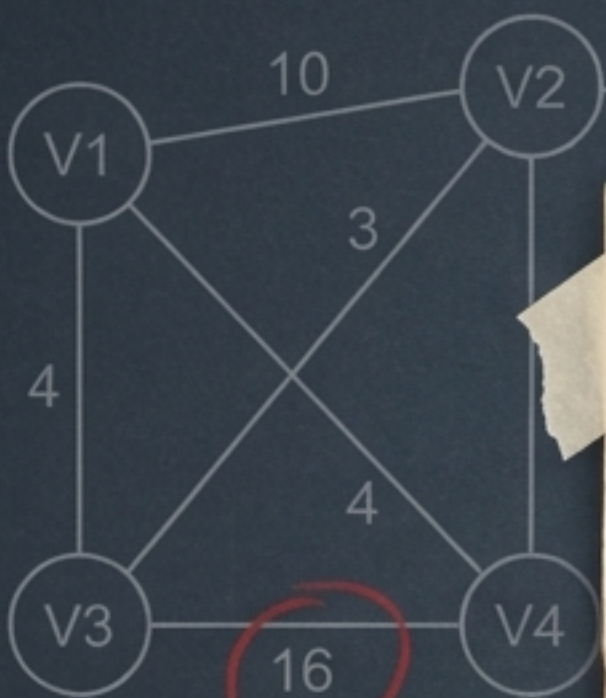


EVIDENCE FILE

【設問3 攻略ガイド】 最短経路を「記録」し 「出力」せよ

令和6年春 応用情報技術者試験 午後 問3
- 解答のないヒント集 -



```
while(true) {  
  if (done[k] == 0  
    dist[curNode]  
    if (done[k] ==  
      dist[curNode]  
  
    if (dist[k] ≠  
      dist[k] == d  
    }  
    dist[curNode]  
  }  
} endwhile  
}
```



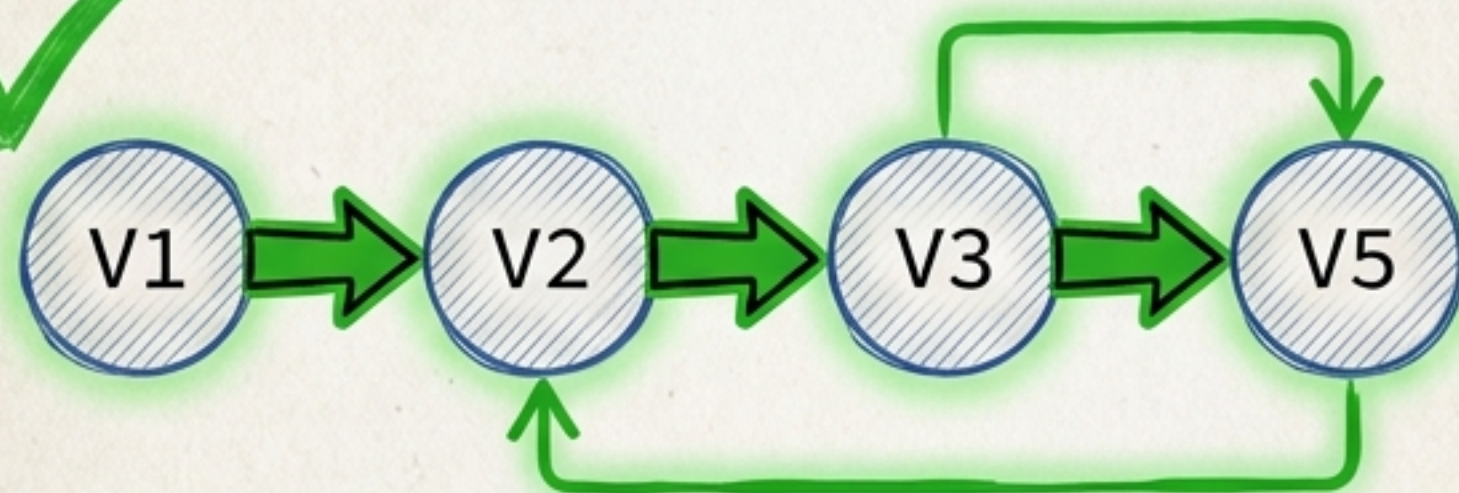
```
    v; ) {  
    s'&ar [v] {  
    curNode == distNode);  
    l) {  
  
    * ar := dist[curNode] {  
    = dist[k][curNode];  
  
    e] = 1;  
  
    endwhile  
  }  
}
```

