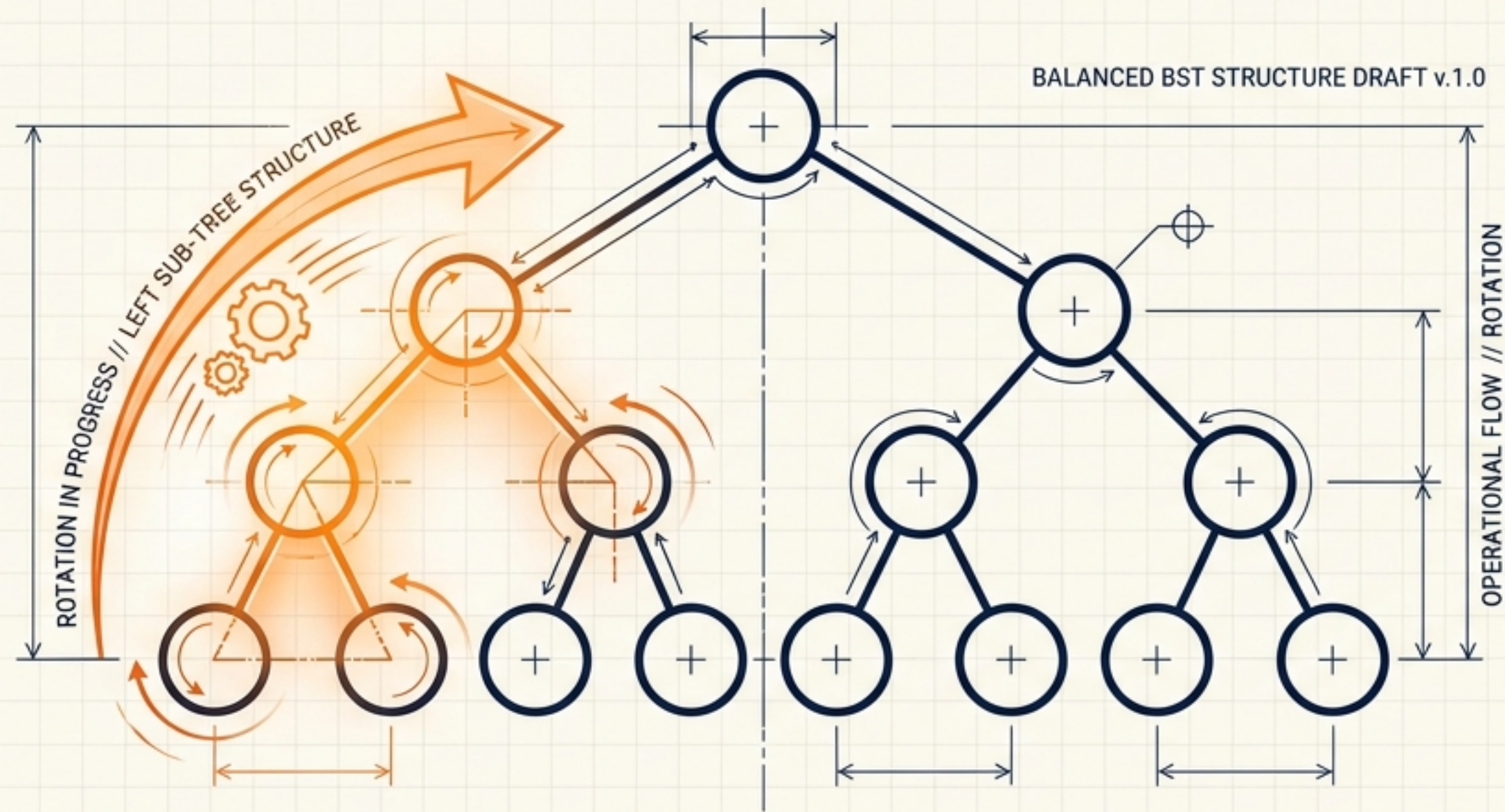


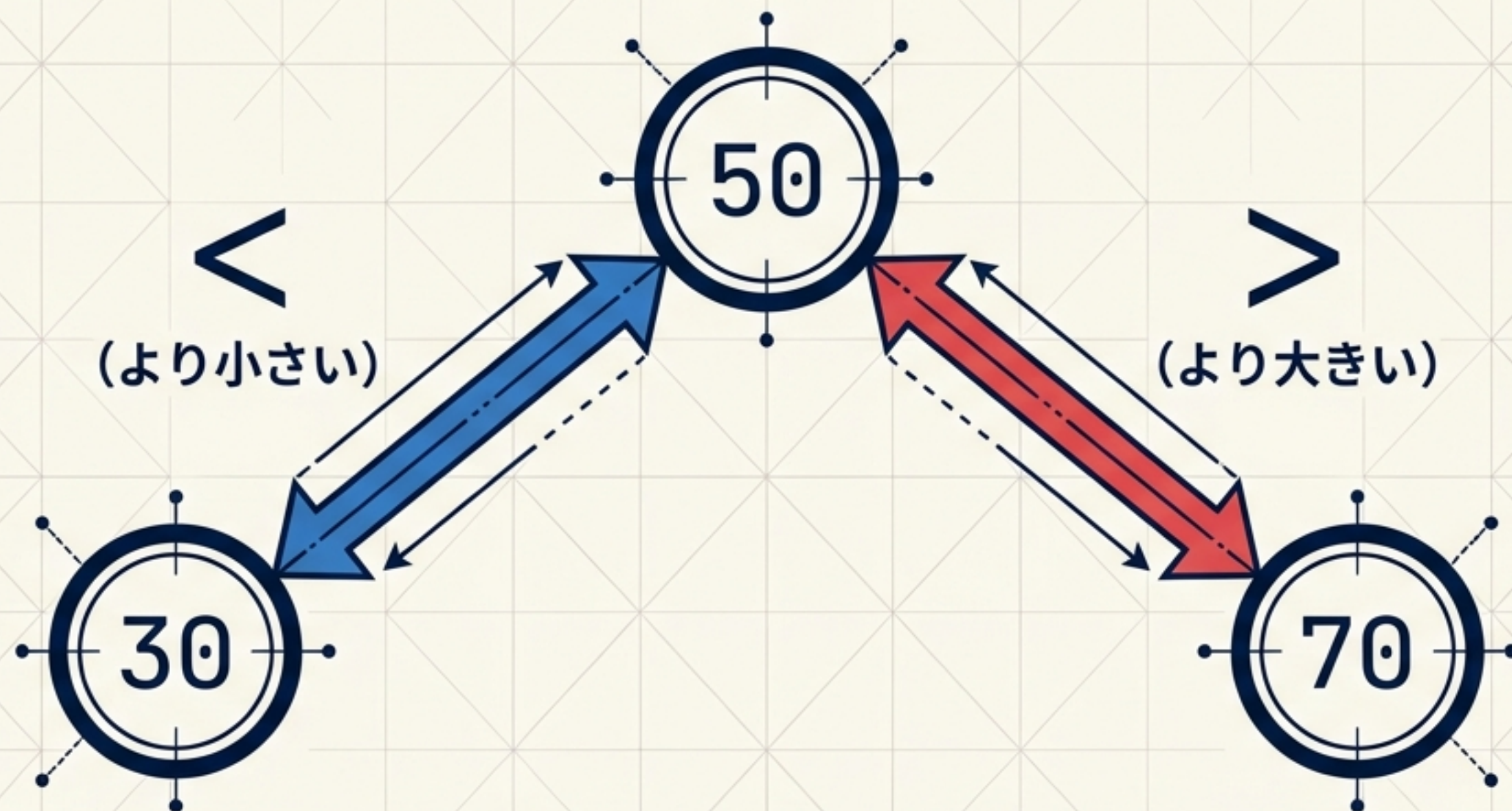
応用情報 午後アルゴリズム攻略

2分探索木の「回転」が見える！自力で解ける！ヒント集



※このスライドに解答は書かれていません。図解をヒントに、問題文と向き合ってみましょう。

基本のキ：2分探索木の「絶対ルール」

