

設問4を解き明かす：ダイクストラ法における計算量の考察

答えを導き出すための「論理的思考プロセス」と「O（オーダー）記法の極意」

```
SYSTEM_READY // INITIATING_LOGIC_TRACE
```

設問4のミッション：アルゴリズムの骨格を俯瞰する

本設問は単なる穴埋め問題ではありません。「アルゴリズムがどこで無駄なく動いているか」をマクロな視点で捉える力が問われています。本資料では解答そのものを提示しません。読者自身が論理的に正解を導き出せる「思考のフレームワーク」を提供します。

関数 distance では、次の [キ] を選ぶために始点ノード距離を計算する回数は最大でも N 回である。

また、[キ] を選ぶ回数は、一度選ばれると当該ノードの最短距離は確定するので、最大でも N 回である。

よって、最悪の場合の計算量は、 $O([ク])$ である。

Mission 1:
アルゴリズムの『主役』
を見極める

Mission 2:
プログラムの『ネスト
(入れ子)』を解剖する

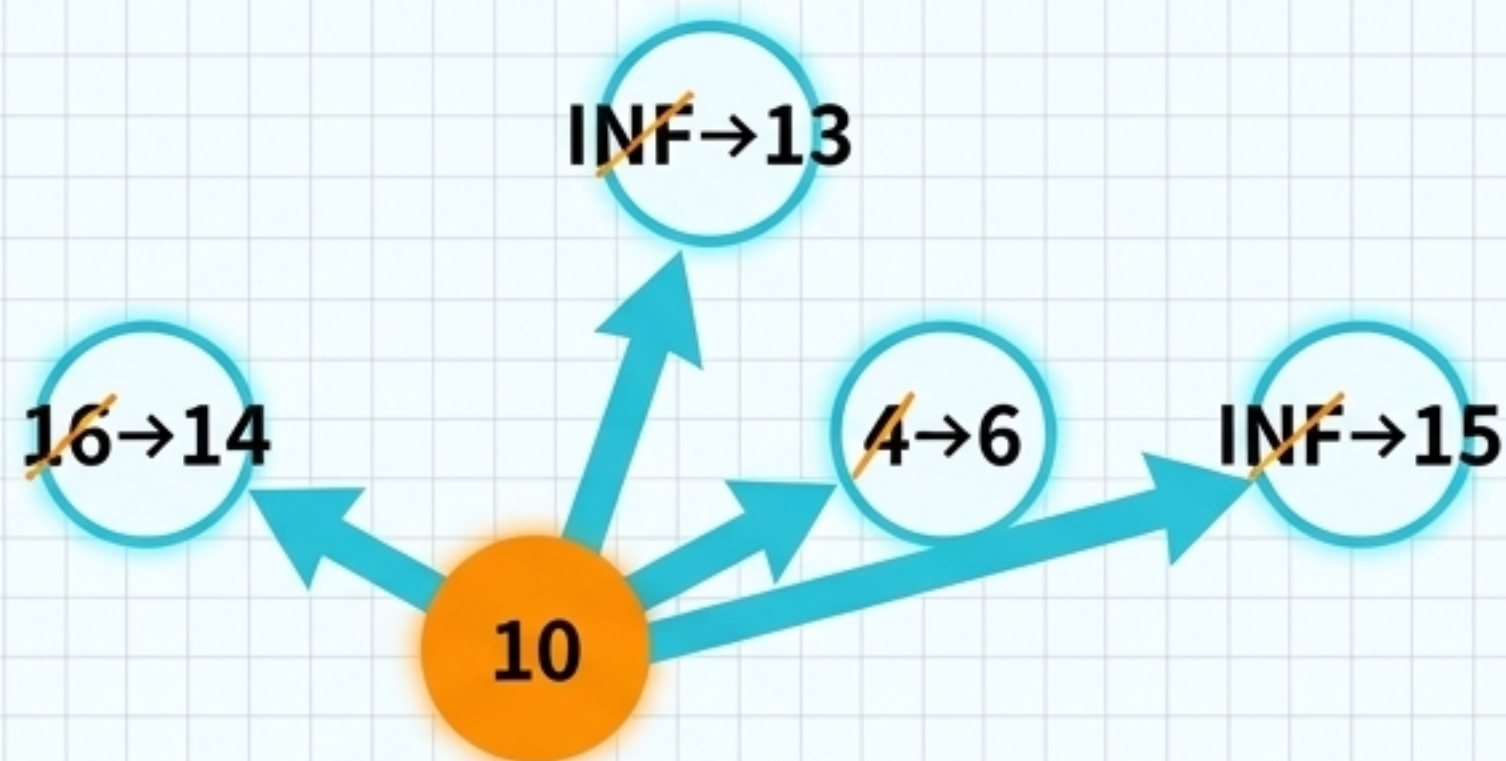
探索のプロセスを可視化する：ダイクストラ法の2つのステップ

Step 1: 最小を探して「確定」する



未確定のノードの中から、現時点で最も距離が短いものを選び出し、そのノードの最短距離を「確定」させる。

Step 2: 確定した地点から「隣接」を更新する

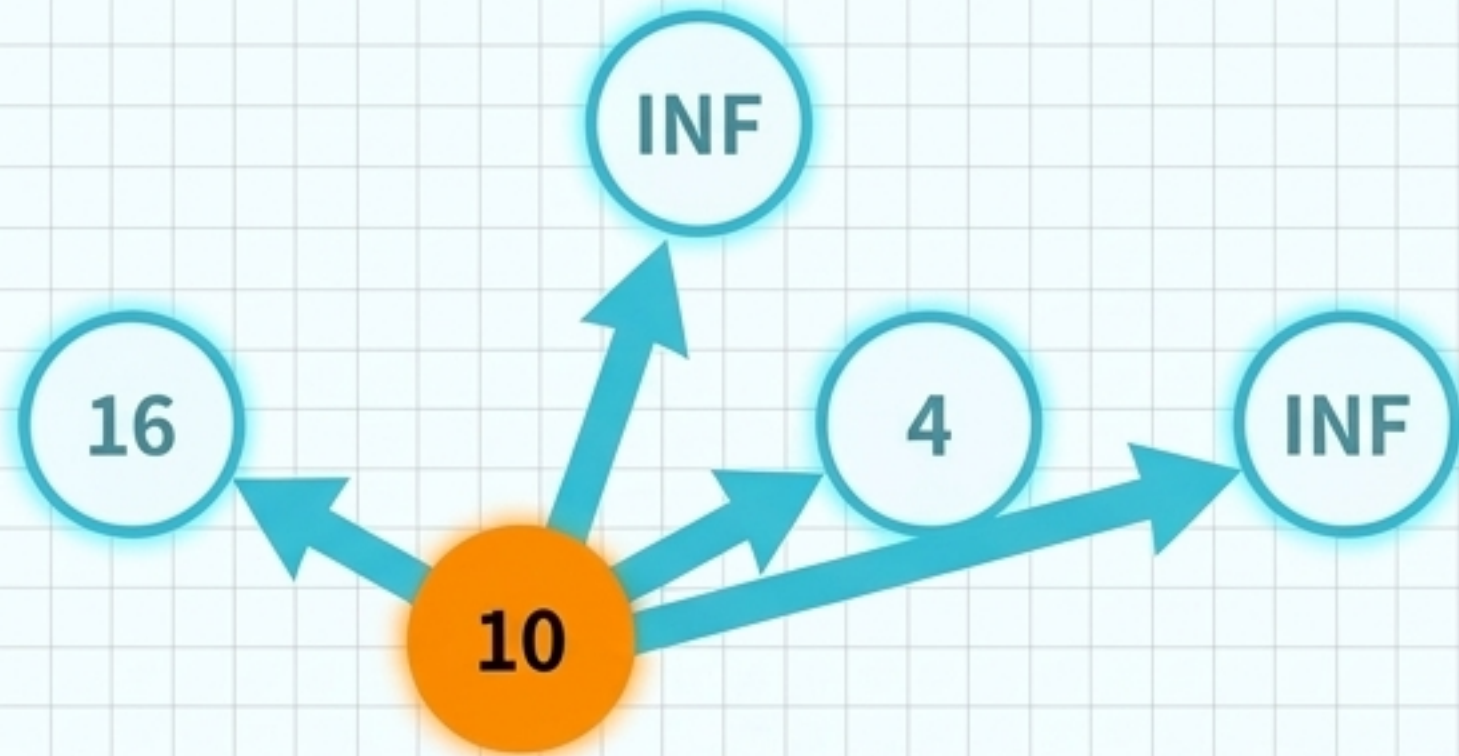


選ばれたノードを踏み台として、隣接するノードの距離を再計算し、より短ければ値を更新する。

Key Takeaway: 距離を計算し更新するためには、必ず「起点となる主役のノード」を毎回選び出す必要がある。

[キ]の導出：要件定義（仕様）へのリバーズエンジニアリング

概念 (Concept)