ミッションの確認：
設問3は何を求めているか？

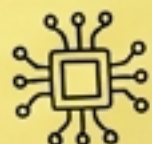
これまでの限界：
最短距離（数値）しか分からない



今回のゴール：
通ってきた「経路そのもの」を出力する



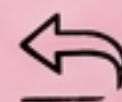
1. 記憶する
仕掛け(変数)
の追加



2. 足跡を記録
する処理
の挿入



3. 記録を
逆順にたどる
出力処理



新たな武器「viaNode」の役割を解剖する

整数型の配列: viaNode

/* 最短経路のノード番号を格納する。初期値は 0。*/

問題文で追加された viaNode は、いわば「足跡リレー」のボタンです。あるノードに向かう最短ルートが判明したとき、その「一つ前の出発元」が誰だったかを専用ボックスに記録しておくための仕掛けです。

ノードY

最短ルート発見！

ノードX

viaNode[X]

Xに来る前は、Yにいた証拠!

設問3 (1) へのアプローチ：
「どこに」「何を」挿入するか

探すもの1 (場所
[オ])：距離の
値が書き換わる
瞬はどこか？

「各ノードの始点ノード距離を更新する
たびに、直前に経由したノード番号を
viaNode に格納する ① 代入文を一、
つ、図2のプログラムの行 [オ] の
直後に挿入する。」

探すもの2 (処
理①)：代入文
の「左辺」と
「右辺」はどう
組み立てるか？

探査ヒント1：挿入場所[オ]を特定せよ

キーワード：「始点ノード距離を更新するたびに」

距離を管理する配列 `dist` の値が、「より小さい」と判定され、実際に新しい値で上書きされている行はどこでしょうか？

その代入が行われている「直後」こそが、足跡を記録するベストタイミングです。

```
24:  for (k を 1 から N まで 1 ずつ増やす)
25:      if ( エ が dist[k] より小さい かつ done[k] が 0 )
26:          dist[k] ← エ
27:      endif
28:  endfor
```

ココ！（更新の直後）

探査ヒント2：代入文①の「右辺」を考えよ

ミッション：「直前に経由した
ノード番号」を記録せよ

今、あなたは「更新起点ノード」
に立って、隣のノードを調べてい
ます。もし隣の最短距離が更新
されたなら、隣のノードにとっての
「直前に経由したノード」は、
今あなたが立っているそのノード
です。

現在地（更新起点ノード）

左辺（どこに） ← 右辺（何を）

```
8: 整数型: curNode /* 更新起点ノードのノード番号 */
9: 整数型: minDist /* 更新起点ノードを求める際に使用する一時変数 */
10: 整数型: k /* 要素番号 */
```

探査ヒント3：代入文①の「左辺」を考えよ

ミッション：viaNodeの「どこ」に記録するか？

足跡は「目的地」側のノードに紐づけて記録する必要があります。現在、距離を更新しようと調べている「隣接するノード（探索先）」の番号は、何という変数でループ処理されていますか？

探索先のノード
(ループ変数)