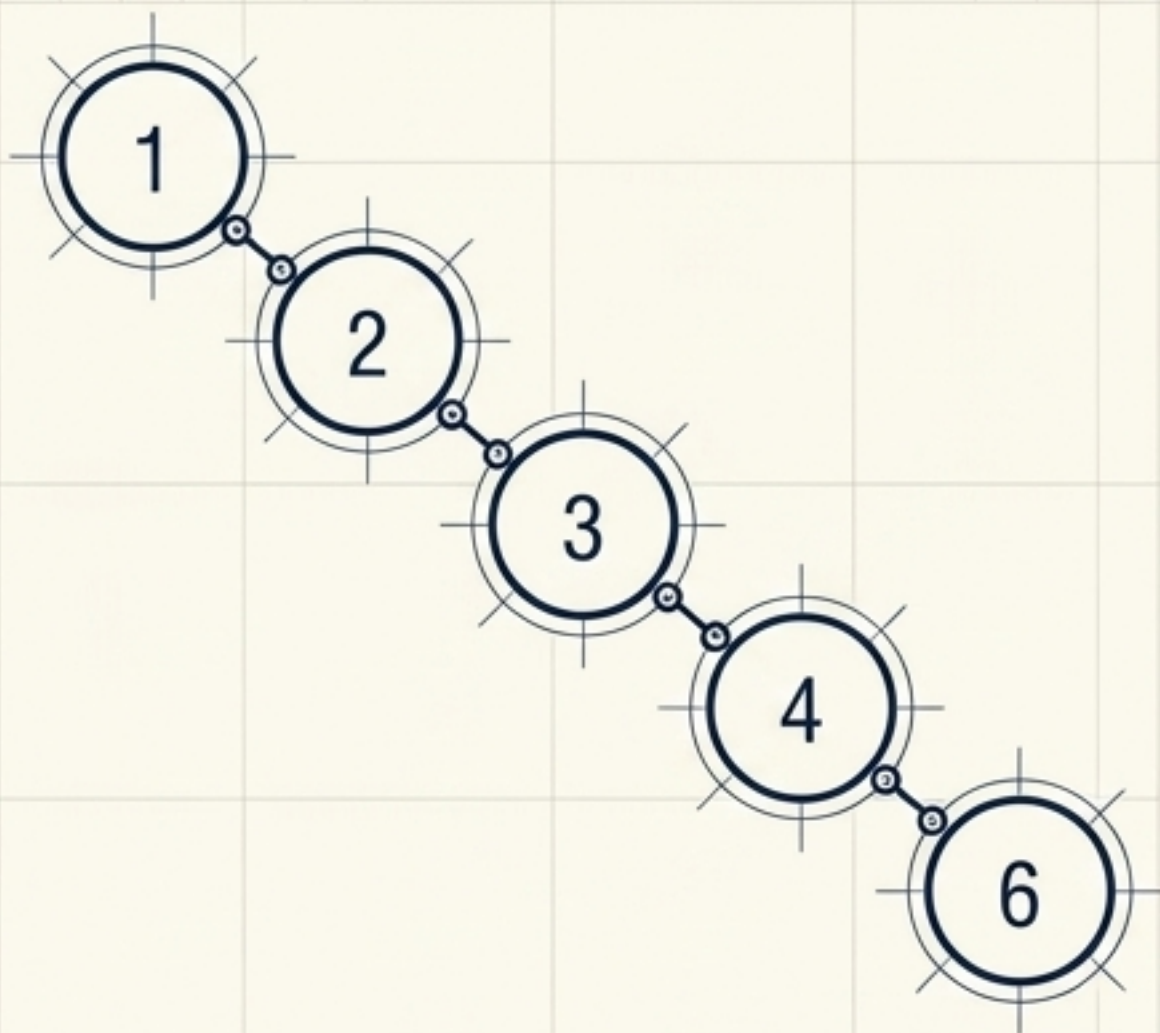


- 左側は常に「親より小さい」
- 右側は常に「親より大きい」

このルールがあるから、
探したいデータに
素早く辿り着ける！

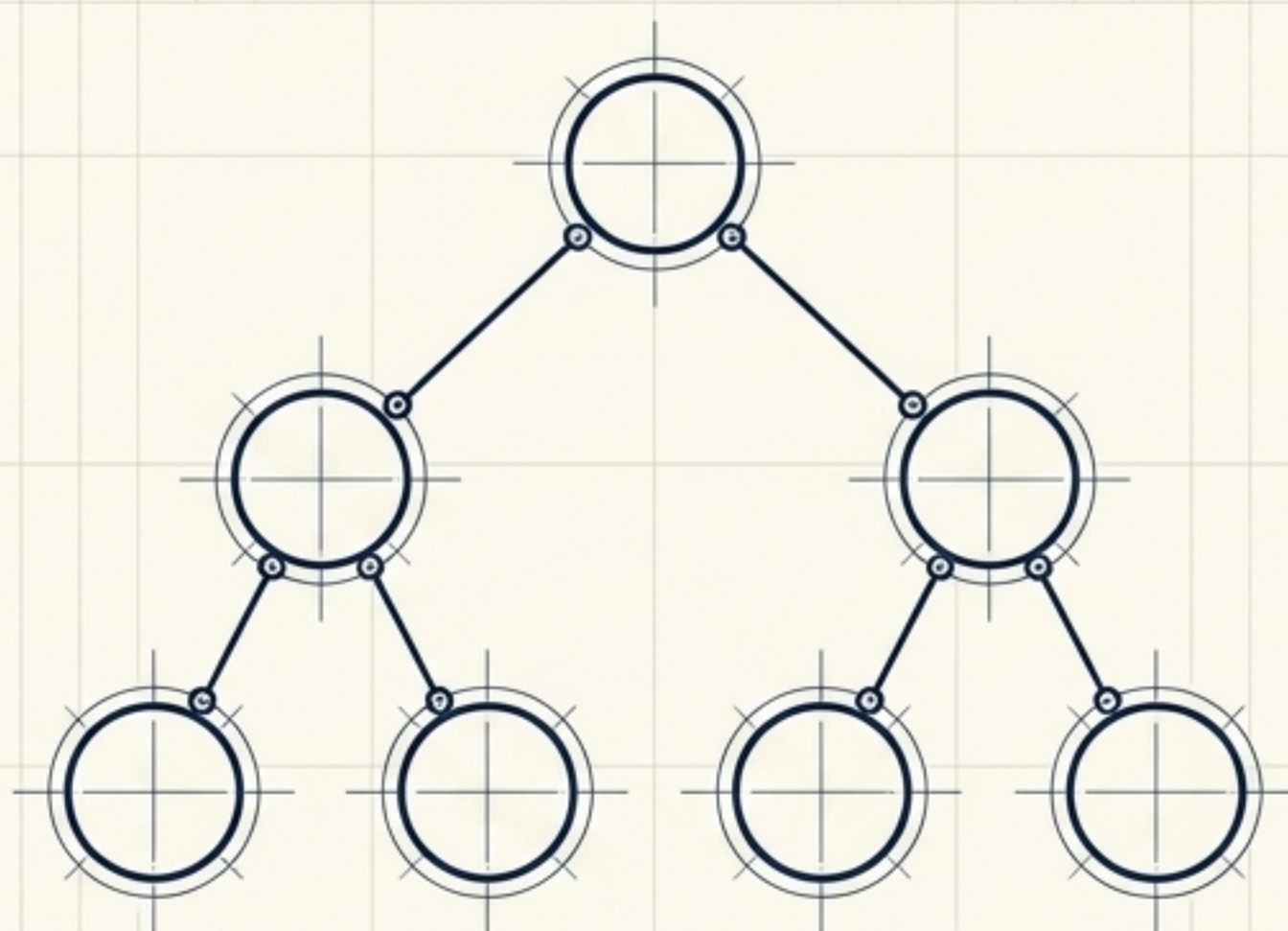
探索スピードは「木の高さ」で決まる ([ア][イ]のヒント)

