

Pythonで解き明かす最短経路問題

～2024年春期 応用情報技術者試験 問3 徹底解説～

--- 最短経路（逆順） ---

ノード 5（終点）

← ノード 3

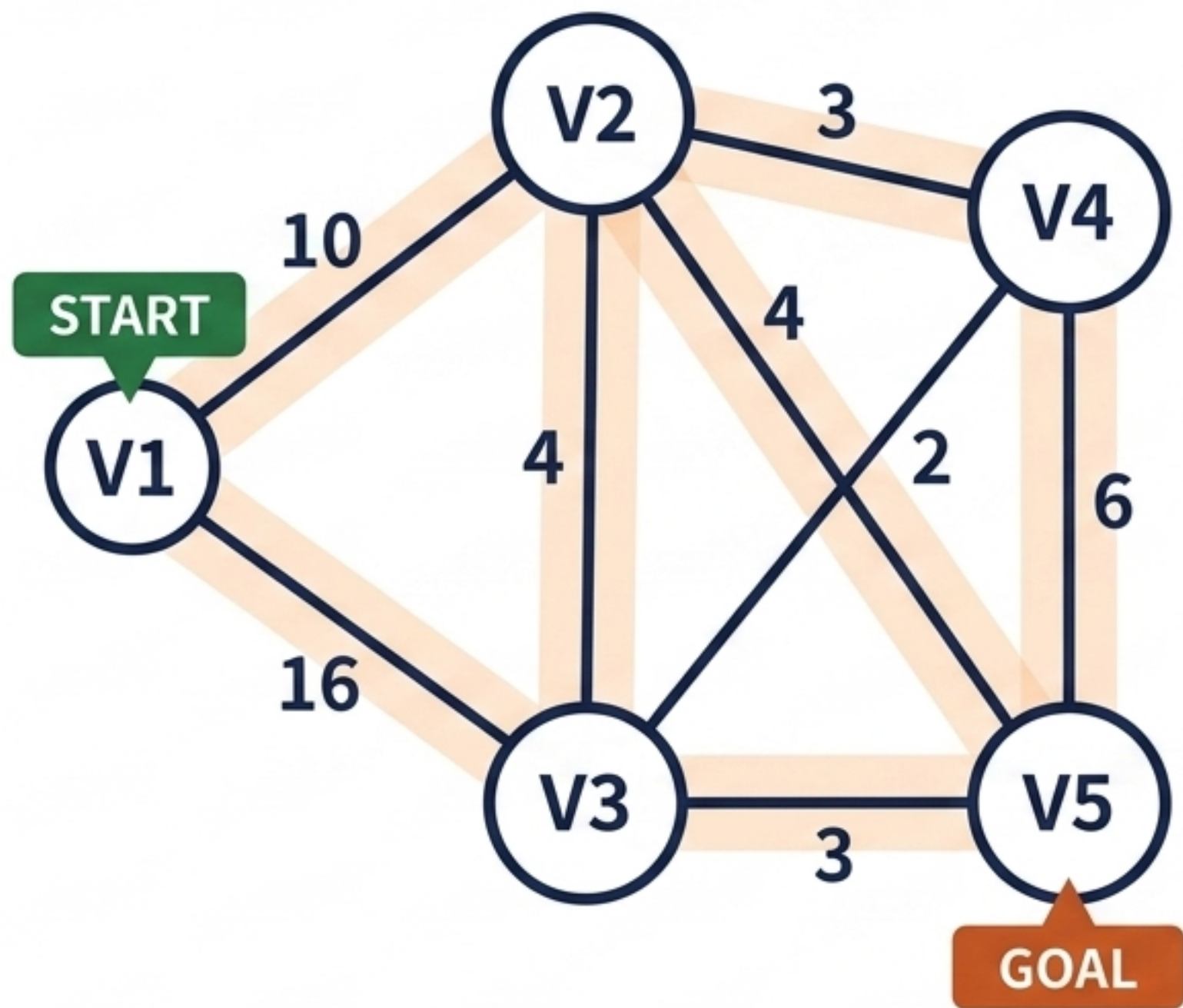
← ノード 2

← ノード 1

始点(V1)から終点(V5)までの最短距離: 17

この鮮やかな結果（距離17、経路5←3←2←1）は、どのようなロジックで導き出されたのか？
アルゴリズムの挙動とPythonコードの裏側を、リバーズエンジニアリングで完全に視覚化します。

