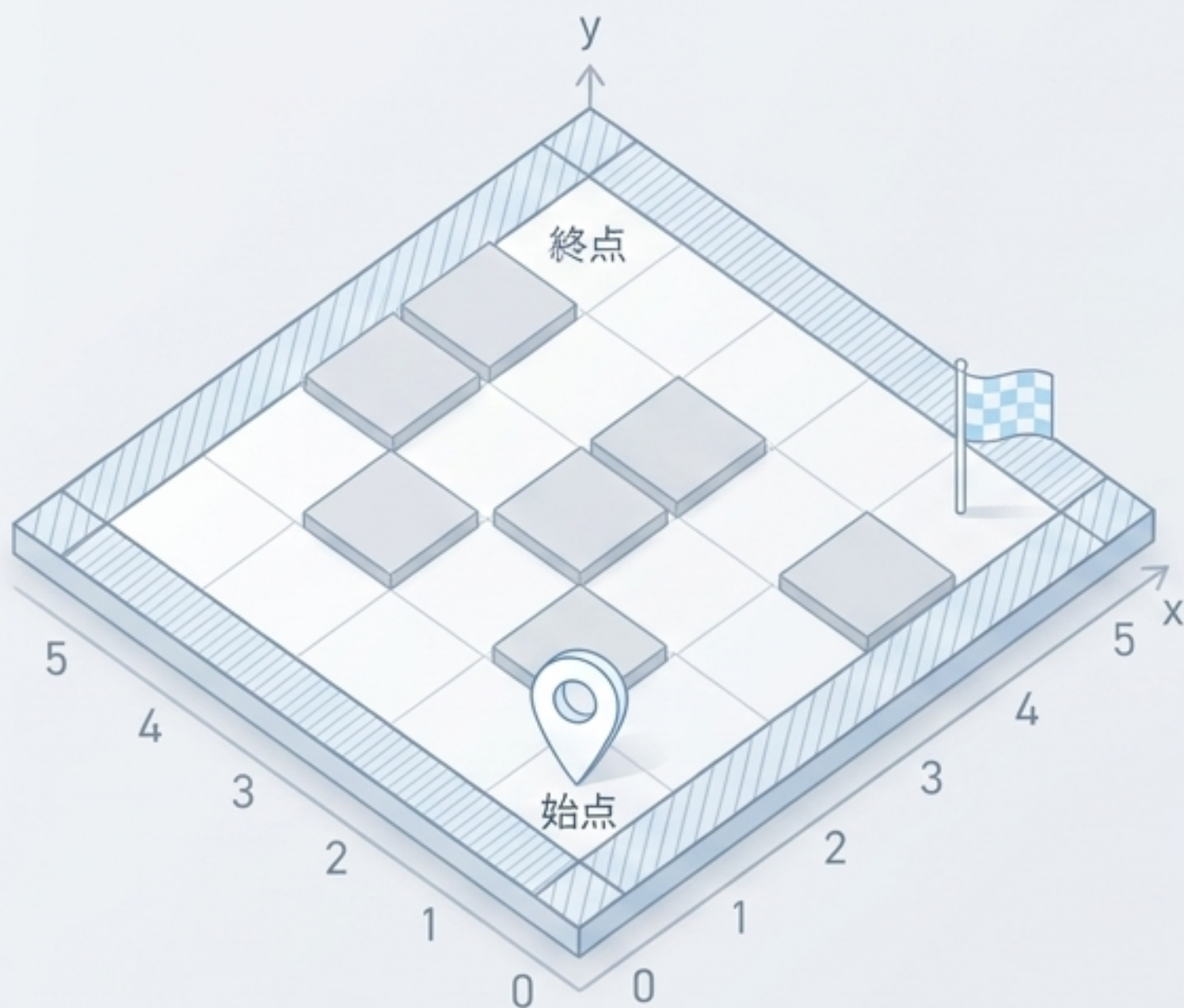




迷路探索アルゴリズムを攻略せよ！

～視覚で身につく「トレースカ」～

令和4年秋期 応用情報技術者
午後問3（設問1・設問2 完全ヒント集）



[RUN SIMULATION]

