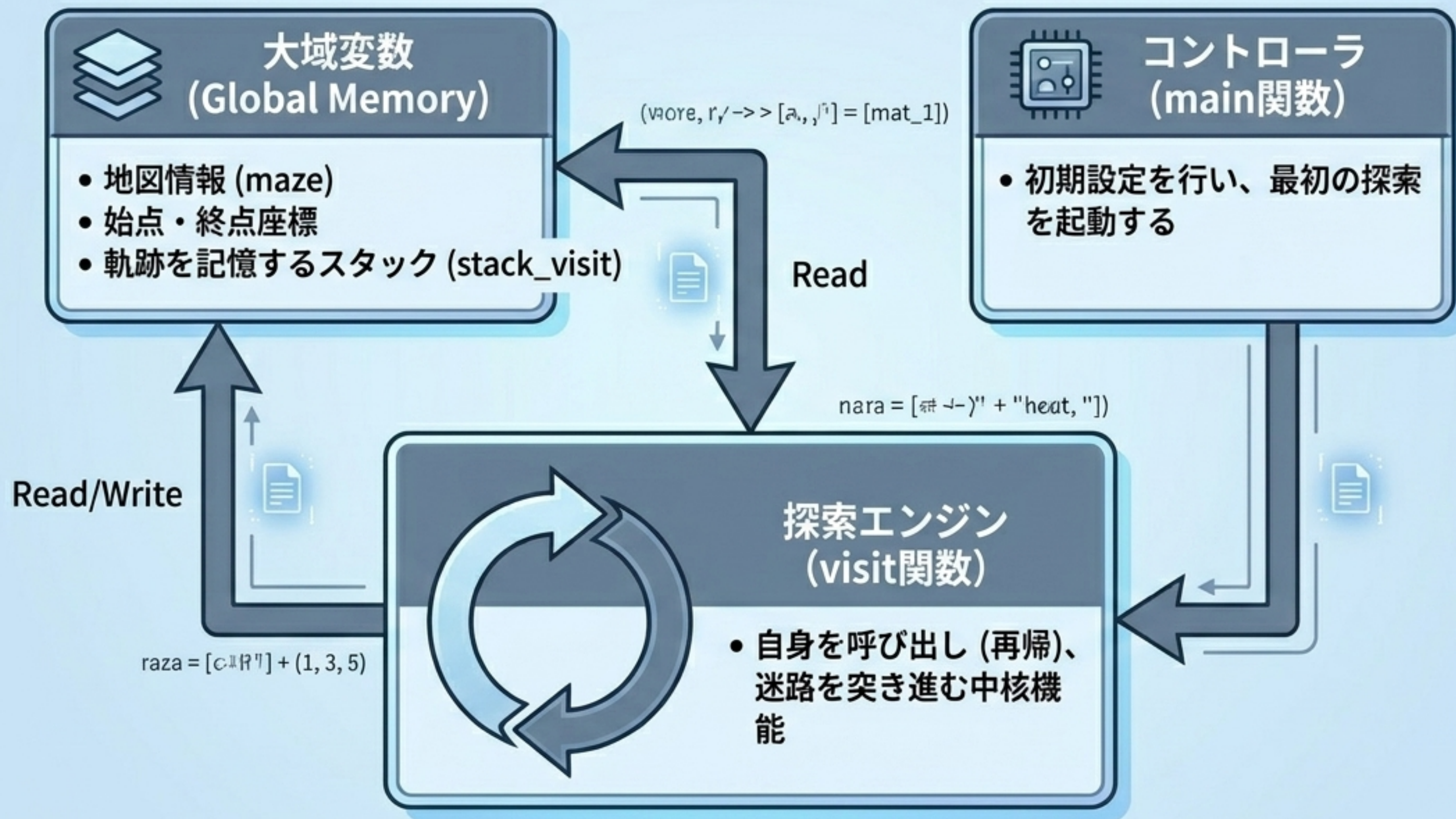


2. NGフラグ（壁・障害物）
は進入不可



3. 一度通ったマス（足跡）
は二度通れない

探索を司る3つのコンポーネント

