

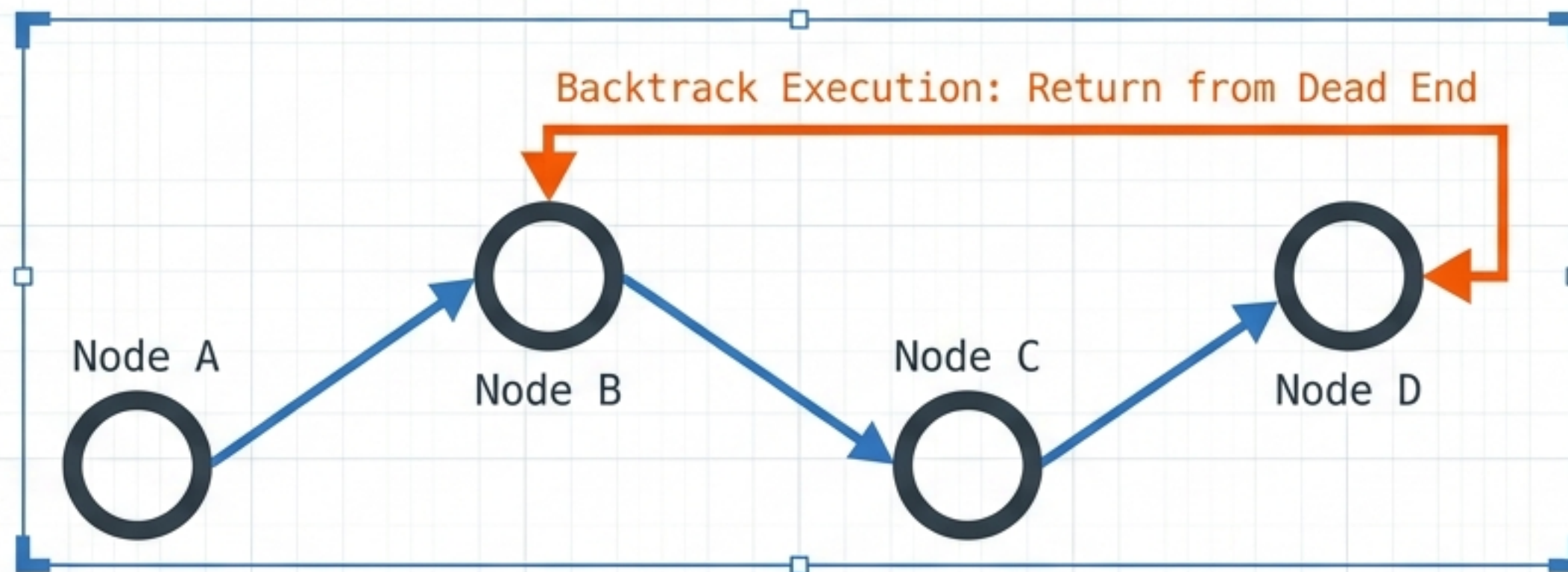
一筆書きアルゴリズムの解剖

令和3年度 秋期 応用情報技術者試験
午後 問3 (設問2~4 構造解説)