プログラミング問題で問われる「3つの力」

1



【国語力】読解

問題文から解答の根拠を
見つける力。

2



【アルゴリズム力】想像

処理構造やアルゴリズムの
動きを推測する力。

3



【トレース力】再現

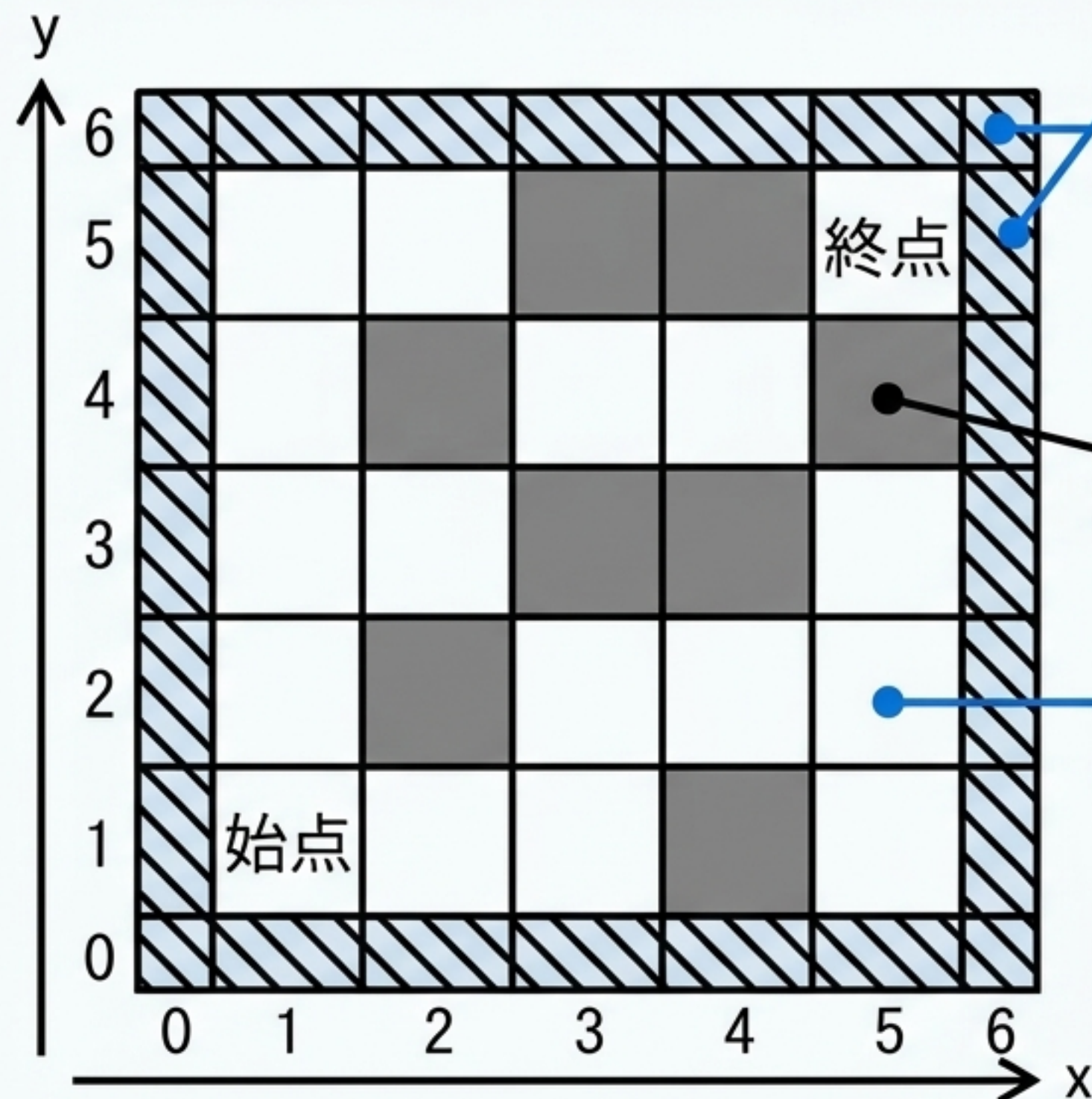
紙とペンでコードの挙動を
一歩ずつ正確に追いかける。

本問（午後問3）は、この『**トレース力**』が合否を分ける
最大の鍵！頭の中の動きを視覚化してデバッグしよう。

