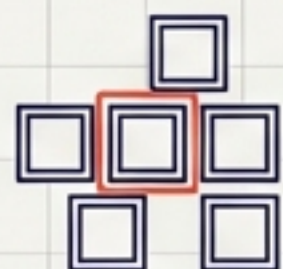


アルゴリズム探索の極意

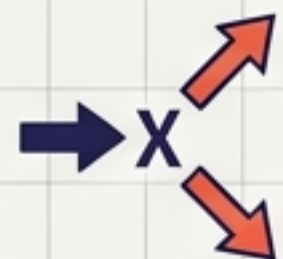
「選ぶ」と「選ばない」が作る分岐の世界

【台本】 本日はアルゴリズムの世界へようこそ。このスライドは、応用情報技術者試験の良問をベースに、皆さんに『アルゴリズムを組み立てる思考力』を身につけてもらうためのものです。スライド上に答えは書いてありません。ヒントを手がかりに、一緒に謎を解いていきましょう。

ミッション：目標「X」に最も近づけよ！



○ Rule 1: 複数のブロックからいくつかを選ぶ



○ Rule 2: 合計値をターゲット「X」に一番近づける（超えても足りなくてもOK）



○ Rule 3: 同じブロックは1回しか使えない

Target
X = 100



10

34

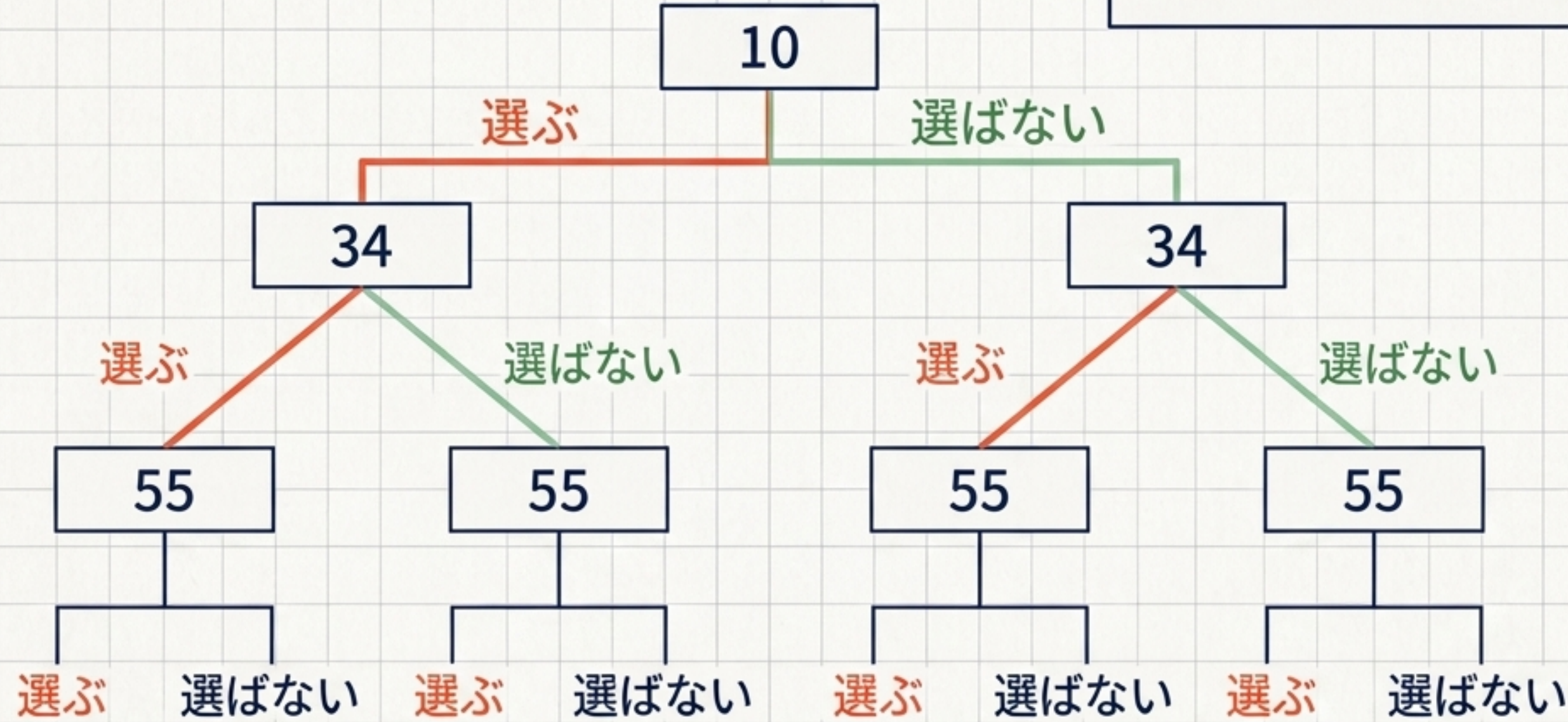
55

77

【台本】 まずはミッションの確認です。たくさんの数字の中からいくつかを選んで、目標の数字に最も近づけるゲームだと考えてください。人間なら勘で選べますが、コンピュータにはどうやって探させれば良いのでしょうか？ 確実なルールを作る必要があります。

探索地図を作る：分岐のルール

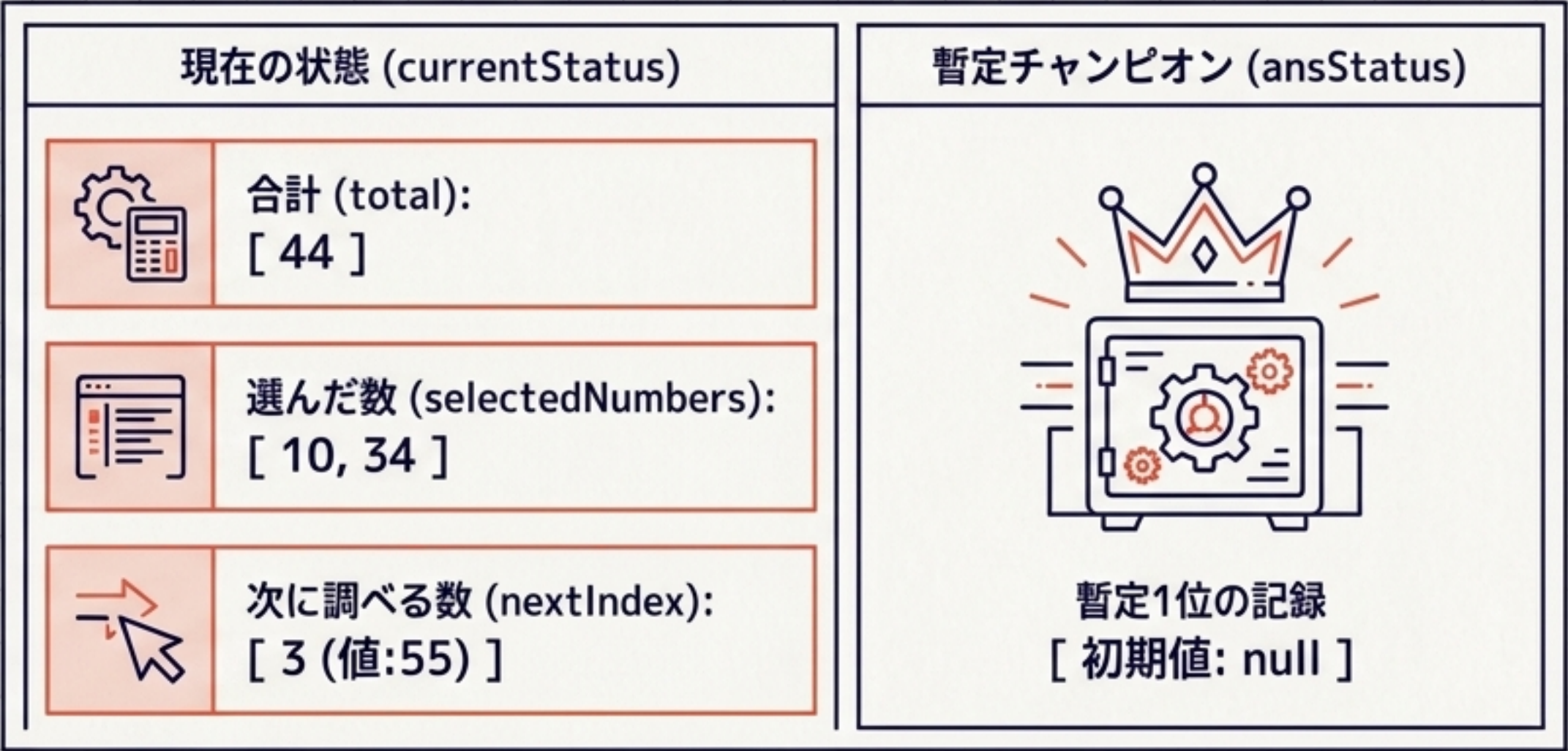
コンピュータの必勝法
= すべての組み合わせを網羅する



【台本】コンピュータの得意技は『全部調べること』です。最初の数字を選ぶか、選ばないか。次の数字を選ぶか、選ばないか。この二者択一を延々と繰り返すと、枝分かれする『木』のような探索地図（状態空間木）が完成します。この地図をどう歩くかが、今回のテーマです。