Algorithm Analysis & Design

※本資料に最終的な解答は記載していません。コードの挙動とデータ構造を紐解き、自ら解答を導き出すための思考プロセス解説資料です。

本アルゴリズム(directedE)の核心は、
深さ優先探索を応用した「バックトラッ
ク法」による経路構築にある。



【設問2】 The Prompt: 評価のサイクル

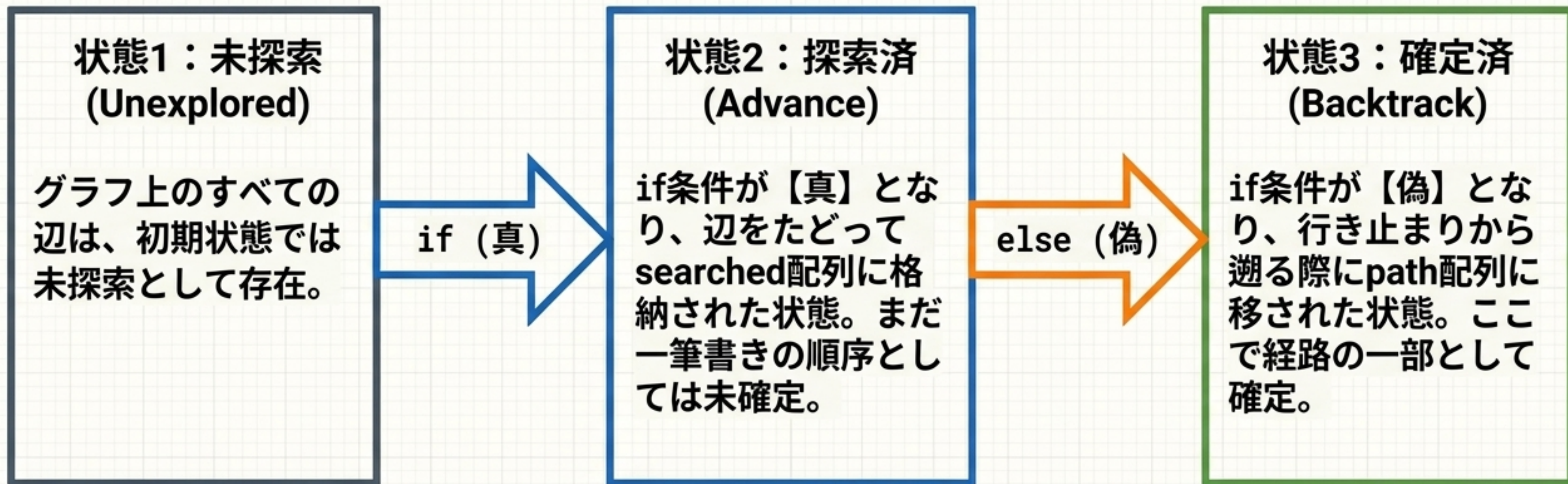
分析の焦点

図1のグラフにおける while 文中の if 文の実行回数。

アルゴリズムが「各辺に対してどのようなアプローチを行うか」を理解する。単に辺の数を数えるのではなく、「進む」処理と「戻る」処理のサイクルの観点から評価回数を導き出す。

