## Today’s Lecture

- Dynamic programming to find highest weight paths in WDAGs

# Weighted Directed Graphs

- A *weighted directed graph* is
  - a directed graph ($V$, $E$) together with
  - a function $w$ from $E$ to the real numbers,
    - i.e. with a numerical *weight* $w(e)$ (which may be positive, negative, or 0) associated to each edge $e$.

A weighted DAG is called a WDAG.

- The (*sum*) *weight of a path* is defined to be the sum of the weights on the edges of the path.
- Similarly, the *product weight of a path* is the product of the edge weights
  - usually only consider this when all weights are non-negative.
- weight of a path P is written $w(P)$
- For a path of length 0 (i.e. consisting of a single vertex):
  - the sum weight is 0
  - the product weight is 1

# Highest Weight Paths on WDAGs

- *Problem*: find a path with the highest possible weight.
- *Solution*:
  - "Brute force" approach
    - i.e. simply enumerating all possible paths and comparing their weights

    is usually impractical (too many paths!)
  - Instead, use the method of *dynamic programming* ('The Fundamental Algorithm of Computational Biology').

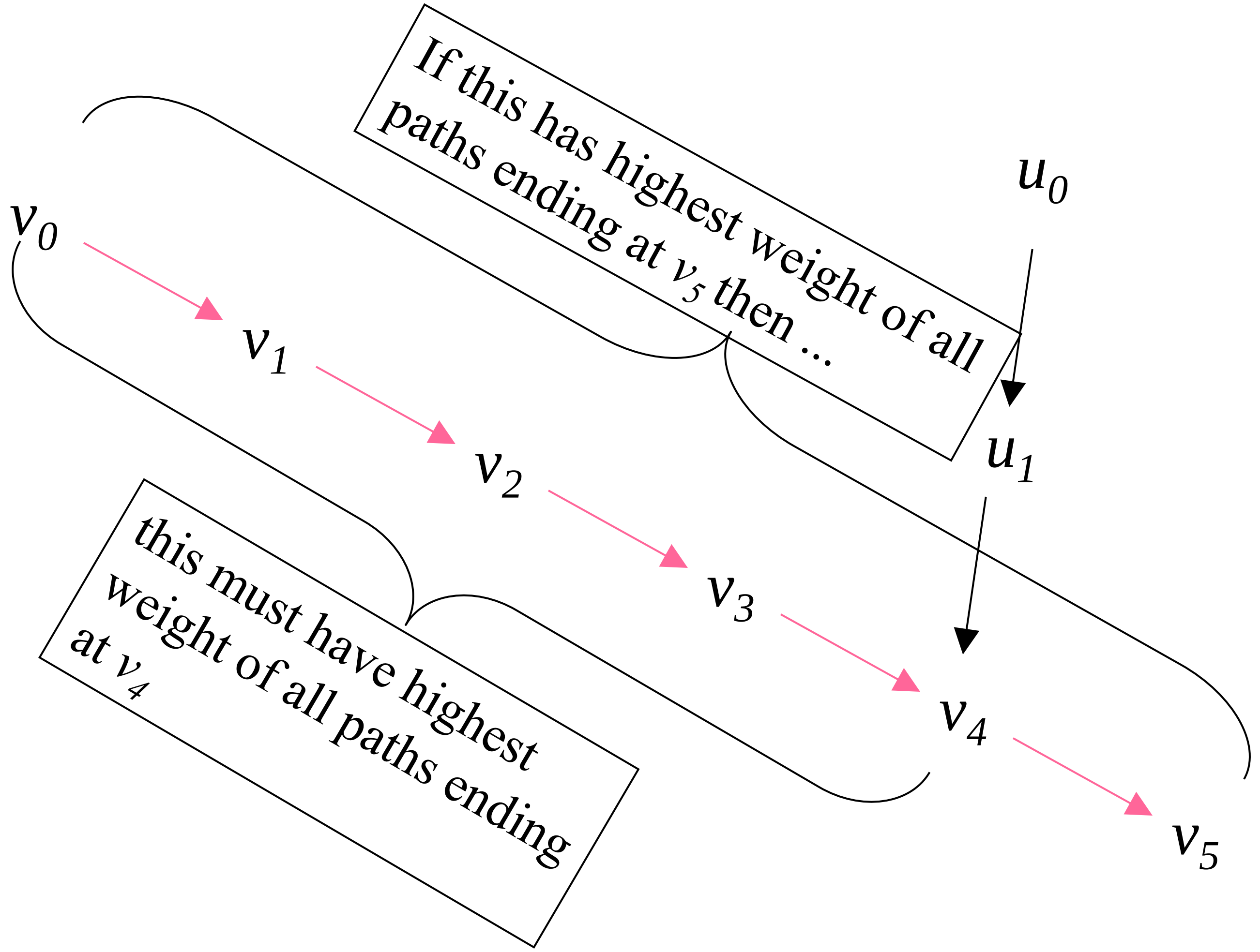
# Highest Weight Paths on WDAGs (cont'd)