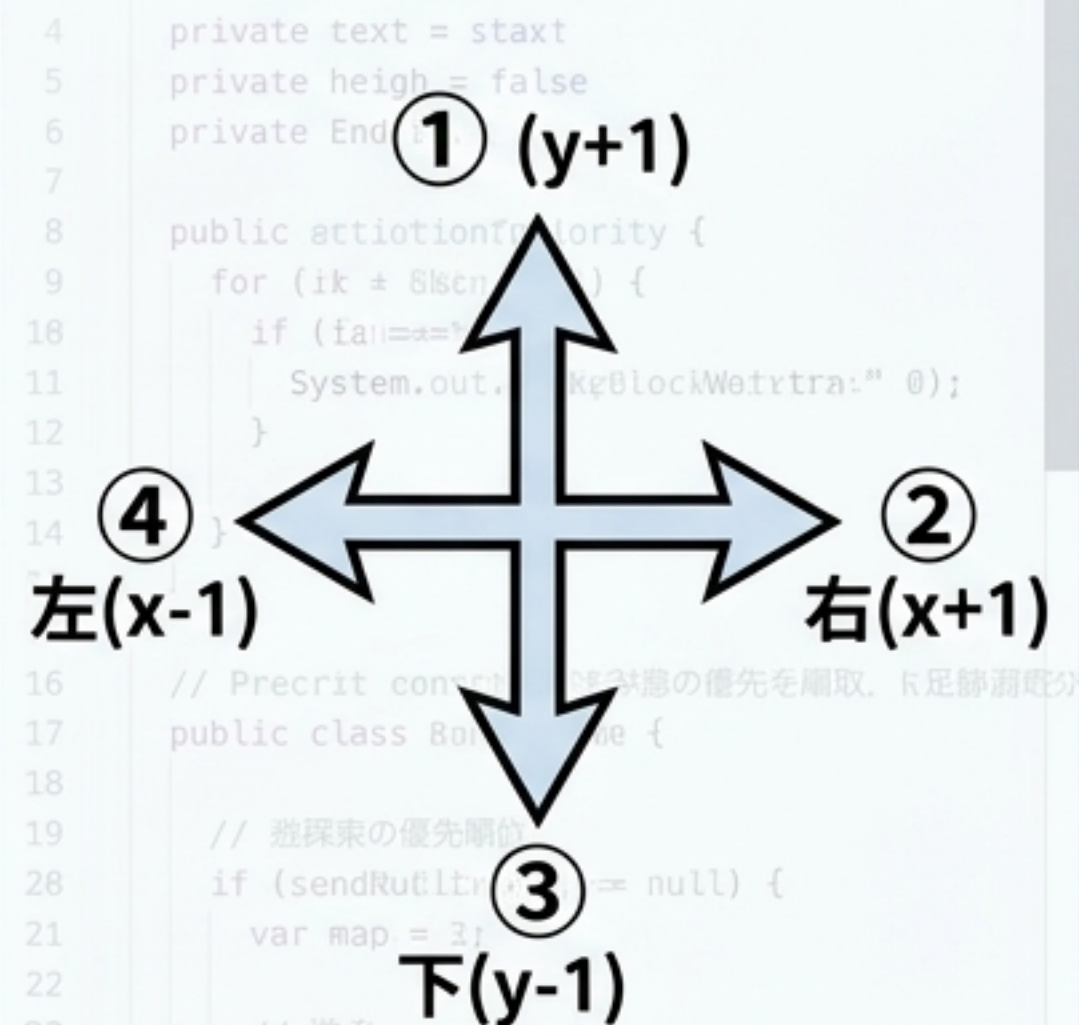
ルール確認：迷路探索の掟（要件定義）

Map View

Parameters



探索の優先順位 (Priority)



※注意：同じマスは2度通れない
(足跡フラグで管理)

探索の2大アクション：「進む」と「戻る」

フェーズ1：前進 (Explore)



- 未訪問のマスへ進む。
- マス情報に「足跡フラグ (VISITED)」を置く。

フェーズ3：後退 (Backtrack)