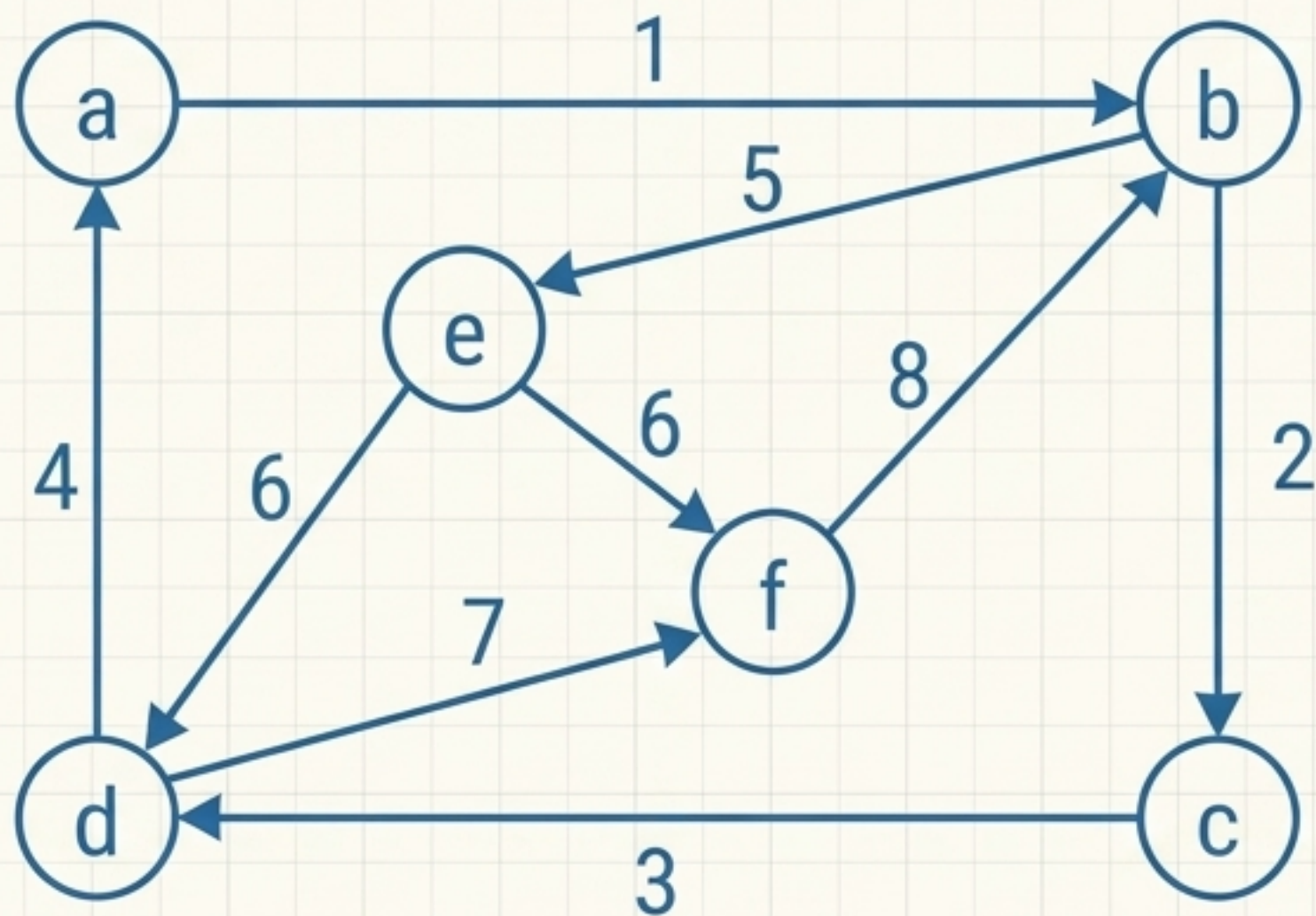
```
while ( ① last が 1 以上 )  
  if ( current[x] が [ア] でない )  
    // 未探索の辺をたどる (進む処理: Advance)  
  else  
    // 行き止まりを遡る (確定処理: Backtrack)  
  endif  
endwhile
```