ナビゲーションの迷路：最短ルートはどう見つけるか



- **アルゴリズムの用途:**

カーナビゲーションシステムにも採用される「ダイクストラ法」の仕組みを解き明かす。



- **解決すべき課題:**

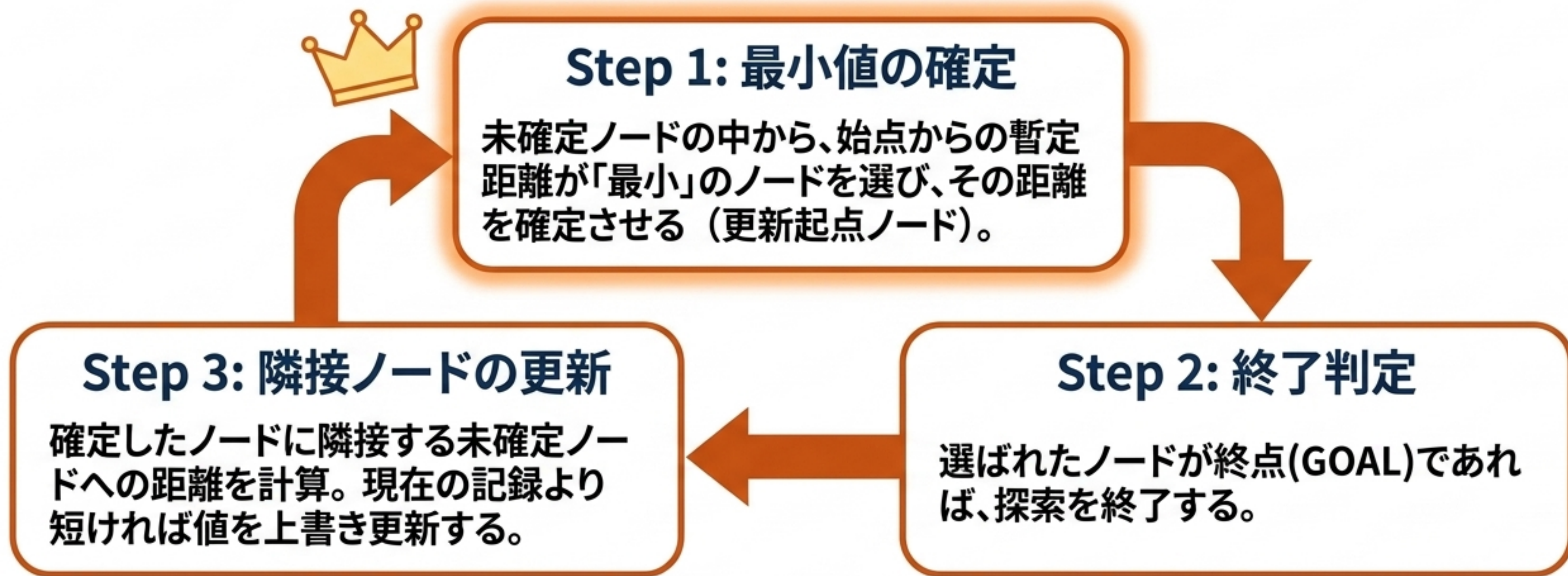
始点(V1)から終点(V5)まで、エッジの距離（コスト）の合計が最小となるルートを見つけ出すこと。



