

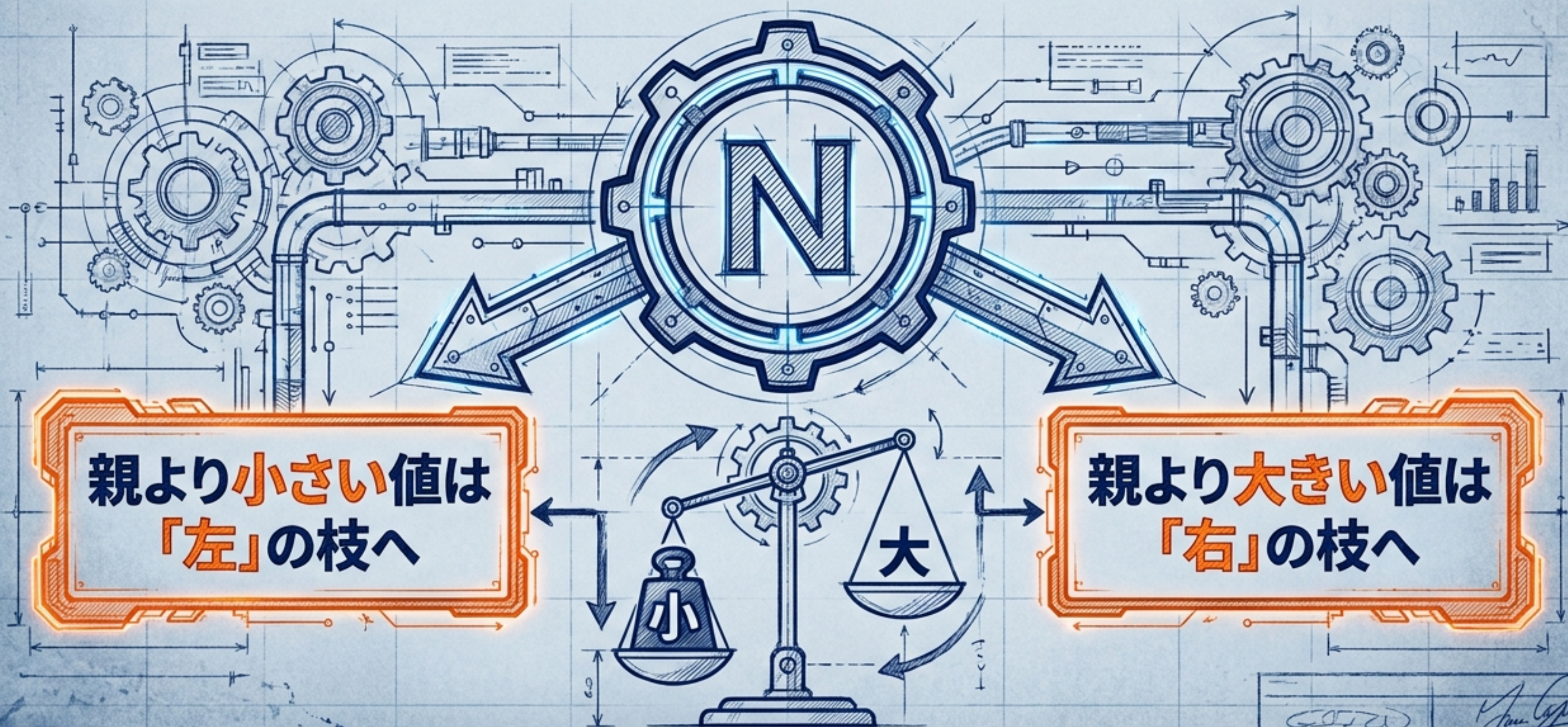
2分探索木 完全攻略ガイド

自力で解ける！設問ア～ケ。
設問4の考え方とヒント

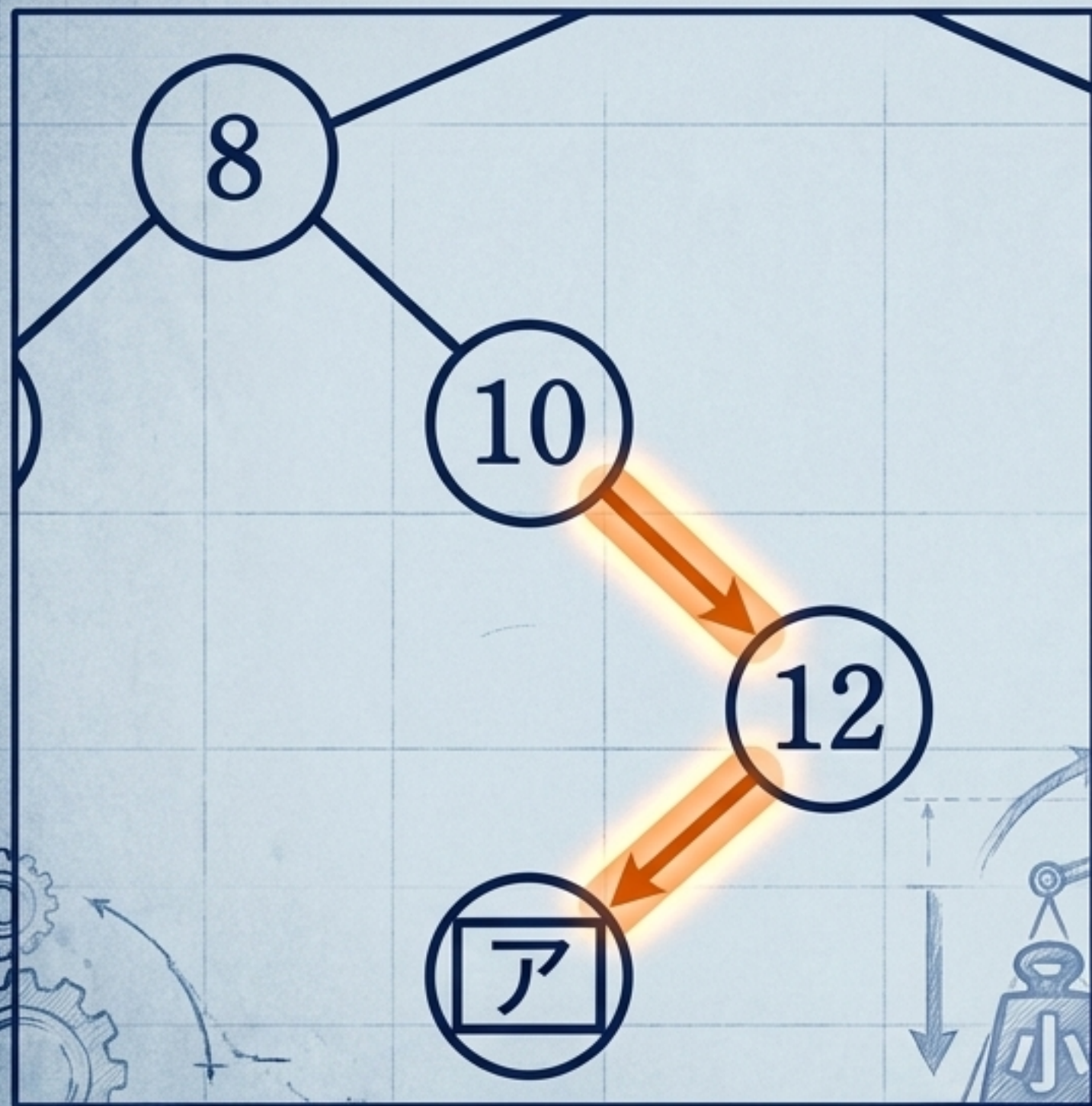
[?]

全てを支配する「基本ルール」

迷ったら必ずこの設計図に戻ること。木の「どこをとっても」この関係が成立する。



空欄[ア]のヒント：範囲を絞り込む



Thinking Steps

Step 1: ノード10との関係は？
(10の「右」にあるということは…?)

Step 2: ノード12との関係は？
(12の「左」にあるということは…?)

結論：
「〇〇より大きく、かつ〇〇より小さい自然数」は？

空欄[イ]のヒント：探索の分かれ道

```
elseif([イ])  
    return search(k, p.left)
```

左 (p.left) に進む!

左に進むためには、
kとp.keyの関係はどうあるべきか?

探したい値
k

今の値
p.key

空欄[ウ][エ]のヒント:ノードの挿入



空欄[ウ]:空き地に到達したら

- 状況:pとnullが等しい=空き場所
- ミッション:ここに新しいデータを保存するための「実体」を作る
- ヒント:問題文の「表1周辺」を確認!