【設問2】 The Concept: 辺のライフサイクル



【設問2】 The Deduction: 評価プロセスと辺の総数

対象グラフ：辺の総数 = 8



Rule 1: 「真」となる評価

未探索の辺を見つけて進むため、すべての辺に対して必ず1回ずつ「真」と評価される瞬間がある。

Rule 2: 「偽」となる評価

進むべき辺がなくなり、後戻りして確定させるために、すべての辺に対して必ず1回ずつ「偽」と評価される（行き止まりを確認する）瞬間がある。

Deduction Hint: 辺の総数に対して、評価は「何倍」行われるか？図の総辺数(8)とアルゴリズムの対称性から、必然的な実行回数が導かれる。

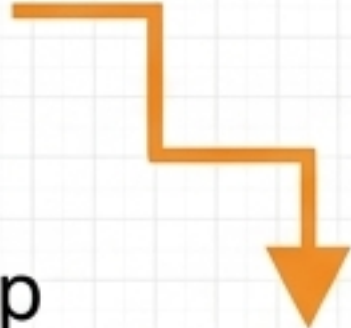
【設問3】 The Prompt: 終了条件と例外

分析の焦点

「一筆書きができない」強連結な有向グラフでプログラムを実行した際の挙動。

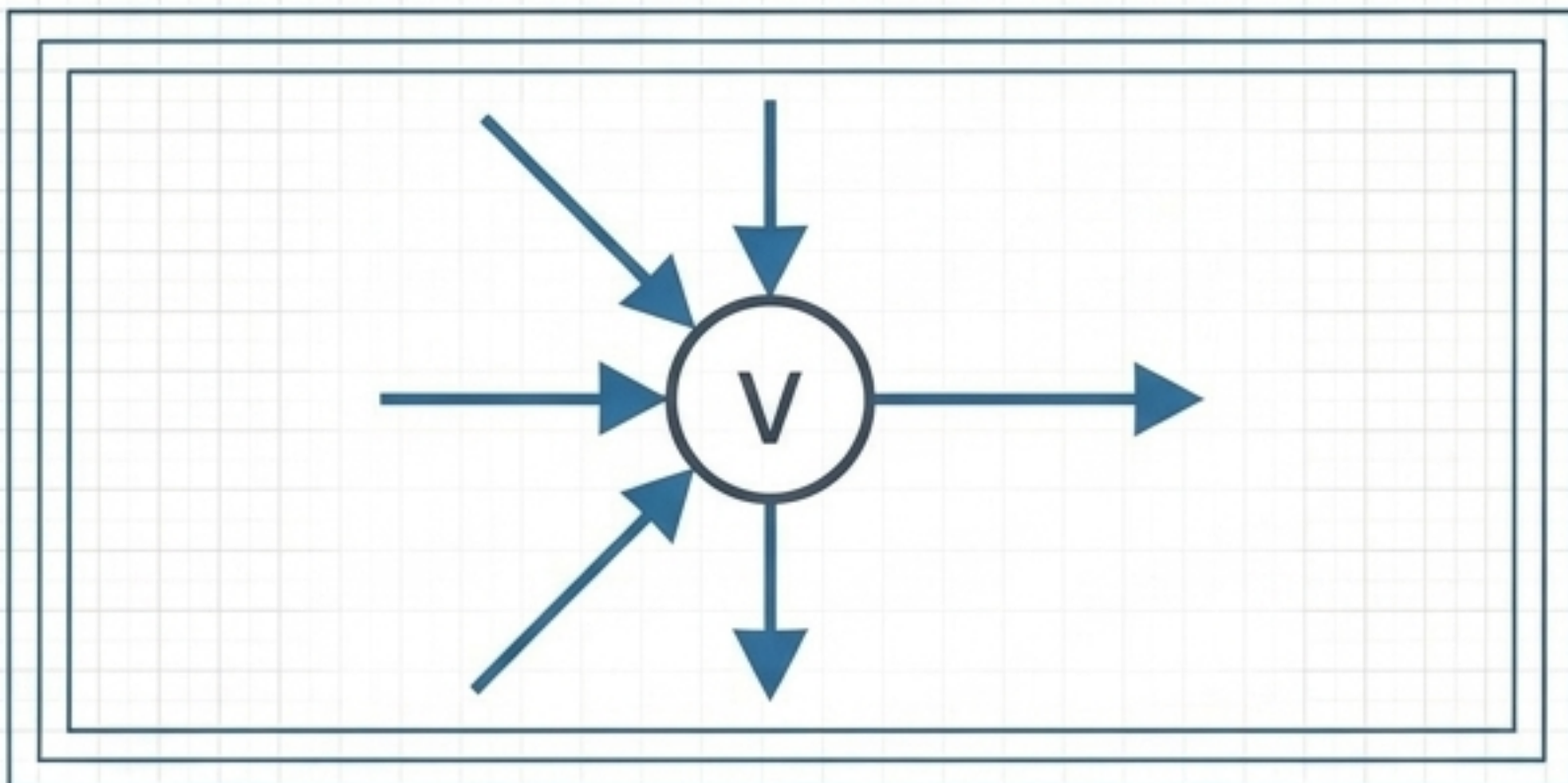
アルゴリズムが持つ「強制的な終了ルール」と、グラフの「位相幾何学的な要件」の不一致が何を引き起こすかを考察する。

```
last ← M // 辺の総数
while (last が 1 以上)
  ...
  else
    path[last] ← temp
    last ← last - 1 // 辺を確定することにより減少
```



【設問3】 The Concept: 理論と実装の乖離

数学的条件 (Blueprint)