- **求められる出力:**

単なる「最短距離」だけでなく、そこに至る「最短経路」も同時にプログラムに記録させること。

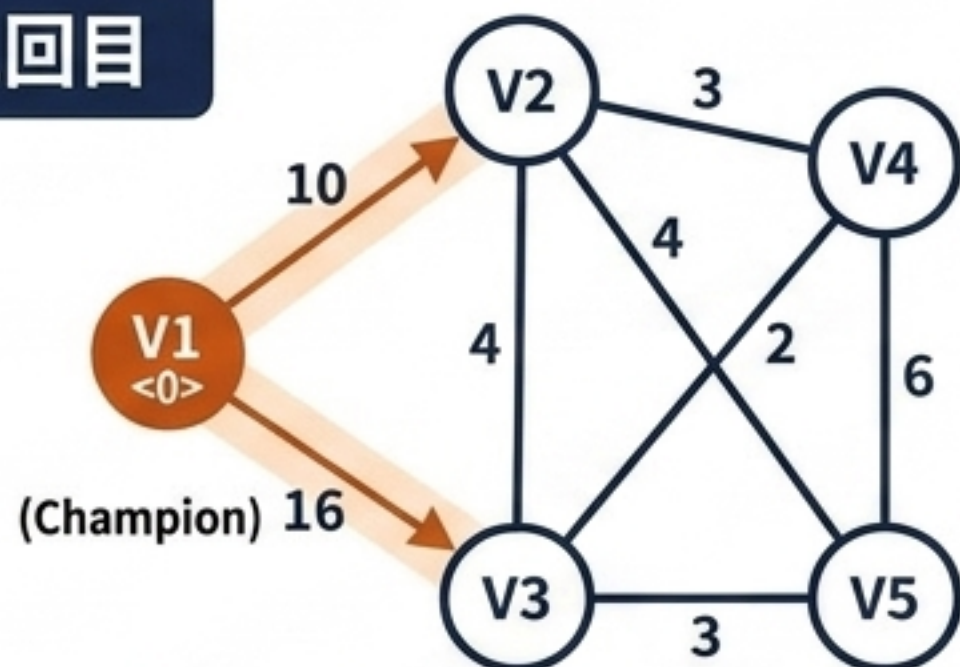
探索のコア・ロジック:「チャンピオン方式」による確定サイクル



アルゴリズムの原則：全体を一度に計算するのではなく、現在地から『一番近い確実なポイント』を次々と勝ち残らせていく（チャンピオン方式）。

探索のトレース（序盤）：未踏のルートを切り拓く

1回目



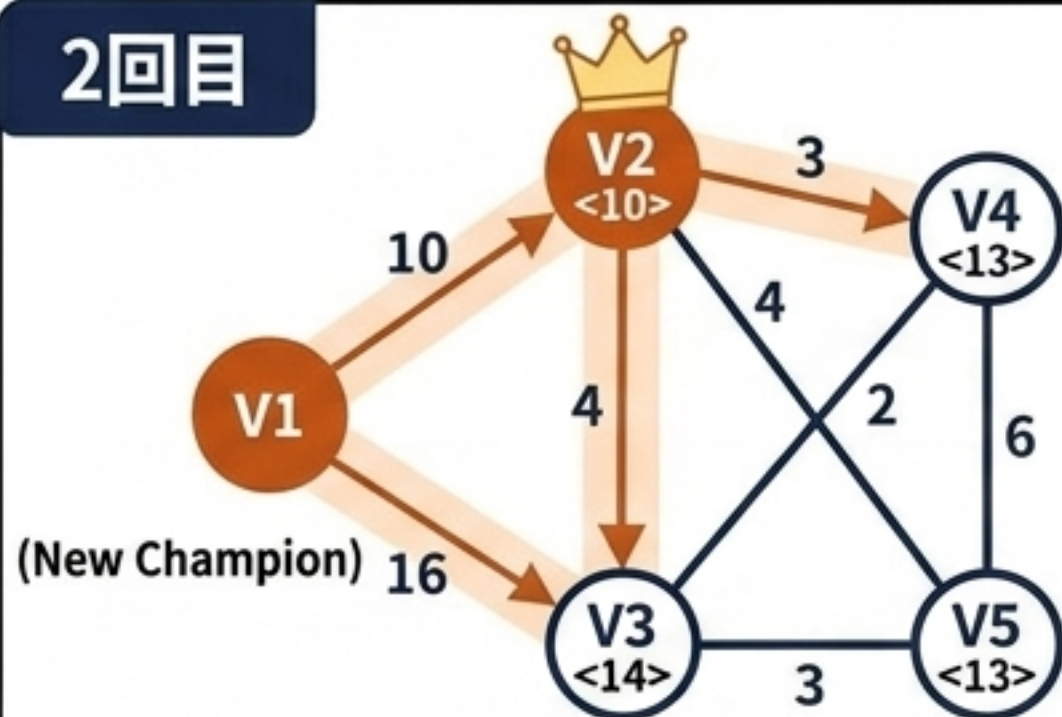