始点との距離を算出する基準となるノード

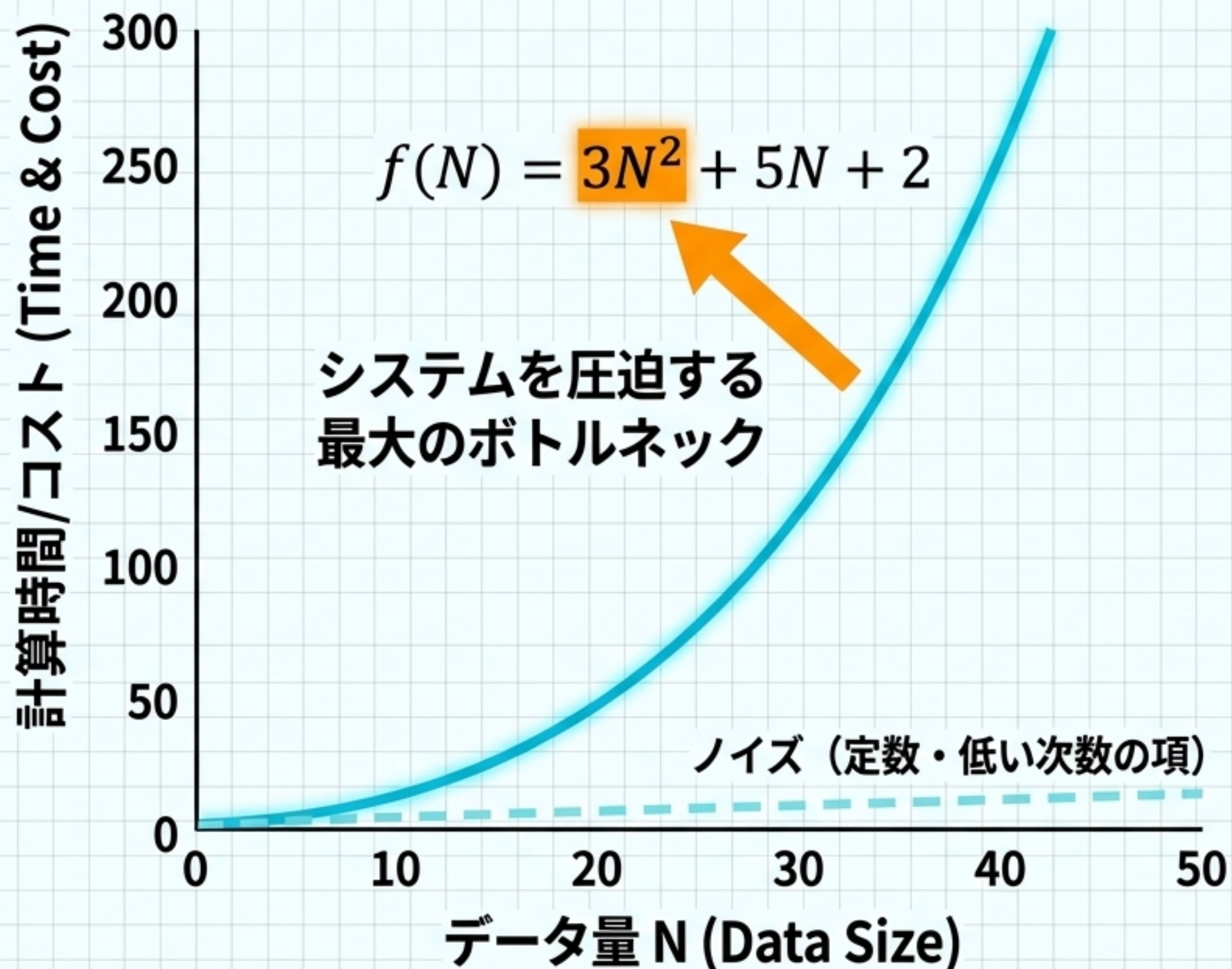
原文 (Source Text)

次に、終点の最短距離が確定するまで、①～③を繰り返す。

ここで、始点との距離を算出する基準となるノードを〇〇という。

プログラミングやアルゴリズムの問題において、独自の概念や役割を持つ対象の名称は、必ず問題文の前半で「仕様（ルール）」として事前定義されています。アルゴリズムの動きと照らし合わせれば、定義文に隠されたキーワードが自然と浮かび上がります。

O (オーダー) 記法とは何か？：成長のボトルネックを見抜く

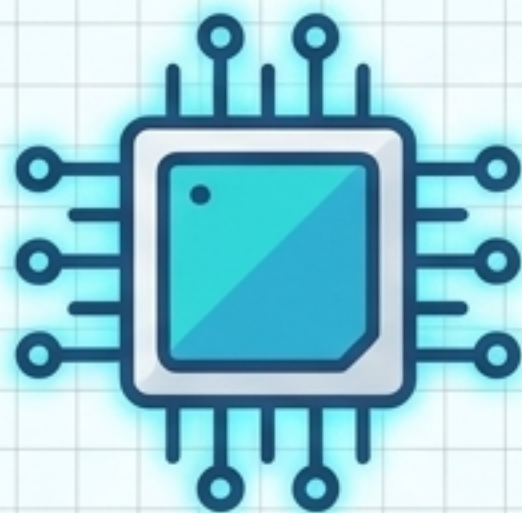


O記法 (Big-O notation)
とは、データ量 N が爆発的に増大したとき、プログラムの実行時間に「最も影響力を持つ項」だけを抽出して表現する手法です。