有向グラフにおける一筆書き（オイラー閉路）の絶対条件: すべての頂点において、入ってくる辺の数（入次数）と出ていく辺の数（出次数）が完全に一致していること。

この条件が崩れているグラフでは、数学的に一筆書きは成立しない。

プログラムの挙動 (Runtime)



directedE 関数の特性: プログラム内には「オイラー閉路が可能かを事前に判定するロジック」が存在しない。

強連結である限り、未探索の辺を探し、行き止まれば遡る処理を「無条件に」継続する。

【設問3】 The Deduction: 盲目的な完了と矛盾

1. 処理の停止保証

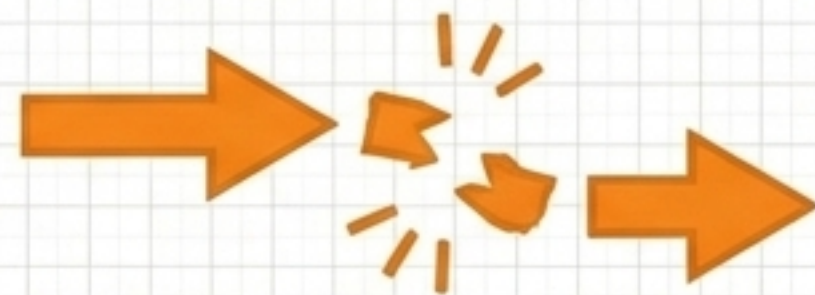
終了条件 last が 1 以上 とは、辺が path に格納されるたびに進むカウントダウン。

強連結である以上、必ず全ての辺を見つけて path に移すため、無限ループには陥らず、処理は必ず「完了（停止）」する。



2. 生成される結果の破綻

出来上がった path 配列の結末：全ての辺が格納されてはいるが、元のグラフが入出次数のバランスを欠いているため、配列の順番通りに辿ろうとすると途中で経路が繋がらない箇所が発生する。



Deduction Hint: 「プログラムの動作」は完了する。しかし、「出力される結果」は数学的な要件を満たさない。解答群の中からこの状態を示す選択肢を選ぶ。

【設問4】 The Prompt: メモリ領域の統合

分析の焦点

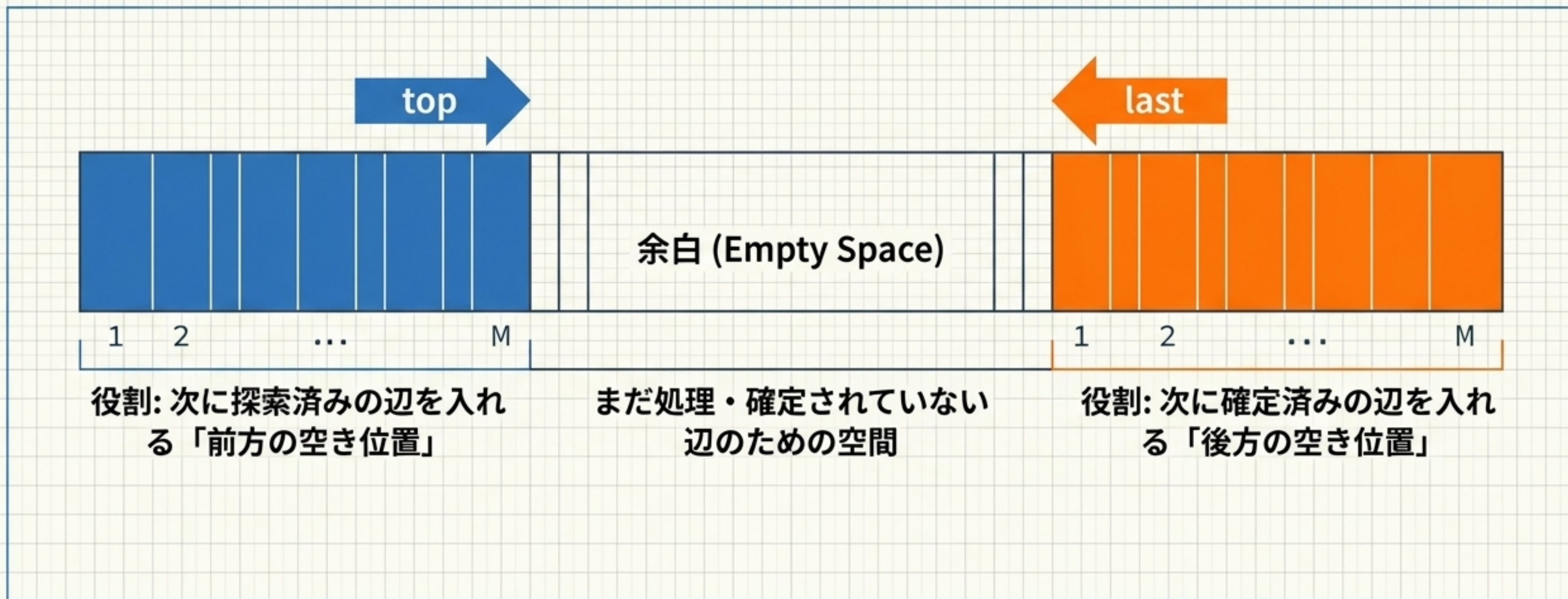
2つの配列（searchedとpath）を1つに統合し、無駄な繰り返しを防ぐための新しい while 終了条件を導く。