表1 構造体 Node の構成要素

構成要素	説明
key	キー値
left	左子ノードへの参照
right	右子ノードへの参照



空欄[エ]:関数の総仕上げ

- 状況:プログラムの最後の行
- ミッション:呼び出し元に結果を「返す」
- ヒント:問題文の指示「...の根のノードへの参照を返す」に注目!
この関数内で木の構造を保持している変数はどれ?

空欄[オ][カ]のヒント: シンプルな削除(子が1つ)

自分が消えたら、残った「唯一の子」に後を継がせる

Before

After

P

右の子

p.left が null

右の子

[オ]: p.left が null (左の子がない) なら、誰を引き上げる?
[カ]: p.right が null (右の子がない) なら、誰を引き上げる?

空欄 [キ] のヒント： 複雑な削除の仕上げ（子が2つ）

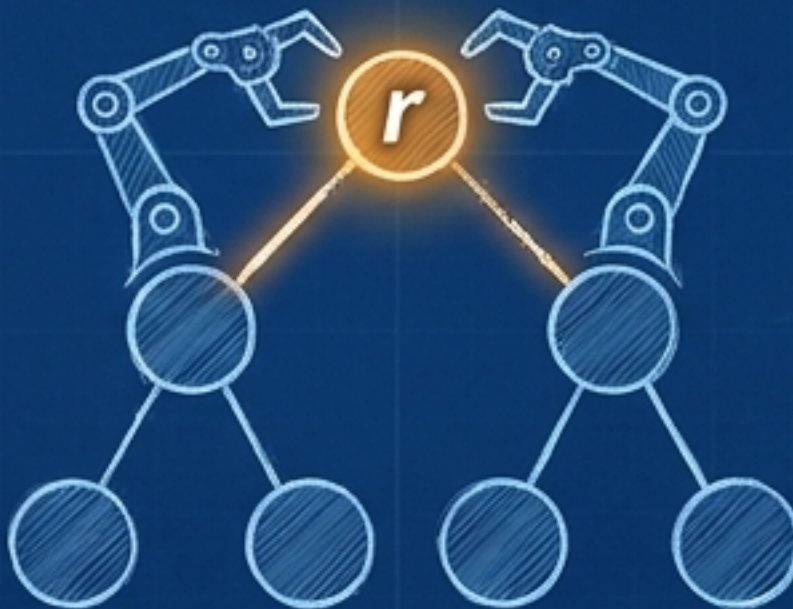
```
else
  p.left ← extractMaxNode(p.left)
  r ← extractedNode
  r.left ← p.left
  r.right ← p.right
  [キ]
endif
```