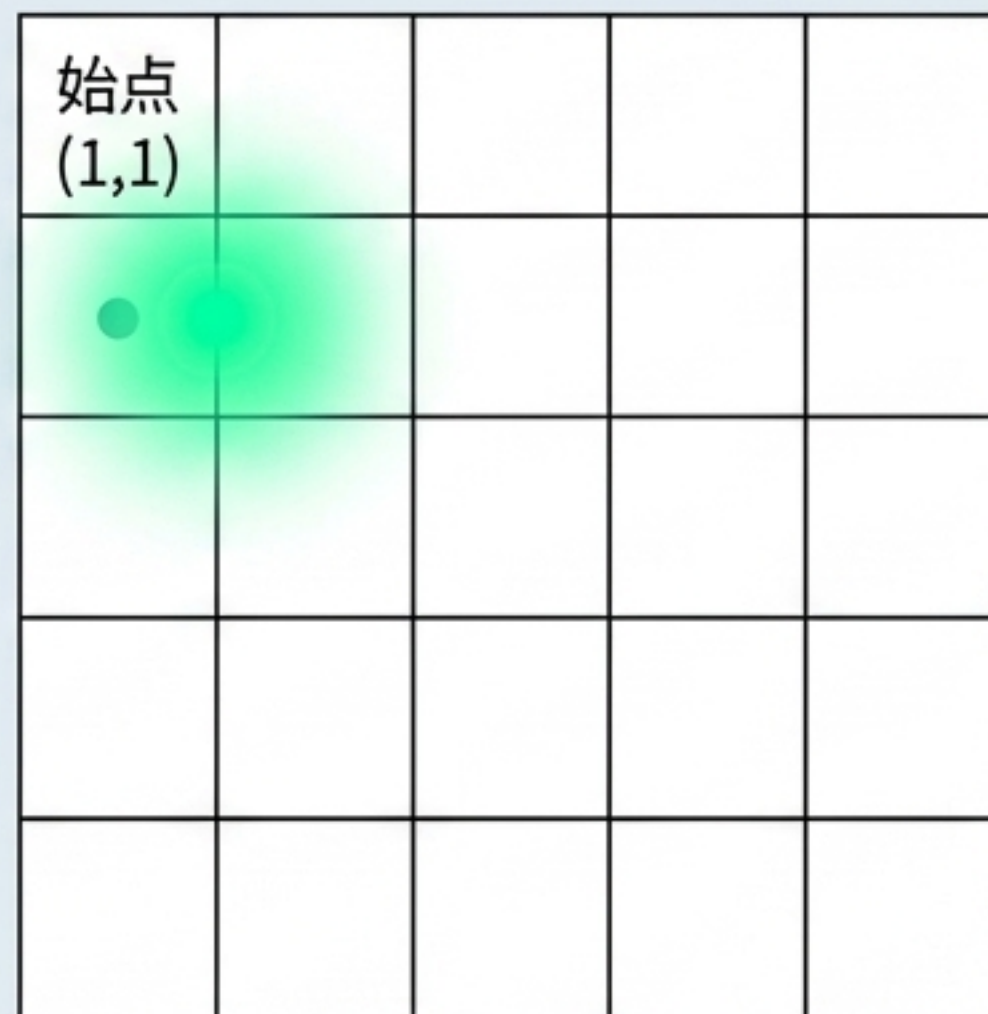


探索の幕開け

```
def main():  
    stack_top = 0  
    sol_num = 0
```

maze[x][y]に迷路図情報を設定する