- 4方向すべて進行不可（行き止まり）。
- 1つ前のマスへ戻る。

「戻る」ときのトリガーは常に【**四方塞がり**】の絶望状態！



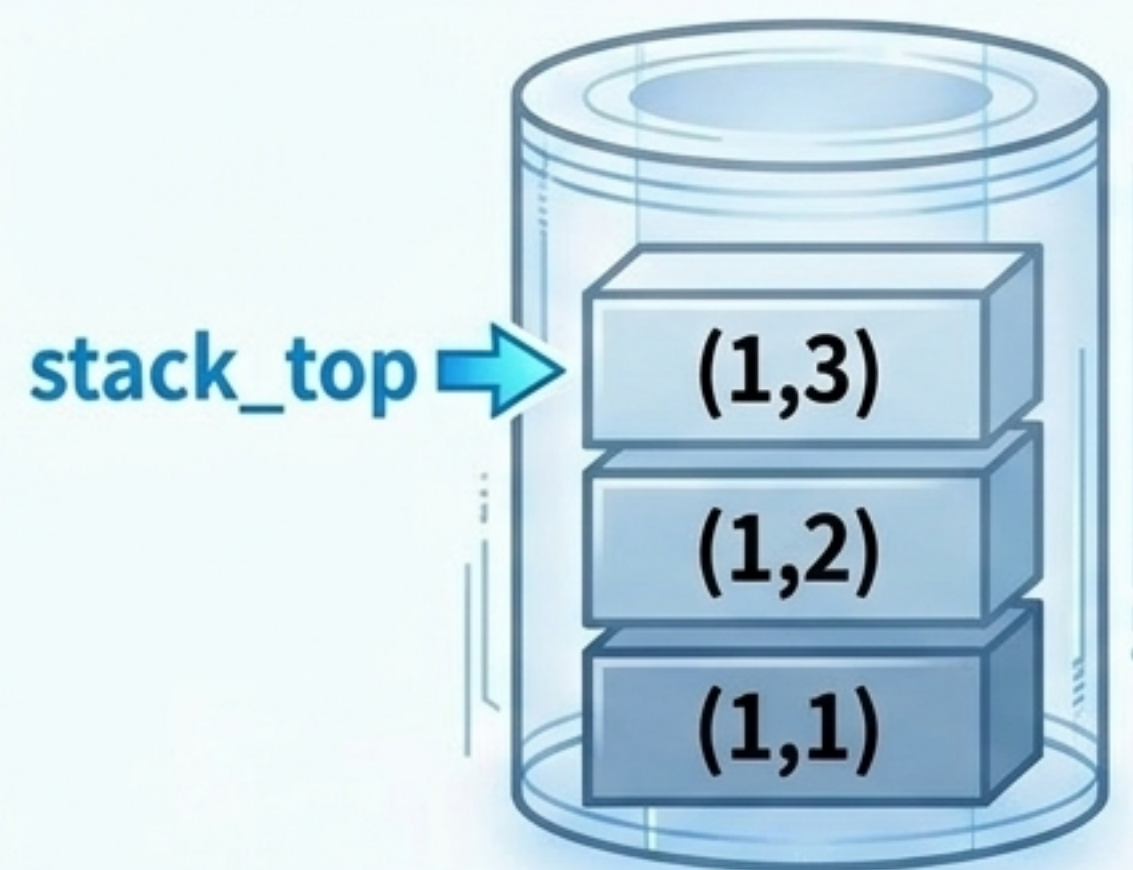
記憶のリュックサック：「スタック」

Map View



移動軌跡

Stack Memory



[進む] = Push :
ブロックを積む
($\text{stack_top} + 1$)