dist配列（始点からの距離）

V1	V2	V3	V4	V5
0	INF	INF	INF	INF

 →

V1	V2	V3	V4	V5
0	10	16	INF	INF

2回目



dist配列（始点からの距離）

V1	V2	V3	V4	V5
0	10	16	INF	INF

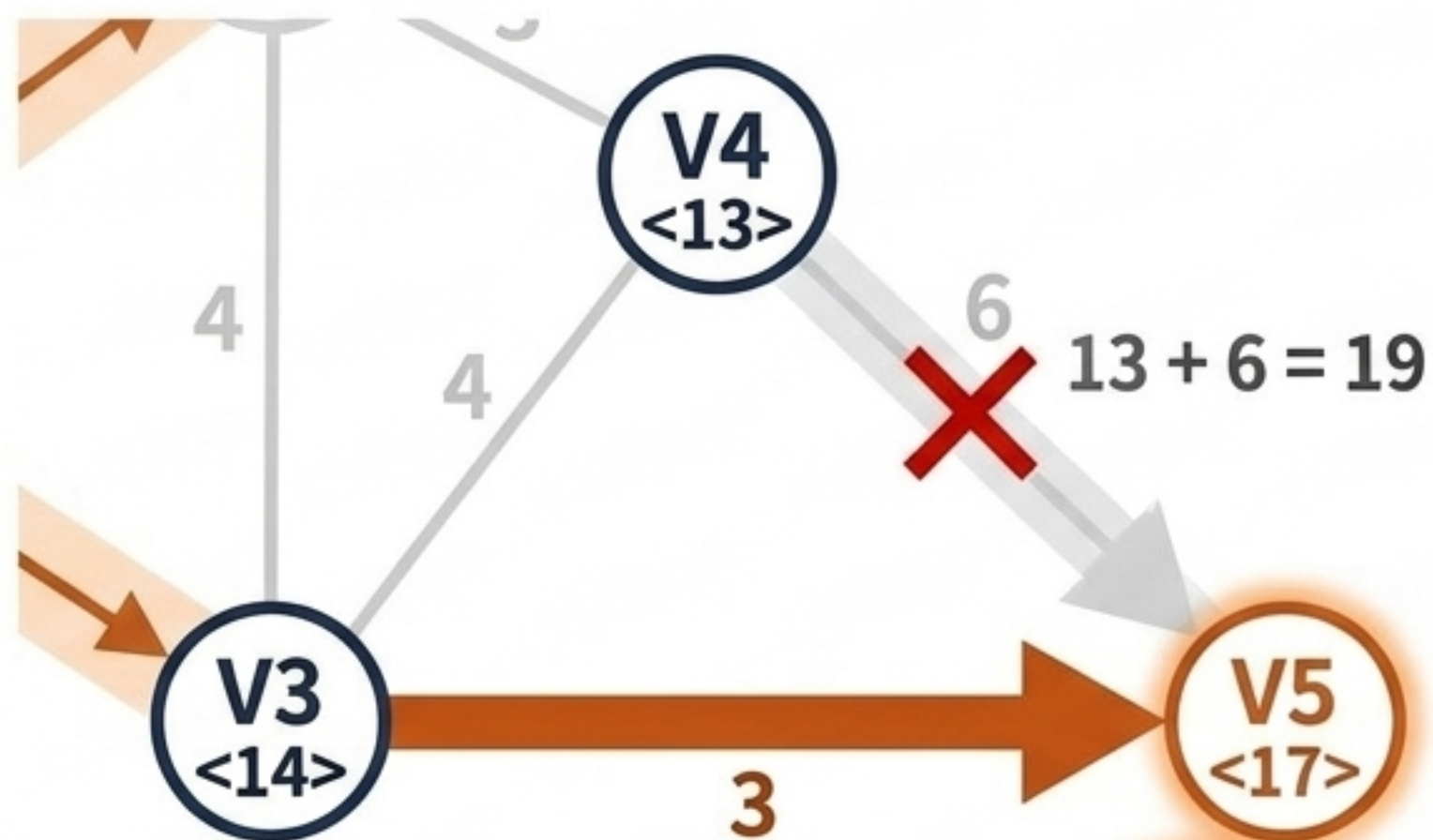
 →

V1	V2	V3	V4	V5
0	10	14	13	INF

Annotations: $\min(16, 10+4) = 14$ (for V3), $10+3 = 13$ (for V4)