```
visit(start_x, start_y)
```



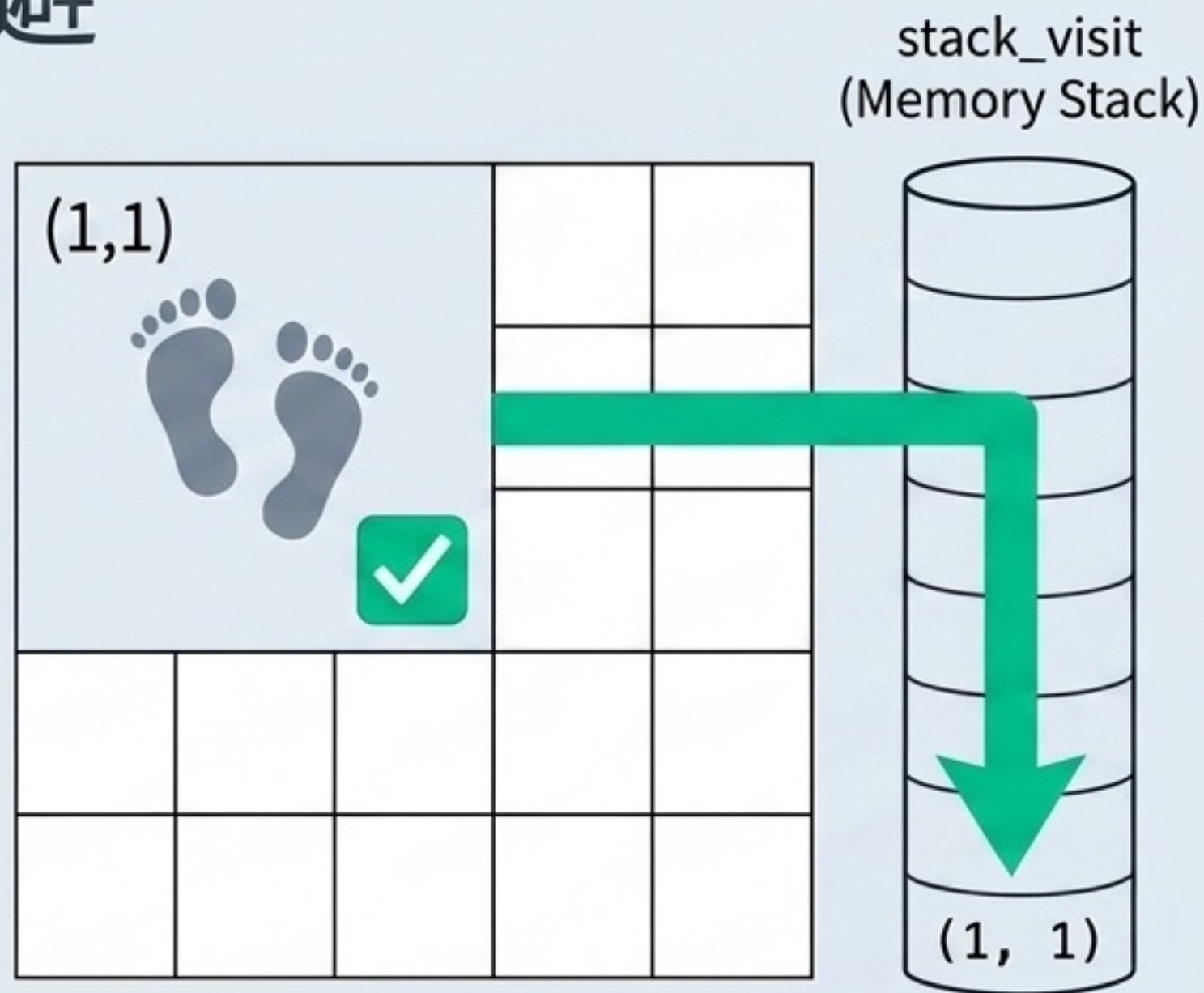
stack_visit
(Memory Stack)