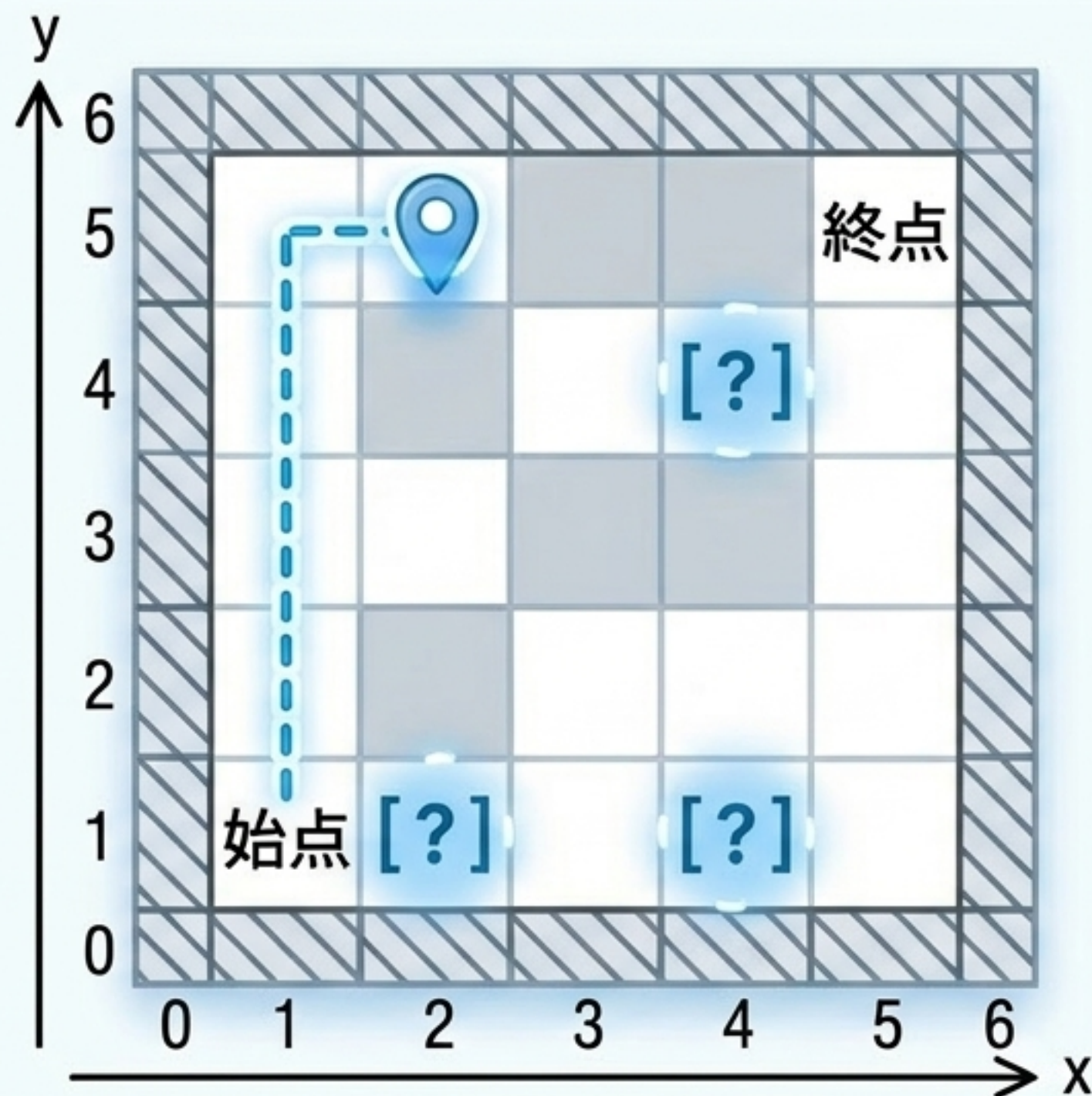


[戻る] = Pop :
ブロックを降ろす
($\text{stack_top} - 1$)

