Figure 20.4

Linear probing hash table after each insertion

```
hash ( 89, 10 ) = 9
hash ( 18, 10 ) = 8
hash ( 49, 10 ) = 9
hash ( 58, 10 ) = 8
hash (  9, 10 ) = 9
```

	After insert 89	*After insert 18*	*After insert 49*	*After insert 58*	*After insert 9*
0			49	49	49
1				58	58
2					9
3					
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

Figure 20.6

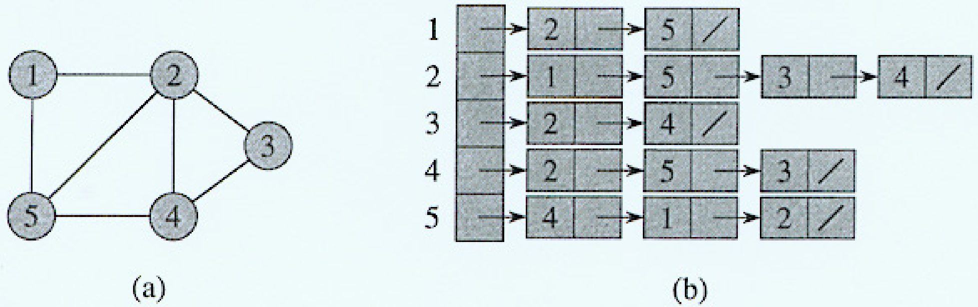
A quadratic probing hash table after each insertion (note that the table size was poorly chosen because it is not a prime number).

```
hash ( 89, 10 ) = 9
hash ( 18, 10 ) = 8
hash ( 49, 10 ) = 9
hash ( 58, 10 ) = 8
hash (  9, 10 ) = 9
```

	After insert 89	*After insert 18*	*After insert 49*	*After insert 58*	*After insert 9*
0			49	49	49
1					
2				58	58
3					9
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

Graphs

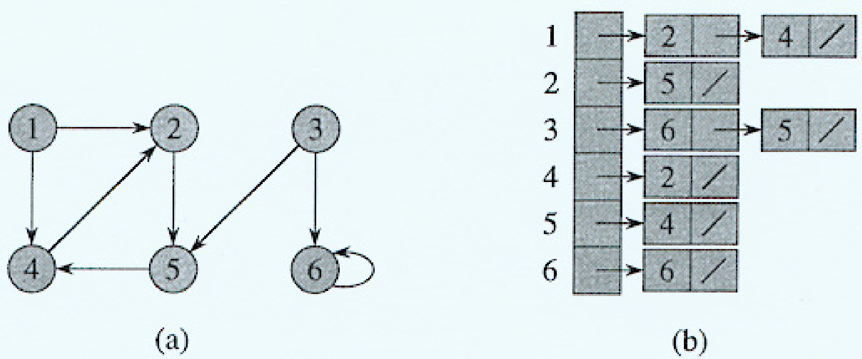
- Graphs model networks of various kinds: roads, highways, oil pipelines, airline routes, dependency relationships, etc.
- Graph G(V,E)
- V Vertices or Nodes
- E Edges or Links: pairs of vertices
- Directed vs. Undirected edges



(c)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G having five vertices and seven edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.



(c)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Graphs

- Graphs can be augmented to store extra info (e.g., city population, oil flow capacity, etc.)
- Weighted vs. Unweighted
- Paths and Cycles

Figure 14.1
A directed graph.