すべてはここから始まる。
自己増殖的な再帰探索のスタート。

訪問の記録とスタックへの退避

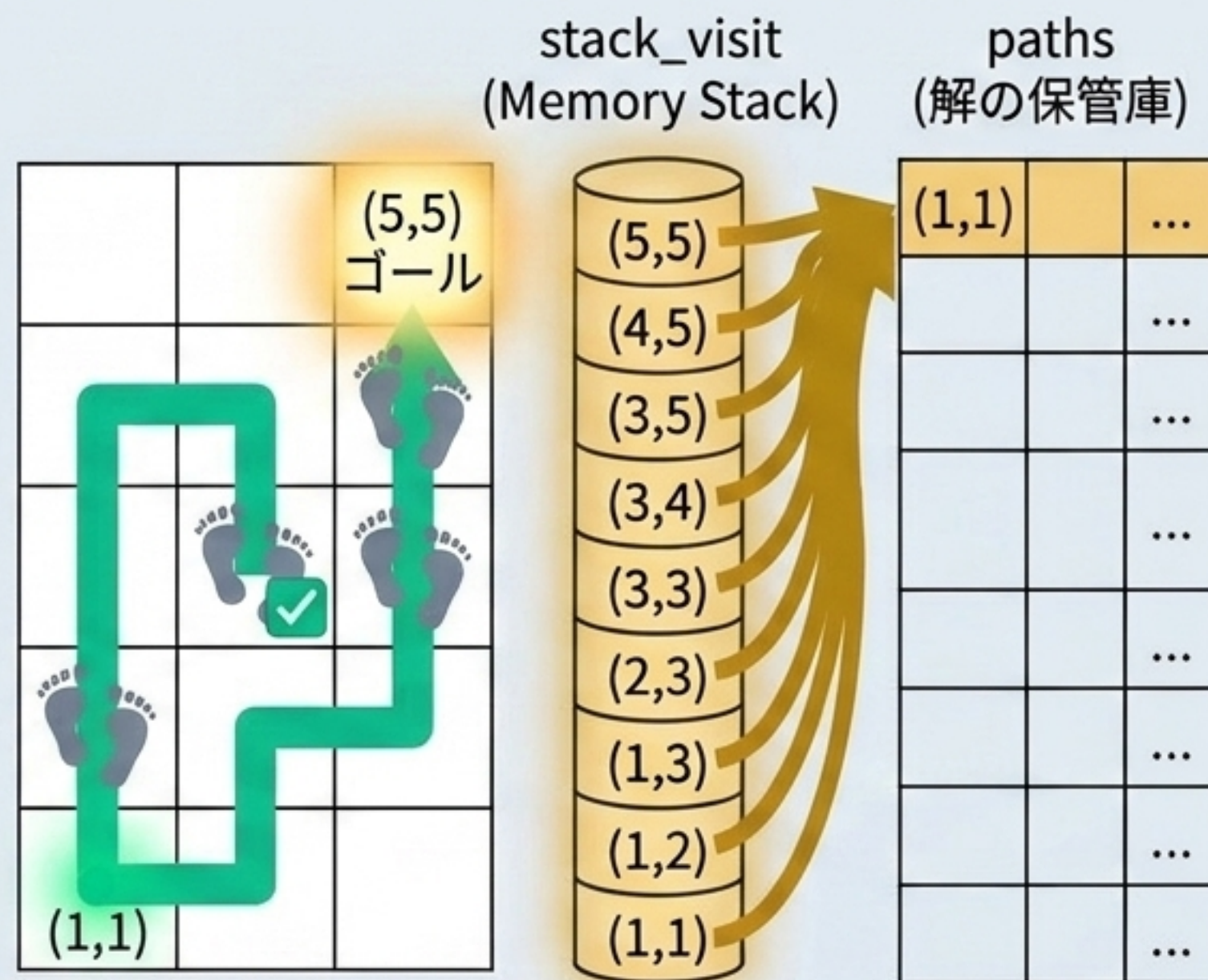
```
def visit(x, y):  
    maze[x][y] = VISITED  
    stack_visit[stack_top] = (x, y)
```



移動した瞬間、現在地に足跡フラグ(VISITED)を置き、経過としてスタックに座標を積む。

ゴール到達！軌跡を記録する

```
if (x == goal_x and y == goal_y):  
    for k = 0 to stack_top:  
        paths[sol_num][k] = stack_visit[k]  
    sol_num = sol_num + 1
```



終点到達！スタックの中身（歩んできた履歴）を一つの「解」として保管庫へ移す。

未知のマスへ進む (再帰呼び出し)

else:

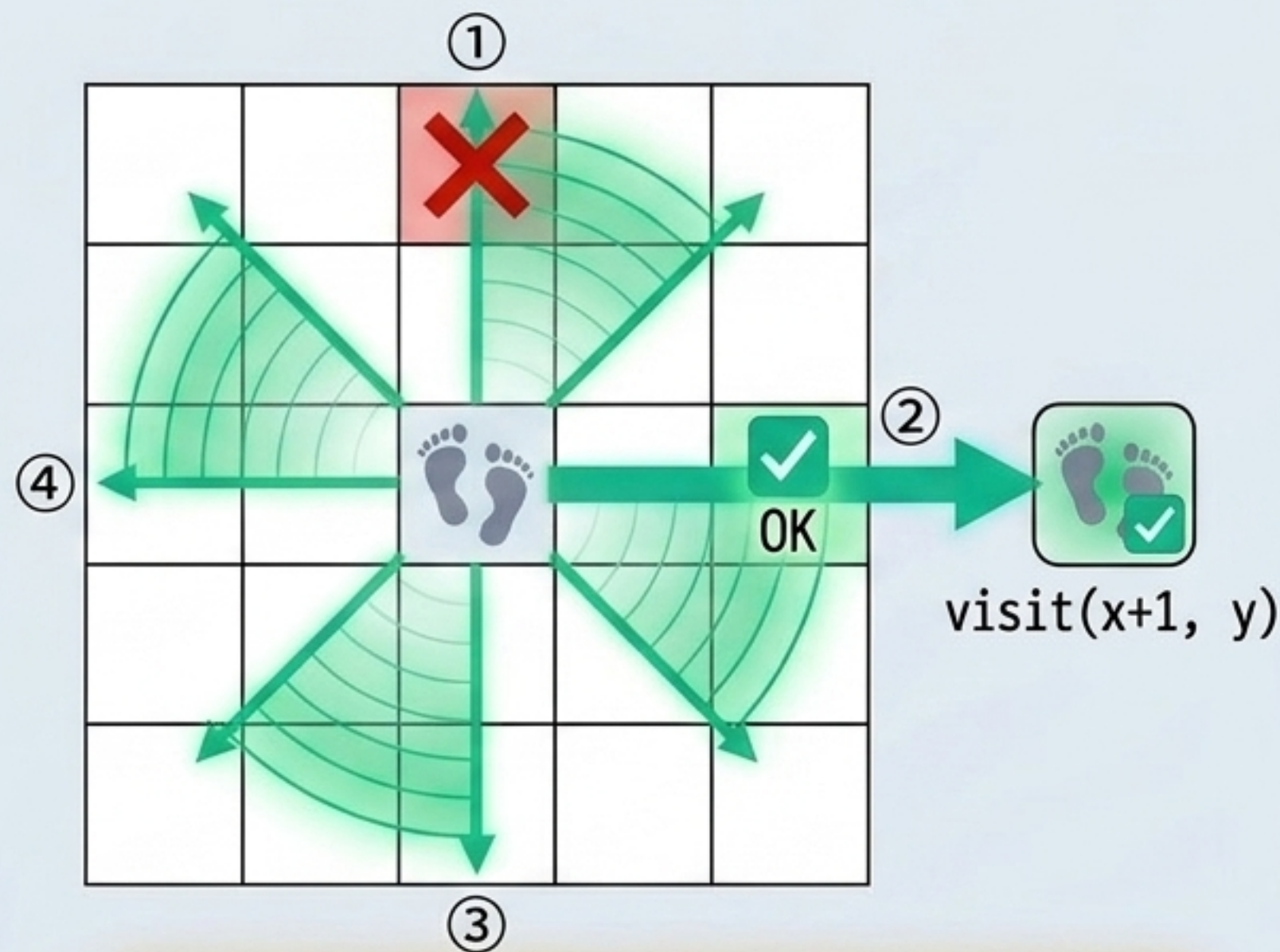
```
stack_top = stack_top + 1
```

```
if (maze[x][y+1] == OK): visit(x, y+1)
```

```
if (maze[x+1][y] == OK): visit(x+1, y)
```

```
if (maze[x][y-1] == OK): visit(x, y-1)
```

```
if (maze[x-1][y] == OK): visit(x-1, y)
```