探索者のステータス（データ構造）

迷子にならないよう、「今の状態」と「暫定1位の記録」を常に持ち歩く。



【台本】迷子にならないよう、各地点で記録しておくべきデータを定義します。今の合計はいくつか？次に調べる数はどれか？そして、暫定1位の記録（ansStatus）を常に持ち歩きながら探索を進めます。このステータスを更新していくのがプログラムの役割です。

[設問1 ヒント] 探索のメインループ

```
while( [ ア ] )  
    if( ansStatus が null である ) {  
        [ イ ]  
    }  
    elseif( 今の状態の方が目標に近い ) {  
        [ ウ ]  
    }  
}
```

? [ア]のヒント: 探索が終わるのは、待機列（キュー）がどうなった時？（関数名 isEmpty() に注目）

🗨 [イ][ウ]のヒント: 暫定チャンピオン（ansStatus）が空の時や、今の状態（currentStatus）の方が目標に近い時、何をすべき？

【台本】 いよいよプログラムの穴埋めです。[ア]はループを続ける条件。探索すべき状態が入っている『キュー』がどんな状態になるまで続けるのでしょうか？ [イ][ウ]は記録更新の瞬間です。今の状態をどう扱えばいいか考えてみましょう。

[設問1 ヒント] 次の道へ進む条件

```
if( [ エ ] )
```

```
    enqueue( nextStatus1(currentStatus) )
```

```
    enqueue( nextStatus2(currentStatus) )
```