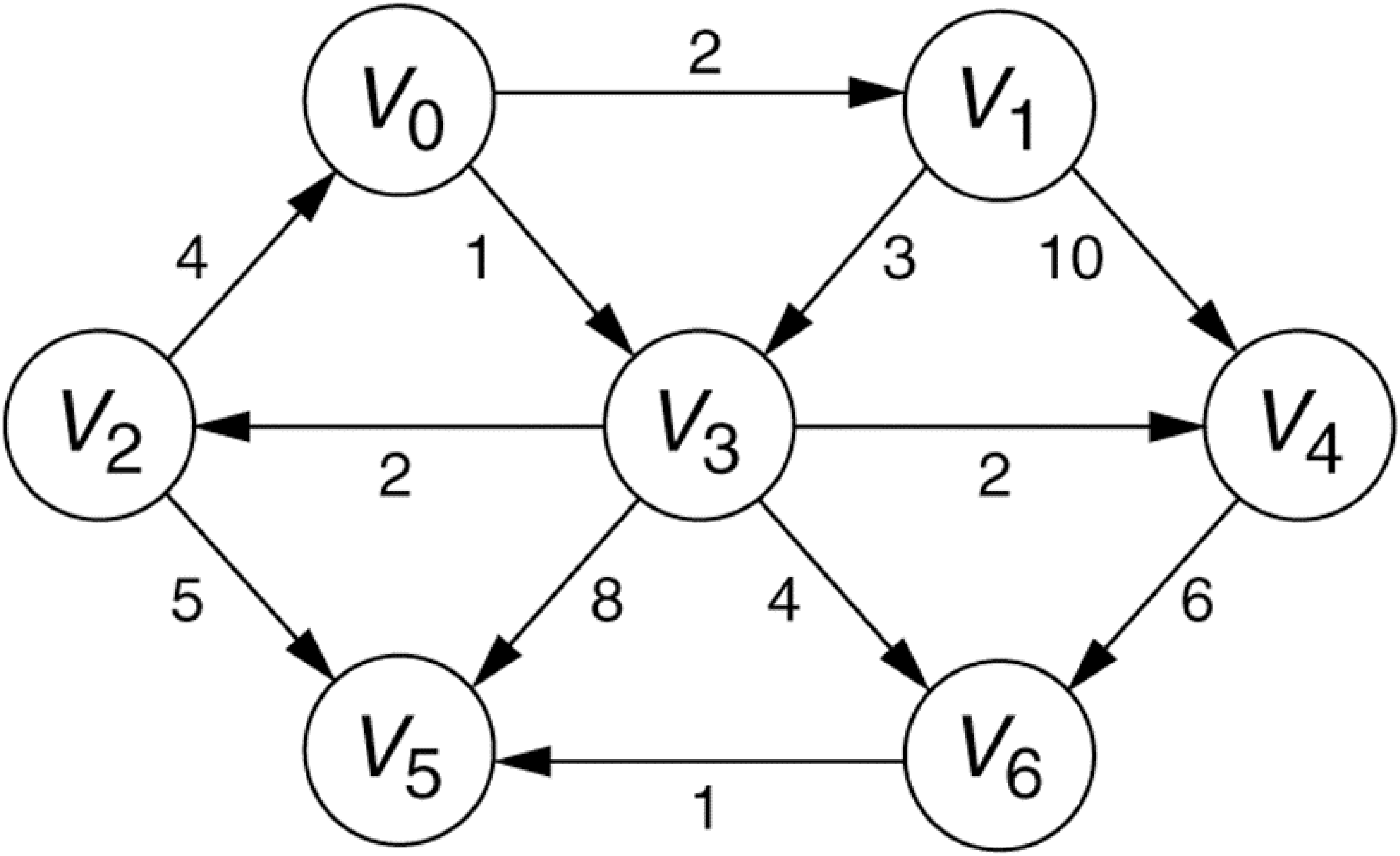
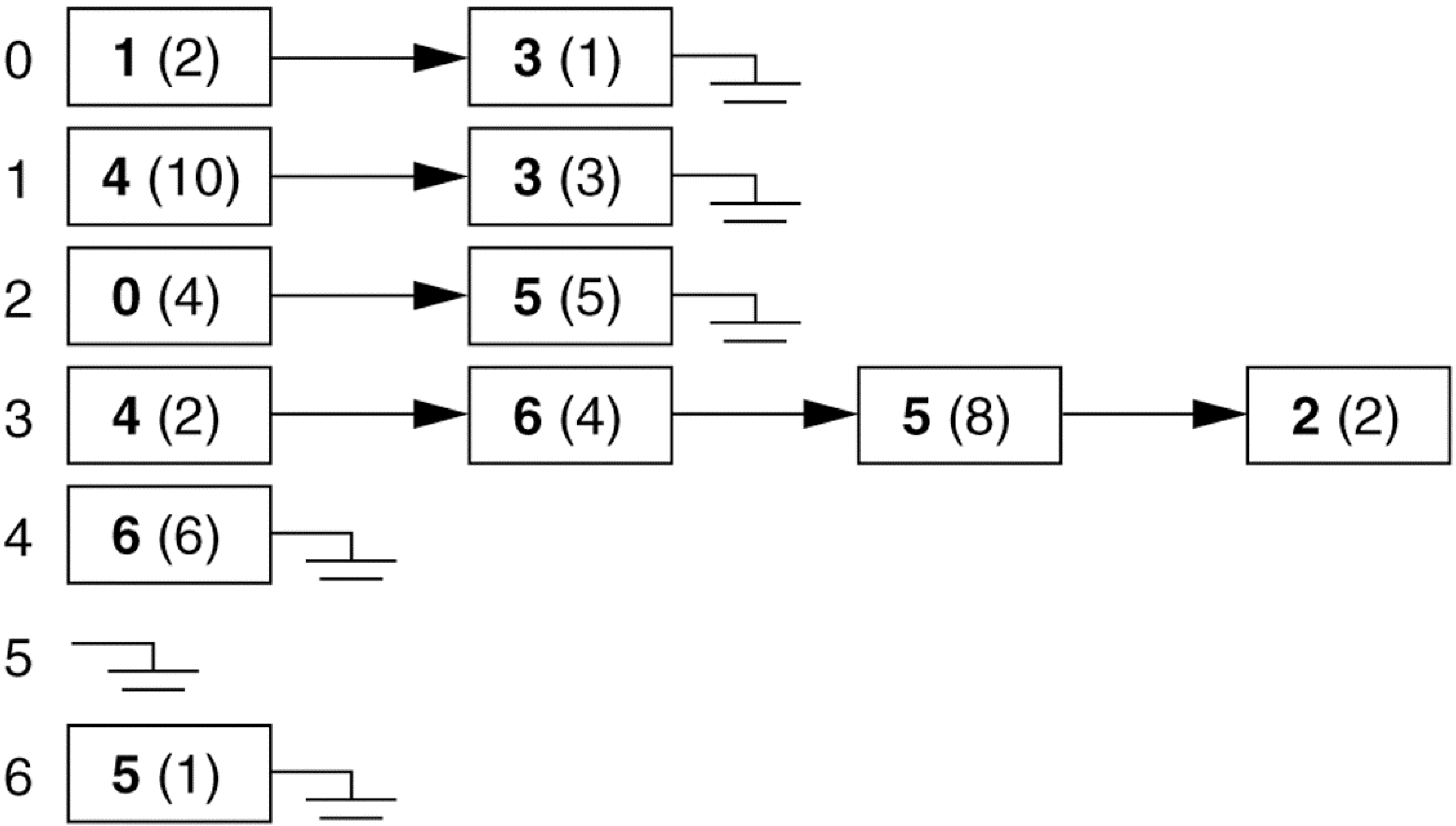


Figure 14.2

Adjacency list representation of the graph shown in Figure 14.1; the nodes in list *i* represent vertices adjacent to *i* and the cost of the connecting edge.



