All-Pairs Shortest Paths

David N. Jansen 杨大卫

名 姓

Quiz

- What are the basic operations for single-source shortest-path algorithms? Why do we only change the data using these basic operations?
- What is the main idea of each of the following algorithms? What are its conditions for use?
  - Bellman–Ford
  - DAG (directed acyclic graph) algorithm
  - Dijkstra

What is “all-pairs shortest paths”?

- Single-Source: A producer of a city map wants to know the distance from Hangzhou Chengzhan to every tourist attraction in the city.
  The map is placed at one specific place (Hangzhou Chengzhan),
  so tourists can see how long it takes to any attraction.

- All-Pairs: A producer of a city map wants to know the distance between any two tourist attraction in the city / between all pairs of tourist attractions.
  A map can be placed at every tourist attraction,
  so tourists can see how long it takes to any other attraction.

Naive Solutions for All-Pairs

- Run Dijkstra's algorithm for every source: $O(|V|^3 + |V| \cdot |E|) = O(|V|^3)$ or
  $O(|V| \cdot |E| \log |V|)$.

- Run Bellman–Ford's algorithm for every source: $O(|V|^2 \cdot |E|) = O(|V|^4)$.

- We will do better than this!

Input and Output

- Input: a weighted graph $G = (V,E)$ with weights $w{:}E \to \mathbb{R}$
  often as adjacency matrix W: V x $V \to \mathbb{R}$,
  where $w_{ii} = 0$ and $w_{ij} = \infty$ if there is no edge from i to j.

- Output: for every pair of vertices v_i,v_j, a shortest path from v_i to v_j and its length.
  We can store the paths efficiently as: π_{ij} = the predecessor of v_j on the path starting at v_i, i.e. on the path $v_i \rightsquigarrow \pi_{ij} \to v_j$

Overview

- All-Pairs shortest paths and matrix exponentiation
  Idea: by an operation similar to matrix exponentiation for matrix *W*,
  one can calculate the length of all shortest paths.

- **Floyd–Warshall algorithm**
  Idea: a shortest path from *s* to *t* has an internal vertex with maximum
  index v_k. All other vertices are in $v_1,\ldots,v_{k-1}$.
  Combine shortest paths $s \to v_k \to t$.

- **Johnson's algorithm**
  Idea: when there are negative-weight edges,
  change the weights and then apply Dijkstra's algorithm.

Matrix Exponentiation

- Assume given a $n \times n$-matrix A. Calculate $A^n = \overbrace{A \cdot A \cdots A}^{n \text{ factors}}$.

- A simple solution calculates $A^2 = A \cdot A$, $A^3 = A^2 \cdot A$, $A^4 = A^3 \cdot A$ etc., until n is reached. Requires time in $O(n^4)$.

- An efficient solution calculates $A^2 = A \cdot A$, $A^4 = A^2 \cdot A^2$, $A^8 = A^4 \cdot A^4$ etc., until n is reached. Requires time in $\mathrm{O}(n^3 \log n)$.

Shortest Path Weights

- W_{ij} = weight of the shortest path from i to j, consisting of ≤ 1 edge

- I want to define an operation ⊗ ("o-times") on matrix W.
  $W^{②} = W \otimes W$ $W^{③} = W^{②} \otimes W$ $W^{④} = W^{③} \otimes W$ etc.
  such that
  $W^{ⓜ}_{ij}$ = weight of the shortest path from i to j, consisting of ≤ m edges

- The operation will be similar to matrix multiplication, and finding $W^{(n-1)}$ is similar to matrix exponentiation.

- Even simpler: any $W^{ⓜ}$ for $m \geq n-1$ is equal to $W^{(n-1)}$
  (if there are no negative-weight cycles).

Matrix Multiplication and ⊗

- Basic step in matrix multiplication:

$$(A^2)_{ij} = \sum_{k=1\ldots n} a_{ik} \cdot a_{kj}$$

- Basic step for ⊗:

$$(A^{②})_{ij} = \min_{k=1\ldots n} a_{ik} + a_{kj}$$

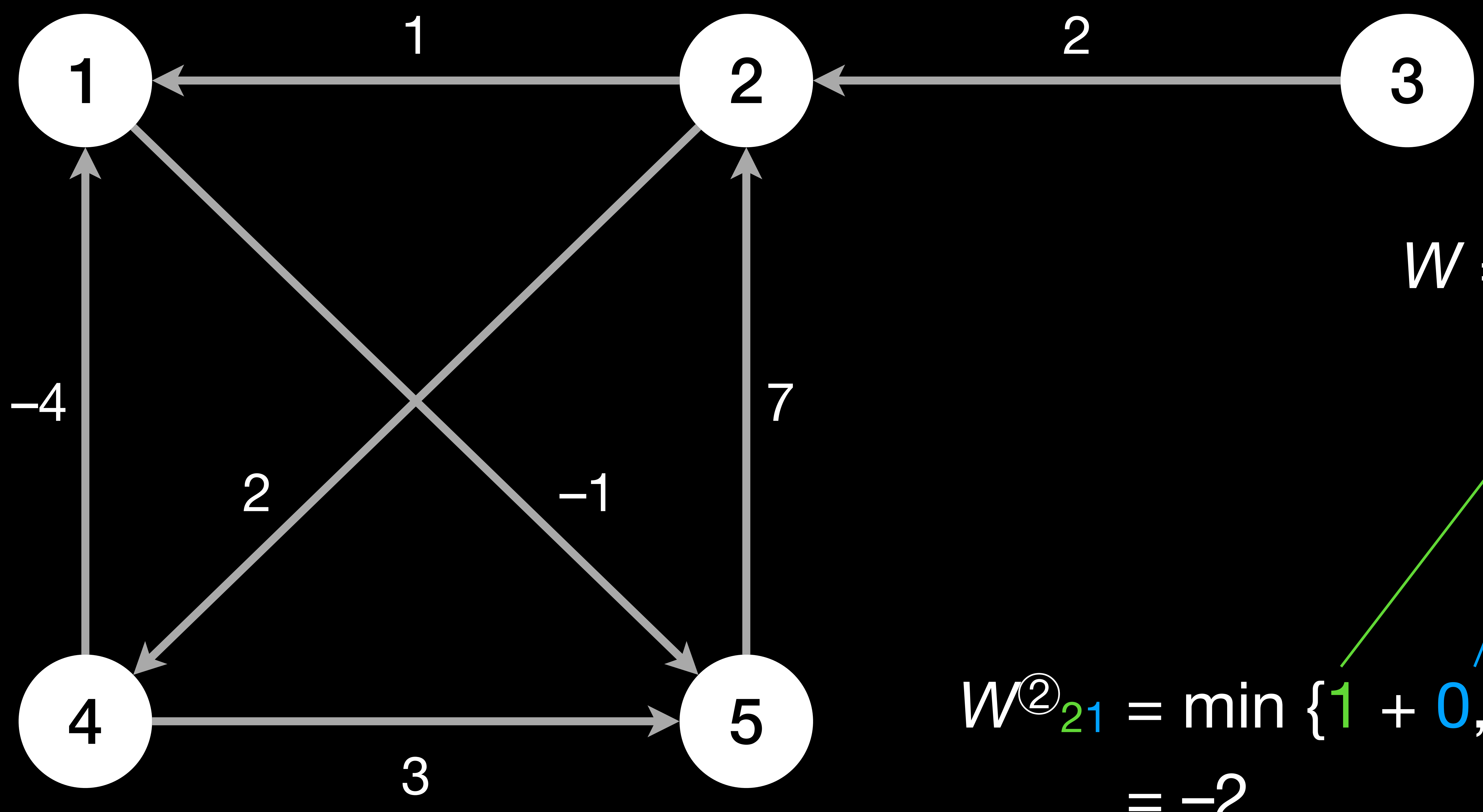
- min-plus algebra: a semiring 半环 over $\mathbb{R} \cup \{\infty\}$ where “min” is the addition and “+” is the multiplication

All-Pairs Shortest Paths with ⊗

- Idea: Calculate $W^{ⓜ}$ for some $m \geq n-1$

FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
$n = W.rows$
$m = 1$
while $m < n-1$
    Let $W^{(2m)}$ be a new $n \times n$-matrix
    $W^{(2m)} = W^{ⓜ} \otimes W^{ⓜ}$
    $m = 2m$
return $W^{ⓜ}$

All-Pairs Shortest Paths with ⊗



$$W = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 \\ 1 & 0 & \infty & 2 & \infty \\ \infty & 2 & 0 & \infty & \infty \\ -4 & \infty & \infty & 0 & 3 \\ \infty & 7 & \infty & \infty & 0 \end{pmatrix}$$

$$W^{\textcircled{2}}_{21} = \min\{1 + 0,\ 0 + 1,\ \infty + \infty,\ 2 + -4,\ \infty + \infty\}$$
$$= -2$$

$\otimes$: Correctness

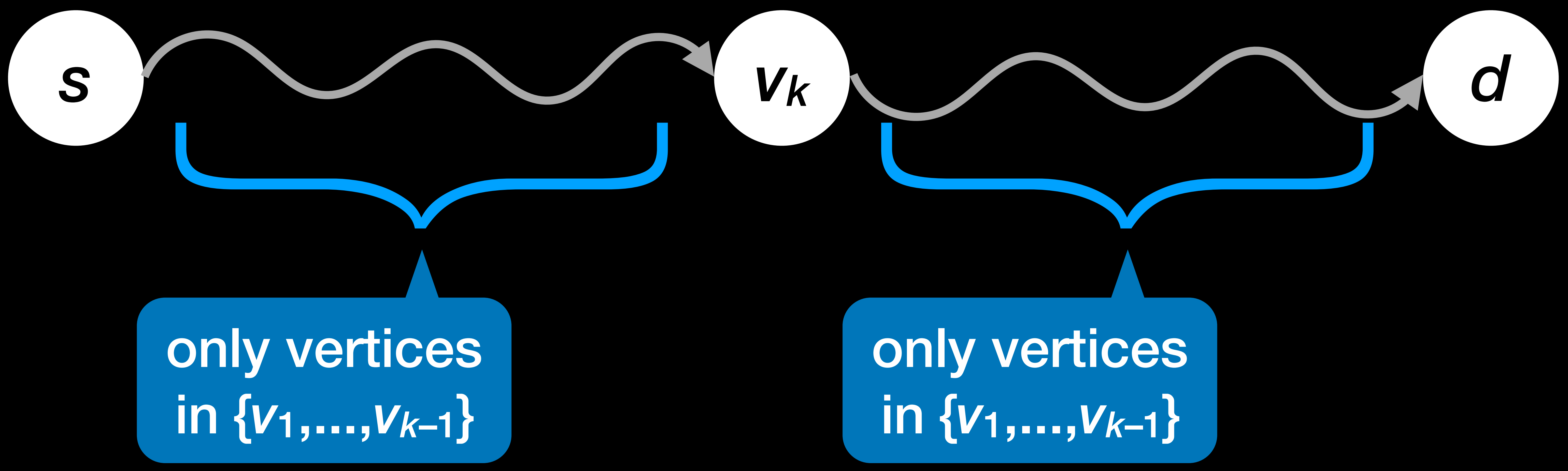
- $W^{(m)}_{ij}$ = weight of the shortest path from i to j, consisting of $\leq m$ edges
- It suffices to search for paths containing $\leq n-1$ edges (if there are no negative-weight cycles).
- Needs proof that the loop body correctly calculates $W^{(2m)}$.