ここに状態を入れる前に
確認すべきことは？

常に「次の数」を選ぶわけではない。

「次に調べる数」のインデックスがどうなっているか、表1を見直してみよう。

【台本】 [エ] は、新しい状態を作ってキューに追加する直前のチェックです。

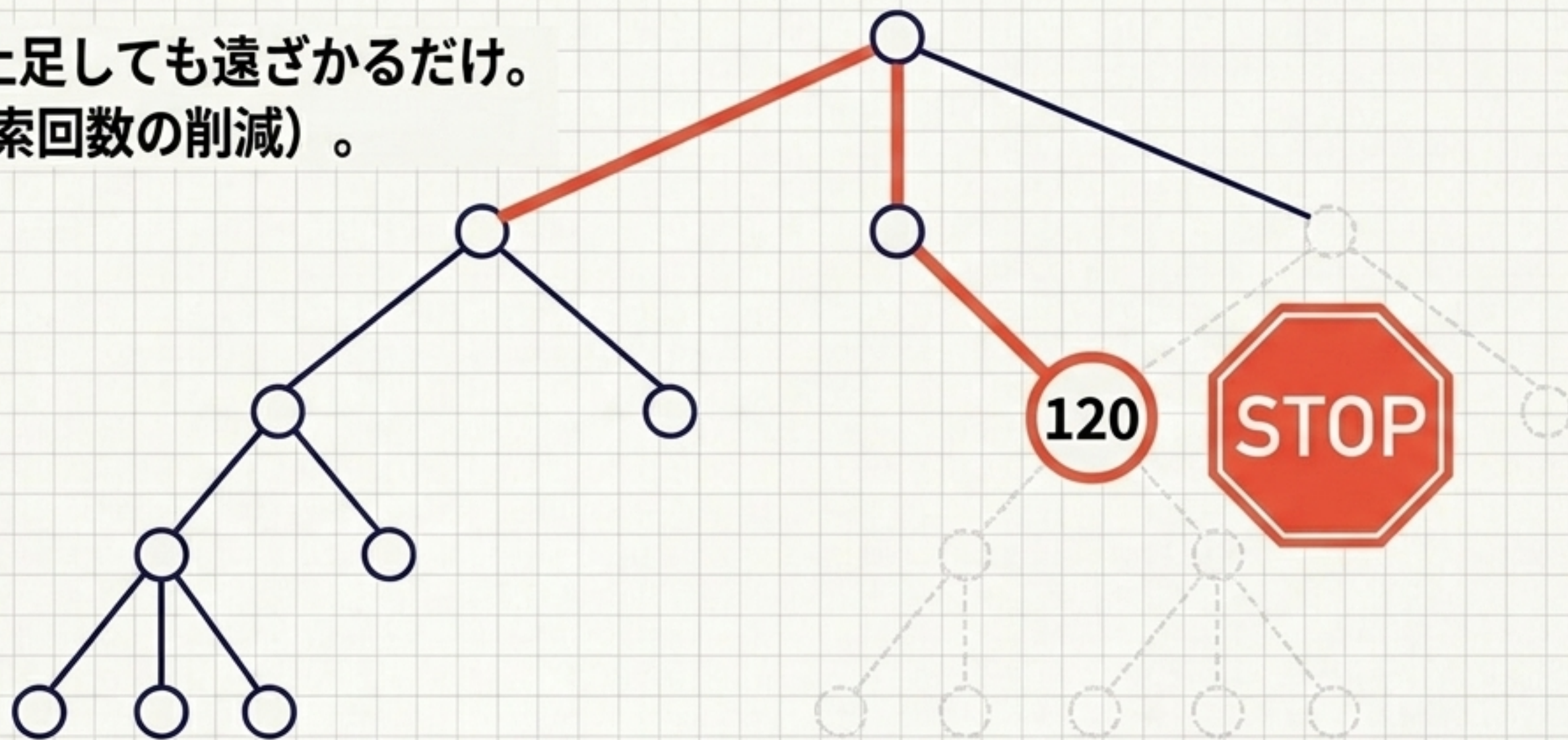
永遠に次の数を選び続けることはできませんね。

『次に調べる数』が存在するかどうかを判定する必要があります。

インデックス変数がどうなればよいのでしょうか？

賢く探す：「枝刈り」のテクニック

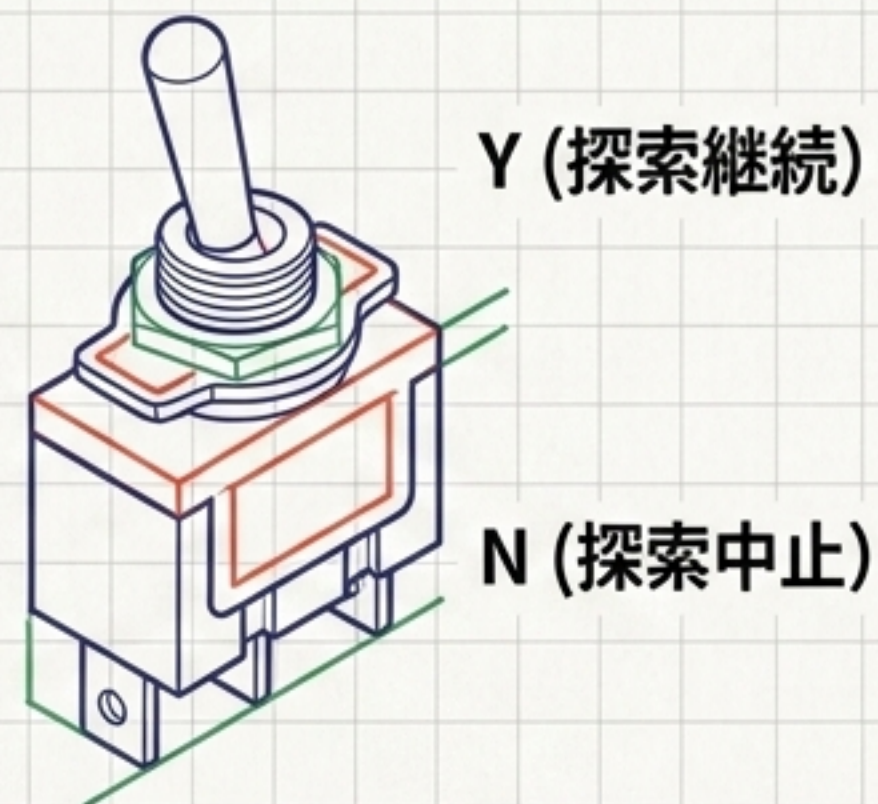
目標Xを超えたら、それ以上足しても遠ざかるだけ。
探索を即座に打ち切る（探索回数の削減）。



【台本】木が大きくなると、全部調べるのは大変です。もし目標が100で、今の合計が120だったらどうしますか？これ以上足しても120より大きくなるだけで無駄ですよ。この無駄ルートを切り捨てることを『枝刈り (Pruning)』と呼びます。