探索の軌跡を一時記憶するメモリ。これが最終的に解を記録するベースとなる。

【設問1(1)】 トレース・チャレンジ開始！

Map View

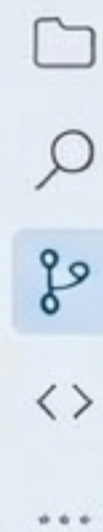


Instructions

ミッション：ルールに従って、指で一筆書きしてみよう！

優先順位リマインダー：上 → 右 → 下 → 左

1. 行き止まりにぶつかったら一つ戻る。
2. ゴール(5,5)に着くまでシミュレーションを続ける。
3. 最終的に「一度も踏まれなかった白マス」の数を数えよ！

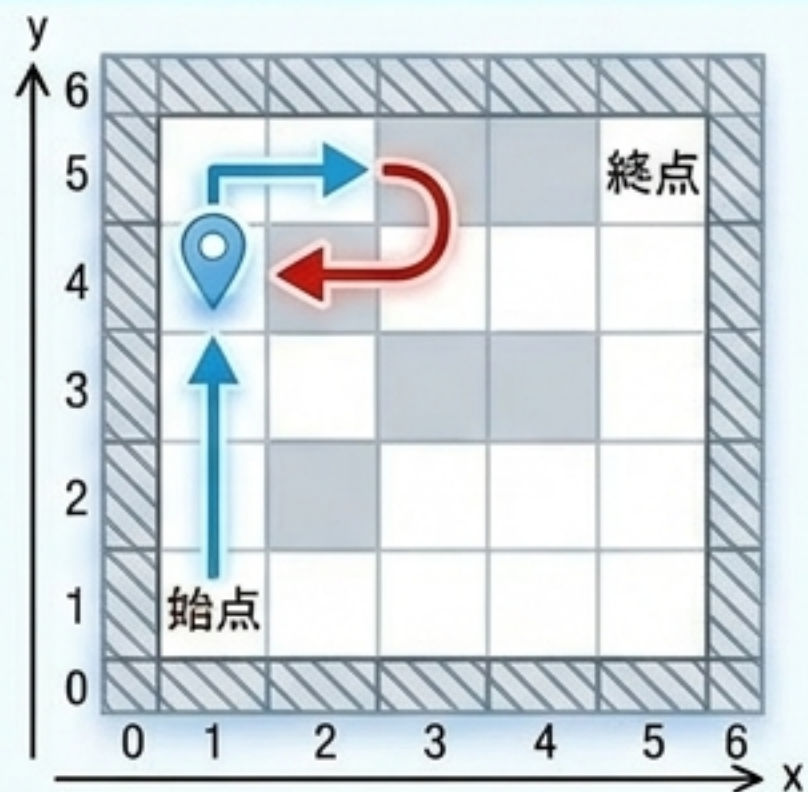


【設問1(2)】 「13回目の移動」の罠



重要ルール：「進む」も「戻る」も、1回の移動としてカウントする！

Map View (at Move 7)



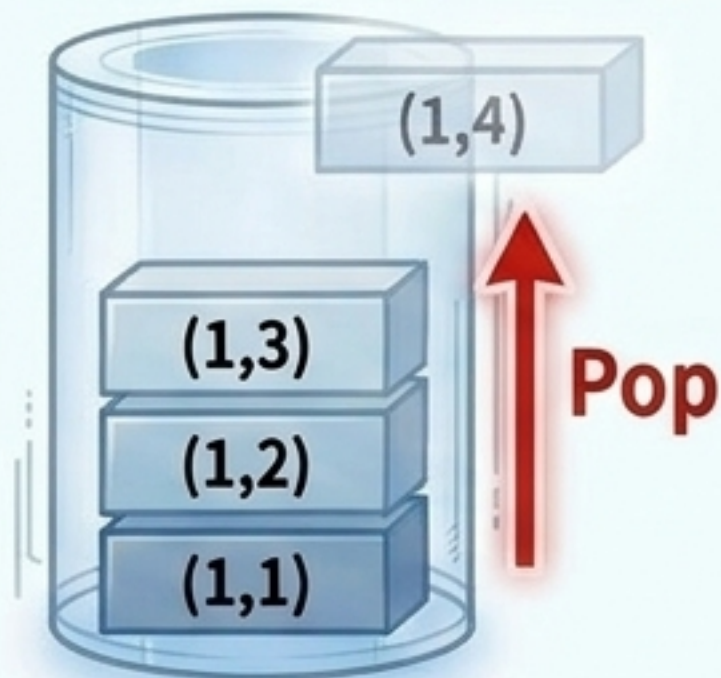
Stack Memory (at Move 7)