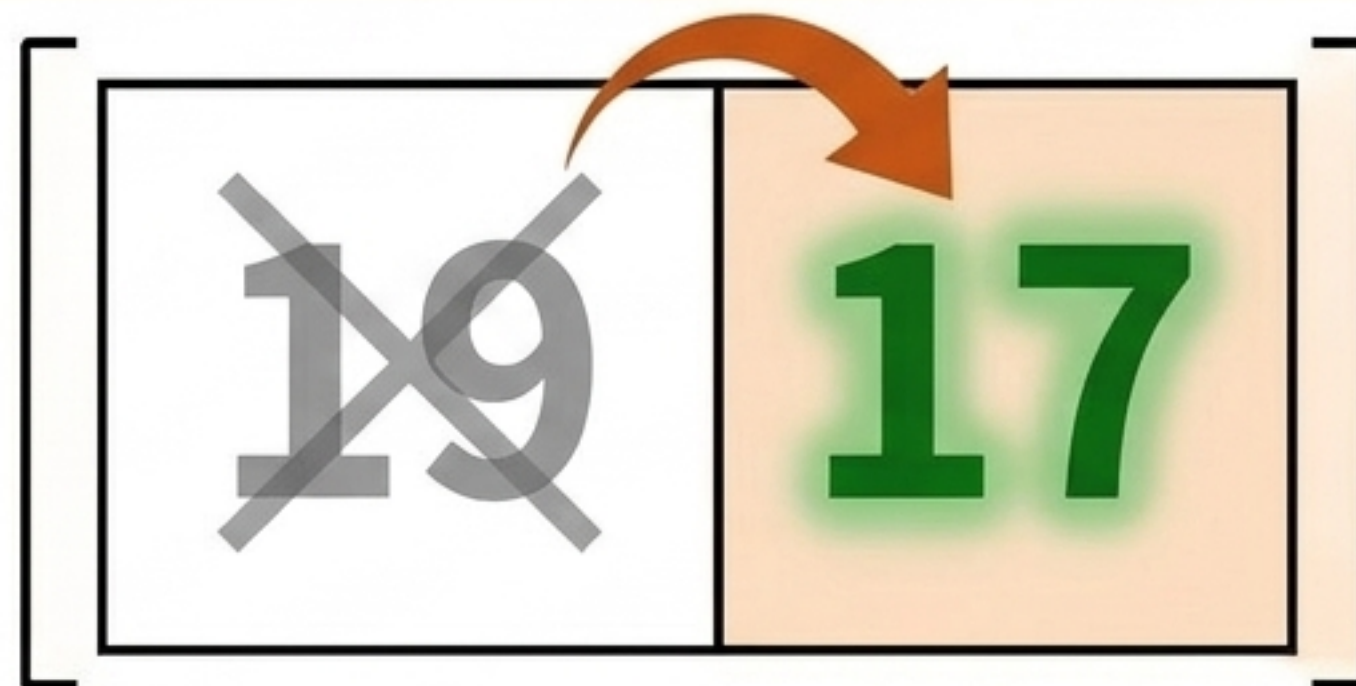
初期状態のINF（無限大）から、更新起点ノードを経由した実際の距離へと配列の値が次々と上書きされていく。

距離の確定：最終的な最短距離「17」への到達



14 (V3までの距離)
+ 3 (V3からV5へのエッジ)
= 17

dist[5] (V5への距離)



【設問1 解答：ア】 17

解説：一時的にV4経由のルートで距離「19」が記録されていたが、V3が更新起点ノード(チャンピオン)に選ばれた際、より短いルート(17)が発見され、配列の値が上書きされる。

ロジックの実装：配列とループによる比較・更新

Code Editor

```
14 1: チャンピオンの選出
15 for (k を 1 から N まで....)
16   if (done[k] が 0 かつ 【イ】)
17     minDist ← 【ウ】
18     curNode ← k
22
23 2: 隣接ノードの距離更新
24 for (k を 1 から N まで....)
25   if (【エ】 が dist[k] より小さい....)
26     dist[k] ← 【エ】
```

dist[k] < minDist
(未確定かつ現在の最小値より小さいか?)

dist[k]
(新しい最小値を記録)

dist[curNode] + edge[curNode, k]
(起点までの距離 + 隣接ノードへの距離)