【設問2 ヒント】 枝刈りの実装

```
if( currentStatus.total が target 以上である ) {  
    nextFlag ← [ オ ]  
} else {  
    nextFlag ← [ カ ]  
}
```



今の合計が目標X以上なら、探索はどうすべき？
フラグに「Y」か「N」を当てはめてみよう。

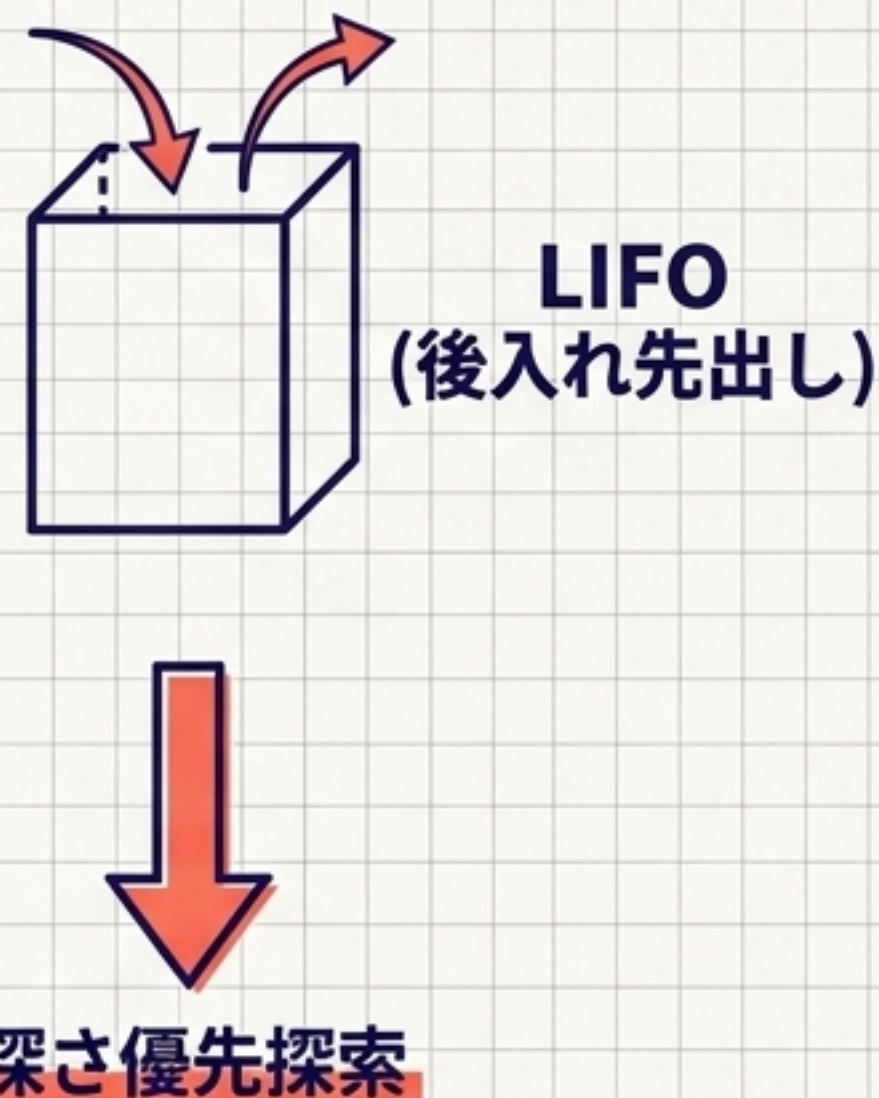
【台本】 設問2です。追加された枝刈りのコードを見てみましょう。条件式は『合計が目標以上である』です。この場合、探索を続けるべきでしょうか、中止するべきでしょうか？『Y』か『N』、どちらが入るか考えてみてください。