問題文ヒント：
「7回目の移動終了時で
スタックには4個の座標」

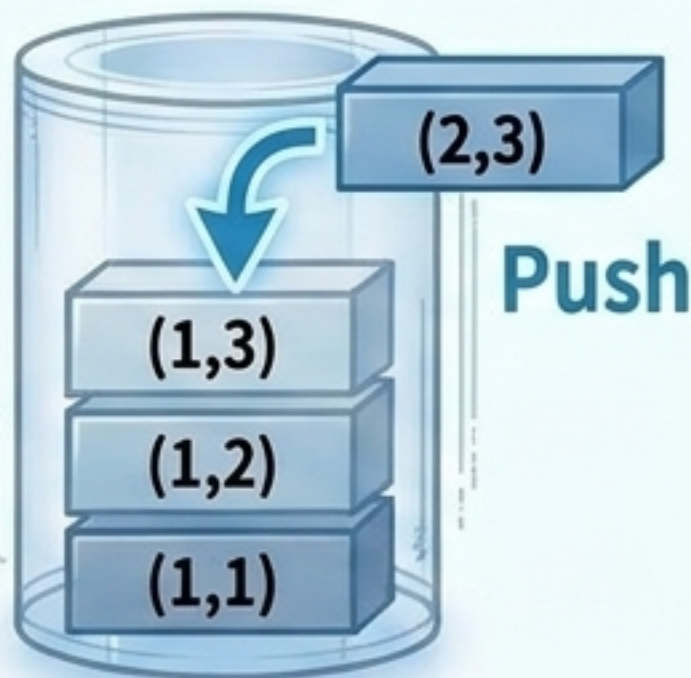
バックトラック（後退）を追え！

8回目の移動



(1,4)から(1,3)へ戻る

9回目の移動



(1,3)から右の(2,3)へ進む

13回目まで指で追う



[ア]=?

ヒント：(1,1)から右(x方向)へ進むのは何回目の移動の後か？
指で数えながら13回目ぴったりでストップしよう。
その瞬間に積まれているブロックの数が答えだ！

コード解読の極意：3つのフェーズに分割せよ

コード解読 - visit(x, y) 関数

```
1  if (x が goal_x と等しい かつ y が goal_y と等しい)
2      for(k を 0 から stack_top まで 1 ずつ増やす)
3          [ イ ] ← stack_visit[k]
4  else
5      stack_top ← stack_top + 1
6      if(maze[x][y+1]が OK と等しい) visit(x, y+1)
7      // ... 他の3方向の探索 ...
8      stack_top ← [ ウ ]
9  endif
10 [ エ ] ← OK
```

フェーズ2：ゴール到達・解の記録

フェーズ1：未訪問マスへ進む（再帰）

フェーズ3：行き止まり・後退

空欄がどのフェーズに属しているかを見極めるのが、コード読解の最短ルート。

【設問2 イ】 ゴール到達時、解をどう記録する？

コード解説 - ヒント

★・フェーズ：到達ゾーン（Gold）

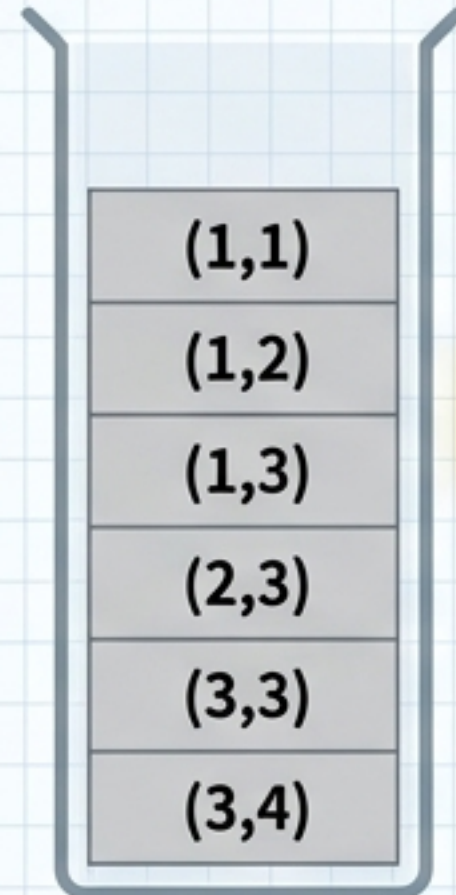
🔍・表1の変数一覧を要確認！

?・「迷路の全ての解の座標を格納する2次元配列」の名前は？

✂・解の番号（u） = sol_num

✂・順番（v） = k

💡・これらを組み合わせると、左辺ははどう書くべき？



一時記憶：スタック
(stack_visit[k])