Adjacency Lists

- Constructing adjacency lists
  - Input: list of edges
  - Output: adjacency list for all vertices
  - Time: O(L), where L is length of list of edges.
- Check if edge exists
  - Input: edge (u,v)
  - Output: does the edge exist in the graph G?
  - Time: $O(d_u)$, where d_u is the number of entries in u's adjacency list. In the worst case it is O(N), where N is the number of vertices
- Need a MAP data structure to map vertex name or ID to (internal) vertex number.

Shortest Paths

- Suppose we are interested in the shortest paths (and their lengths) from vertex “Miami“ to all other vertices in the graph.
- We need to augment the data structure to store this information.

Figure 14.4

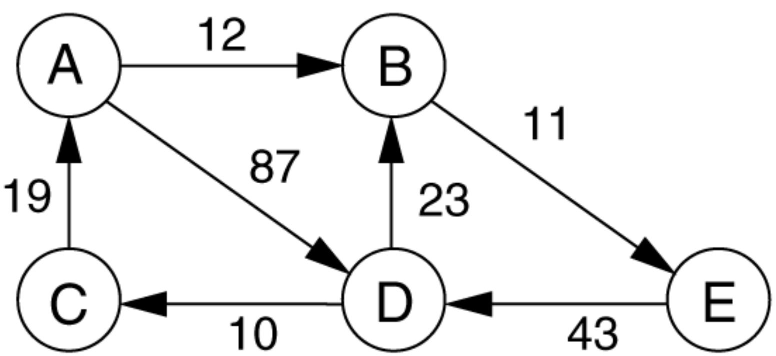
An abstract scenario of the data structures used in a shortest-path calculation, with an input graph taken from a file. The shortest weighted path from A to C is A to B to E to D to C (cost is 76).

D	C	10
A	B	12
D	B	23
A	D	87
E	D	43
B	E	11
C	A	19

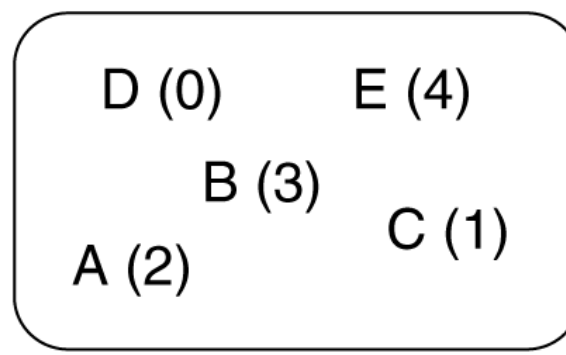
Input

	dist	prev	name	adj
0	66	4	D	→ **3** (23),**1** (10)
1	76	0	C	→ **2** (19)
2	0	-1	A	→ **0** (87),**3** (12)
3	12	2	B	→ **4** (11)
4	23	3	E	→ **0** (43)

Graph table



Visual representation of graph



Dictionary

Figure 14.5

Data structures used in a shortest-path calculation, with an input graph taken from a file; the shortest weighted path from A to C is A to B to E to D to C (cost is 76).

Legend: Dark-bordered boxes are Vertex objects. The unshaded portion in each box contains the name and adjacency list and does not change when shortest-path computation is performed. Each adjacency list entry contains an Edge that stores a reference to another Vertex object and the edge cost. Shaded portion is dist and prev, filled in after shortest path computation runs.
Dark arrows emanate from vertexMap.
Light arrows are adjacency list entries.
Dashed arrows are the prev data member that results from a shortest-path computation.