探索の2大ツール：キュー vs スタック

QUEUE

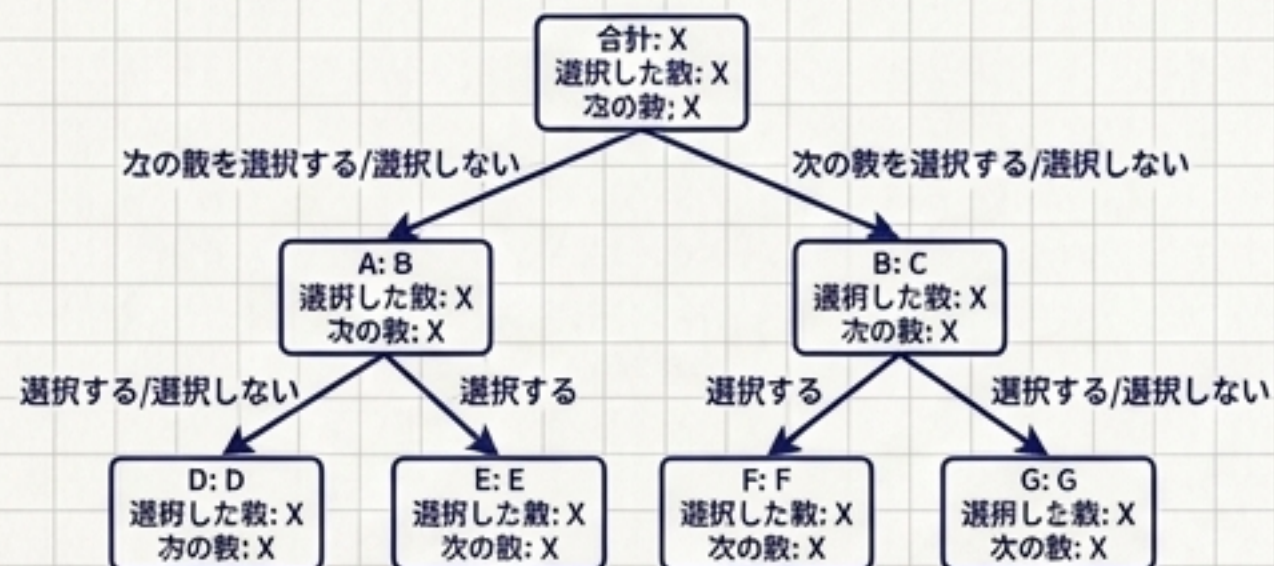


STACK

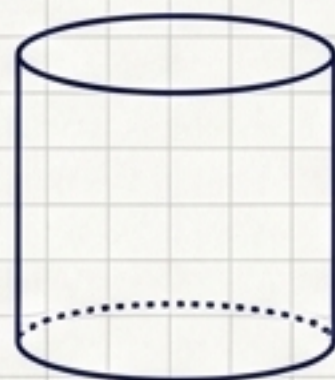


【台本】次に進む前に、状態を保存する『入れ物』の性質を比較しましょう。入れた順に出てくるのがキュー、最後に入れたものが最初に出てくるのがスタックです。この性質の違いが、探索の順番に決定的な違いを生み出します。