$\otimes$: Running Time

- The line "$W^{(2m)} = W^{(m)} \otimes W^{(m)}$" hides $O(n^3)$ operations.
- To reach $m \geq n-1$, we need $O(\log n)$ iterations through the loop.

Floyd–Warshall Algorithm

- How can a path be decomposed into simpler parts?
  - Algorithm using ⊗: two parts with equal (maximal) number of edges
  - Floyd–Warshall: two parts that only visit a subset of vertices $\{v_1,\ldots,v_{k-1}\} \subset V$ and a highest-numbered vertex v_k



Floyd–Warshall Algorithm

- Idea: use dynamic programming.
  Calculate the shortest path from s to d that only passes through vertices $\{v_1,...,v_k\} \subset V$, for every $k = 1,2,...,n$

- $d^{(k)}_{ij}$ = distance from v_i to v_j when passing only through vertices $\{v_1,...,v_k\}$
  $= \min\left(d^{(k-1)}_{ij} \ ,\ d^{(k-1)}_{ik} + d^{(k-1)}_{kj} \right)$

Floyd–Warshall Algorithm

FLOYD–WARSHALL(W)
$n = W.rows$
$D^{(0)} = W$
for $k = 1$ **to** n
    Let $D^{(k)}$ be a new $n{\times}n$-matrix
    for $i = 1$ **to** n
        for $j = 1$ **to** n
            $d^{(k)}_{ij} = \min\left(d^{(k-1)}_{ij} , d^{(k-1)}_{ik} + d^{(k-1)}_{kj} \right)$
return $D^{(n)}$

Floyd–Warshall Algorithm: π

- One can calculate the paths together with the D values:
  $\pi^{(k)}_{ij}$ = last vertex visited before v_j on the shortest path found for $d^{(k)}_{ij}$

if $d^{(k-1)}_{ij} \leq d^{(k-1)}_{ik} + d^{(k-1)}_{kj}$

$$d^{(k)}_{ij} = d^{(k-1)}_{ij}$$

$$\pi^{(k)}_{ij} = \pi^{(k-1)}_{ij}$$

else

$$d^{(k)}_{ij} = d^{(k-1)}_{ik} + d^{(k-1)}_{kj}$$

$$\pi^{(k)}_{ij} = \pi^{(k-1)}_{kj}$$

- Initialization: $\pi^{(0)}_{ij} =$

Floyd–Warshall Algorithm: π

- One can calculate the paths together with the D values:
  $\pi^{(k)}_{ij}$ = last vertex visited before v_j on the shortest path found for $d^{(k)}_{ij}$

if $d^{(k-1)}_{ij} \leq d^{(k-1)}_{ik} + d^{(k-1)}_{kj}$

$d^{(k)}_{ij} = d^{(k-1)}_{ij}$

$\pi^{(k)}_{ij} = \pi^{(k-1)}_{ij}$

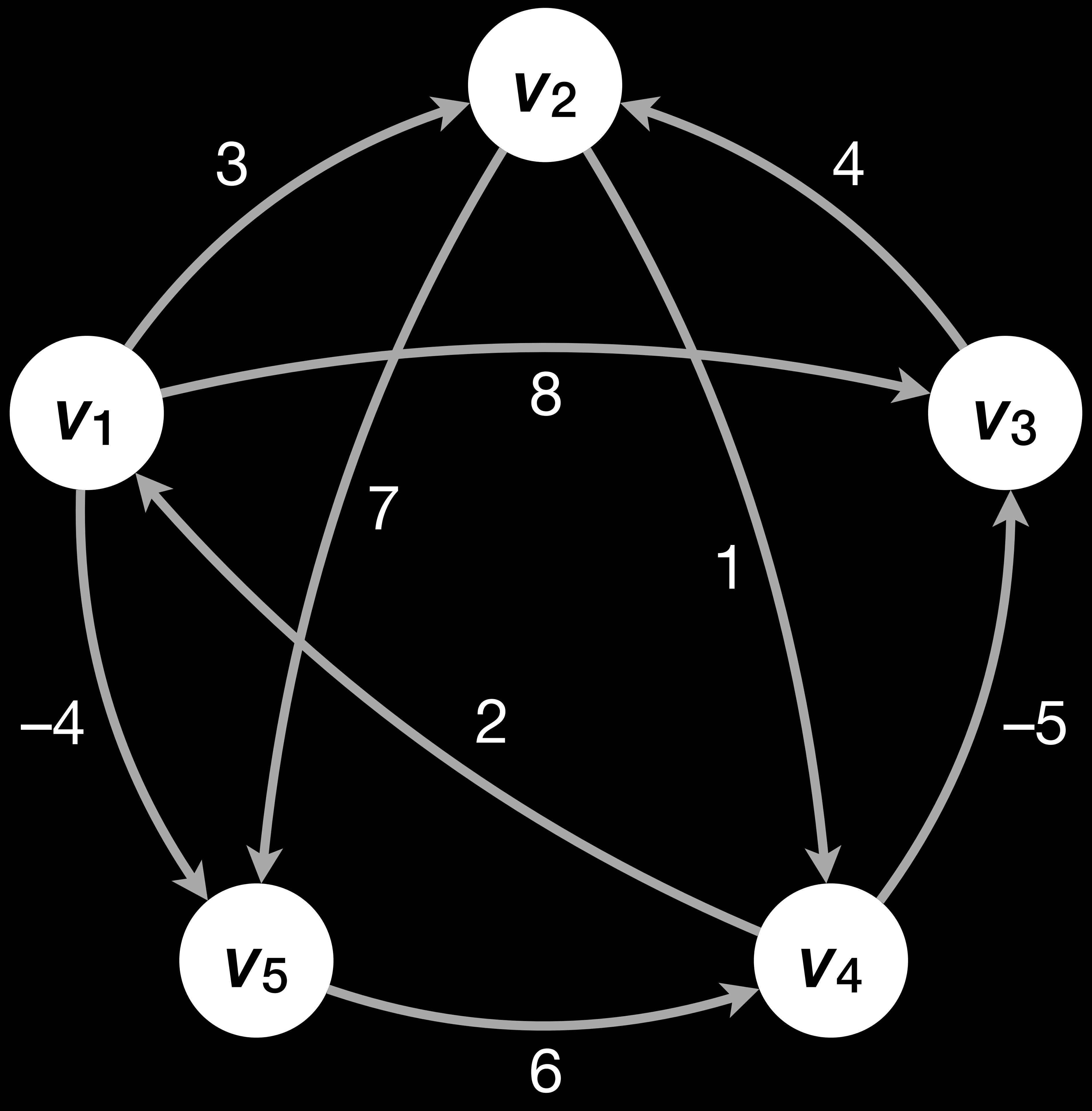
else

$d^{(k)}_{ij} = d^{(k-1)}_{ik} + d^{(k-1)}_{kj}$

$\pi^{(k)}_{ij} = \pi^{(k-1)}_{kj}$

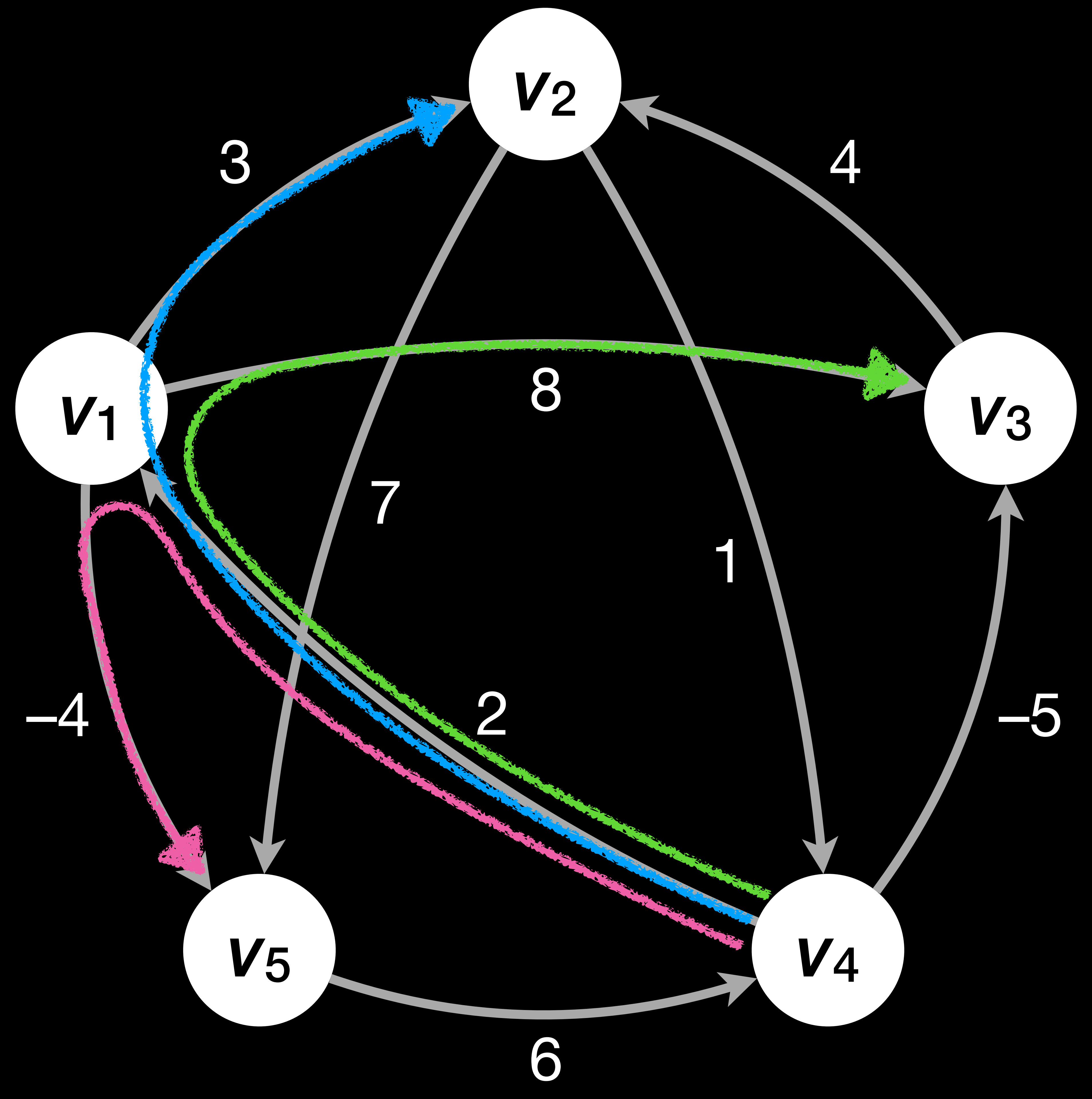
- Initialization: $\pi^{(0)}_{ij} = \begin{cases} \text{NIL} & \text{if } i = j \\ v_i & \text{if } i \neq j \end{cases}$