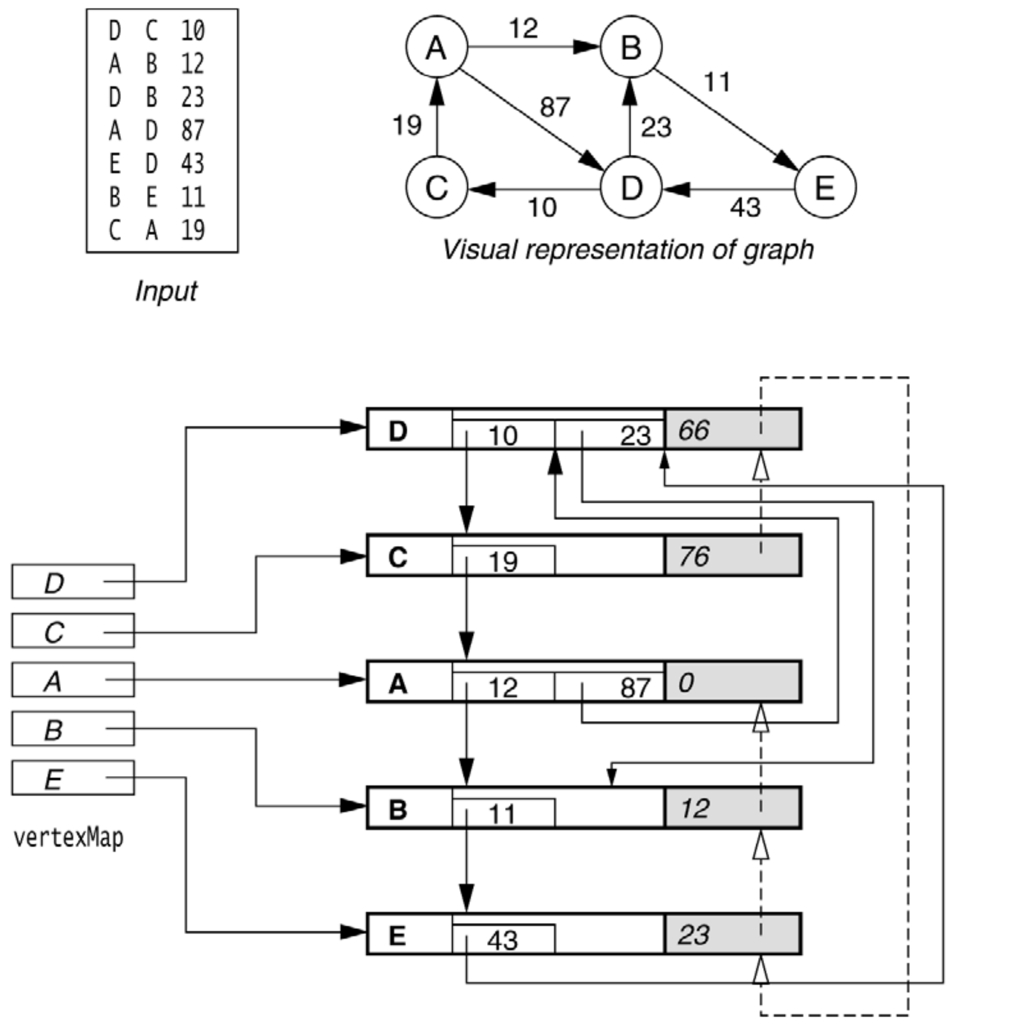


Figure 14.16

The graph, after the starting vertex has been marked as reachable in zero edges

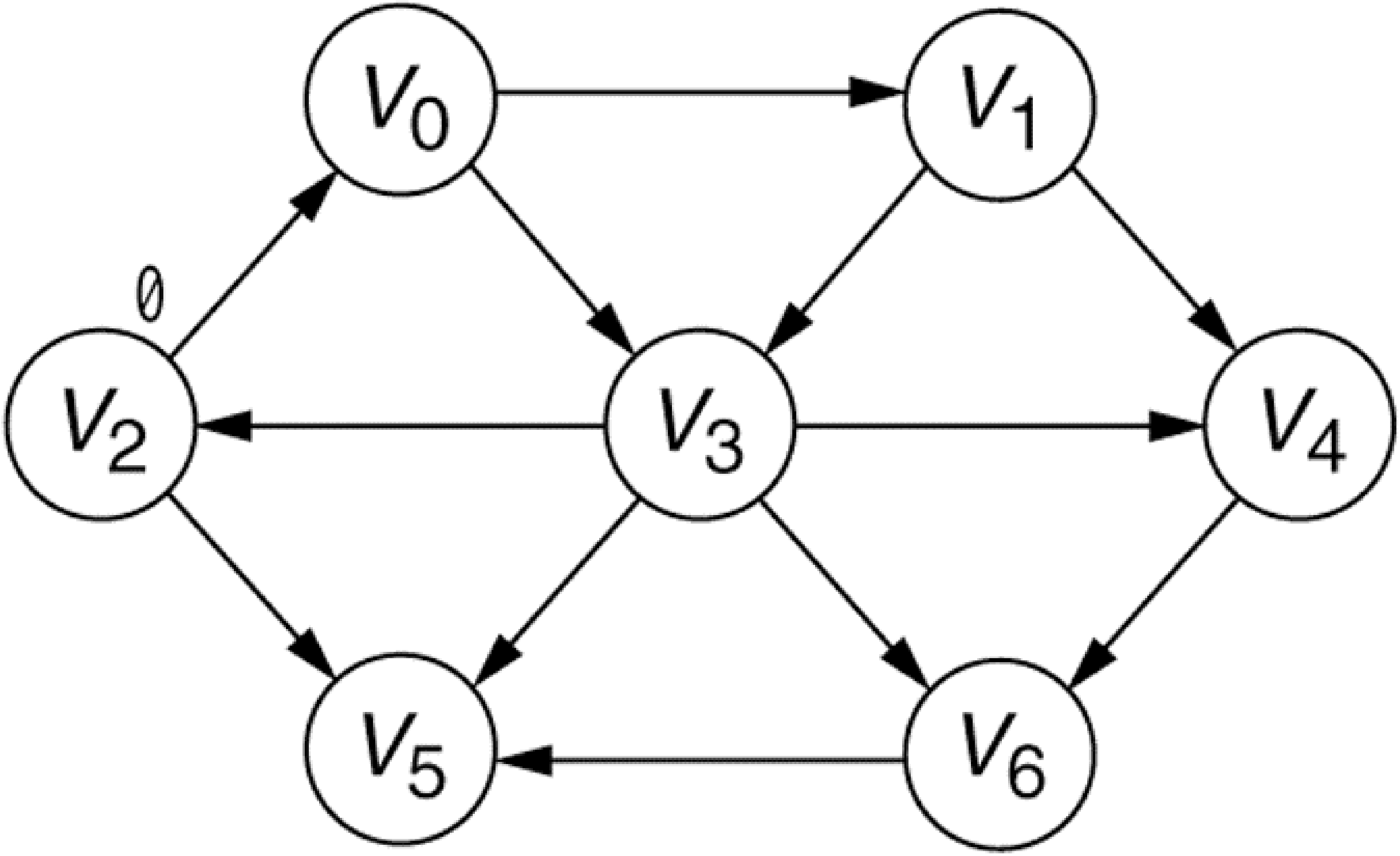


Figure 14.17