地獄 (最悪のケース) ⚠



[ア]のヒント: ノードが n 個一直線に並んでいる場合、一番端を探すには何回比較する?

天国 (理想のケース) ★



[イ]のヒント: 綺麗な木なら、1回の比較で探す範囲が「半分」になる。データ量が n の時、半分にし続けると何回で1になるかを表す数学記号は?

なぜ「回転」が必要なの？（平衡2分探索木の目的）

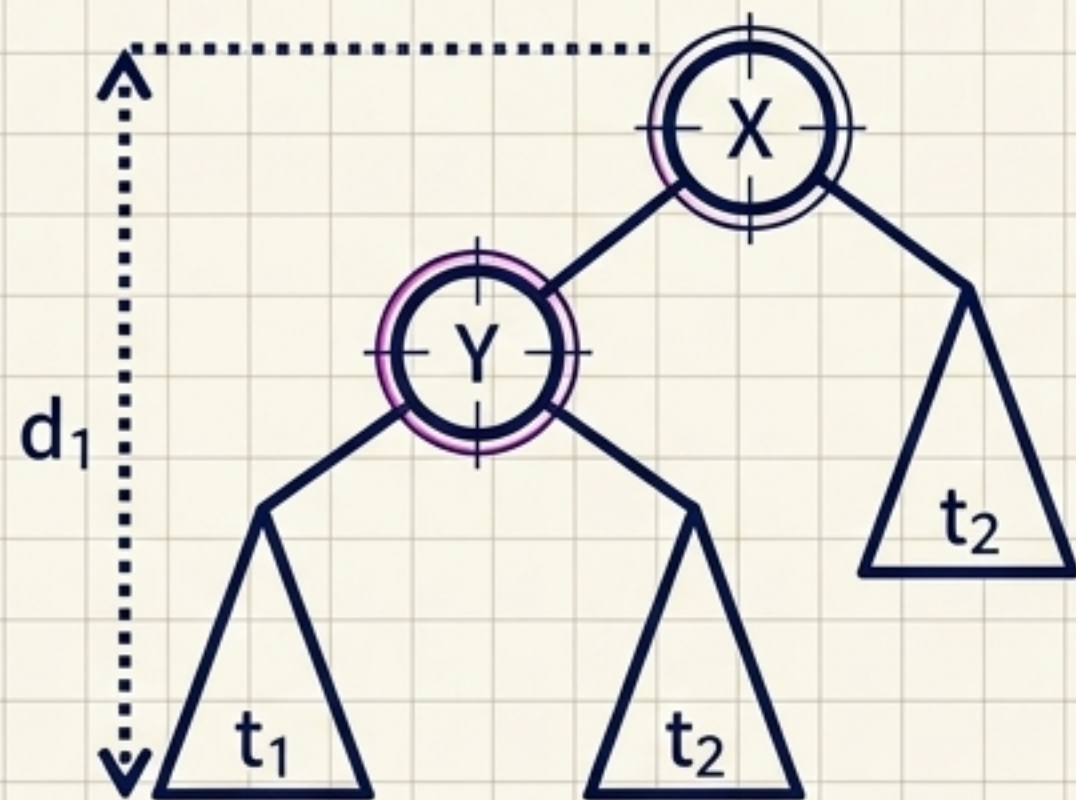


「すべてのノードについて左右の部分木の高さの差が1以下」

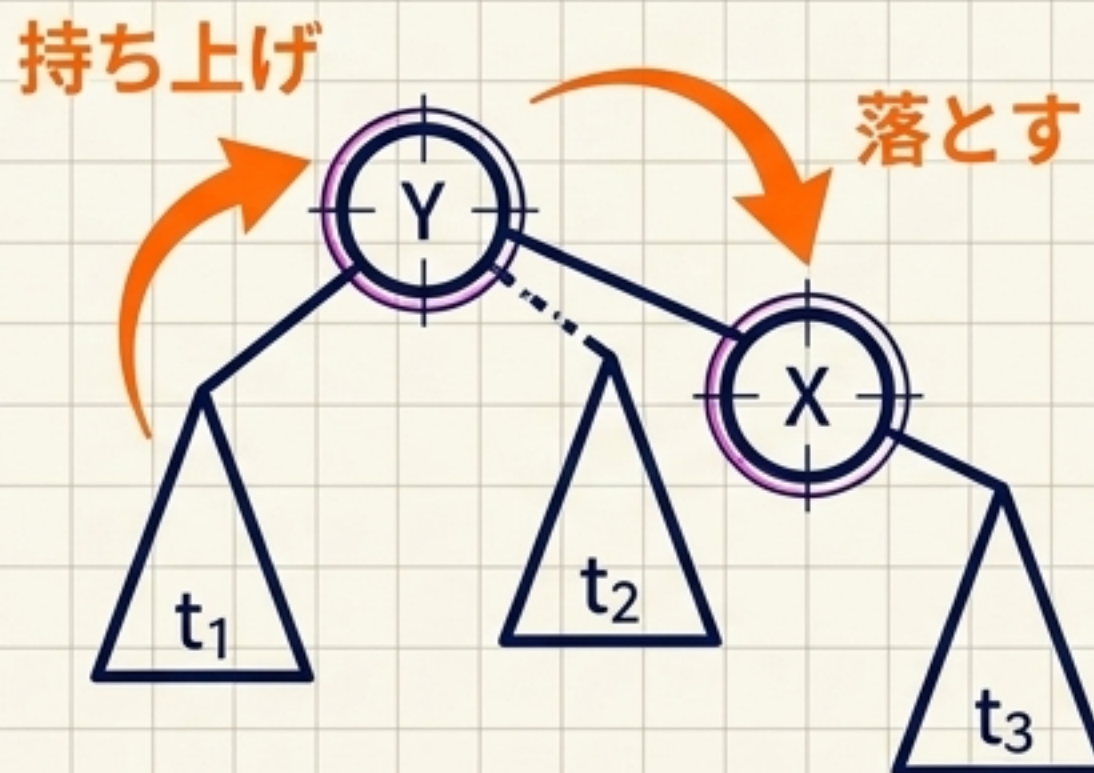
図の木は、**左の高さと右の高さの差が「2」**になっています。
これを直す「**大手術**」「**大手術**」が**回転操作**です。