Floyd–Warshall: Example



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

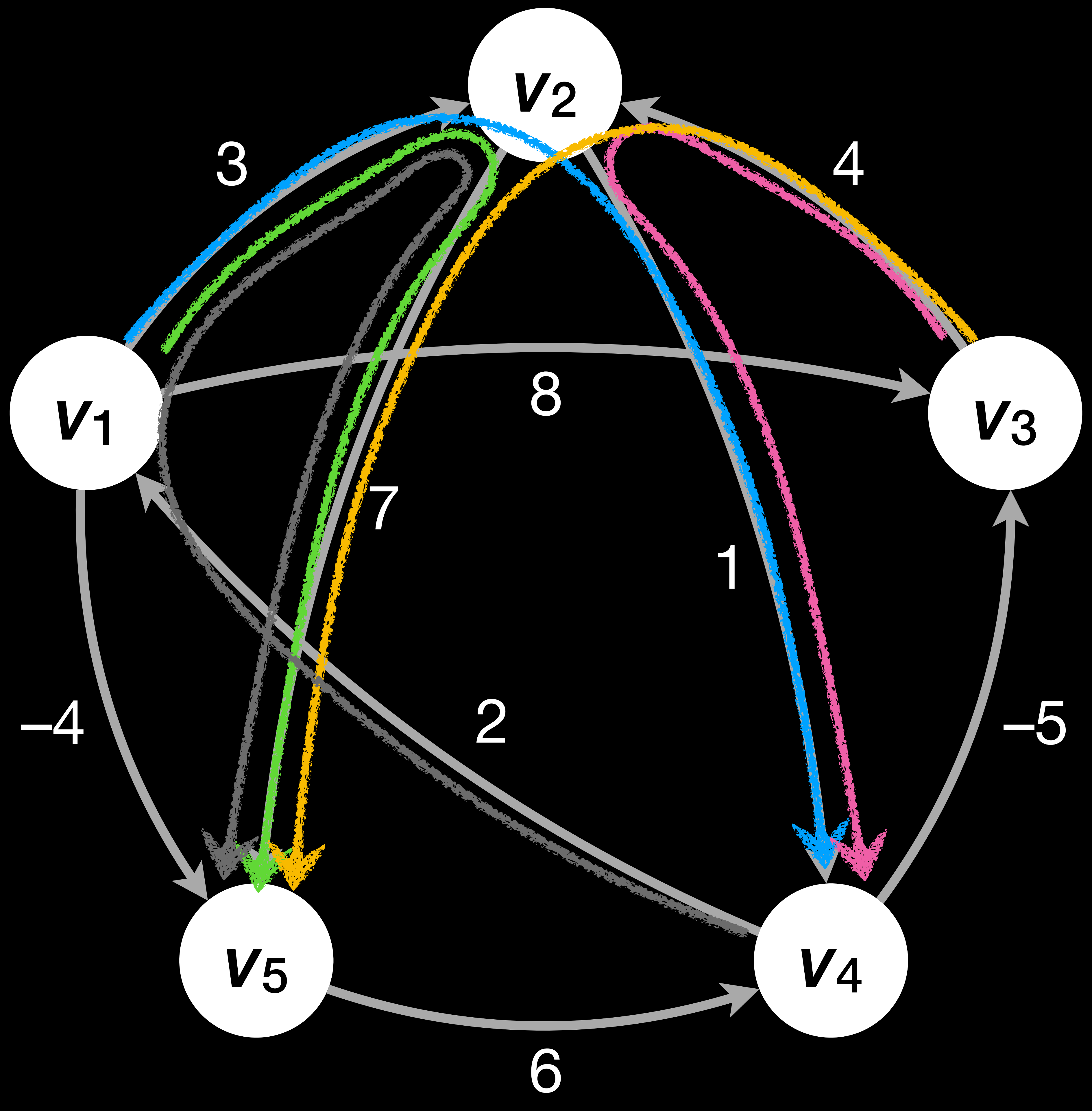
Floyd–Warshall: Example



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

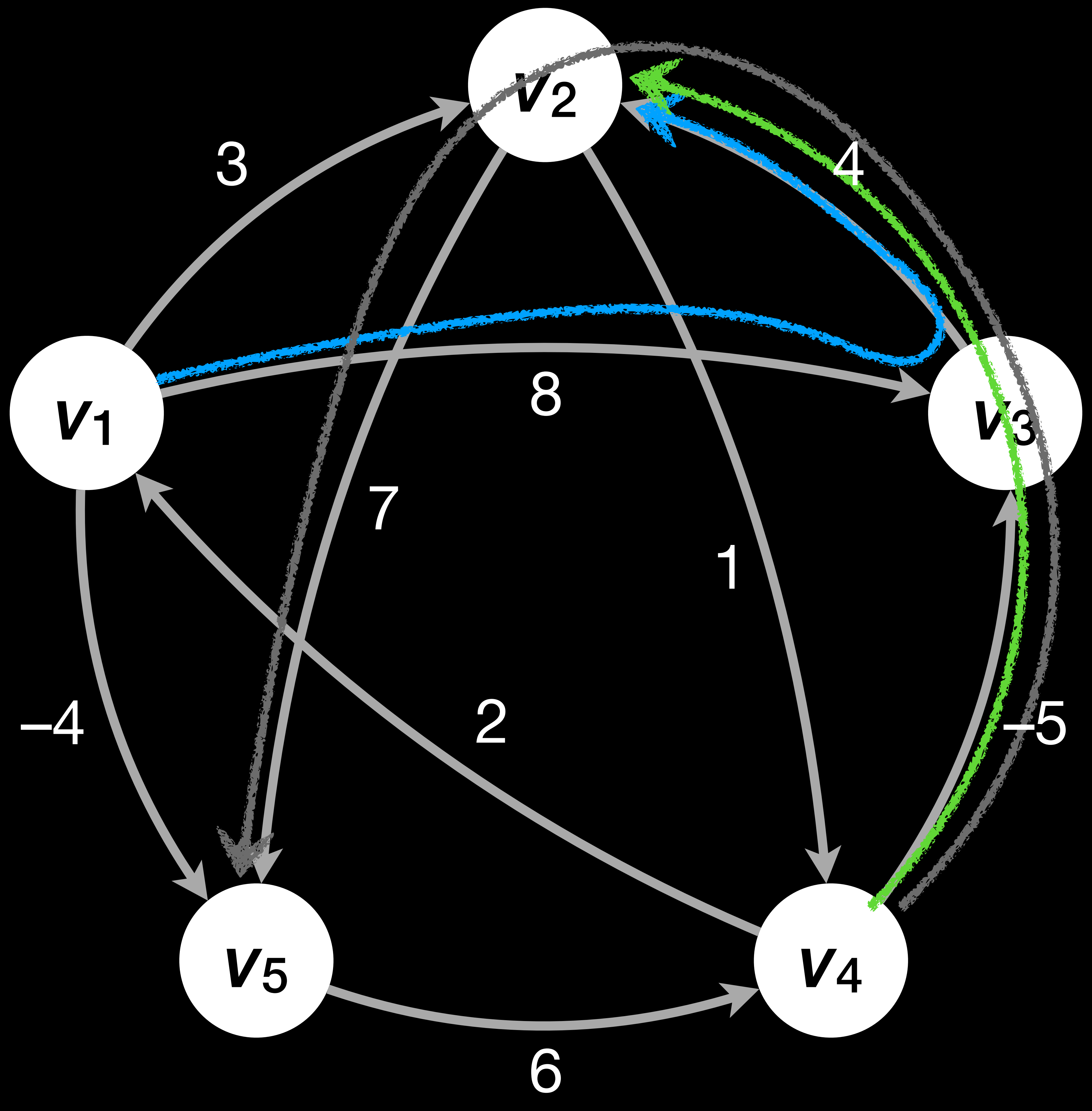
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Floyd–Warshall: Example