組み立てイメージ： `viaNode[ 探索先のノード ] ← 今いる起点ノード`

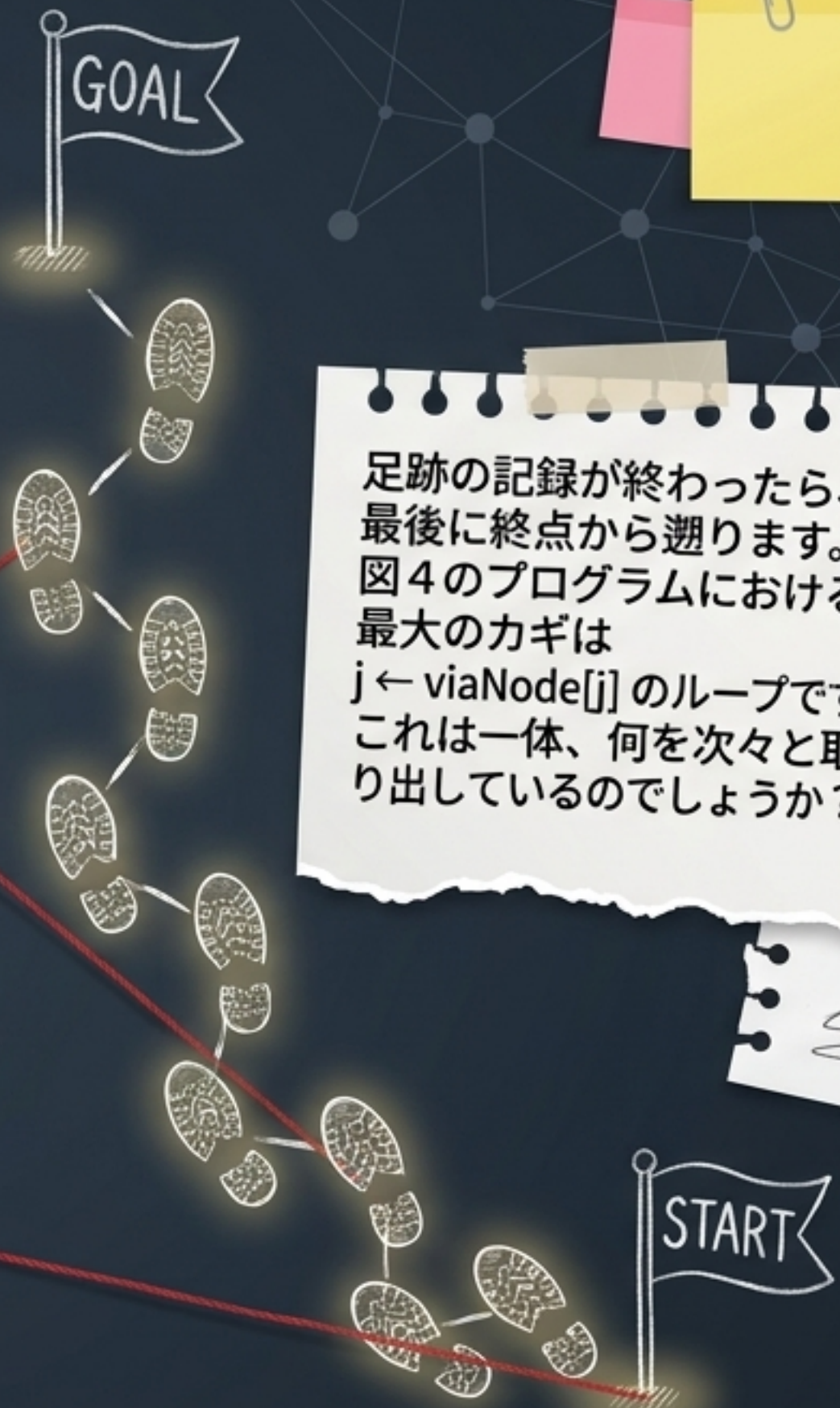
設問3 (2) へのアプローチ： 記録から経路を復元する

「終点のノード番号を起点として[カ]たどることで、最短経路のノード番号を逆順に出力する。」

```
j ← GOAL
GOAL を出力
while (j が 1 より大きい) /* 最短経路の出力 */
  viaNode[j]を出力
  j ← viaNode[j]
endwhile
```

(A magnifying glass highlights the line `j ← viaNode[j]` in the code block, with a red arrow and a question mark pointing to it.)

足跡の記録が終わったら、最後に終点から遡ります。図4のプログラムにおける最大のループは `j ← viaNode[j]` のループです。これは一体、何を次々と取り出しているのでしょうか？



探査ヒント4：
コードをトレースして[カ]を暴け

仮に GOAL が V5 だとします。

まず $j = 5$ となり、 $\text{viaNode}[5]$ の中身 (例えば 3) を取り出します。

取り出した「3」を次の j に代入し、次は $\text{viaNode}[3]$ を見に行きます。

つまり、単純に要素番号の大小で規則的に移動しているわけではありません。

要素番号 (インデックス): 5

格納されている値: 3

要素番号 (インデックス): 3

格納されている値: 2

$j \leftarrow \text{viaNode}[j]$

ア. viaNode に格納してあるノード番号を
イ. viaNode の要素番号を大きい方から
ウ. viaNode の要素番号を小さい方から

最終確認：点と点が繋がったか？

1. 挿入場所：
距離が更新される瞬間



3. 遡る仕組み：
箱の中の値を頼りに
数珠つなぎで遡る

2. 記録の仕組み：
隣接ノード箱に、今いる
ノードを書き込む

すべての証拠が出揃いました。
これがダイクストラ法において「経路を記録・出力する」ためのオーソドックスで美しい手法です。
プログラム内の「**変数が今、何を指しているか**」を追跡する力が、最大の武器になります。健闘を祈ります！