「右回転」とは何か？（図4の解説）

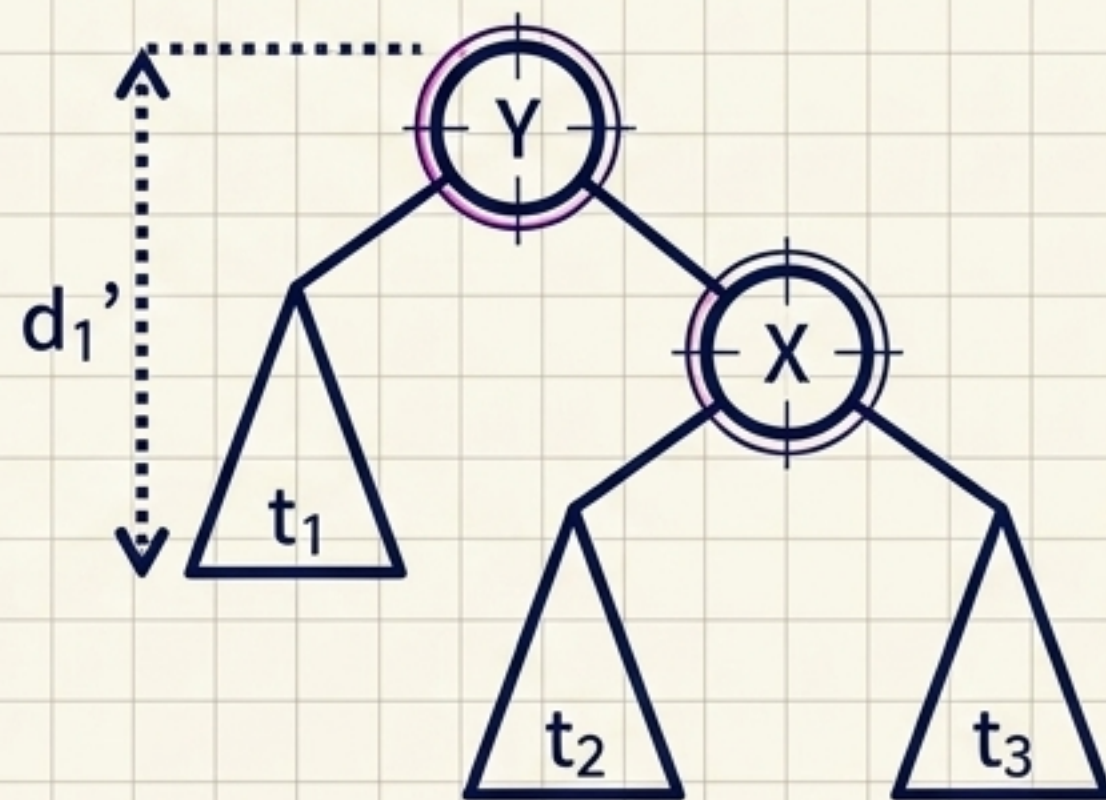
Step 1: 左が重い



Step 2: 持ち上げ&落とす



Step 3: 新しい根



Step 1: 左が重い

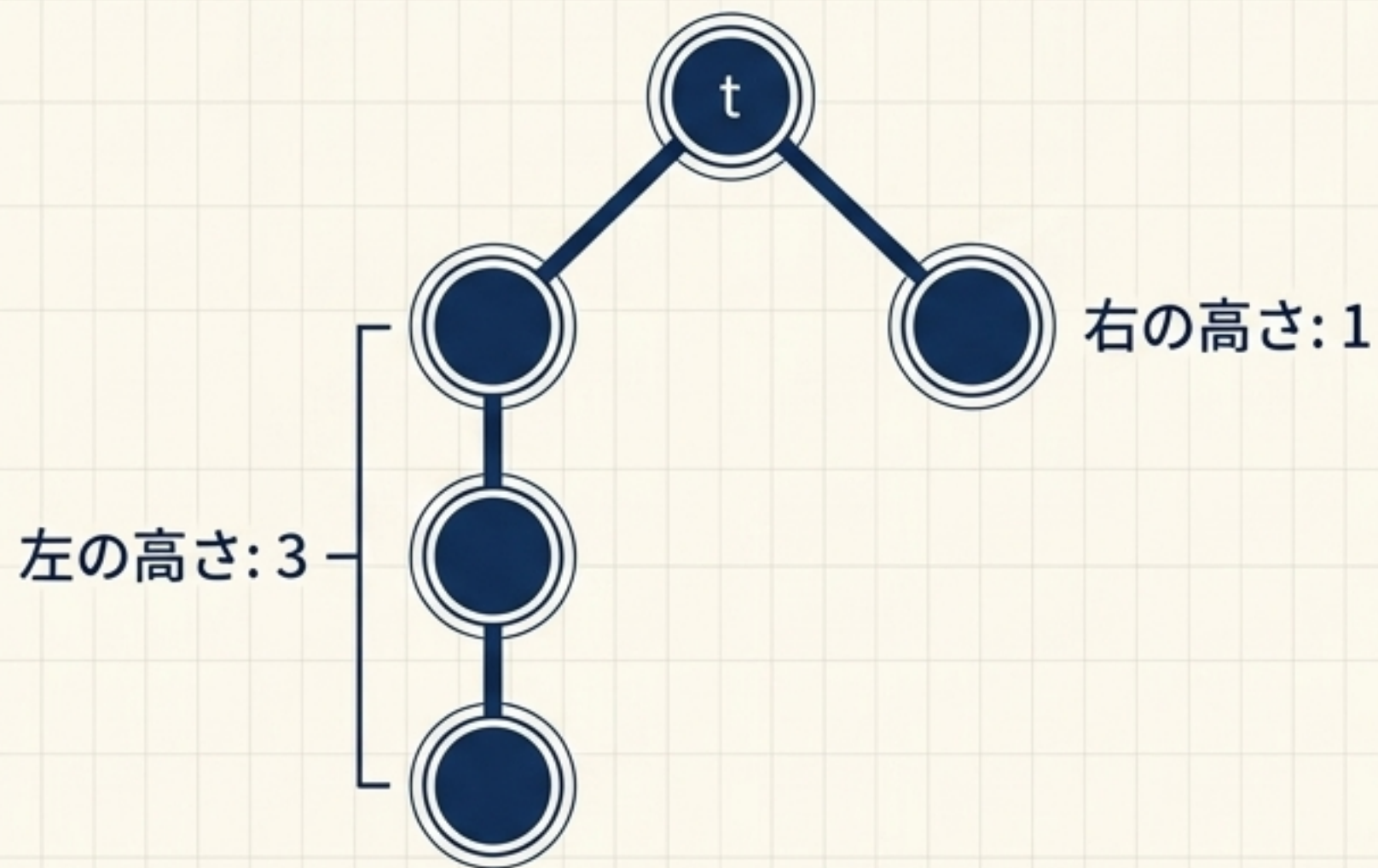
Step 2: 持ち上げ&落とす

Step 3: 新しい根

左側が重すぎる（長すぎる）時、左側の子（Y）を新しい「根」に持ち上げ、元の根（X）を右下に落とす操作。

回転しても「左<親<右」のルールは壊れません！

[ウ]のヒント：どこで「バランスが崩れた」と判定する？



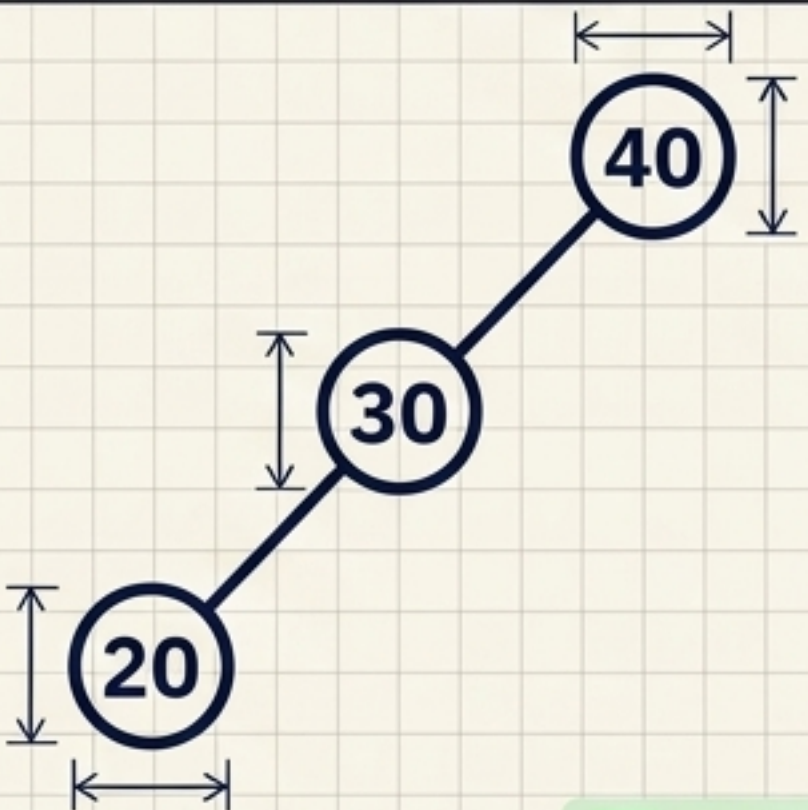
```
1 function balance(t)