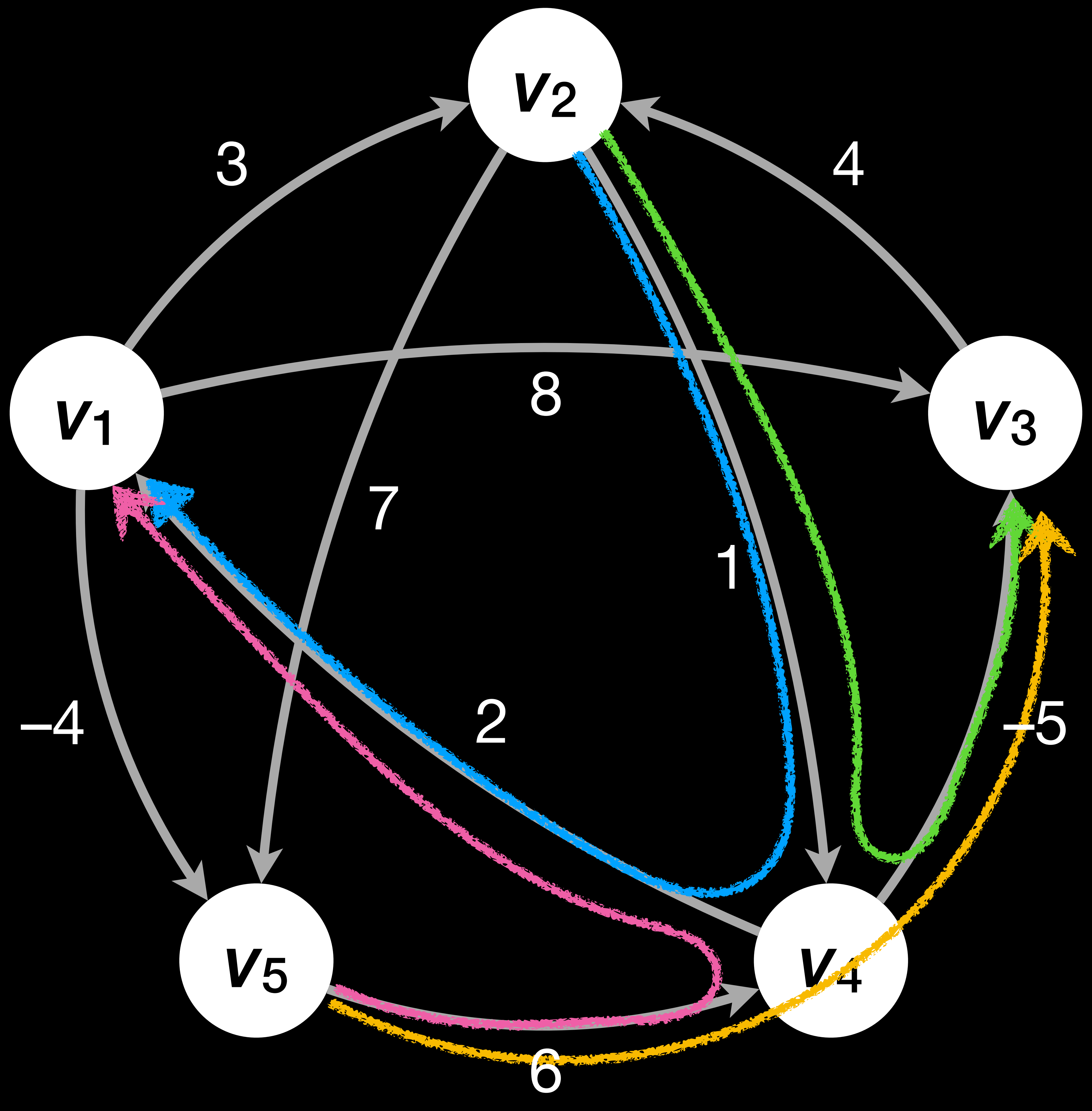
$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Floyd–Warshall: Example



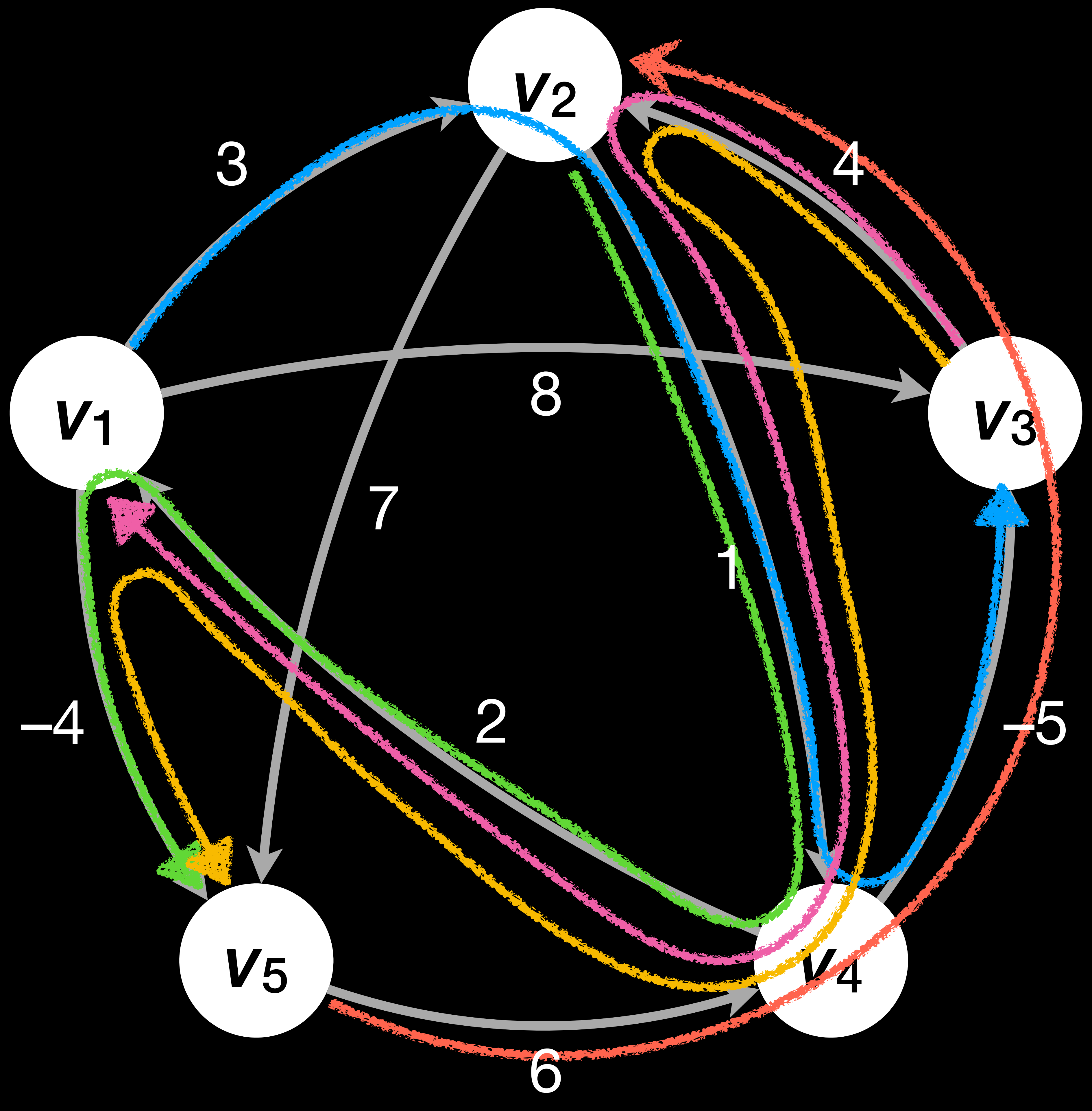
$$D^{(3)} = \begin{pmatrix} 0 & \boxed{3} & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & \boxed{-1} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Floyd–Warshall: Example



$$D^{(4)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \boxed{3} & 0 & \boxed{-4} & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \boxed{8} & \infty & \boxed{1} & 6 & 0 \end{pmatrix}$$