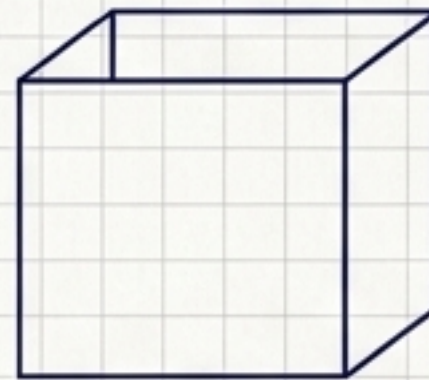
[設問3 ヒント] 探索ルートを追跡せよ



分岐では『左側』のノードから先に格納する！



キュー用



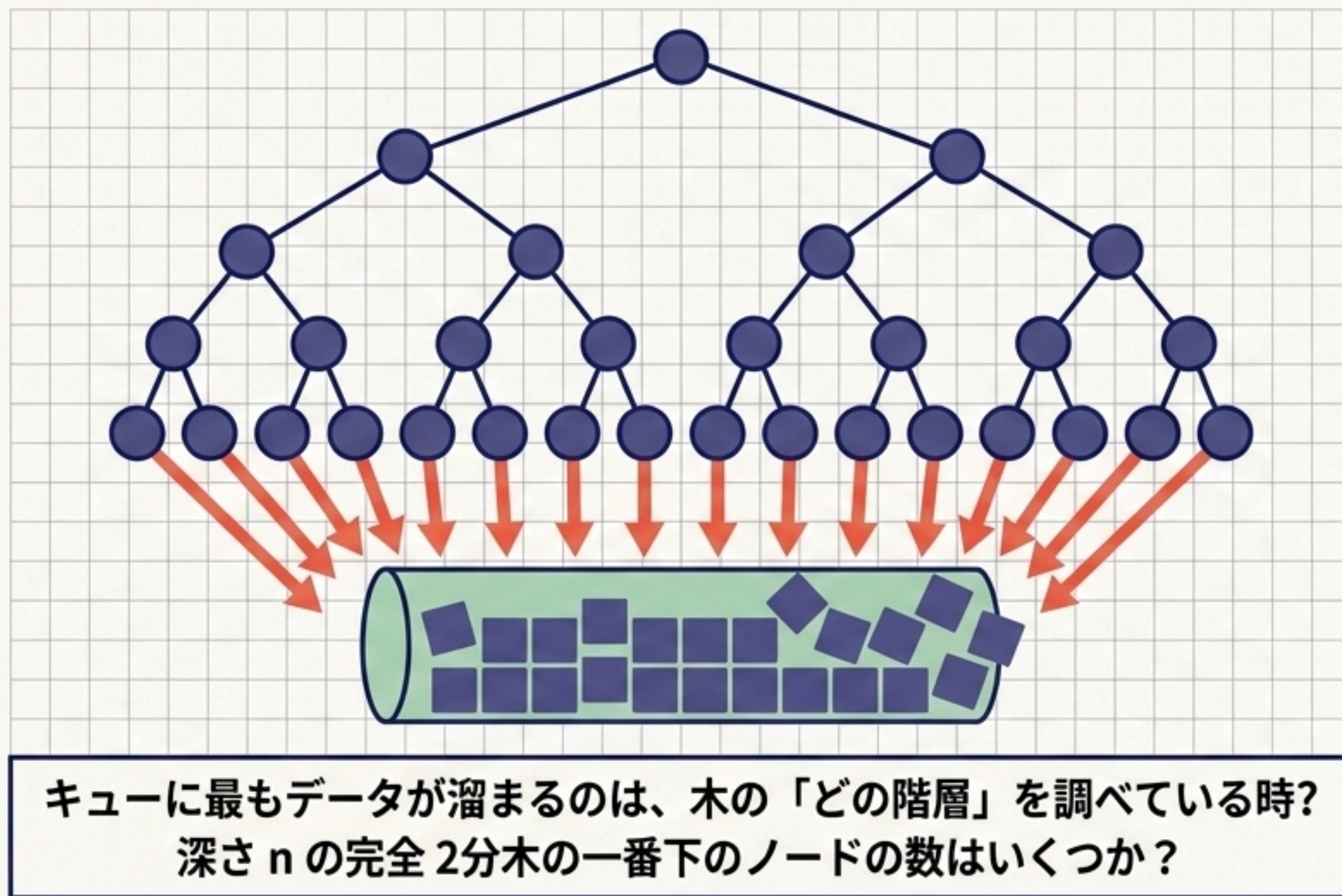
スタック用

キューの場合：同じ階層を左から横に舐めるように進む（幅優先）。

スタックの場合：一番新しいルートを奥へ奥へと深く進む（深さ優先）。

【台本】設問3です。実際にAから順に評価してみましょう。分岐では『左側』から先にデータ構造に格納するというルールに注意してください。キューの筒とスタックの箱をメモ帳に描いて、手作業でシミュレーションするのが一番の近道です。

[設問4 ヒント] メモリの限界を知る



キューに最もデータが溜まるのは、木の「どの階層」を調べている時?
深さ n の完全 2分木の一番下のノードの数はいくつか?

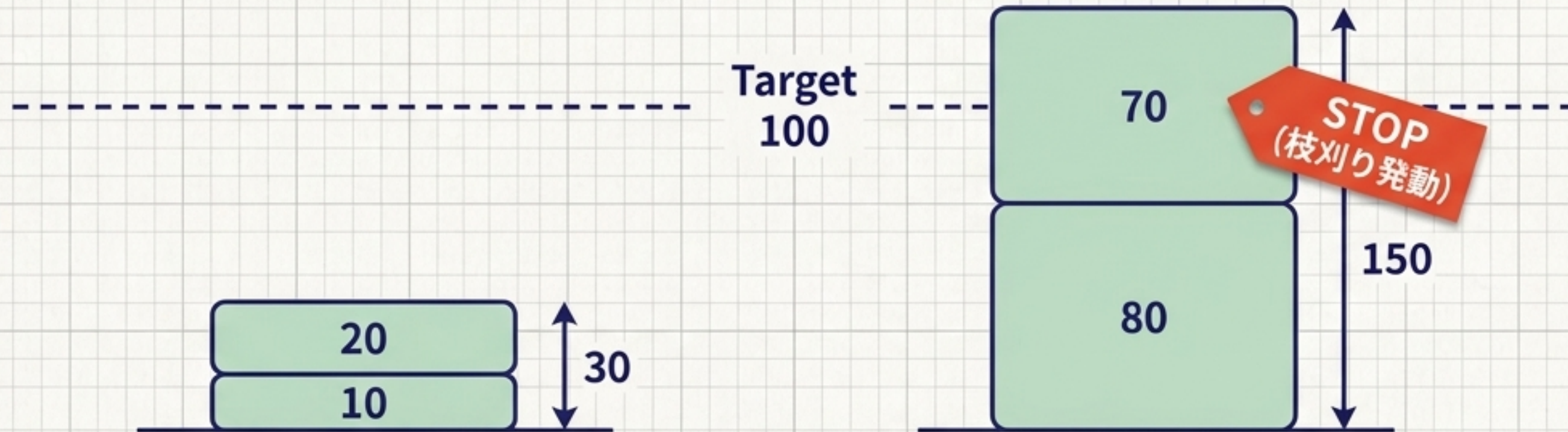
【台本】設問4はメモリ使用量の問題です。キューを使った幅優先探索では、同じ階層のノードが横並びで全てキューに入ります。木構造が最も横に広がるのはどこでしょうか?