Step 1:
代役(r)の選出



左部分木の中で最大の
ノードを代役 r にする

Step 2:
腕の付け替え



元のノード p が持っていた
左右の枝を r に繋ぎ直す

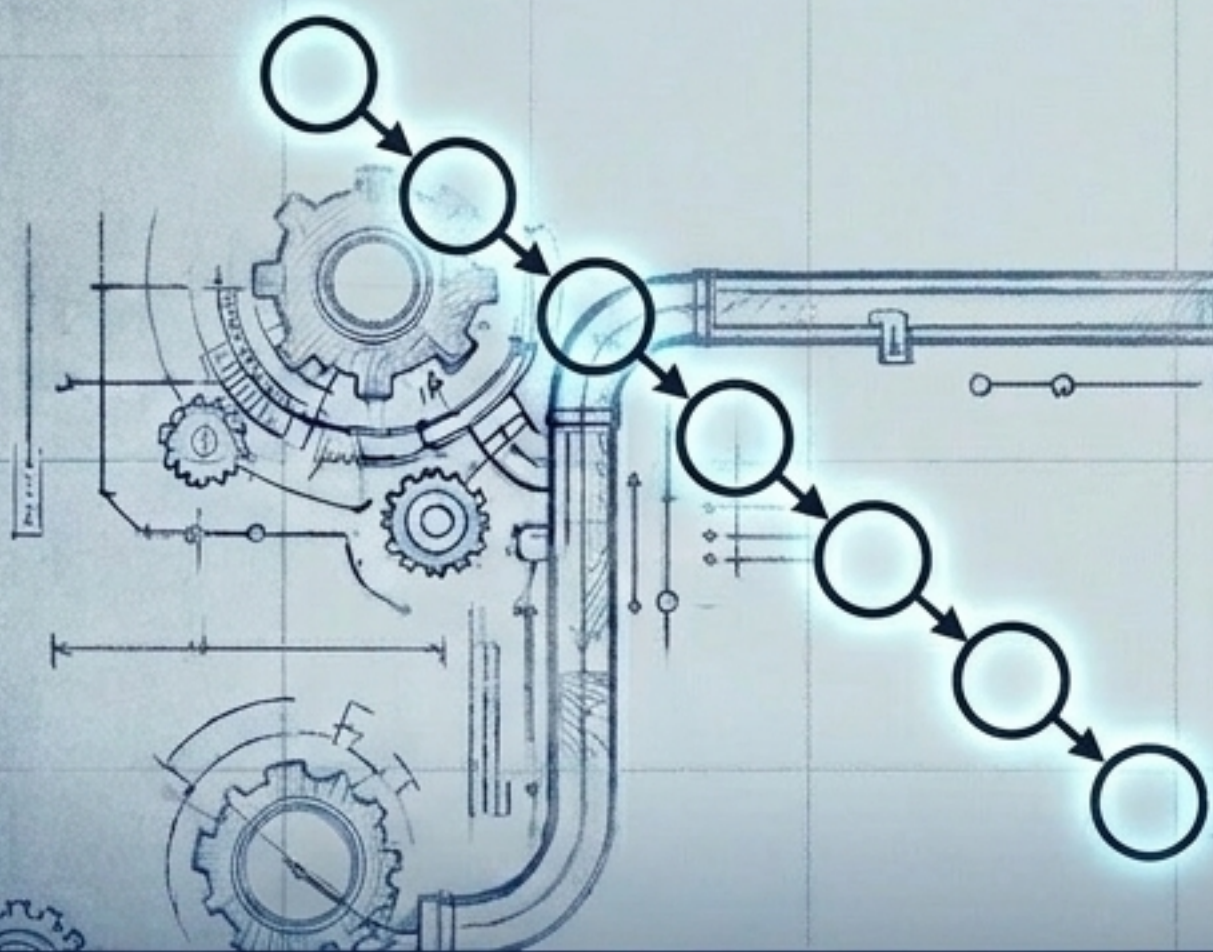
Step 3: 空欄 [キ]
新しい根の決定



この関数は最後に「変数 p」を返す。新しい親である「r」を正しく返すためには、直前で p に対してどんな代入操作が必要？

空欄[ク][ケ]のヒント:木の形と計算量

最悪のケース(一直線)