- Let $P_n = (v_0, v_1, \ldots, v_n)$ be a path of highest weight.
- Then for each $k < n$, the sub-path $P_k = (v_0, v_1, \ldots, v_k)$ must have highest weight of all paths ending at $v_k$, because
  - *if* $Q = (u_0, u_1, \ldots, v_k)$ were another path ending at $v_k$ and having higher weight than $P_k$,
  - *then* the path $(Q, v_{k+1}, \ldots, v_n)$ would have weight

$$w((Q, v_{k+1}, \ldots, v_n)) = w(Q) + w((v_k, \ldots, v_n))$$
$$> w(P_k) + w((v_k, \ldots, v_n)) = w(P_n),$$

contradicting assumption that $P_n$ has highest weight.

# Subpaths of a highest-weight path can’t be improved:



If this has highest weight of all paths ending at $v_5$ then ...
this must have highest weight of all paths ending at $v_4$
$v_0$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$u_0$
$u_1$

# Highest Weight Paths on WDAGs (cont'd)

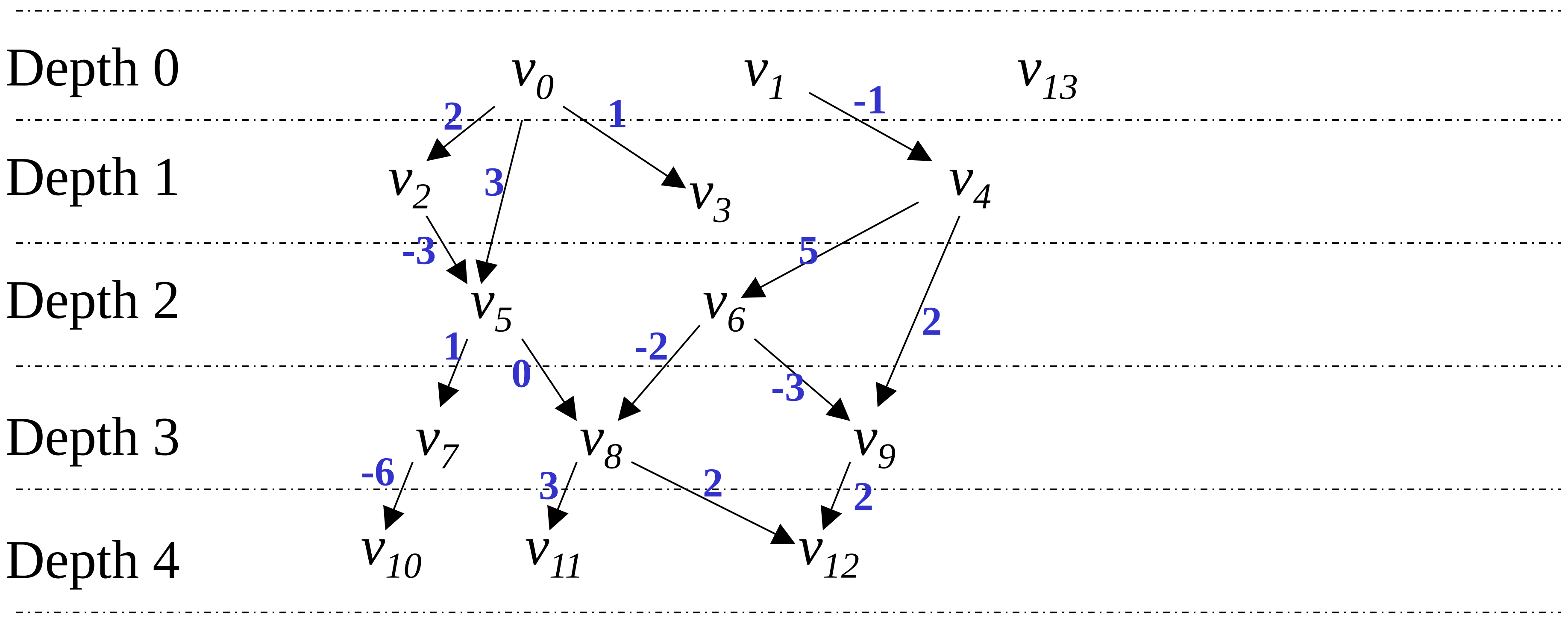
- So generalize the problem as follows:
- find, for *each* vertex $v$, the highest weight of all paths ending at $v$ – call this $w(v)$
- Can find $w(v)$ in single pass through $V$, as follows:
  - process the $v$ in depth order (*or any order in which parents precede children*)
  - if $v$ has no parents, $w(v) = 0$ (the only path ending at $v$ is $(v)$).
  - for any other $v$, except for the path $(v)$ (which has weight 0), any path ending at $v$ is of form $(v_0, v_1, \ldots, v_k, u, v)$. Then
  - $u$ is a parent of $v$, so $w(u)$ has already been computed, and