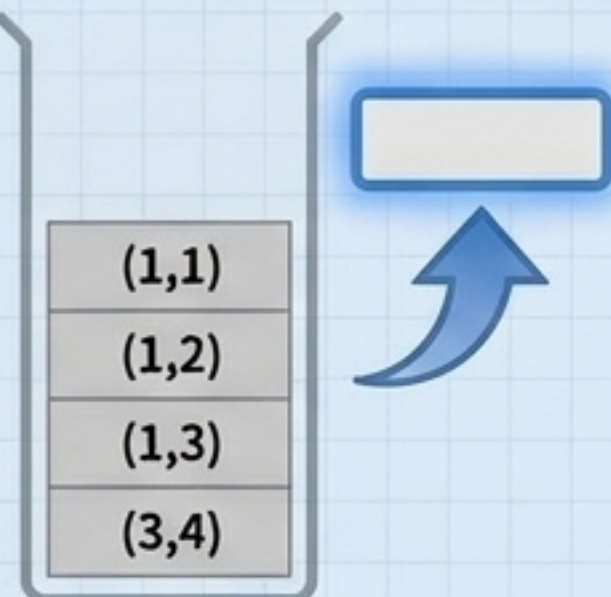
u \ k	0	1	2	3	4	5	...
sol_num=0	paths[0] [0]=(1,1)	paths[0] [1]=(1,2)	paths[0] [2]=(1,3)	paths[0] [3]=(1,4)	paths[4] [4]=(3,5)	paths[5] ...	...
sol_num=1							
...							
...							

永久保存：アルバム
(paths配列)

[イ] ← stack_visit[k]

【設問2 ウ】 行き止まり！ スタックをどう操作する？

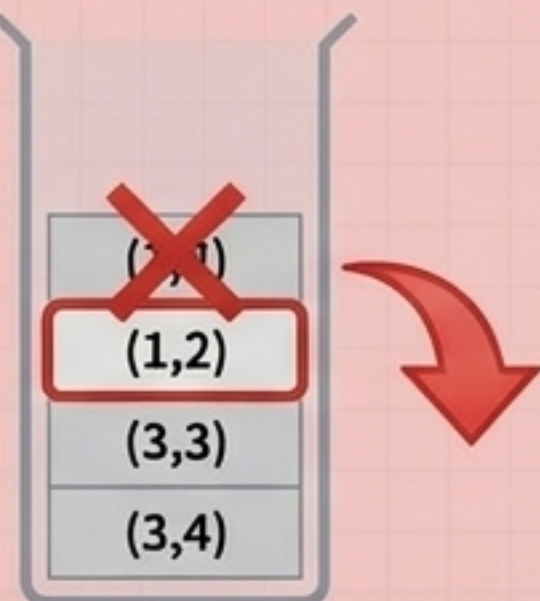
前に進む時（フェーズ1）



```
stack_top ← stack_top + 1
```

ポインタを上に進め、新しい座標を格納する準備をする。

1つ前に戻る時（フェーズ3）



```
stack_top ← [ ウ ]
```

4方向すべて進行不可。記録した直近の座標を破棄し、1つ前の状態に戻る。



ヒント：前に進むときに『+1』したなら、戻るときはポインタをどう計算すべきか？

【設問2 工】 戻るなら、足跡を消して更地にせよ



コード解説 - ヒント

```
function new_func(x, y) {  
  for ([x,] = 0; y[i++]) {  
    [ 工 ] ← OK  
  }  
  return 0;  
}
```

問題文のヒント: 「現在いるマスのマス情報をOKフラグに戻し」



- 「現在いるマス(x, y)の情報」を管理している2次元配列の名前は何だったか？
- 表1の配列一覧を見直そう！

振り返り：トレース(手作業)とコード(自動化)は表裏一体



挑戦の前の最終確認

☐ Checklist

- ☐ 進む優先順位（上・右・下・左）のコンパスをイメージできているか？
- ☐ 「戻る」動きも移動1回分としてカウントしているか？
- ☐ コードの空欄補充は、表1の「変数・配列の定義」と照らし合わせたか？

準備は整いました。あとは紙とペンで、迷路の探索者となってじっくり手を動かしてください。Visual Debuggerを頭の中で起動させれば、必ず解けます。Good Luck!