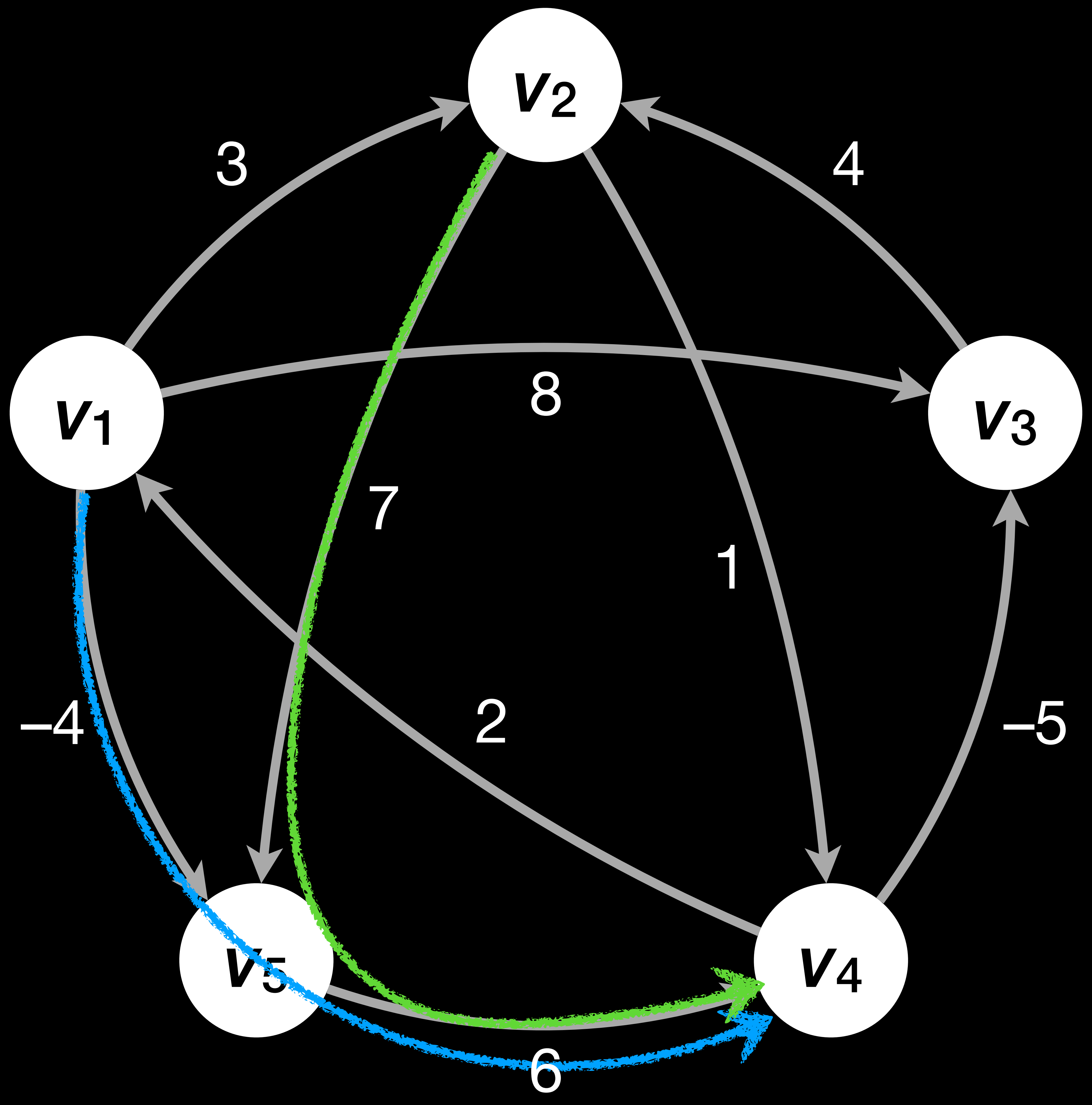
Floyd–Warshall: Example



$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

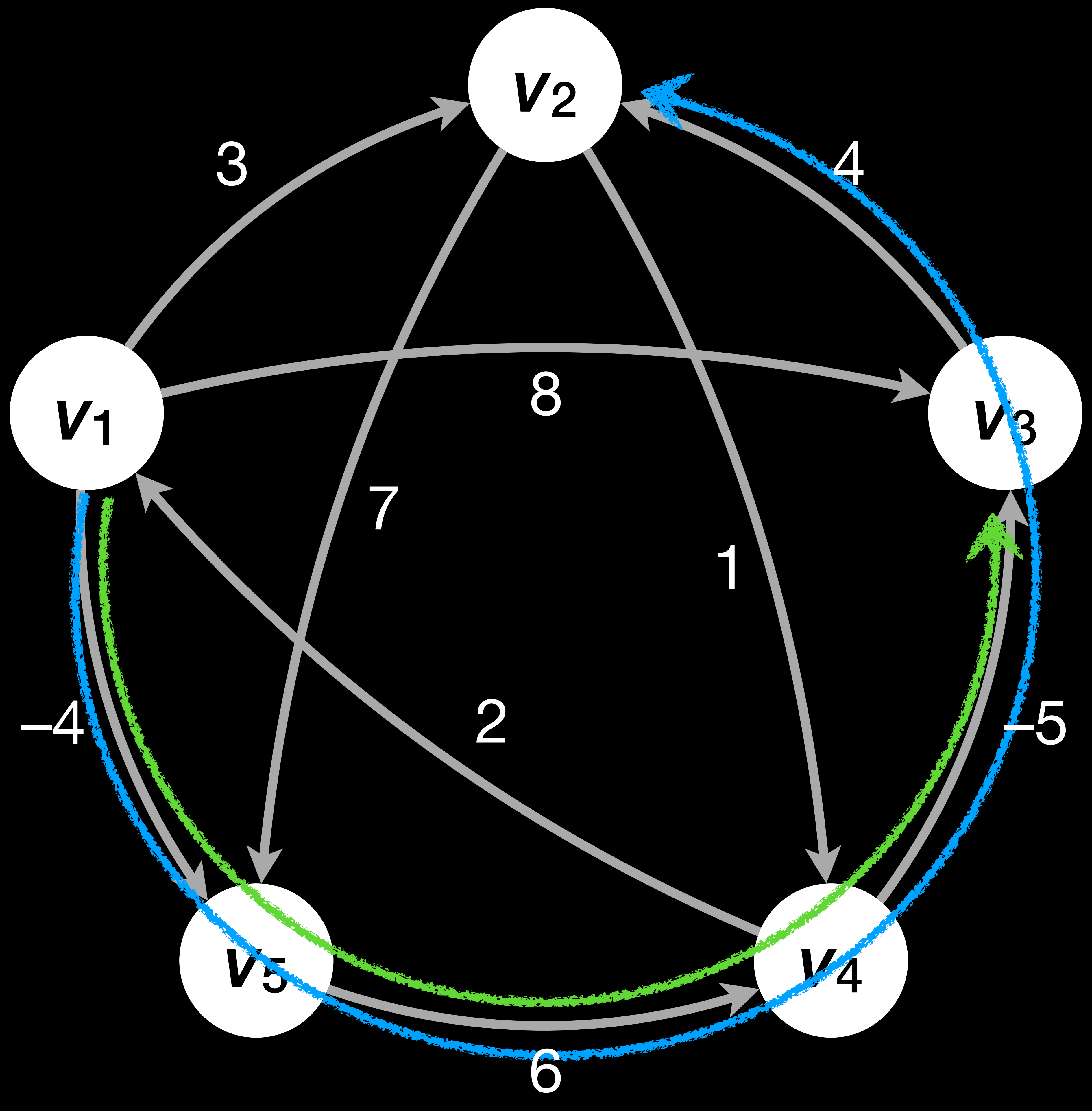
$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Floyd–Warshall: Example



$$D^{(5)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

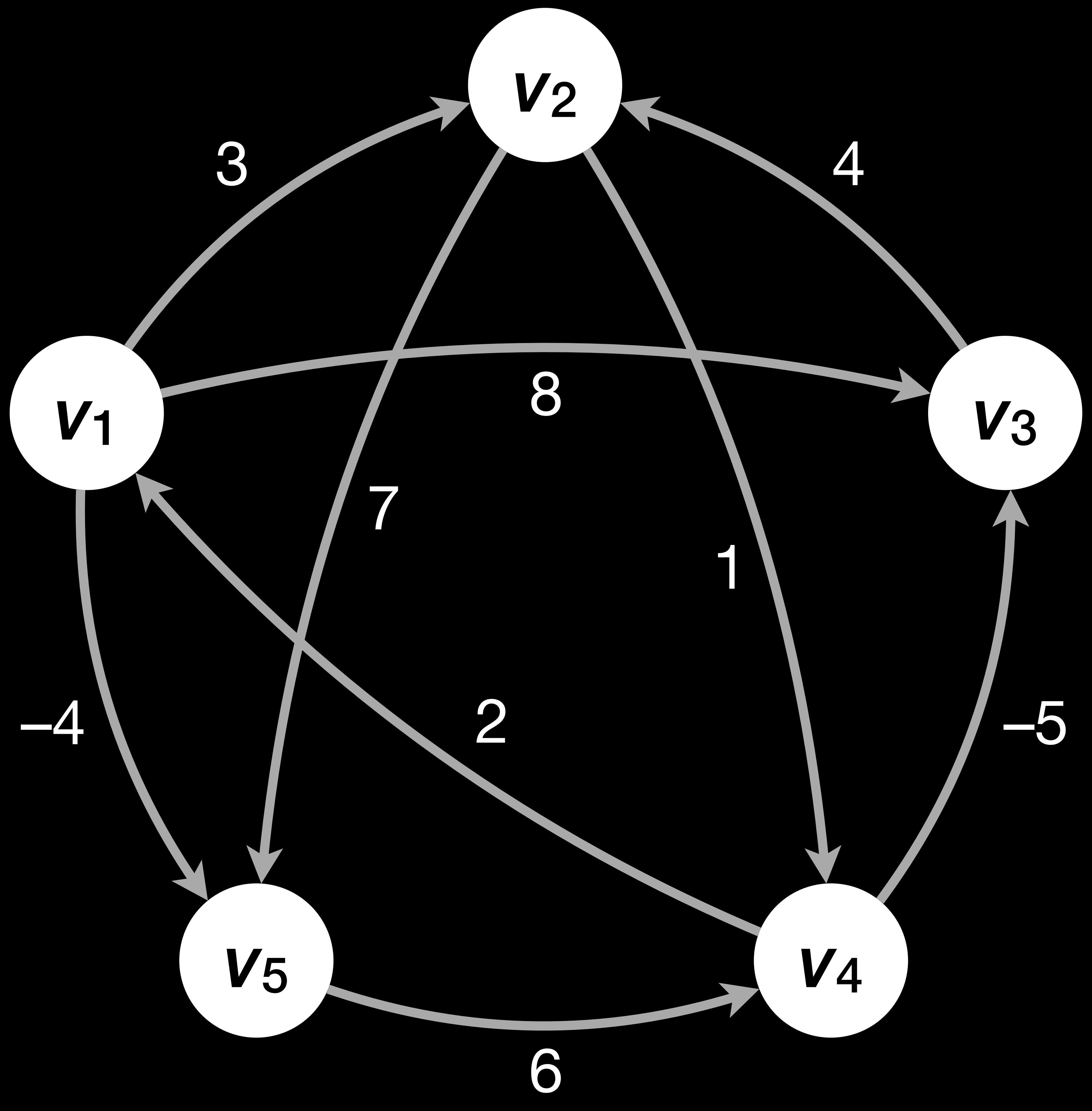
Floyd–Warshall: Example



$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Floyd–Warshall: Example



$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Floyd–Warshall: Partial Correctness

The correctness of Floyd–Warshall depends on these three parts:

- The definition of distance is correct:
  $d^{(k)}_{ij}$ = distance from v_i to v_j when passing only through vertices $\{v_1,...,v_k\}$
  $= \min\left(d^{(k-1)}_{ij} \;,\; d^{(k-1)}_{ik} + d^{(k-1)}_{kj} \right)$

- The algorithm correctly computes $d^{(0)}_{ij}, ..., d^{(n)}_{ij}$.
  (Proof by induction over the loop body.)

- The distance $d^{(n)}_{ij}$ is the true shortest-path distance $\delta(v_i,v_j)$.