$$w((v_0, v_1, \ldots, v_k, u, v)) \leq w(u) + w((u,v))$$

    with equality for an appropriate choice of $v_i$.
  - Therefore we may compute $w(v)$ as

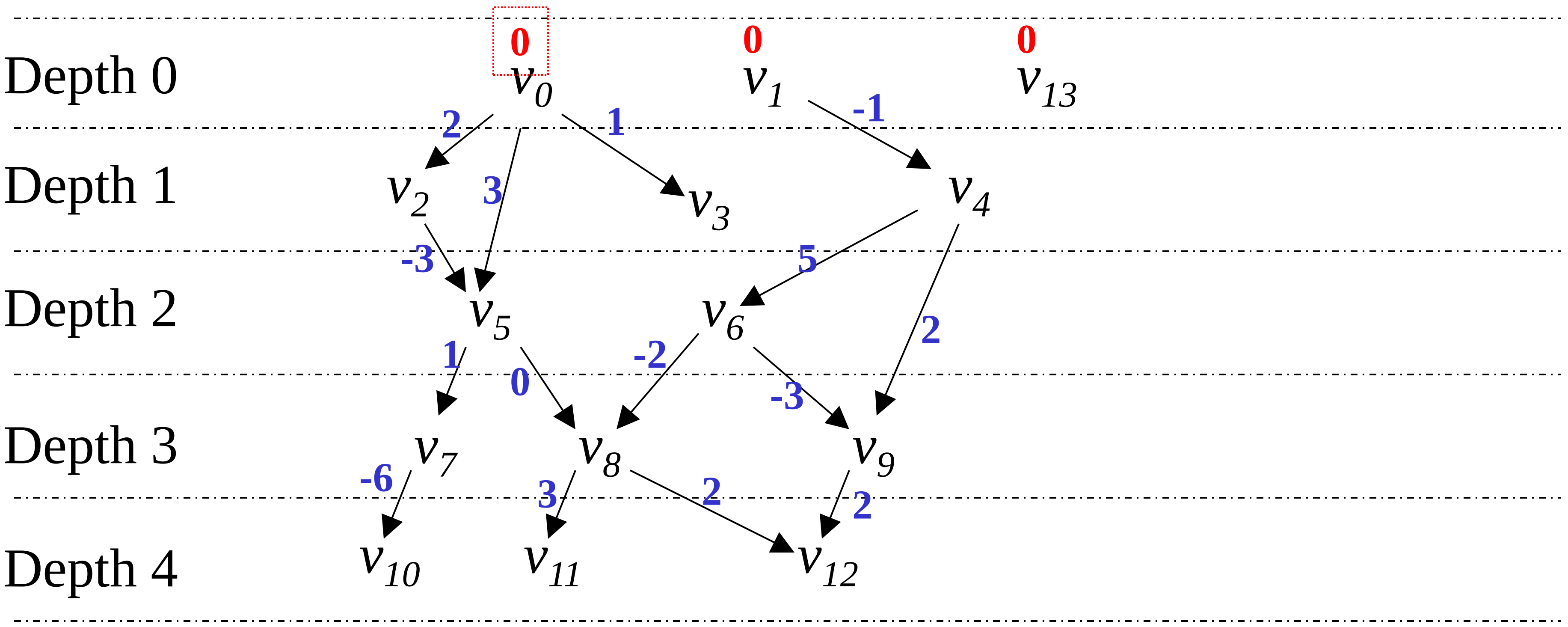
$$w(v) = \max(0, \max_{u \in parents(v)} (w(u) + w((u,v))))$$