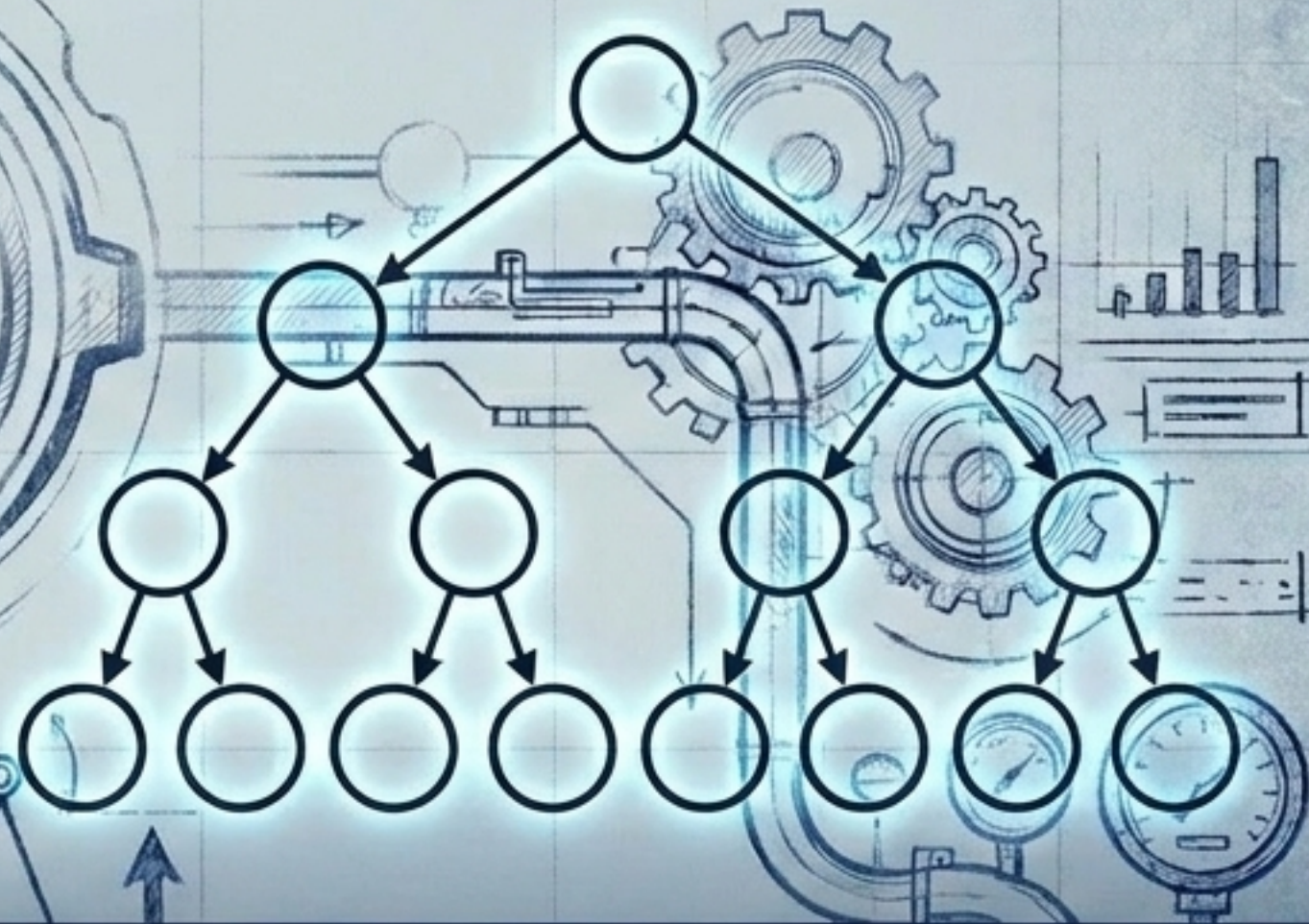


探索の仕組み:先頭から順番に1つずつ確認するしかない

ノード数 n と最大比較回数の関係は?

$$= O([ク])$$

理想のケース(完全2分木)



探索の仕組み:1回比較するごとに、探す範囲が「半分」になる

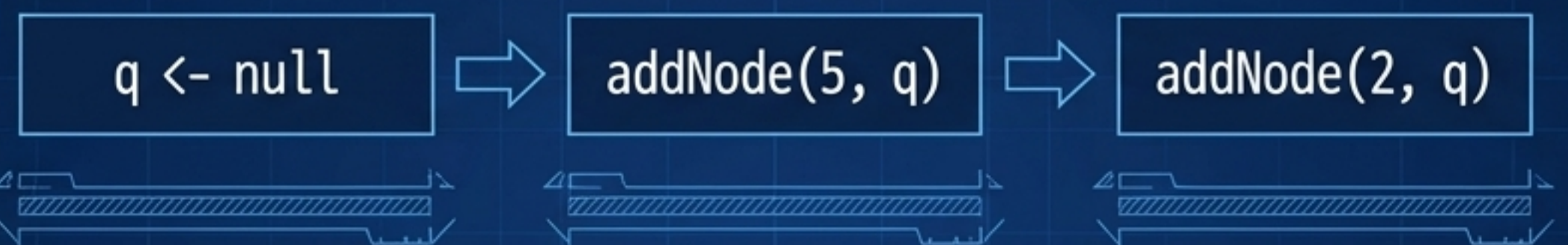
毎回半分 ($1/2$) になる性質を表す高校数学の記号は?

$$= O([ケ])$$

設問4のヒント（挿入編）：木を育てよう

頭の中で考えず、必ず紙に書きながら1ステップずつ進めましょう。

```
q <- null
q <- addNode(5, q)
q <- addNode(2, q)
q <- addNode(7, q)
q <- addNode(1, q)
q <- addNode(8, q)
q <- addNode(4, q)
q <- addNode(3, q)
q <- addNode(12, q)
q <- removeNode(5, q)
q <- removeNode(7, q)
```