2   h1 ← height(t.left) - height(t.right)
3   h1 ← height(t.left) - height(t.right)
4   if([ウ])
5 }
```

$h1 = (\text{左の高さ}) - (\text{右の高さ})$ 。センサーの役割を果たします。

「条件Bal」は「差が1以下」でしたね。では、左側が重すぎて右回転の修正が必要になる境界線（数値）は、 $h1$ がいくつになった時でしょうか？

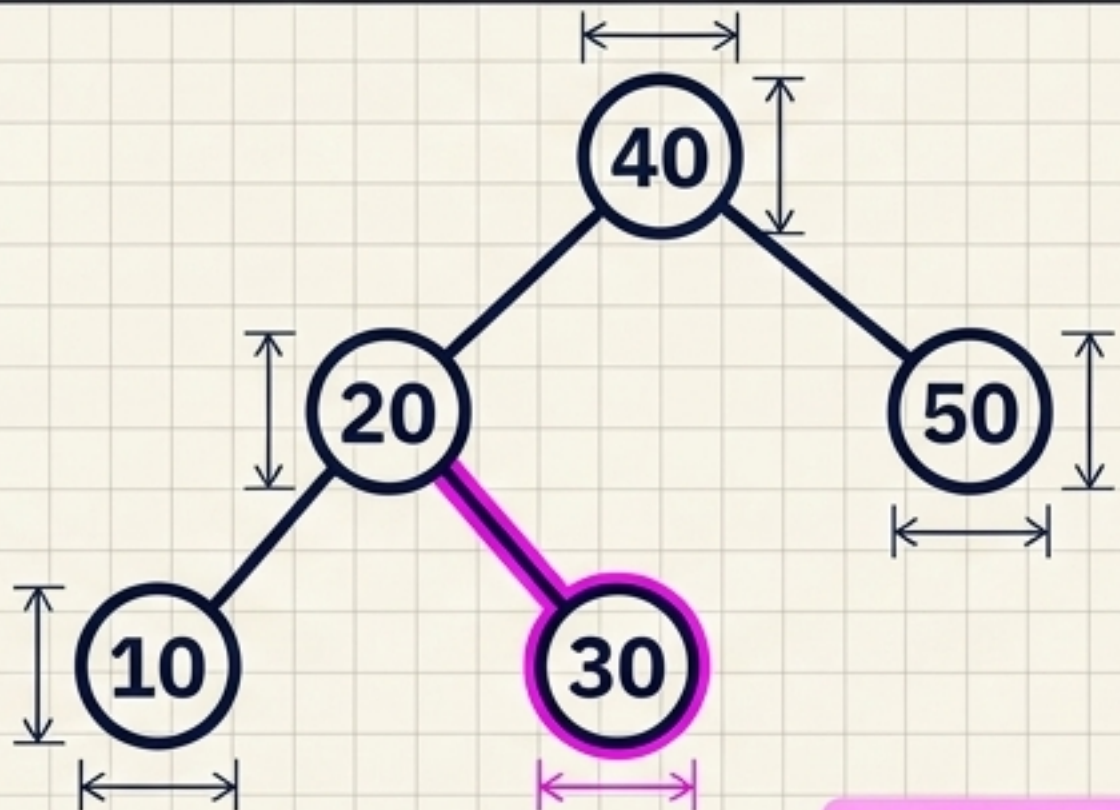
[エ]のヒント：左回転を「先に」やるのはどんな時？

LL Pattern (Straight)



✓ 一直線（1回で直る）

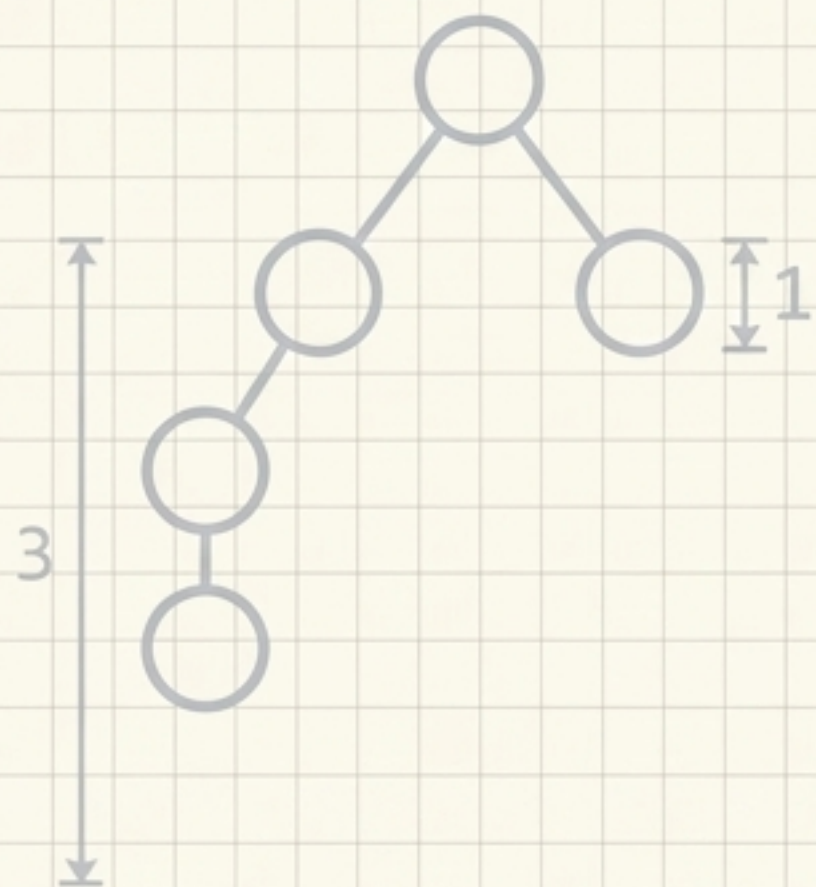
LR Pattern (Zigzag)