中学校の数学を思い出して、深さ n の完全 2分木の最下層のノード数を計算してみてください。

[設問5 ヒント] さらなる最適化（事前処理）

パターンA：小さい数から足す（昇順）

パターンB：大きい数から足す（降順）

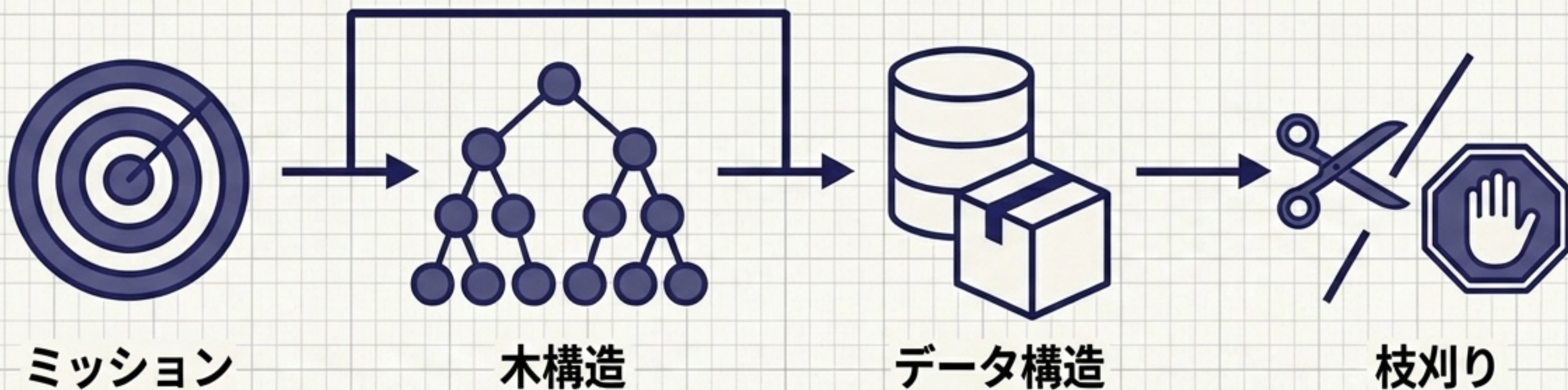


枝刈り（合計が目標X以上でストップ）をより「早く」発動させるには、探索前に数のリストをどう並べ替えておけばよい？

【台本】最後の設問です。

枝刈りは『目標を超えたら即終了』という強力な技ですが、10, 20...と小さい数から足しては、なかなか目標に届かず無駄な探索が続きます。では、探索を始める前に配列に対してどんな『準備』をしておけば、早く目標を超えて無駄な探索をカットできるでしょうか？

探求のまとめ：Algorithm Blueprint



1. 複雑な条件は「木構造」で視覚化できる。
2. キューとスタックが「探索の性格」を決定づける。
3. アルゴリズムの真の美しさは「無駄を省く（枝刈り）」ことにある。

【台本】お疲れ様でした！一見複雑なアルゴリズムの問題も、こうして視覚的な設計図（Blueprint）として捉え、ルールに沿って追跡していけば必ず構造が見えてきます。ぜひ、ご自身の解答をもう一度見直し、実際のコードの動きと照らし合わせてみてください。探索の世界を制覇するのはあなたです。