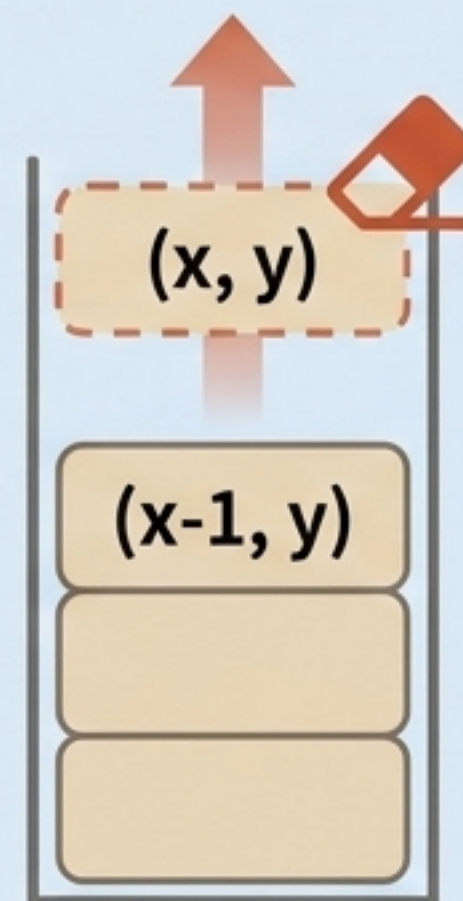
進める(OK)マスがあれば直ちに移動し、
自身の分身 (再帰) を呼び出す。

進めないなら、元に戻る（後戻り処理）

```
stack_top = stack_top - 1  
maze[x][y] = OK
```



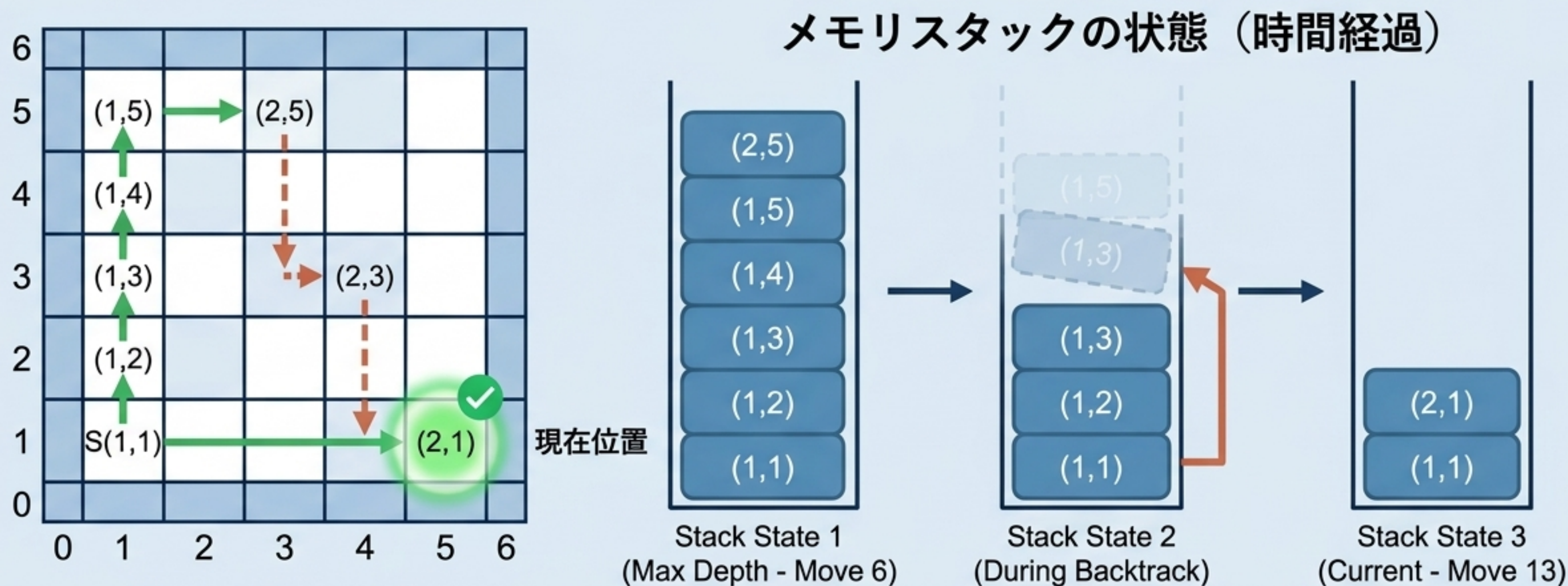
stack_visit
(Memory Stack)



stack_visit
(Memory Stack)