【設問2 解答】 イ：dist[k] が minDist より小さい / ウ：dist[k] / エ：dist[curNode] + edge[curNode, k]

言語間の翻訳：試験疑似コードからPythonへ

試験問題の疑似言語 (図2)

実際のPython実装

整数型の配列: `dist`

Pythonでは
要素番号を1から
合わせるため
N+1 個確保する
テクニックを使用

`dist = [INF] * (N + 1)`

for (k を 1 から N まで 1 ずつ増やす)

`for k in range(1, N + 1):`

`dist[k] ← dist[curNode] + edge[...]` → `dist[k] = dist[curNode] + edge[...]`

代入と比較のロジックはPythonでもほぼ同一の構造で表現可能

IPAの疑似言語はPythonの構造に極めて近い。配列のインデックス（0始まりか1始まりか）の違いに注意すればスムーズに翻訳できる。

経路の喪失：「どこから来たのか」をどう記憶するか？



⚠ 現在のデータ構造の限界

距離の記録 (dist) と確定フラグ (done) だけでは、「どのルートを通してゴールに辿り着いたか」という足跡情報が一切残っていないため、経路を出力できない。

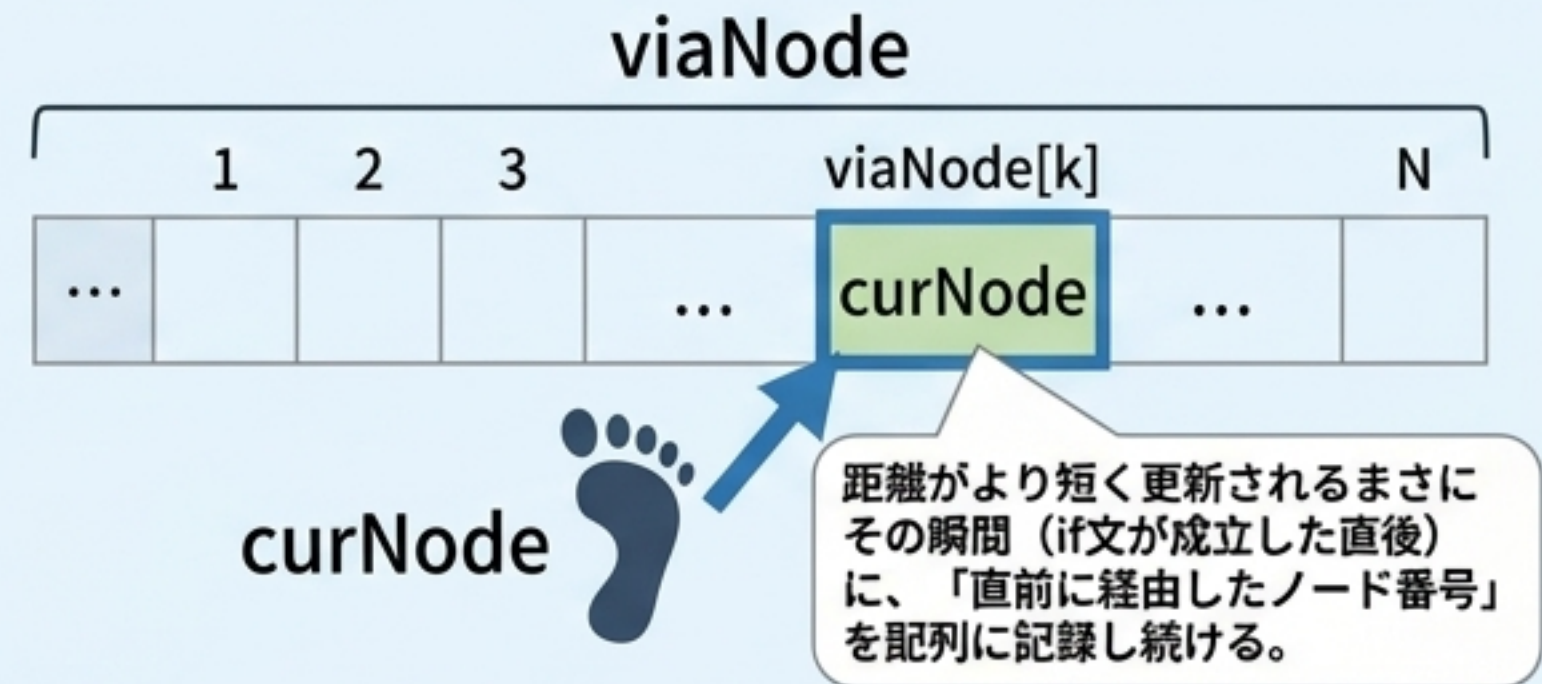
dist配列：[0, 10, 14, 13, 17]

done配列：[1, 1, 1, 1, 1]