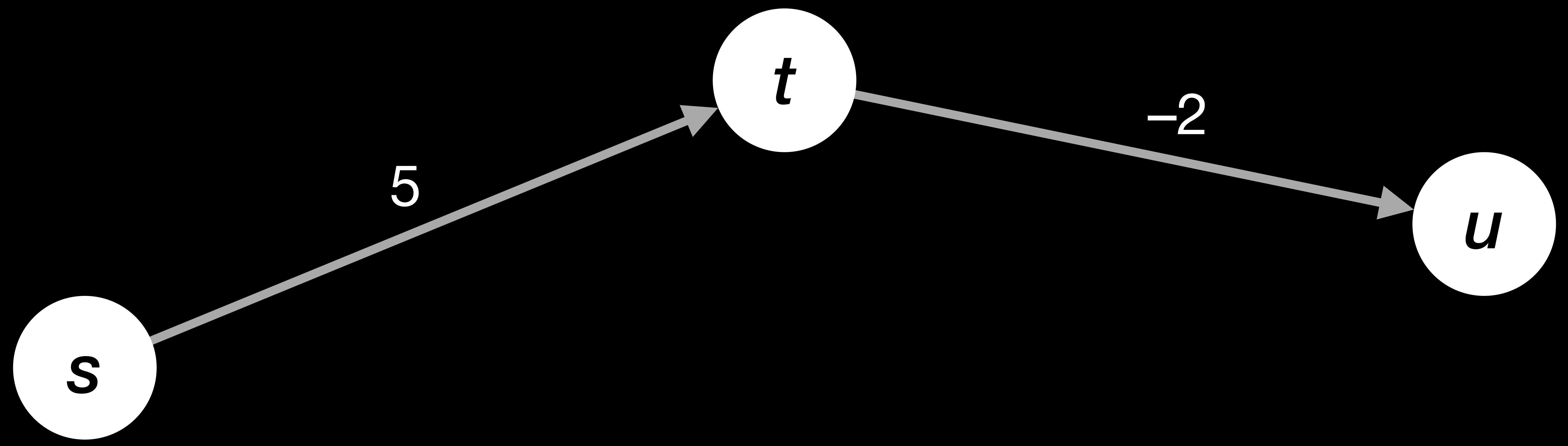
Floyd–Warshall: Running Time

- Three nested **for** ... 1 **to** n loops; simple operation within the loop.
- We need to allocate new $n \times n$-matrices $D^{(k)}$.
  In fact, only one matrix is needed.
  The allocation and initialisation of the new matrix may require time $O(n^2)$.
- The total running time is in $O(n^3)$.

Johnson’s Algorithm

- Idea:
  Change the graph to make it suitable for the efficient Dijkstra’s algorithm.

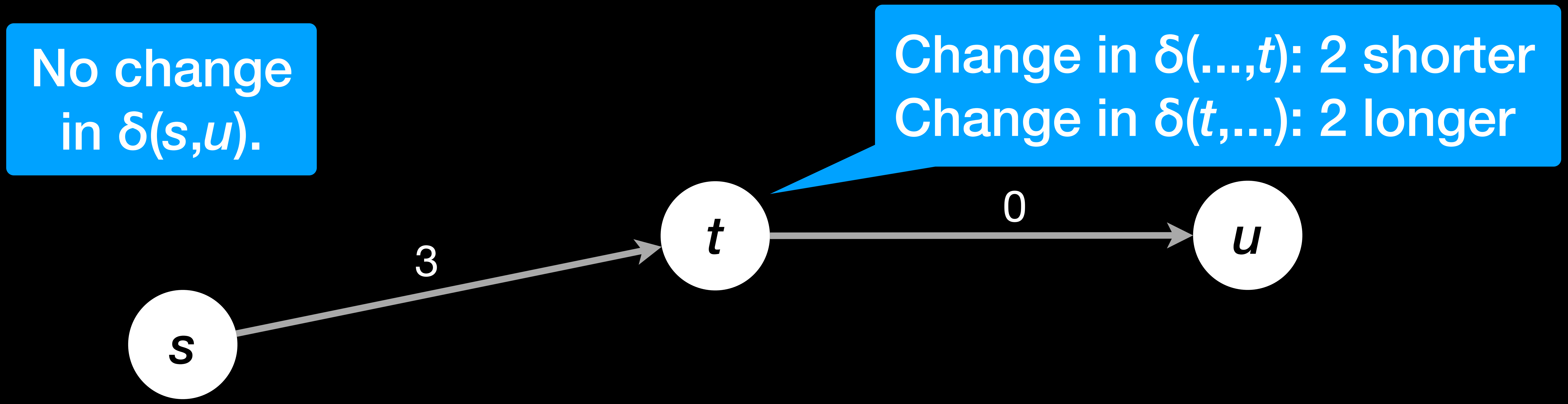
- Dijkstra’s algorithm requires that all weights be ≥0.
  If there are negative weights,
  change the “height” of some nodes to make all weights ≥0.



Johnson's Algorithm

- Idea:
  Change the graph to make it suitable for the efficient Dijkstra's algorithm.

- Dijkstra's algorithm requires that all weights be ≥0.
  If there are negative weights,
  change the "height" of some nodes to make all weights ≥0.



Johnson's Algorithm

- Changing the height = "reweighting"

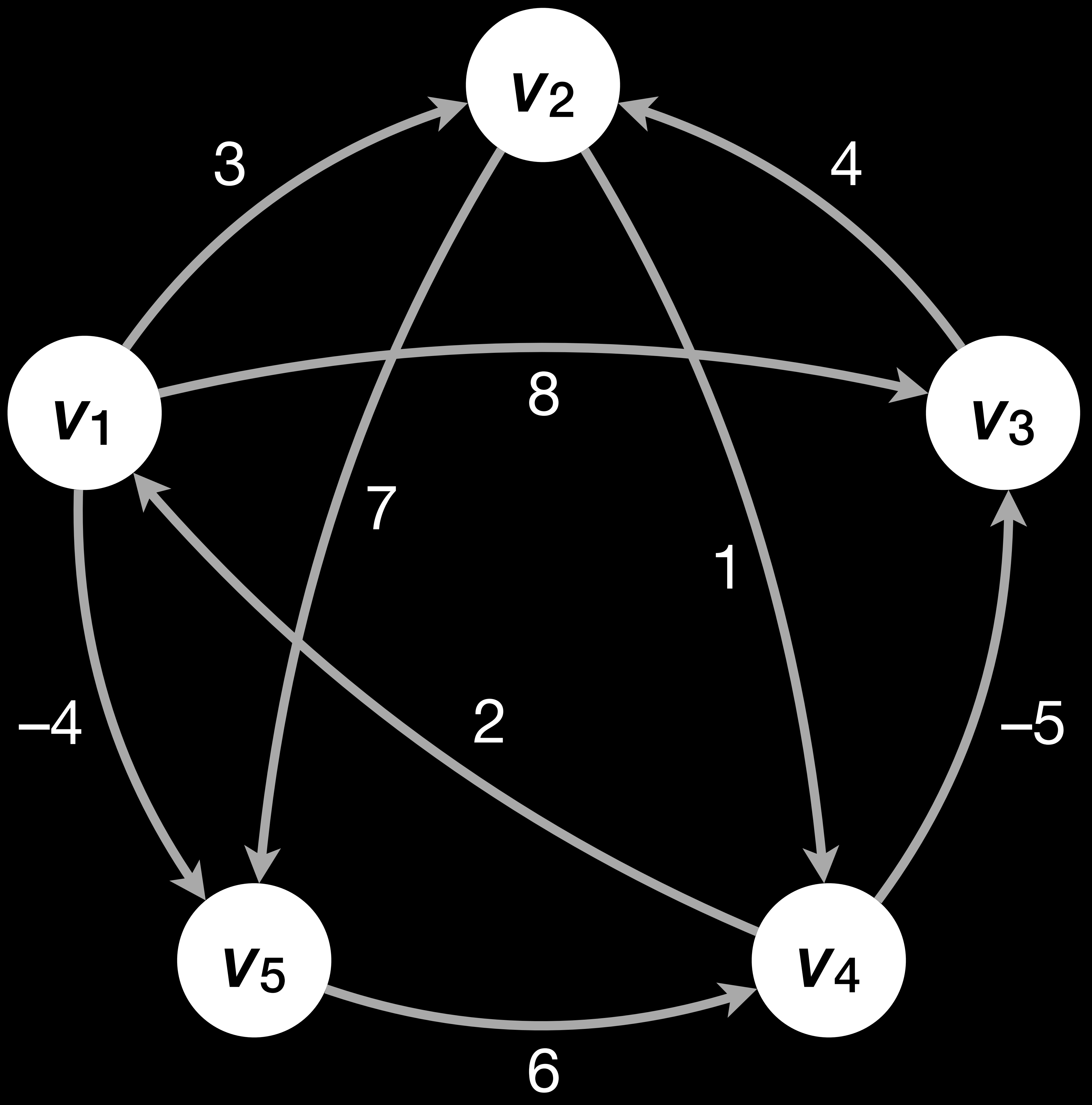
- Q: How can we calculate the correct height for every vertex v?
  A: Find the shortest path from a neutral vertex to v.