ここがアルゴリズムの肝。
足跡を消して戻ること、別のルートで
このマスを使う可能性を残す！

メモリ（スタック）は過去を記憶する



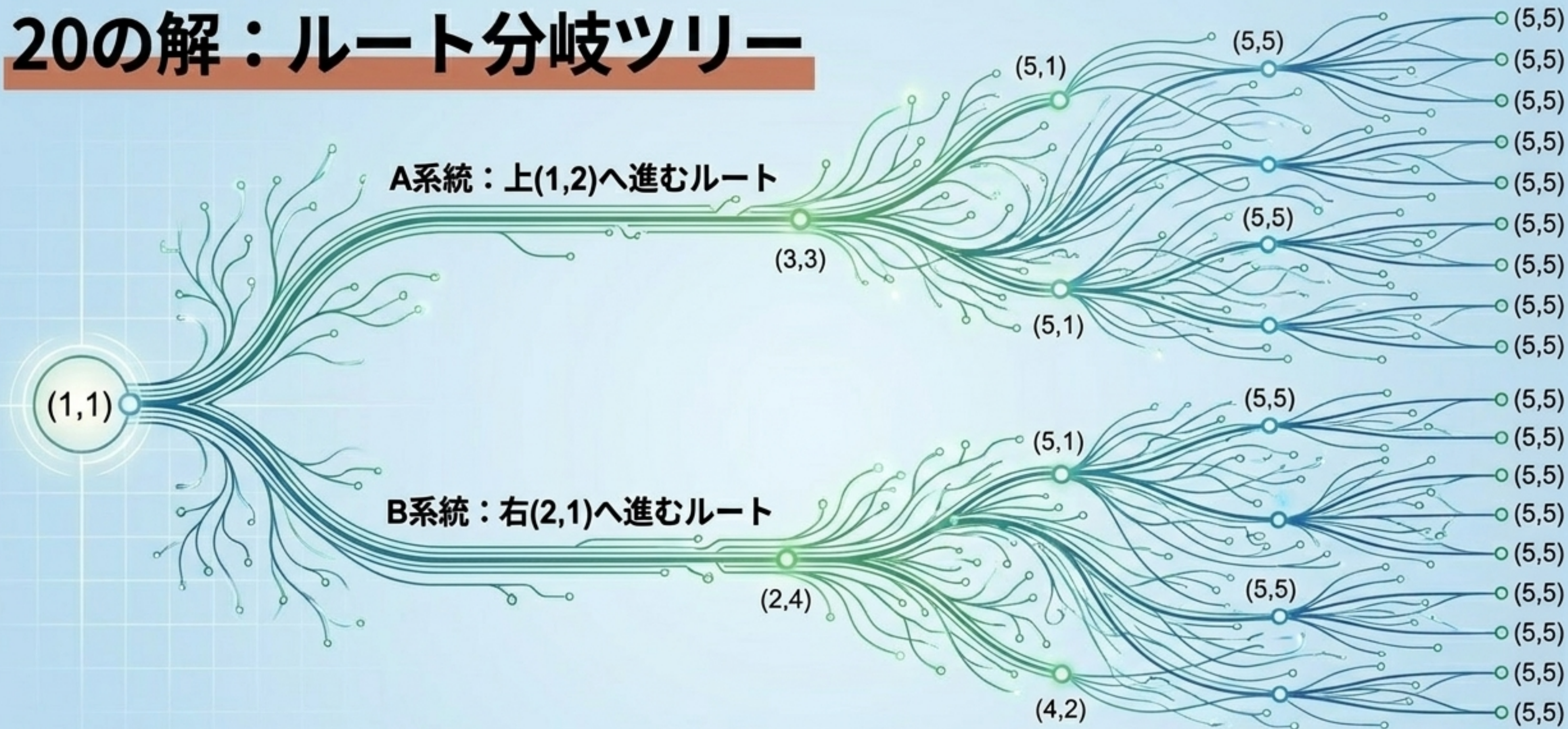
後戻りしたルートはスタックから破棄される。
現在進行中の正しい歴史だけが記憶される。

1つの解で満足しない。全ルートを網羅せよ

終点での処理が終わると、関数は1つ手前のマスに戻り、残りの探索方向を順に試す。ゴール地点の足跡も消されるため、別のルートから再びゴールへ進入することが可能になる。



20の解：ルート分岐ツリー



DFS（深さ優先探索）は、可能な限りの深部まで進み、行き止まれば一つ戻って別の枝を試す。このツリーはその全探索の軌跡である。

Complete Solution Set (全20解の軌跡)

解 1: (1,1) → (1,2) → (1,3) → (1,4) → (1,5) → (2,5) → (3,5) → (3,4) → (4,4) → (5,4) → (5,5)
解 2: (1,1) → (1,2) → (1,3) → (1,4) → (1,5) → (2,5) → (3,5) → (3,4) → (3,3) → (3,2) → (4,2) → (5,3) → (5,4) → (5,5)
解 3: (1,1) → (1,2) → (1,3) → (1,4) → (1,5) → (2,5) → (3,5) → (3,4) → (3,3) → (3,2) → (4,2) → (4,2) → (4,1) → (5,1) → (5,2) → (5,3) → (5,4) → (5,5)
解 4: (1,1) → (1,2) → (1,3) → (1,4) → (1,5) → (2,5) → (3,5) → (3,4) → (3,3) → (3,2) → (3,1) → (3,1) → (4,1) → (4,2) → (5,2) → (5,3) → (5,4) → (5,5)
解 5: (1,1) → (1,2) → (1,3) → (1,4) → (1,5) → (2,5) → (3,5) → (3,4) → (3,3) → (3,2) → (3,1) → (3,1) → (4,1) → (5,1) → (5,2) → (5,3) → (5,4) → (5,5)
解 6: (1,1) → (1,2) → (1,3) → (2,3) → (3,3) → (3,4) → (4,4) → (5,4) → (5,5)
解 7: (1,1) → (1,2) → (1,3) → (2,3) → (3,3) → (3,2) → (4,2) → (5,2) → (5,3) → (5,4) → (5,5)
解 8: (1,1) → (1,2) → (1,3) → (2,3) → (3,3) → (3,2) → (4,2) → (4,1) → (5,1) → (5,2) → (5,3) → (5,4) → (5,5)
解 9: (1,1) → (1,2) → (1,3) → (2,3) → (3,3) → (3,2) → (3,1) → (4,1) → (4,2) → (5,2) → (5,3) → (5,4) → (5,5)
解 10: (1,1) → (1,2) → (1,3) → (2,3) → (3,3) → (3,2) → (3,1) → (4,1) → (5,1) → (5,2) → (5,3) → (5,4) → (5,5)