足跡の記録：新たな配列『viaNode』の導入

```
24:  for (k を 1 から N まで...)
25:    if ( 新しい距離 が dist[k] より小さい... )
    【挿入】
    viaNode[k] ← curNode
26:    dist[k] ← 新しい距離
27:  endif
```

足跡を記録するスタンプ機能

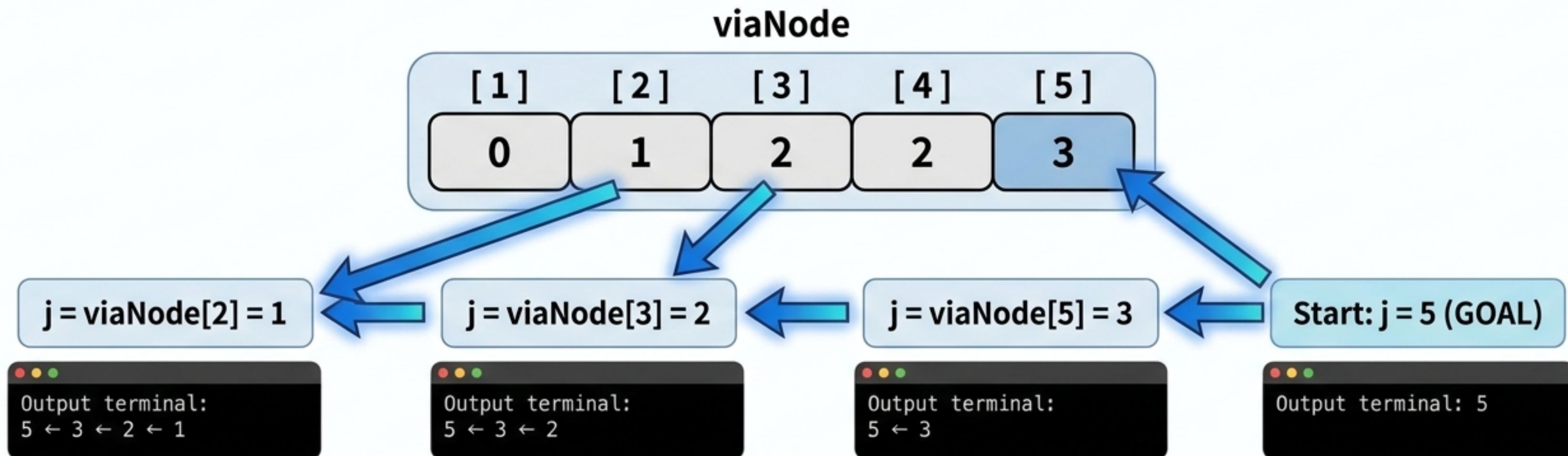


【解説】『viaNode』は「経由ノード」を意味し、あるノードkに到達する直前にいたノード（curNode）を記録します。

【設問3(1) 解答】

✓ 挿入すべき代入文：viaNode[k] ← curNode / 挿入行 **[オ] : 25**

リバース・トレース：目標の出力を生み出す連鎖

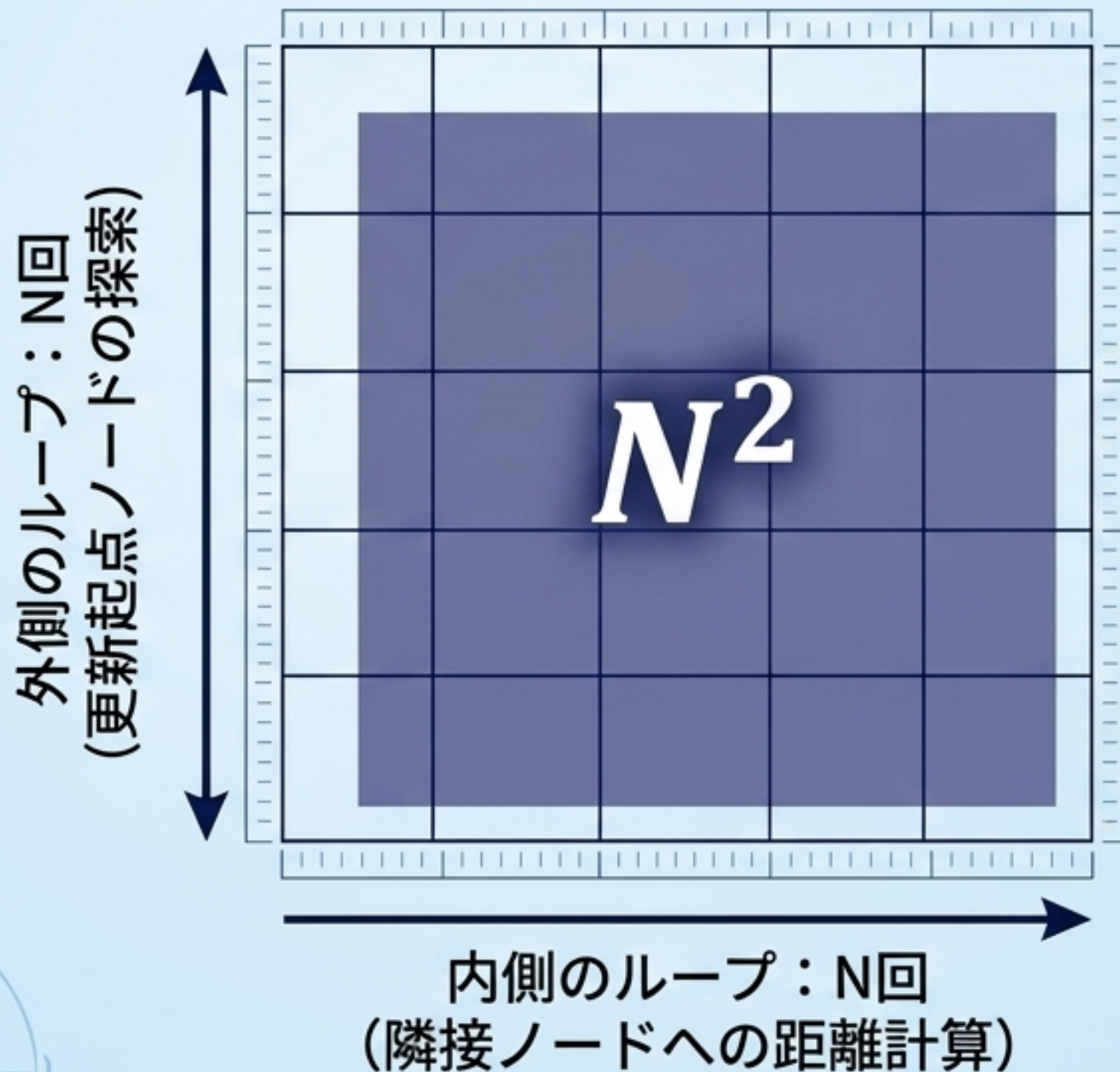


出力結果: ノード 5 ← 3 ← 2 ← 1

✓ 【設問3(2) 解答：カ】 ア (viaNodeに格納してあるノード番号を...)

解説：終点からスタートし、配列の値を次のインデックスとしてバケツリレーのように渡していくことで、最短経路が逆順に復元される。

計算量の考察：アルゴリズムの効率 $O(N^2)$



- 未確定ノードの中から最小距離を選ぶ操作にN回。
- 選ばれたノードに隣接するノードの距離を計算・更新する操作にN回。
- 単純な配列を用いたダイクストラ法では、この2重ループ構造により最悪計算量はノード数の2乗に比例する。



【設問4 解答】

[キ] 更新起点ノード / [ク] N^2

総括：Pythonで解き明かすナビゲーションの裏側

複雑なネットワークの最短経路探索も、3つの配列（距離・確定フラグ・経路）と簡潔なPythonコードに分解することで、完全に制御・計算が可能になります。

本資料が、動的計画法の本質と実社会で稼働するアルゴリズムの解像度を上げる一助となれば幸いです。