細かい定数や係数 ($\times 3$ や $+5$) は無視し、成長の支配的な原因のみに焦点を当てます。

評価対象となる「計算コストの単位」

関数 `distance` の計算量を導き出すためには、「何が1単位の計算か」を定義する必要があります。



1単位の計算 (Basic Operation)

- 変数への代入 (`minDist ← INF`)
- 単純な比較 (`if done[k]` が 0)
- 四則演算 (+, -, ×, ÷)

※ これらの基本操作自体は一瞬で終わるため、単体では全体の計算量に大きく影響しない。



繰り返し (Loop Multiplier)

計算量を決定づける真の要因は、「その基本操作が何回繰り返されるか (ループの回数)」に依存する。

Mission: どのループ構造が、どれだけの回数、基本操作を回しているかを解剖する。

プログラムの解剖 ①：最初の壁となる「外側のループ構造」

外側の全体ループ

```
12: while (true)
13:   minDist ← INF
14:   for (k を 1 から N まで 1 ずつ増やす)
15:     if (done[k]が 0 かつ )
16:       minDist ← 
17:       curNode ← k
18:     endif
19:   endfor
20:   done[curNode] ← 1
21:   if (curNode が GOAL と等しい)
22:     return dist[curNode]
23:   endif
24:   for (k を 1 から N まで 1 ずつ増やす)
25:     if (が dist[k] より小さい かつ done[k]が 0)
26:       dist[k] ← 
27:     endif
28:   endfor
29: endwhile
```

- プログラム全体を包み込む while(true) の大枠に注目します。
- このループが1周するごとに、1つのノードの最短距離が「確定 (done = 1)」します。
- ノードの総数は N 個です。したがって、すべてのノードが確定して処理が終了するまで、この外側のループは【最大で N 回】繰り返されることになります。
- これは設問4の「選ぶ回数は最大でも N 回である」という記述と完全に一致します。

プログラムの解剖 ②：2つの「N回探索」を担う内側ループ

内側処理 A: チャンピオン決定戦

```
14: for (k を 1 から N まで 1 ずつ増やす)
15:   if (done[k] が 0 かつ ...)
      ...
19: endfor
```

未確定のノード全てを比較し、最小距離を持つノードを探し出す処理。ノード数 N だけ比較を繰り返すため、最大 N 回の操作。

内側処理 B: 隣接ノードの距離更新

```
24: for (k を 1 から N まで 1 ずつ増やす)
25:   if ( ...)
      ...
28: endfor
```

選ばれたノードから、全ノードに対して隣接しているかを確認し距離を再計算する処理。ここでもノード数 N だけ確認を行うため、最大 N 回の操作。

外側のループの「1回の周回」の中で、内部では N に比例する計算が2セット ($N + N$) 行われています。Big-O記法では定数倍は無視されるため、内側のトータル処理コストは $O(N)$ と評価できます。

[ク]の導出：計算量のマトリクス合成（ループの掛け算）

外側ループ：最大 N 回の繰り返し

毎回 $O(N)$ の計算コスト
(探索 + 更新)

(外側ループの最大周回数) $\times$ (内側の 1 周あたりの計算量)

$$= N \text{ 回} \times (N \text{ 回} + N \text{ 回})$$

$$\approx N \times O(N)$$

$$= O([ク])$$

「N 回繰り返される作業」の箱の中に、「毎回 N 回に比例する計算」が格納されている「ネスト（入れ子）」構造です。データ量 N が増えたとき、全体の計算回数はこの掛け算の構造に従ってスケール（爆発）します。数式を展開すれば、自ずと [ク] の項が導き出されます。

The Deduction Framework：あらゆる計算量問題に通用する思考総括

Step 1: Define

定義の特定。

問題文の仕様定義
ブロックに立ち
返り、問われて
いる「役割」の
正式名称を特定す
る。

Step 2: Outer Boundary

外枠の把握。

処理全体が何回繰
り返されるか
(最大周回数)を
プログラムコード
の大枠から読み
解く。

Step 3: Inner Cost

内側コストの評価。

1回の周回内で発生
する「探索」や
「更新」のループ
回数(比例する
計算量)を評価す
る。

Step 4: Synthesis

マトリクス合成。

ネスト構造を掛け
合わせ、Big-O記
法のルール(係数
・定数の除外)
に従って支配的な
項を抽出する。

今回ダイクストラ法を解き明かしたこの思考プロセスは、ソートや探索など、他のあらゆるアルゴリズムの計算量・仕様読解問題に応用可能です。コードの動きを視覚的な「構造」として捉える視点を持ち続けましょう。