⚠ ジグザグは危険

Context:	if([ウ]) の中で、先に左回転を行う条件が if(h2が0より大きい)。
Translation:	問題文の条件「左側の子ノードUについて、Uの右側の部分木の方が高い場合は…」
Hint Prompt:	h2 にはどんな計算式が入れば、Uの「右の方が左より高い」ことを判定できる？(h1の計算式を参考に！)

[オ][カ]のヒント：コードの後半は前半の「鏡」

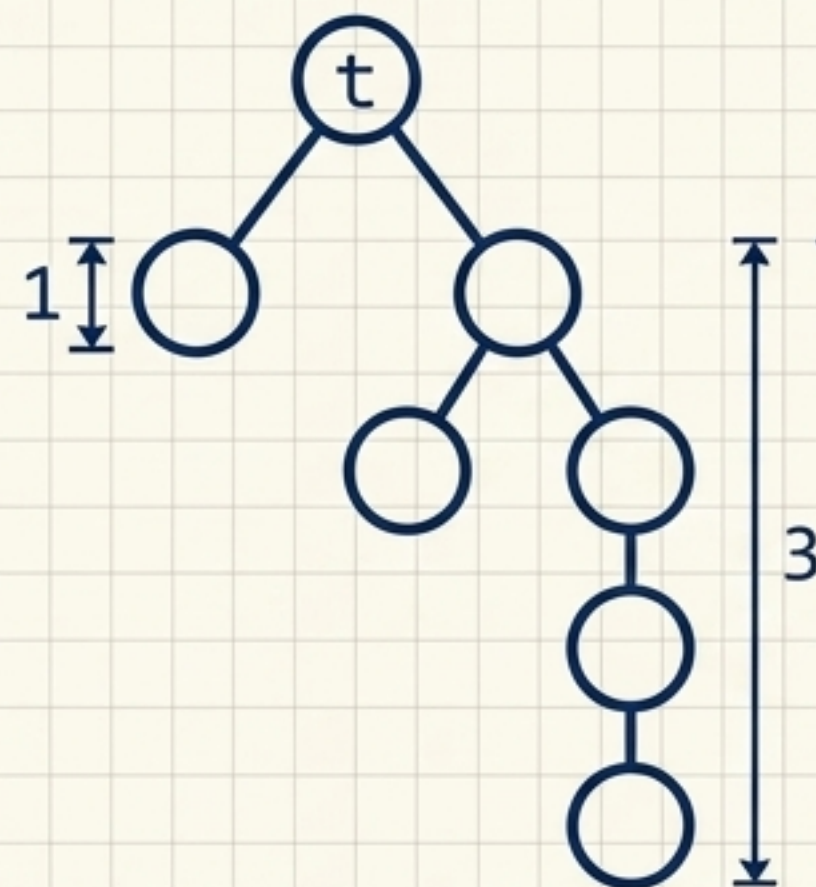


[オ]のヒント:

$h1 = (\text{左の高さ}) - (\text{右の高さ})$ でした。

今度は右側が長すぎて左回転が必要なケースです。

$h1$ はどんな値になる？ (プラス？ マイナス？)



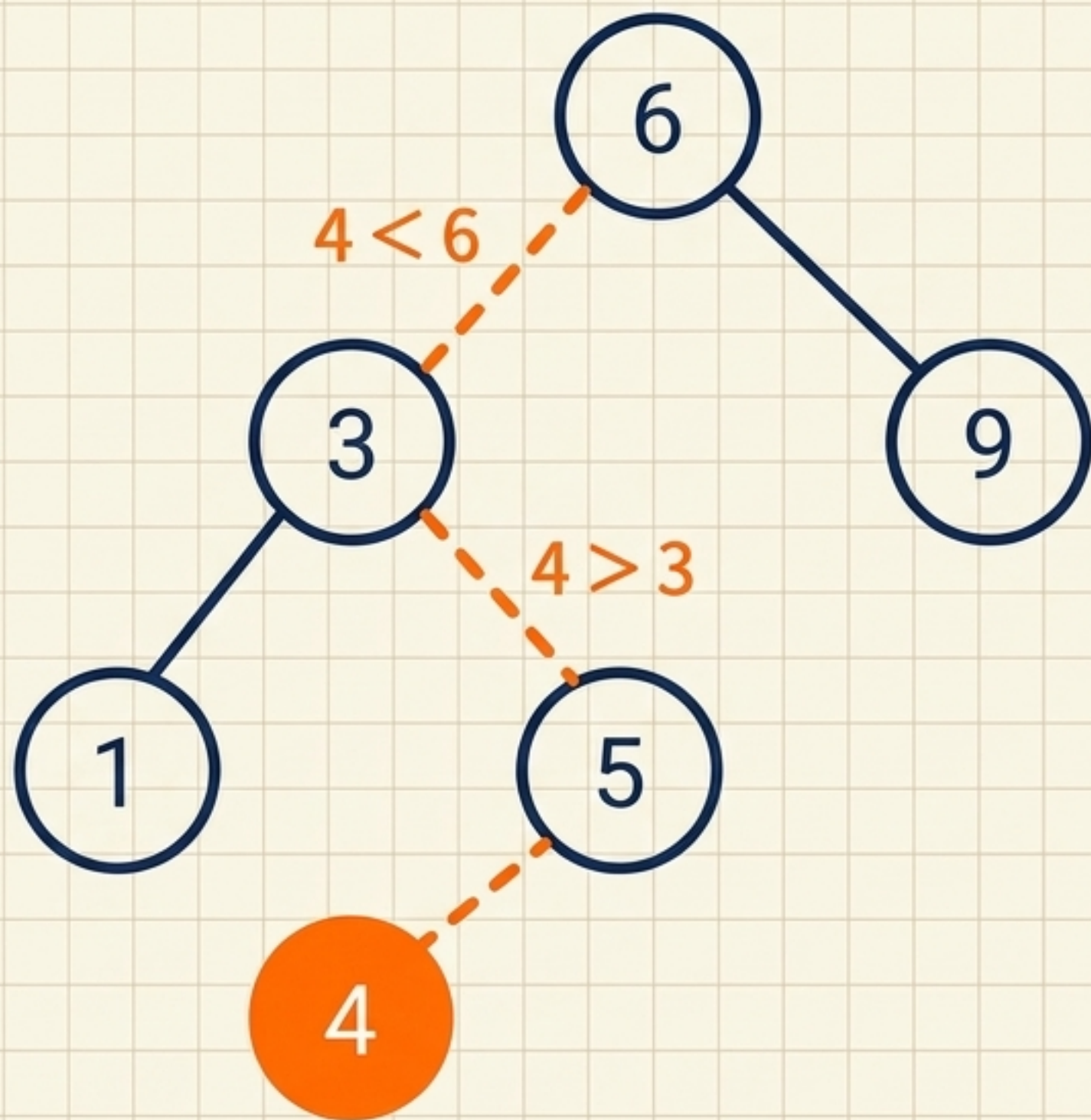
```
elseif([オ])  
  h3 ← [カ]
```

[カ]のヒント:

今度は右側の子ノード(V)の「左の方が高い」かどうかを判定したい。

$h2$ の式をどう左右反転させればいい？

設問2(2) トレース開始：まずは insert(r, 4)



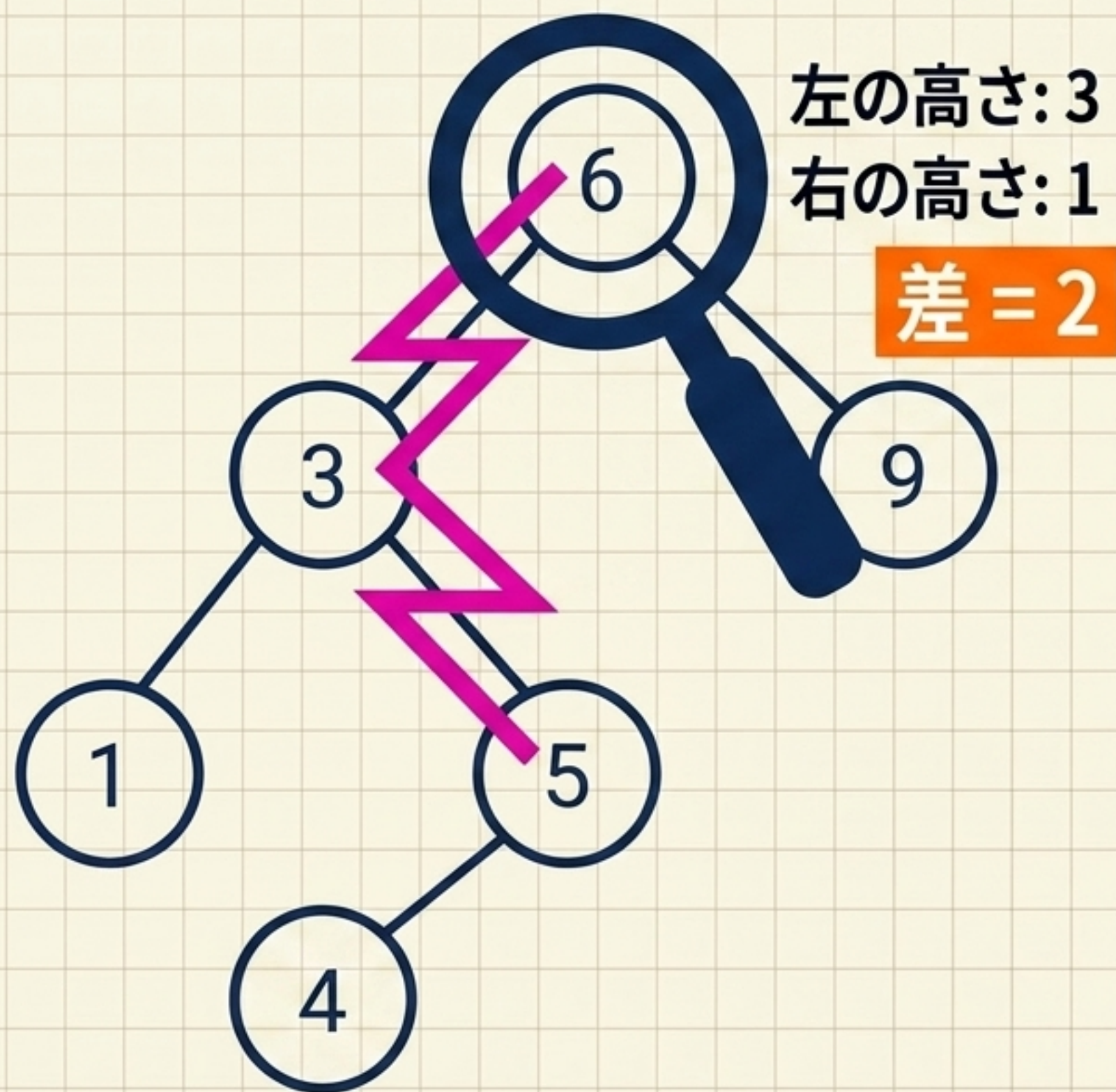
Action Step:

木を辿るルール「左<親<右」に従って「4」を入れる場所を探そう。

Check Point:

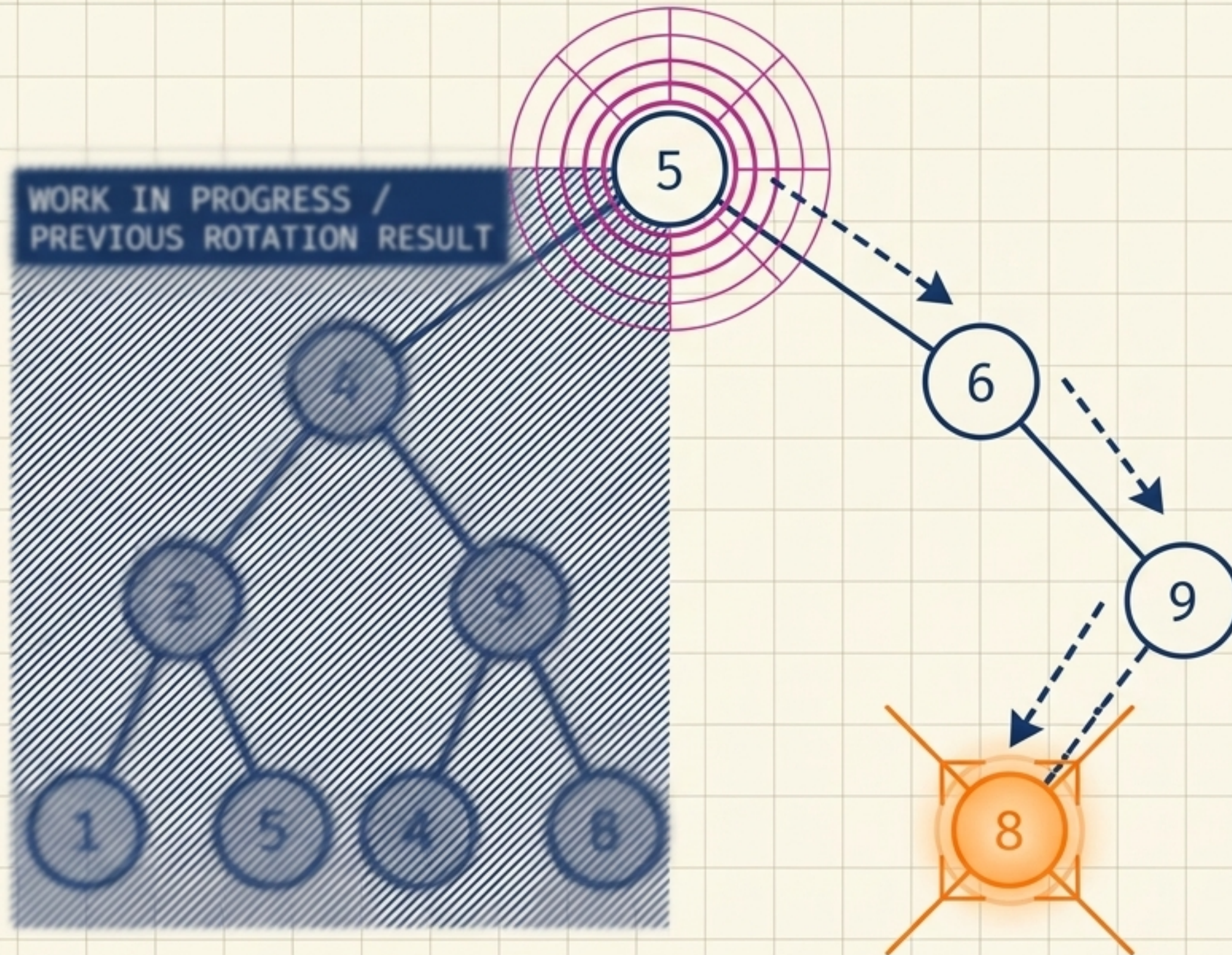
4を入れた後、下から順に親へと遡り、「条件Bal」が崩れていないか高さの差を数えてみよう。