The graph, after all the vertices whose path length from the starting vertex is 1 have been found

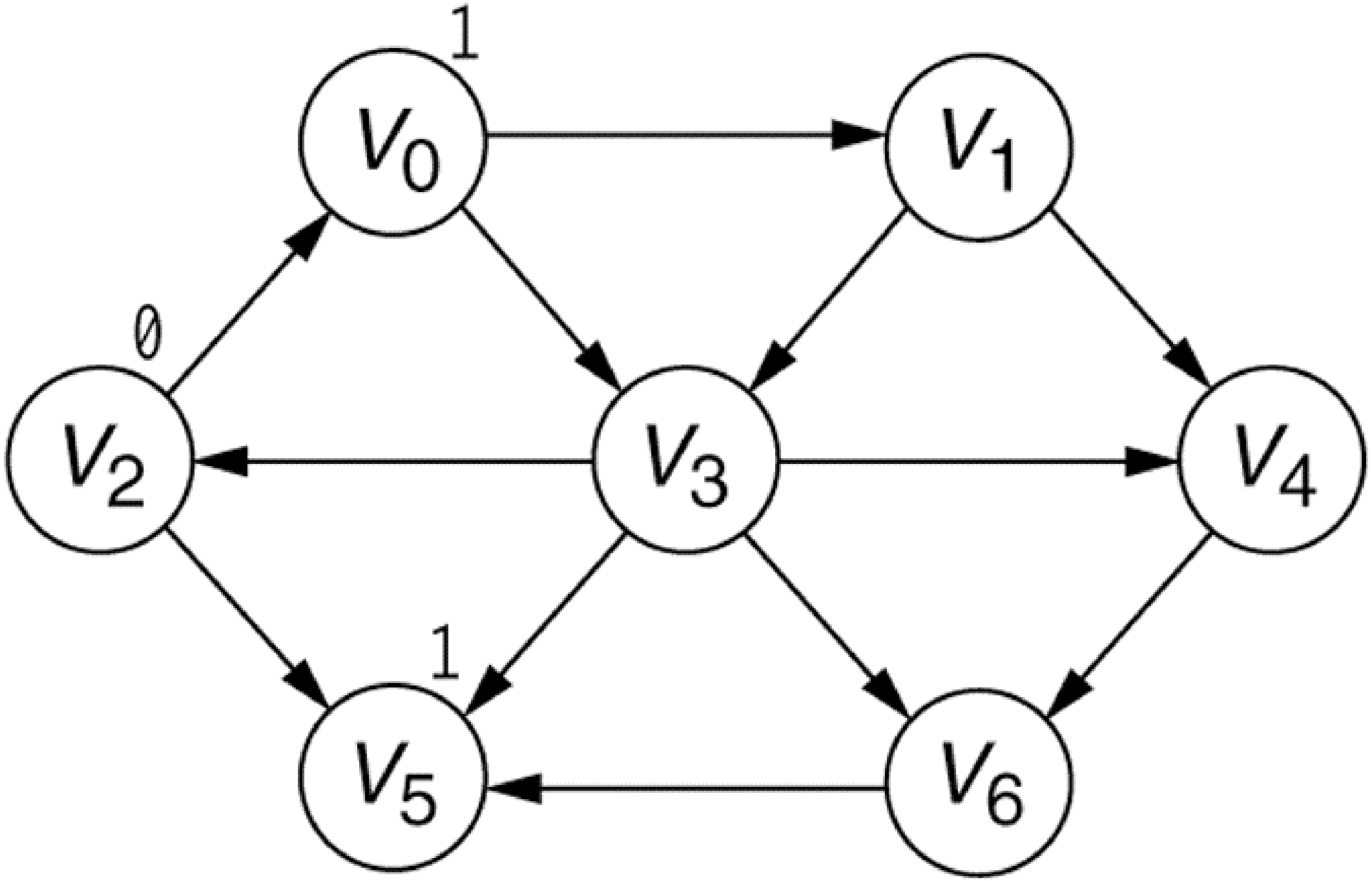


Figure 14.18

The graph, after all the vertices whose shortest path from the starting vertex is 2 have been found