- Overview of Johnson's Algorithm:
  1. calculate the heights $h(v)$ (using the Bellman–Ford algorithm)
  2. calculate new weights $\hat{w}$ for every edge
  3. Repeat Dijkstra's algorithm for every source, with weights $\hat{w}$, to calculate $\hat{d}(u,v)$ for every pair of vertices
  4. calculate $d(u,v)$ using $\hat{d}$ and h

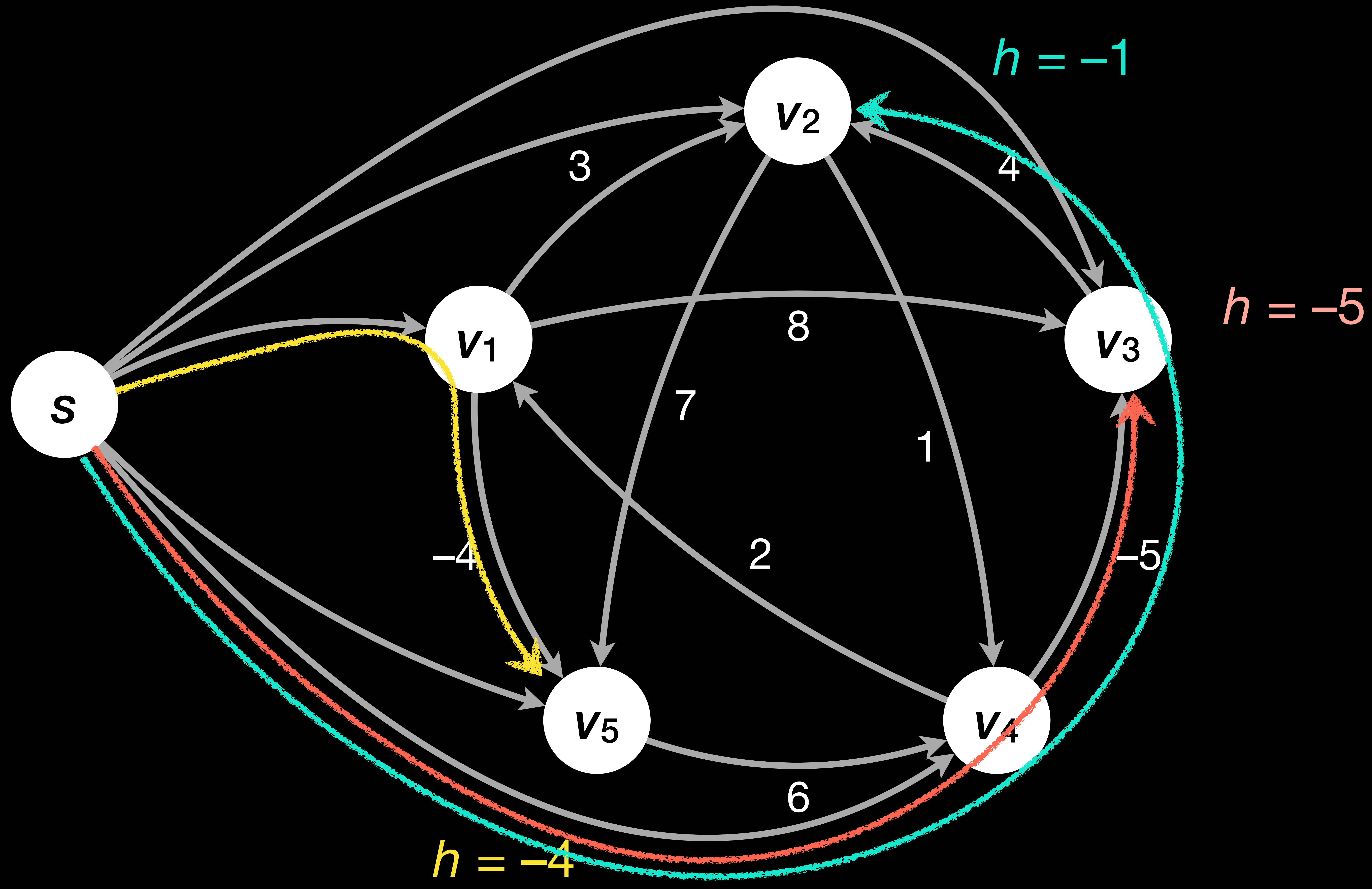
Johnson's Algorithm

JOHNSON(G,w)
Let $G' = (G.V \cup \{s\}, G.E \cup \{(s,v) \mid v \in G.V\})$ and $w(s,v) = 0$
BELLMAN-FORD(G',w,s)
if there is a negative-weight cycle
    return "There is a negative-weight cycle."
for each vertex $v \in G.V$
    $h(v) = v.d$
for each edge $(u,v) \in G.E$
    $\hat{w}(u,v) = w(u,v) + h(u) - h(v)$
Let $D = (d_{uv})$ be a new $|G.V| \times |G.V|$-matrix
for each vertex $u \in G.V$
    DIJKSTRA(G,$\hat{w}$,u)
    for each vertex $v \in G.V$
        $d_{uv} = v.d + h(v) - h(u)$
return D