設問2(2) 最初の試練：バランス崩壊！



- Status: ノード6の時点で、
左(3) - 右(1) = 2。崩れました！
- Diagnosis: 形を見てみよう。
「左に重い」が、その子は「右に重い」...これは[エ]で見たジグザグパターンだ！
- Challenge: どのノードを軸にして、
どういう順番で回転させるんだった？（問題文の手順(1)を読んで図を描き直そう）

設問2(2) 最終ステップ：insert(r, 8)



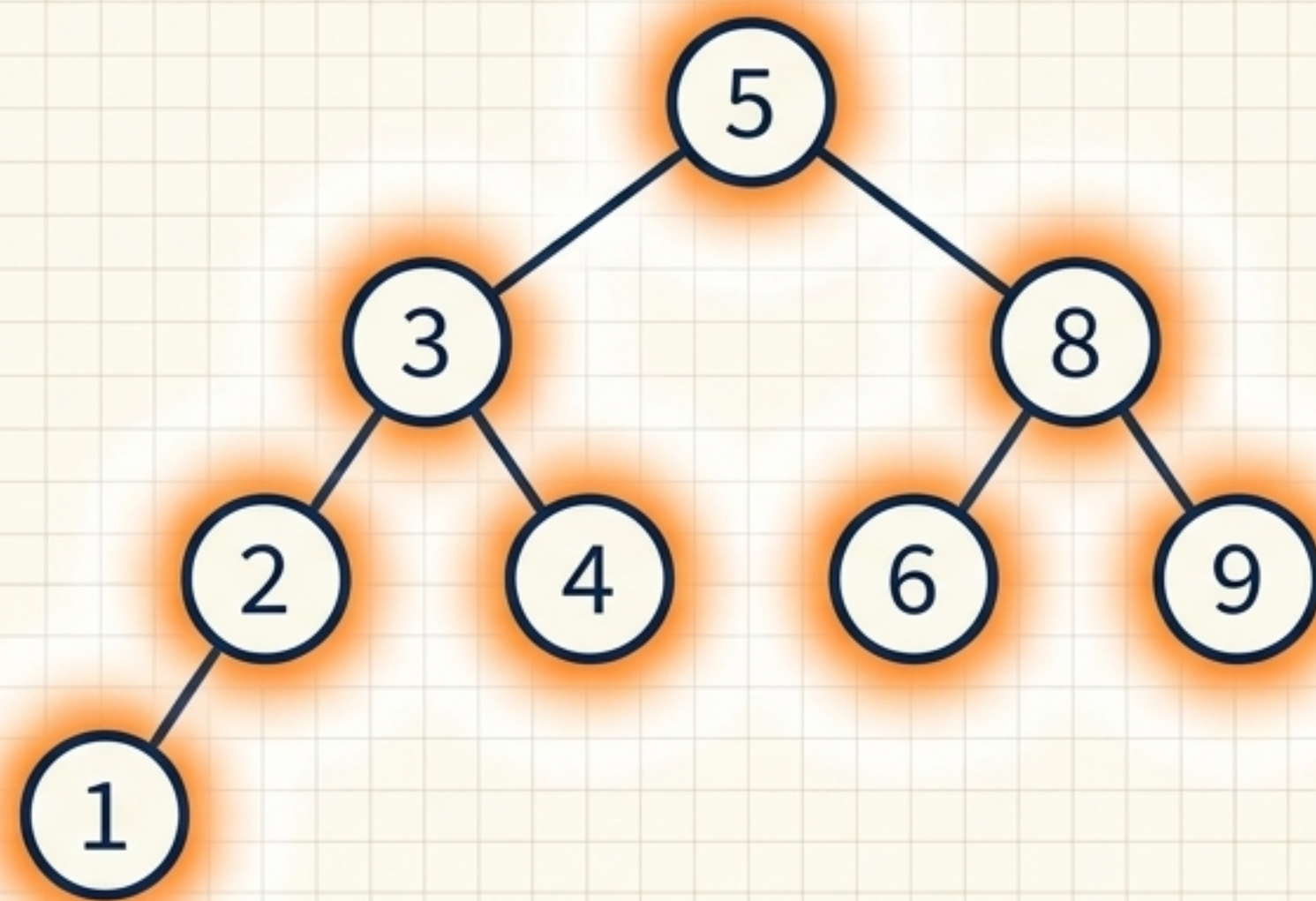
Action Step:

先ほど自分で作り直した新しい木の根からスタートし、「8」を正しい位置に挿入しよう。

Final Check:

8を入れた後、再び「条件Bal al」をチェック。今度は右側が長すぎるはず。最後に回転を適用して、完成図を描き出そう！

[キ]のヒント：なぜこんな面倒な「回転」をしたのか？



$O([\neq])$

Synthesis: 挿入するたびに balance 関数で回転を行い、「条件Bal」を維持し続けました。

The Aha Moment: その結果、木は常に「ほぼ完全な2分探索木」に近い状態に保たれます。ということは、探索や挿入にかかる最悪の計算量は、スライド3で考えた[イ]と同じになるはずです！

アルゴリズム攻略の鉄則

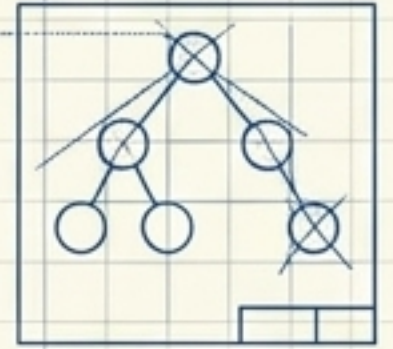


1. ルールを図で描く



2. 極端なケースを想像する

```
node.left = temp;  
node.left = roate;  
node.rght = 8;  
node.right= temp;  
...
```



3. 変数と図を翻訳する

プログラムの字面だけを追うと迷子になります。
「コードが木をどう動かそうとしているのか」、
図を描いて翻訳するクセをつけましょう。

さあ、実際の解答用紙に答えを書き込んでみましょう！