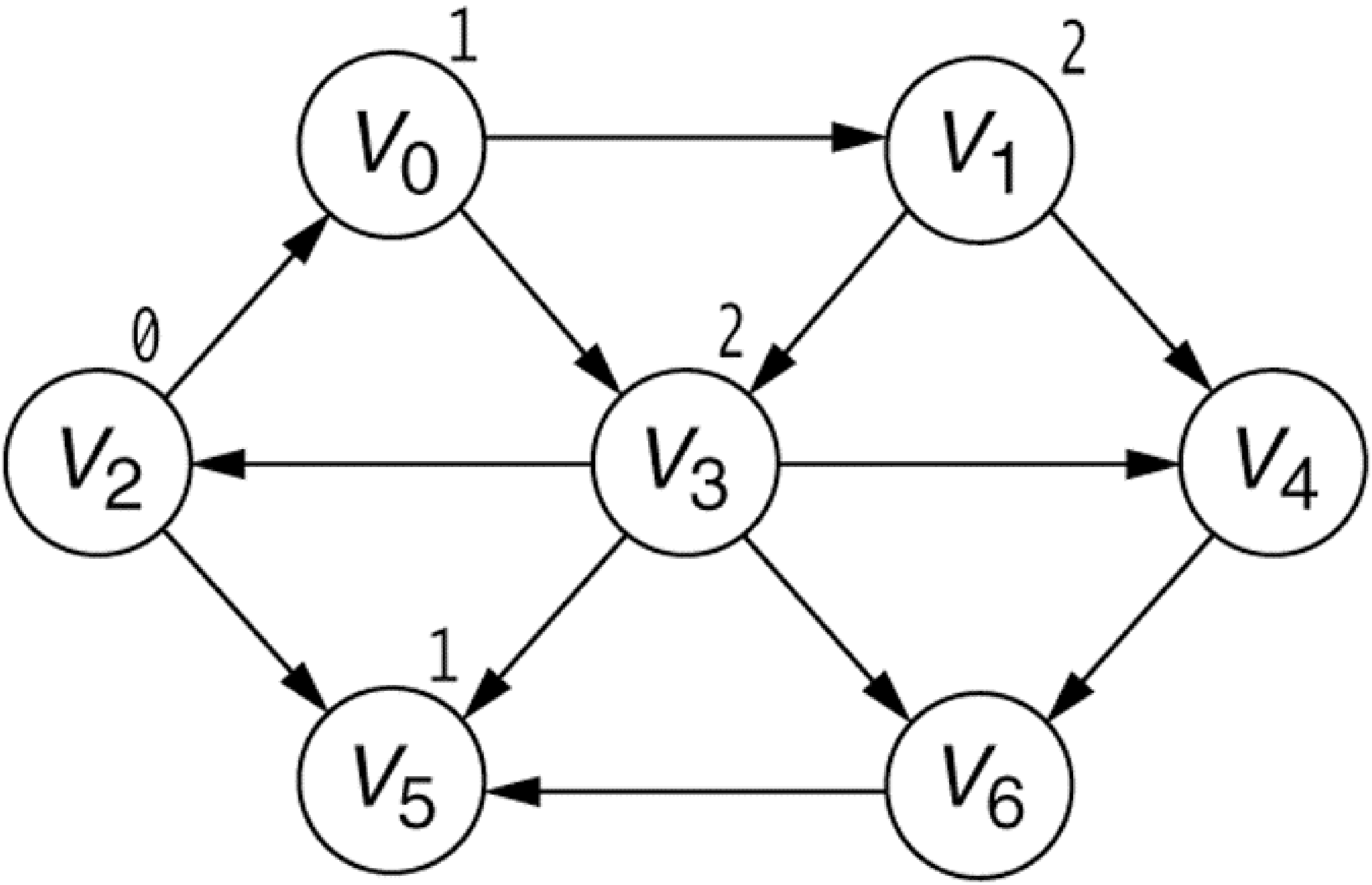


Figure 14.19

The final shortest paths

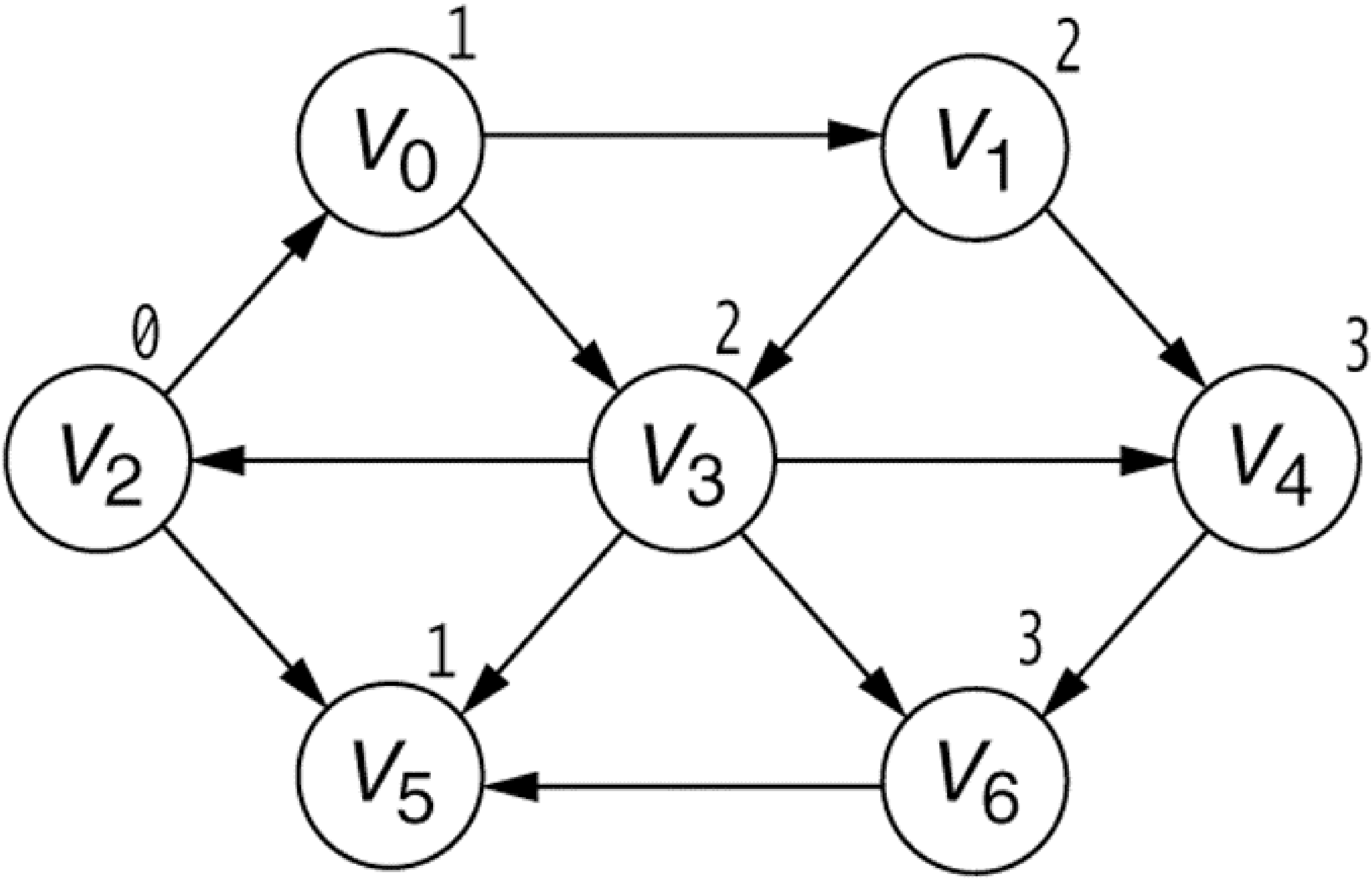


Figure 14.20

If *w* is adjacent to *v* and there is a path to *v*, there also is a path to *w*