解 11: (1,1) → (2,1) → (3,1) → (3,2) → (3,3) → (3,4) → (4,4) → (5,4) → (5,5)
解 12: (1,1) → (2,1) → (3,1) → (3,2) → (3,3) → (2,3) → (1,3) → (1,4) → (1,5) → (2,5) → (3,5) → (4,4) → (5,4) → (5,5)
解 13: (1,1) → (2,1) → (3,1) → (3,2) → (4,2) → (5,2) → (5,3) → (5,4) → (5,5)
解 14: (1,1) → (2,1) → (3,1) → (3,2) → (4,2) → (4,1) → (5,1) → (5,2) → (5,3) → (5,4) → (5,5)
解 15: (1,1) → (2,1) → (3,1) → (4,1) → (4,2) → (5,2) → (5,3) → (5,4) → (5,5)
解 16: (1,1) → (2,1) → (3,1) → (4,1) → (4,2) → (3,2) → (3,3) → (3,4) → (4,4) → (5,4) → (5,5)
解 17: (1,1) → (2,1) → (3,1) → (4,1) → (4,2) → (3,2) → (3,3) → (2,3) → (1,3) → (1,4) → (1,5) → (2,5) → (3,5) → (3,4) → (3,4) → (4,4) → (5,4) → (5,5)
解 18: (1,1) → (2,1) → (3,1) → (4,1) → (5,1) → (5,2) → (5,3) → (5,4) → (5,5)
解 19: (1,1) → (2,1) → (3,1) → (4,1) → (5,1) → (5,2) → (4,2) → (3,2) → (3,3) → (3,4) → (4,4) → (5,4) → (5,5)
解 20: (1,1) → (2,1) → (3,1) → (4,1) → (5,1) → (5,2) → (4,2) → (3,2) → (3,3) → (2,3) → (1,3) → (1,4) → (1,5) → (2,5) → (3,5) → (3,5) → (4,4) → (5,4) → (5,5)

Pythonコードのわずか数十行が、この複雑な20の軌跡を完璧に、そして漏れなく導き出す。

深淵での迷走と復帰（移動22回・訪問3回）

前半（行き止まり探索 12回）



後半（新たなルート発見 10回）



座標(4,2)が3回訪問される理由：前半の往復（2回）
+ 後半の通過（1回）＝ 計3回

Cheat Sheet: DFS & Backtracking

迷路探索の全貌は、この2つの状態遷移の繰り返しに他ならない。

	前進する (進む)	後戻りする (戻る)
条件	未訪問(OK)のマスがある	4方向すべて進行不可 (または終点)
スタック操作	<code>stack_top += 1</code> (記録の追加)	<code>stack_top -= 1</code> (記録の破棄)
フラグ操作	VISITED (足跡をつける)	OK (足跡を消す)

The Power of Recursion

```
def visit(x, y):  
    if not is_valid(x, y):  
        return path.append((x, y))  
    if (x, y) == (GOAL_X, GOAL_Y):  
        solutions.append(list(path))  
    else:  
        visit(x+1, y);  
        visit(x-1, y);  
        visit(x, y+1);  
        visit(x, y-1);  
        path.pop()
```

進む。記録する。ダメなら戻して、足跡を消す。

たったこれだけのシンプルなルールの反復（再帰）が、

20通りもの複雑な解答を一つ残らず網羅する。

アルゴリズムの真の力は、単純な規則から生まれる無限の問題解決能力にある。