# Example

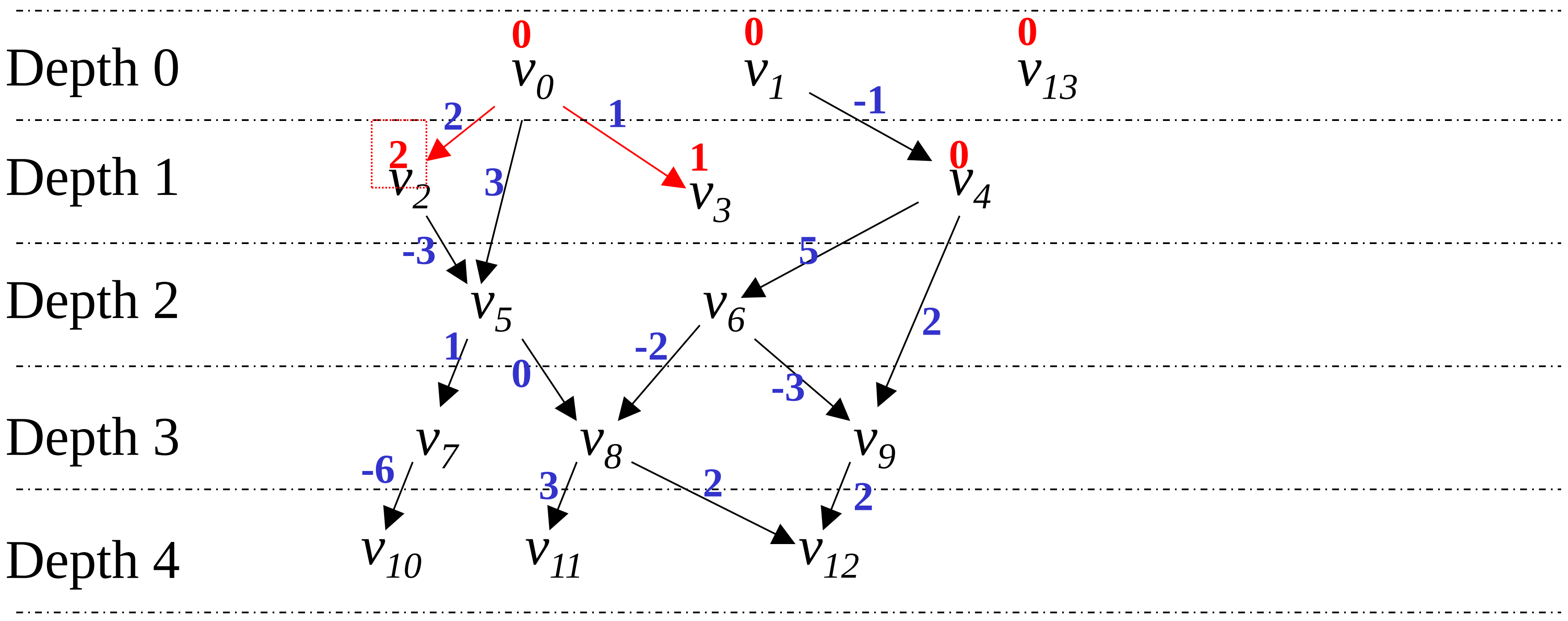


Depth 0
Depth 1
Depth 2
Depth 3
Depth 4
$v_0$
$v_1$
$v_{13}$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
2
1
-1
3
-3
5
2
1
0
-2
-3
-6
3
2
2