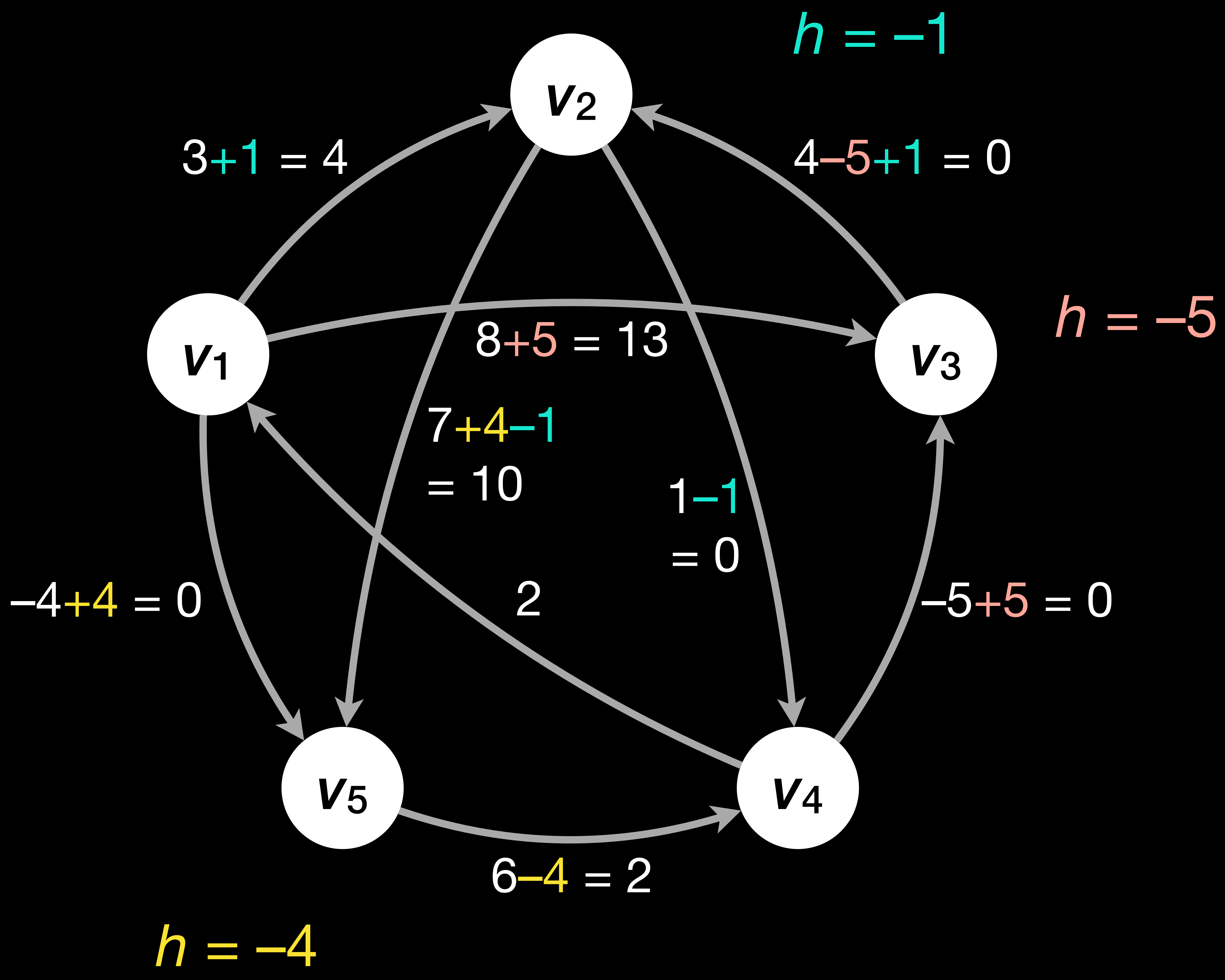
Johnson's Algorithm: Example



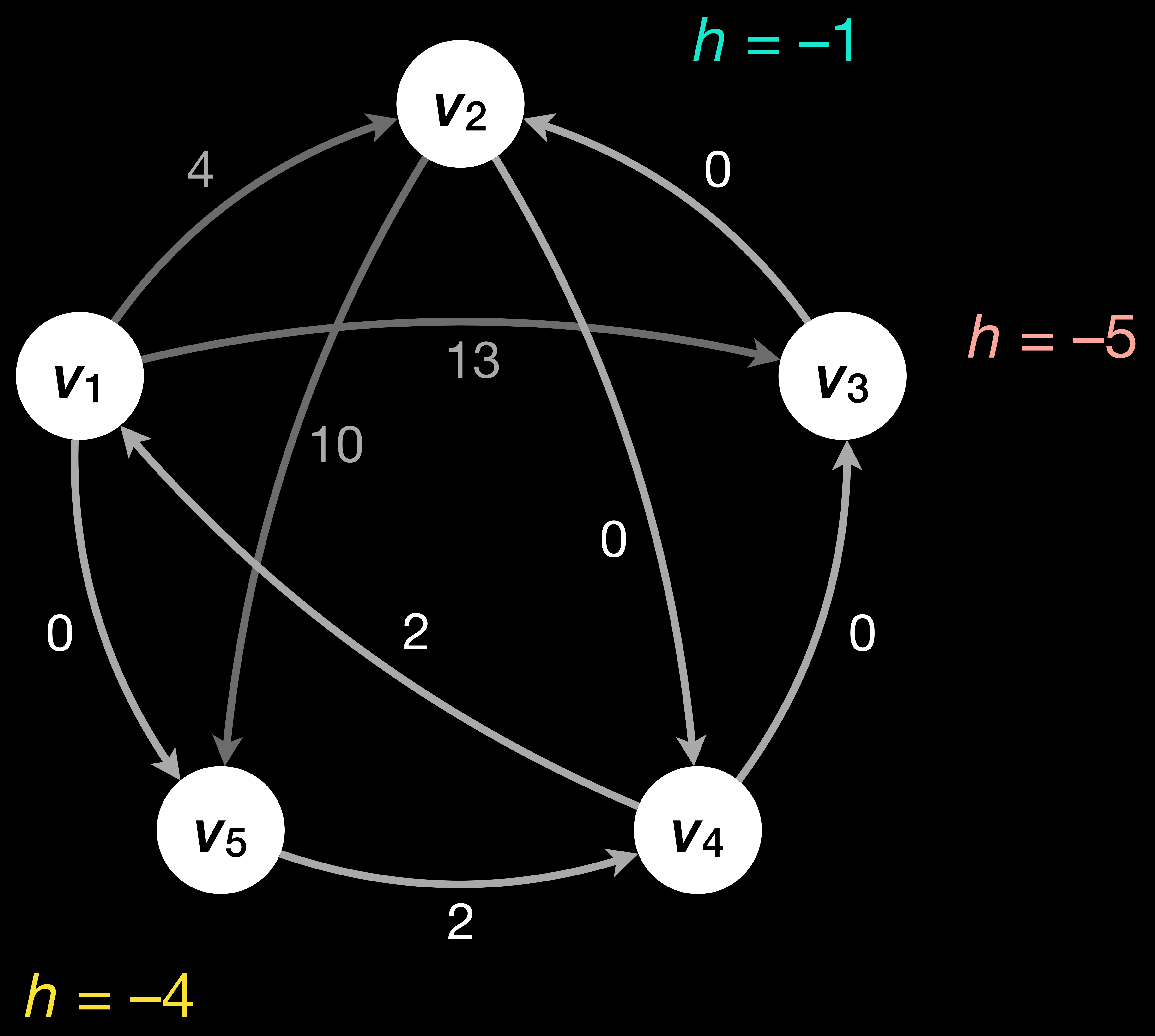
Johnson's Algorithm: Example



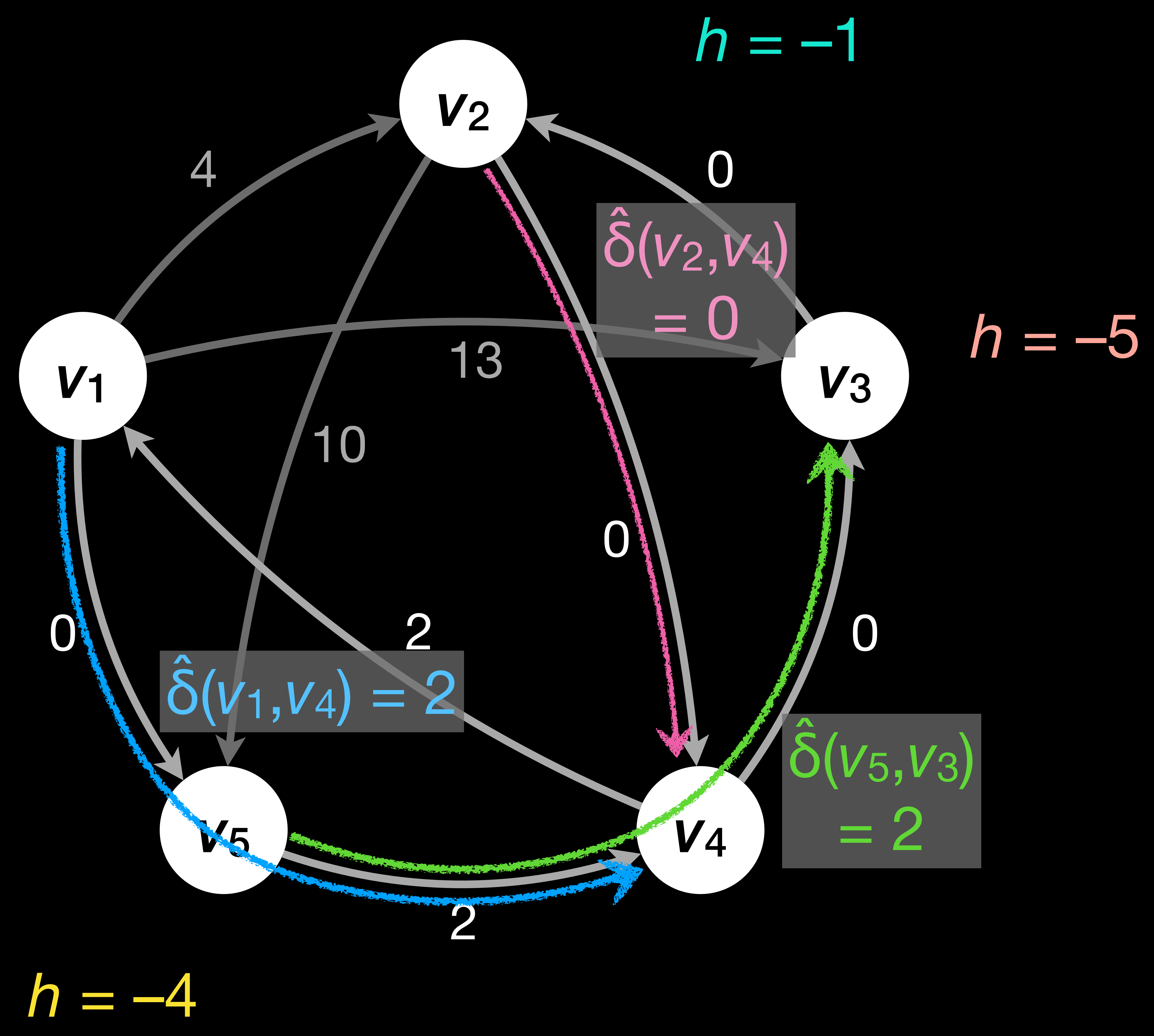
Johnson's Algorithm: Example

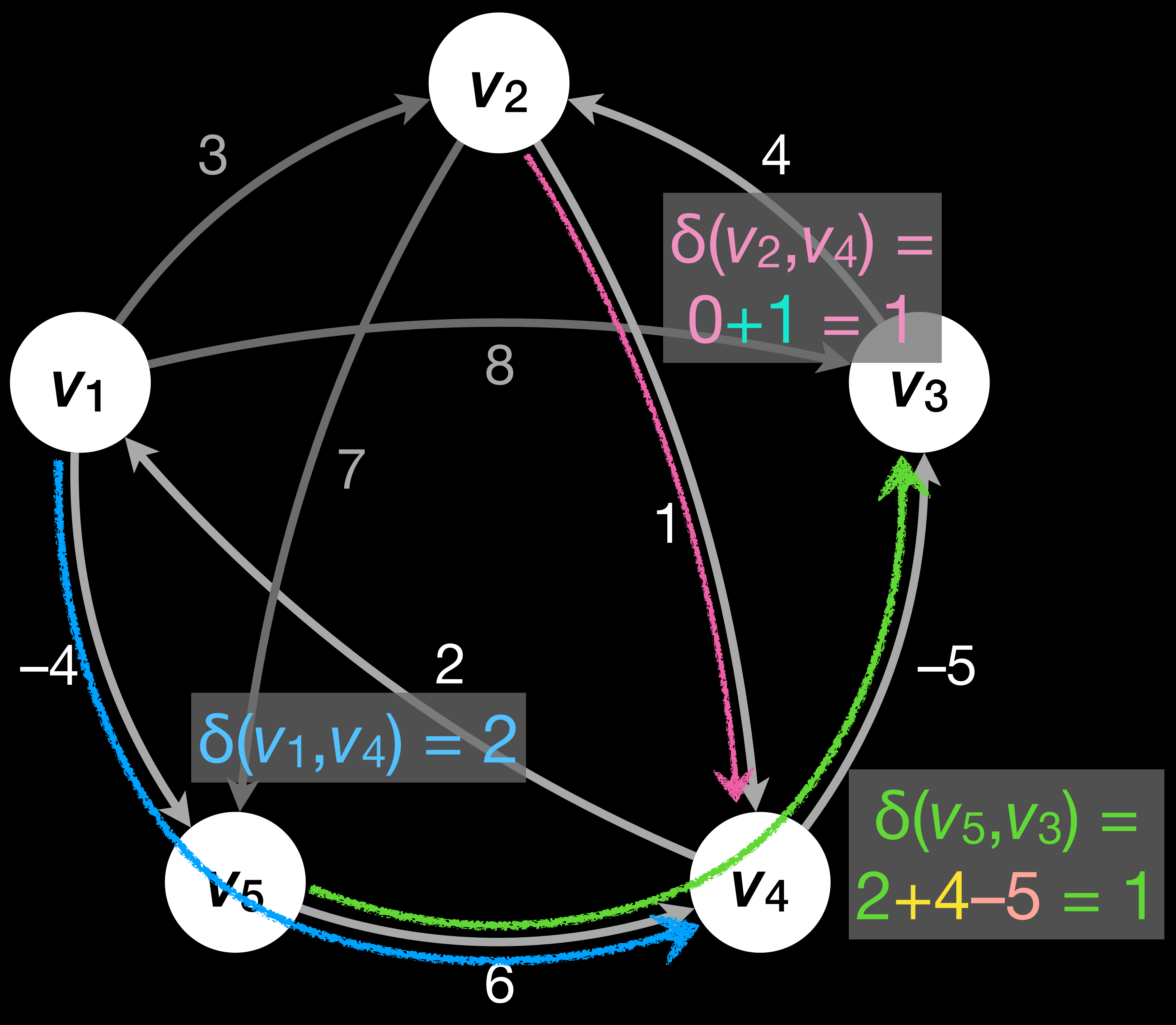


Johnson's Algorithm: Example



Johnson's Algorithm: Example





Johnson's Algorithm: Partial Correctness

- Lemma 25.1 (Reweighting does not change shortest paths)
  Given a weighted directed graph $G = (V,E)$ with $w{:}E \to \mathbb{R}$.
  Let $h{:}\mathrm{V} \to \mathbb{R}$ be any function assigning heights to vertices.
  Let $\hat{w}(u,v) = w(u,v) + h(u) - h(v)$.
  Let p be any path from vertex v_0 to vertex v_k.
  Then p is a shortest path in (G,w) iff p is a shortest path in $(G,\hat{w})$.
  Furthermore, (G,w) has a negative-weight cycle iff $(G,\hat{w})$ has a negative-weight cycle.

Johnson's Algorithm: Running Time

- The algorithm calls BELLMAN–FORD once, requiring time in $O(|V|\cdot|E|)$.
- It calls DIJKSTRA $|V|$ times, each call requiring time in $O(|E| \log |V|)$.
- Other operations require less time: $O(|V|^2 + |E|)$.
- Therefore, the overall running time is in $O(|V|\cdot|E| \log |V|)$.

Quiz

- What is the main idea of each of the following algorithms?
  - Floyd–Warshall
  - Johnson
- Is is possible to use Johnson’s reweighting idea for a single-source shortest path algorithm?