Stage 1:

最初は何もありません。
まっさらな状態。

Stage 2:

最初の値「5」が木の
「根」になります。

Stage 3:

根（5）と比べ小さいので
左へ。常に一番上から
ルールに従って配置します。

続く 7, 1, 8...
はどこに入るか？
基本ルールに従
って
最後まで描いて
みよう！

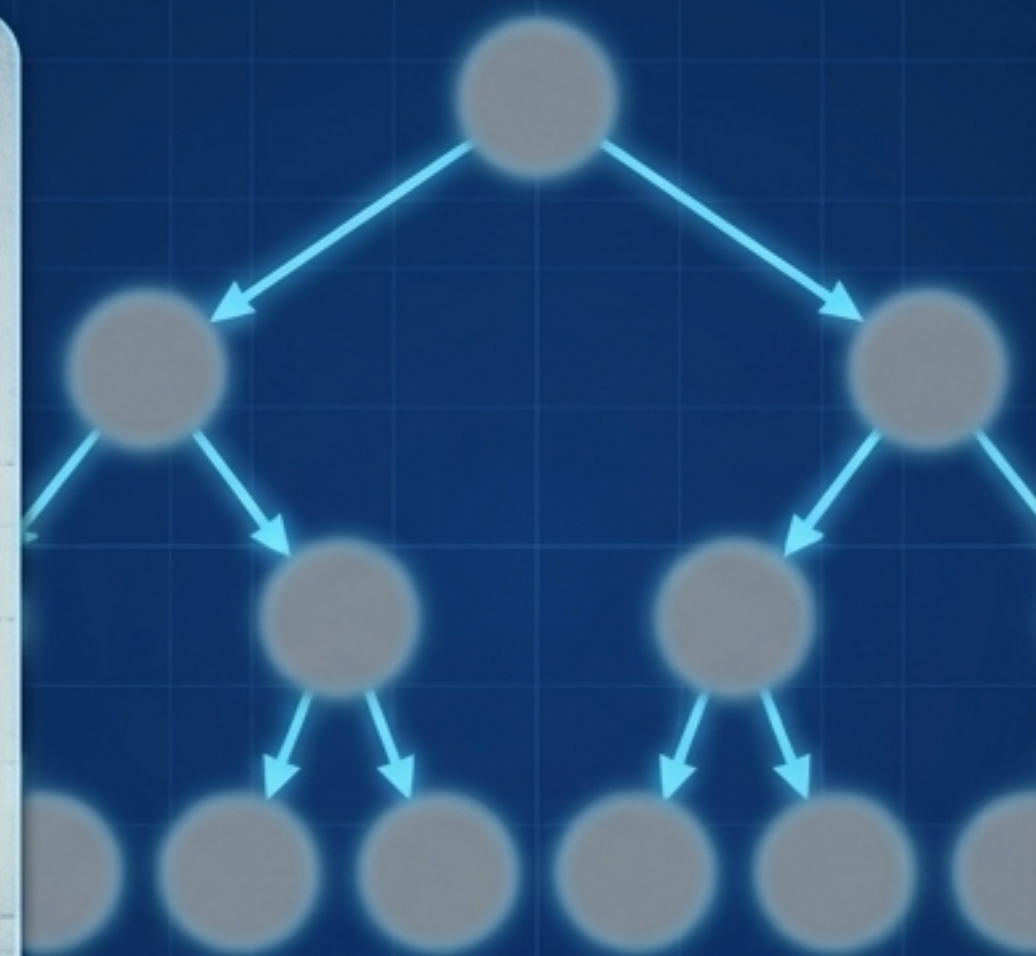
設問4のヒント（削除編）：パターンの見極め

Check 1: removeNode(5, q)

✓ 「5」のノードに子はいくつありますか？



→ 対応する削除ルール
(スライド6または7) を
適用して木を再構築！



Check 2: removeNode(7, q)

木が変化した後、「7」のノードに子はいくつありますか？



→ 再度正しいルールを適用すれば、最終形態の完成です！

ポイント：消したいノードの子の数を冷静に数えること。