1つの配列空間の中で、2つのポインタ（topとlast）がどのように動くかを可視化し、境界条件を特定する。

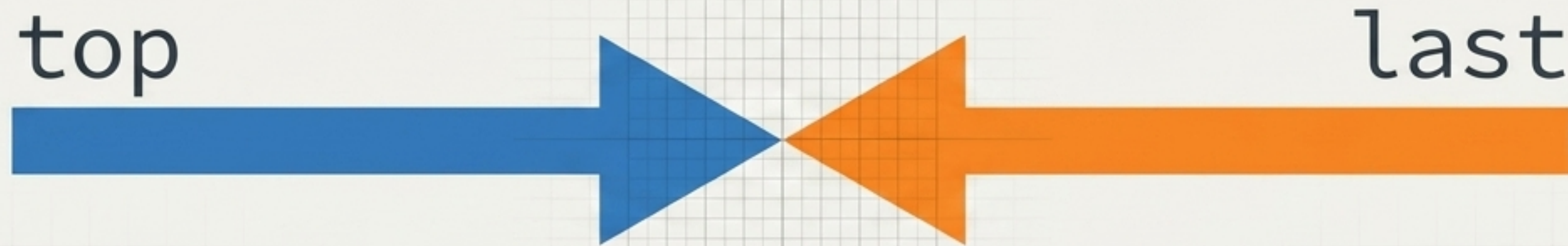
Original Architecture



【設問4】 The Concept: 統合配列のポインタ動態



すべての辺の処理が完了した瞬間とは、
両端から迫るポインタ（topとlast）が衝突し、
配列内に「余白」がなくなった状態を指す。



【設問4】 The Deduction: 境界条件の確定

ループの「継続」条件

while文に記述すべきは「終了条件」ではなく「ループを継続する条件」。

処理すべき辺（配列の余白）が残っている状態とは、前進する `top` が、後退する `last` をまだ追い越していない状態。



境界の越境

もし `top` が `last` を追い越してしまった場合、すべての要素が「探索済」または「確定済」に割り当てられ、ループを終了させる必要がある。

Deduction Hint:

変数 `top` が変数 `last` に対して、空間的にどのような関係性（不等式）を保っている間、ループを回し続けるべきか？

The Architecture of Algorithm

【設問2の示唆】 対称性の美学

「進む(Advance)」と「戻る(Backtrack)」という対称的な処理によって、グラフ全体の網羅性が保証される。評価回数の倍率は、このアーキテクチャに由来する。

【設問3の示唆】 プログラムの盲目性

アルゴリズムはグラフの位相幾何学的条件（オイラー閉路の可否）を関知せず、自身のコードのルールに従って停止するまで動き続ける。

【設問4の示唆】 データ構造の最適化

スタックと確定リストという相反するデータの流れは、ポインタの衝突条件を再定義することで、同一のメモリ空間に美しく統合できる。

コードの表面的な挙動を追うだけでなく、
「データ構造の動態」と「グラフ理論の法則」
をリンクさせることで、
真の最適化の青写真（Blueprint）が見えてくる。