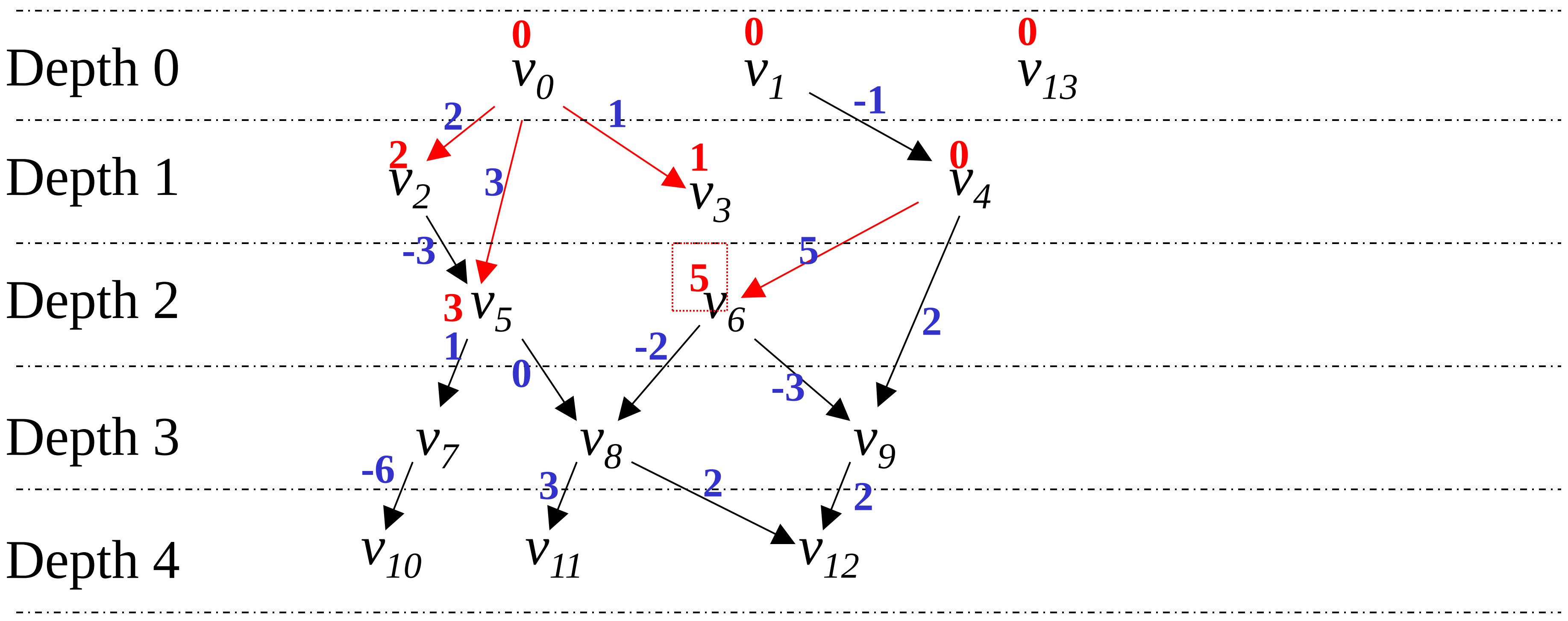
# $w(v)$ – depth 0 nodes



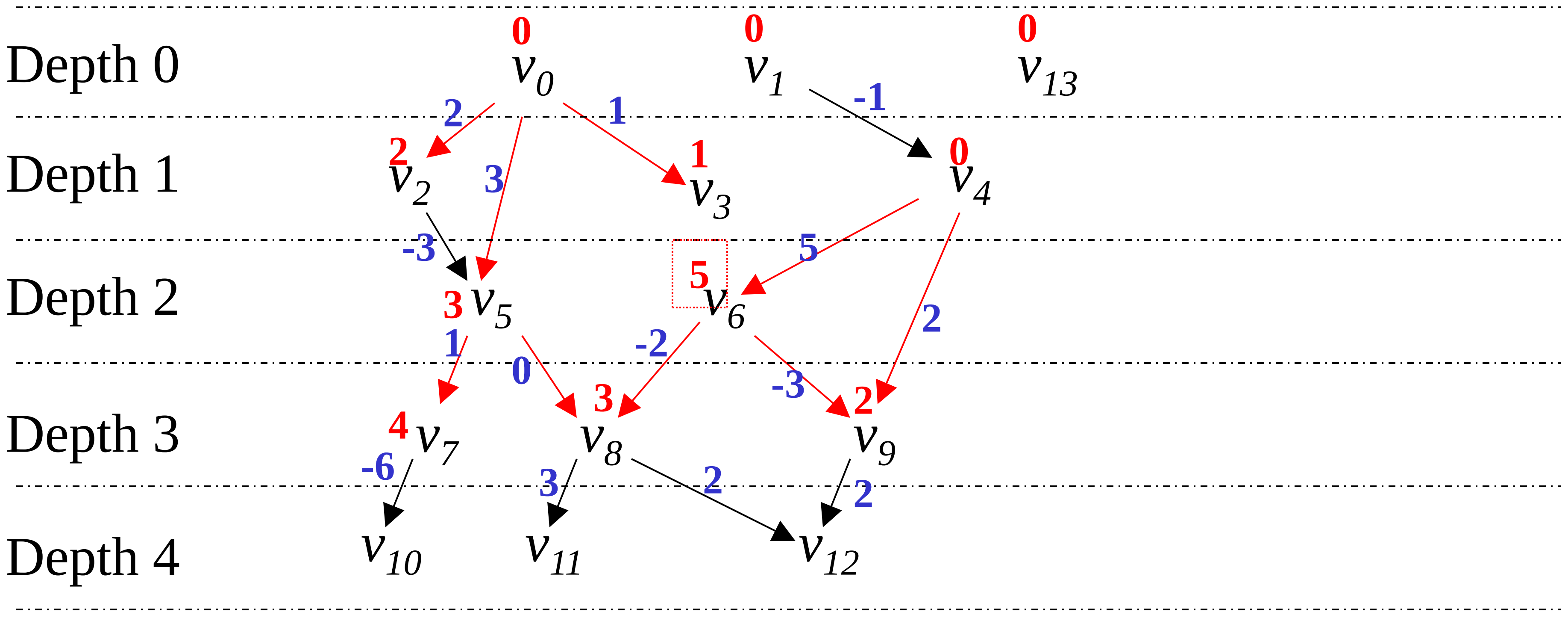
# *w*(*v*) – depth 1 nodes



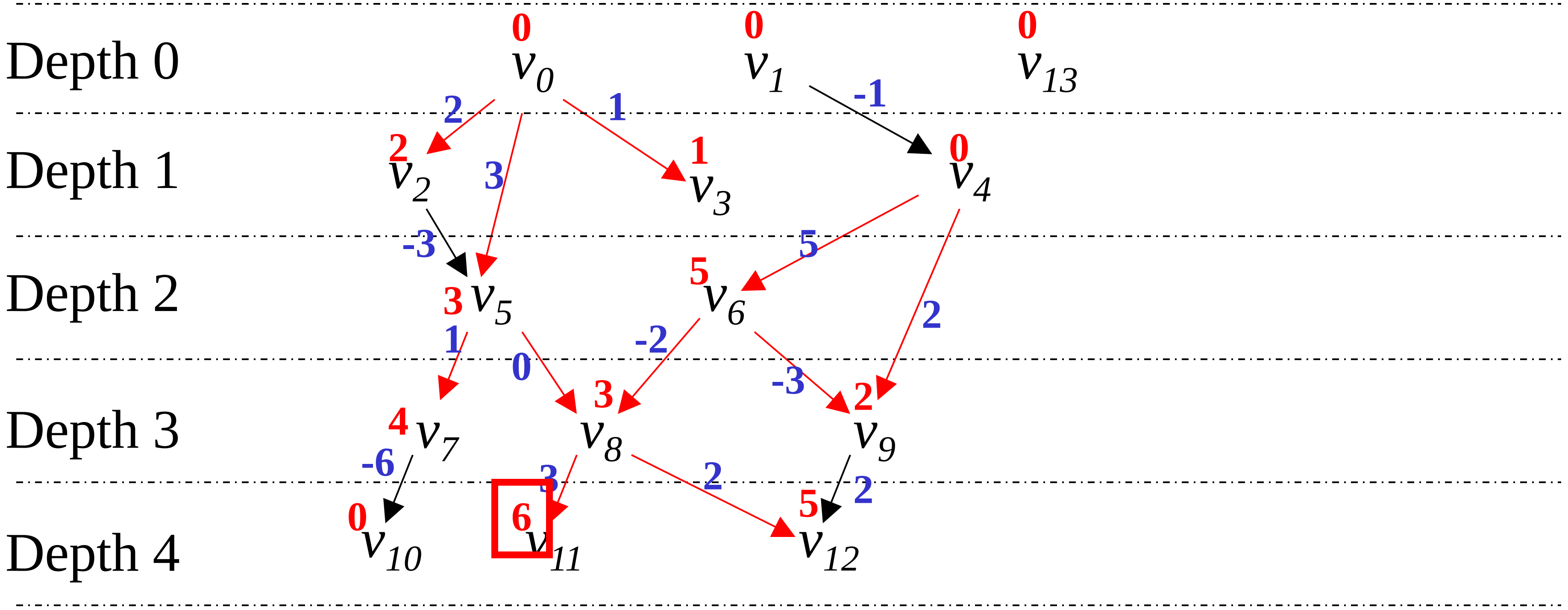
## *w*(*v*) – depth 2 nodes



# $w(v)$ – depth 3 nodes



# $w(v)$ – depth 4 nodes



# Highest Weight Paths on WDAGs (cont'd)

- To reconstruct best path, need "traceback" pointer to immediate predecessor of *v* in best path:

$$T(v) = \begin{cases} v & w(v) = 0 \\ \arg\max\limits_{u \in \text{parents}(v)} (w(u) + w((u,v))) & w(v) \neq 0 \end{cases}$$

  – in preceding graph, *T*(*v*) is the *parent* on *red edge* coming into *v*
    - if more than one such edge, pick one at random;
    - if no such edge, *T*(*v*) = *v*

- Sometimes useful to record *beginning* of best path:

$$B(v) = \begin{cases} v & w(v) = 0 \\ B(T(v)) & w(v) \neq 0 \end{cases}$$

# Highest Weight Paths on WDAGs (cont'd)

- Then highest weight of any path in graph is

$$\max_{v \in V} (w(v))$$

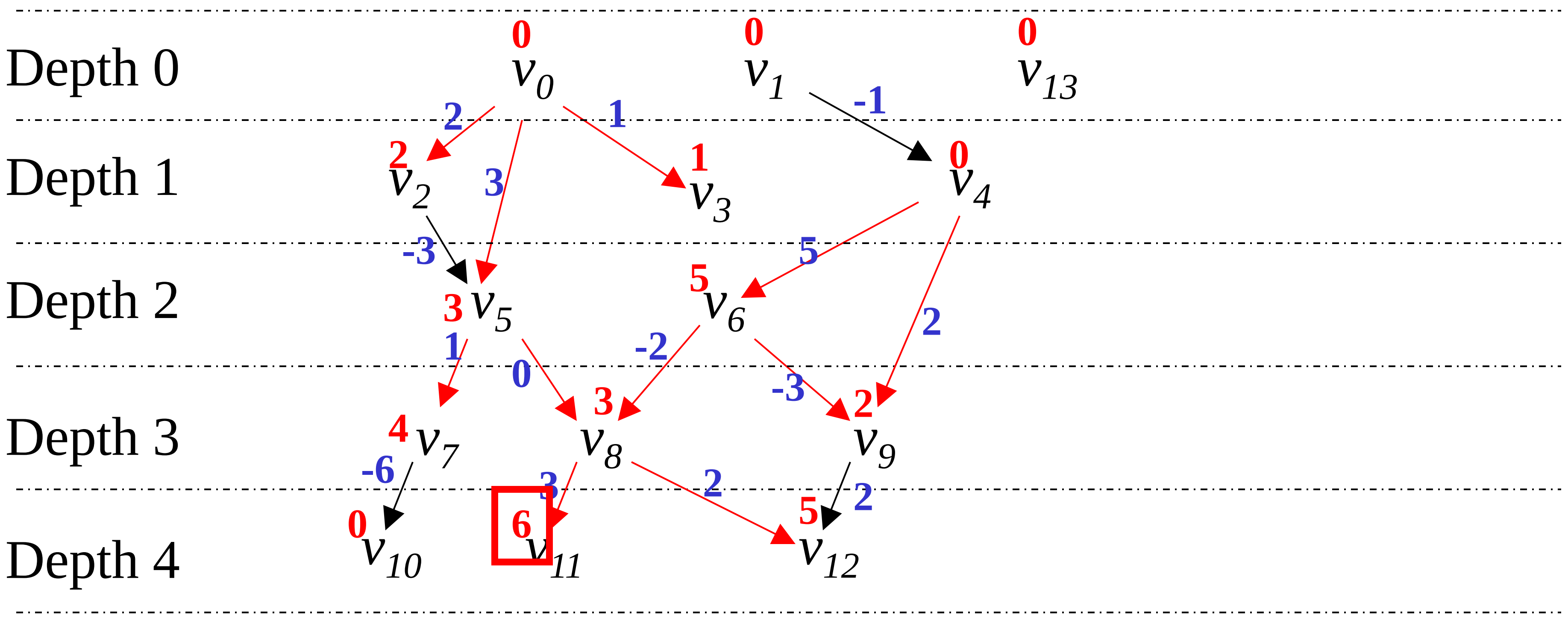
  - updated as each node is visited
    - indicated by $\boxed{\phantom{v}}$ in preceding graph –

  and so doesn't require additional pass through vertices

- if $u = \mathrm{argmax}_{v\,\in V}\,(w(v))$, can reconstruct highest weight path by tracing back from $u$, using $T$:
  - path ends at $u$;
  - immediate predecessor of $u$ is $T(u)$;
  - predecessor of $T(u)$ is $T(T(u))$; etc.
  - stop when $T(v) = v$.

- In preceding example, highest weight is *6* and $u = v_{11}$

# Dynamic programming on WDAGs



# Complexity of Dynamic Programming

- Time to find a best path is $O(|E|+|V|)$:
  - in initial pass, visit each node, and each edge into that node: $O(|E|+|V|)$
  - in traceback, visit subset of nodes, and unique edge from each node: $O(|V|)$

(Complexity to find *all* highest weight paths can be higher)

For very large graphs, even $O(|E|+|V|)$ may be unacceptable!

# Complexity Analysis (cont'd)

- Space requirements:
  - If only want *weight* of best path, and beginning and end, then
    - don't need $T(v)$, and
    - only need retain $w(v)$ and $B(v)$ until have processed all children of $v$ (or when best path found so far ends at $v$).

    Space depends on graph structure, but usually $\ll O(|V|)$.
  - If want path itself, must store $T(v)\ \forall\ v$
    - space = $O(|V|)$
    - $\exists$ algorithms (for some graphs) to reduce this, but may take more time.

# Implementing Dynamic Programming in a Computer Program

- Storing entire graph has space complexity = $O(|V|+|E|)$
- If graph has regular structure, can often "create" and process vertices and edges on the fly, without storing in memory
  - cf. edit graph (to be defined later) for aligning sequences

## Same dynamic programming approach can be used to find:

1. Highest product weight path (if weights are $\geq 0$)
2. Highest weight path that
   - *starts* in particular subset $V'$ of vertices,
     - don't consider paths that start outside $V'$ :

       i.e. when computing $w(v)$, don't consider trivial path unless $v \in V'$
   - and/or *ends* in particular subset $V''$
     - only scan for the maximum $w(v)$ over $V''$
3. Sum of product weights of all paths ending at particular vertex
   - *sum* over all edges coming into $v$, instead of *maximizing*
   - this useful for probability calculations

- Will use the above variants later!