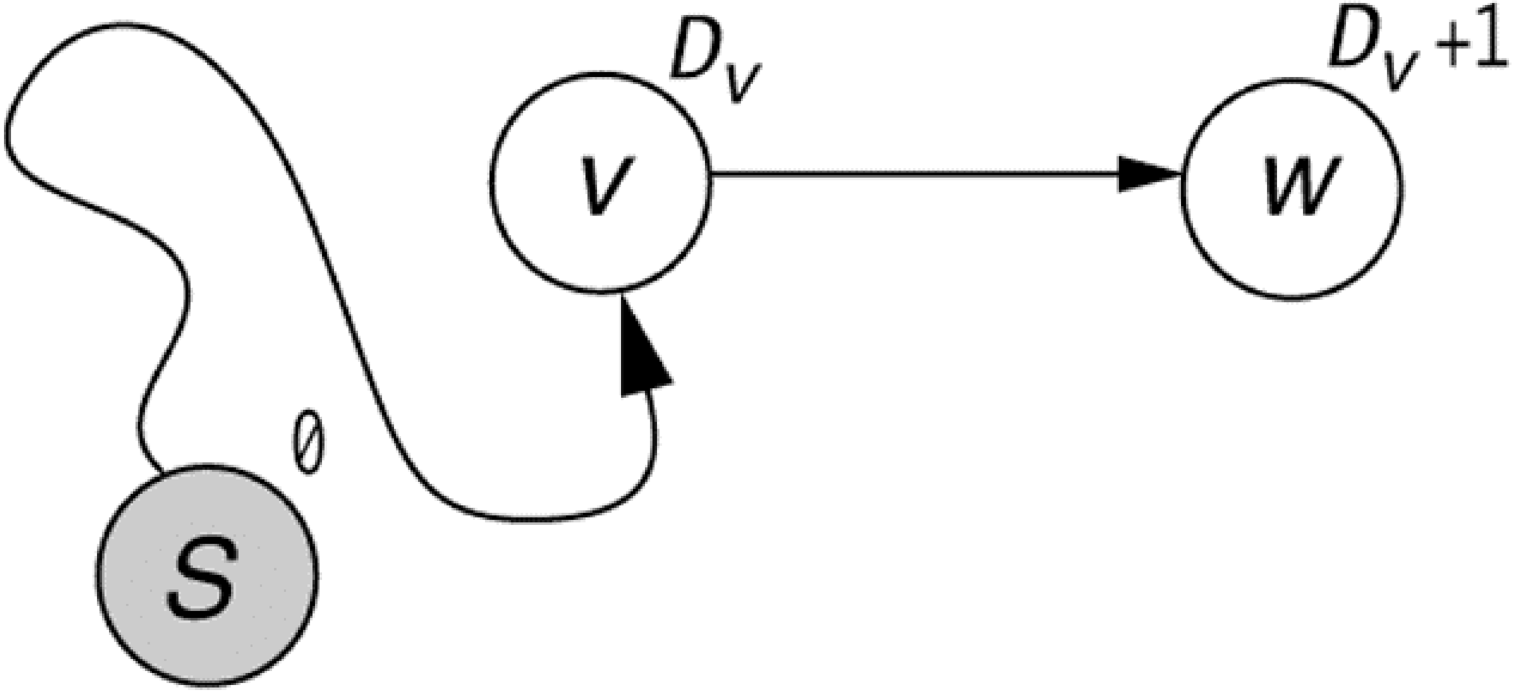


Figure 14.21A

Searching the graph in the unweighted shortest-path computation. The darkest-shaded vertices have already been completely processed, the lightest-shaded vertices have not yet been used as *v*, and the medium-shaded vertex is the current vertex, *v*. The stages proceed left to right, top to bottom, as numbered *(continued).*

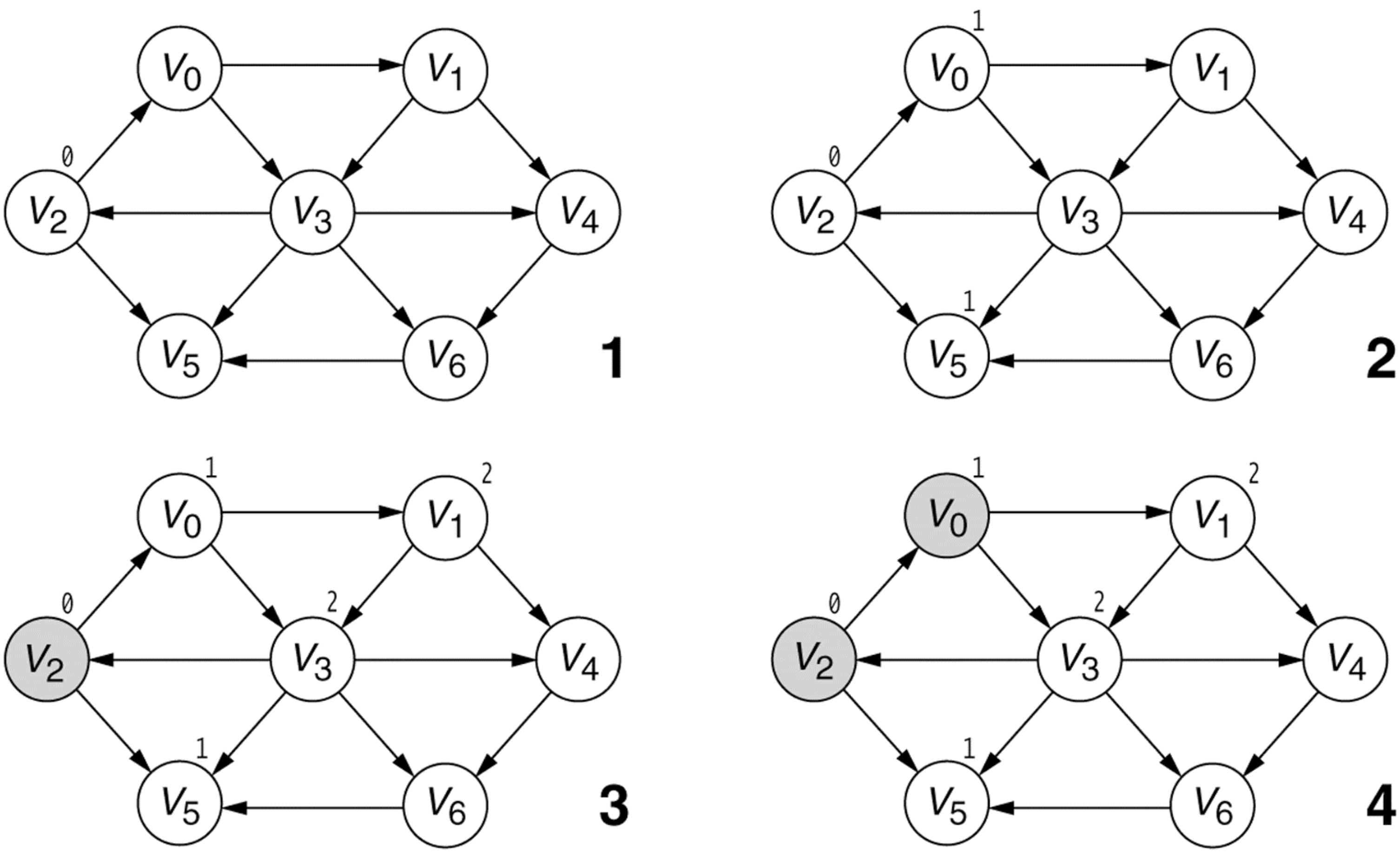


Figure 14.21B

Searching the graph in the unweighted shortest-path computation. The darkest-shaded vertices have already been completely processed, the lightest-shaded vertices have not yet been used as *v*, and the medium-shaded vertex is the current vertex, *v*. The stages proceed left to right, top to bottom, as numbered.

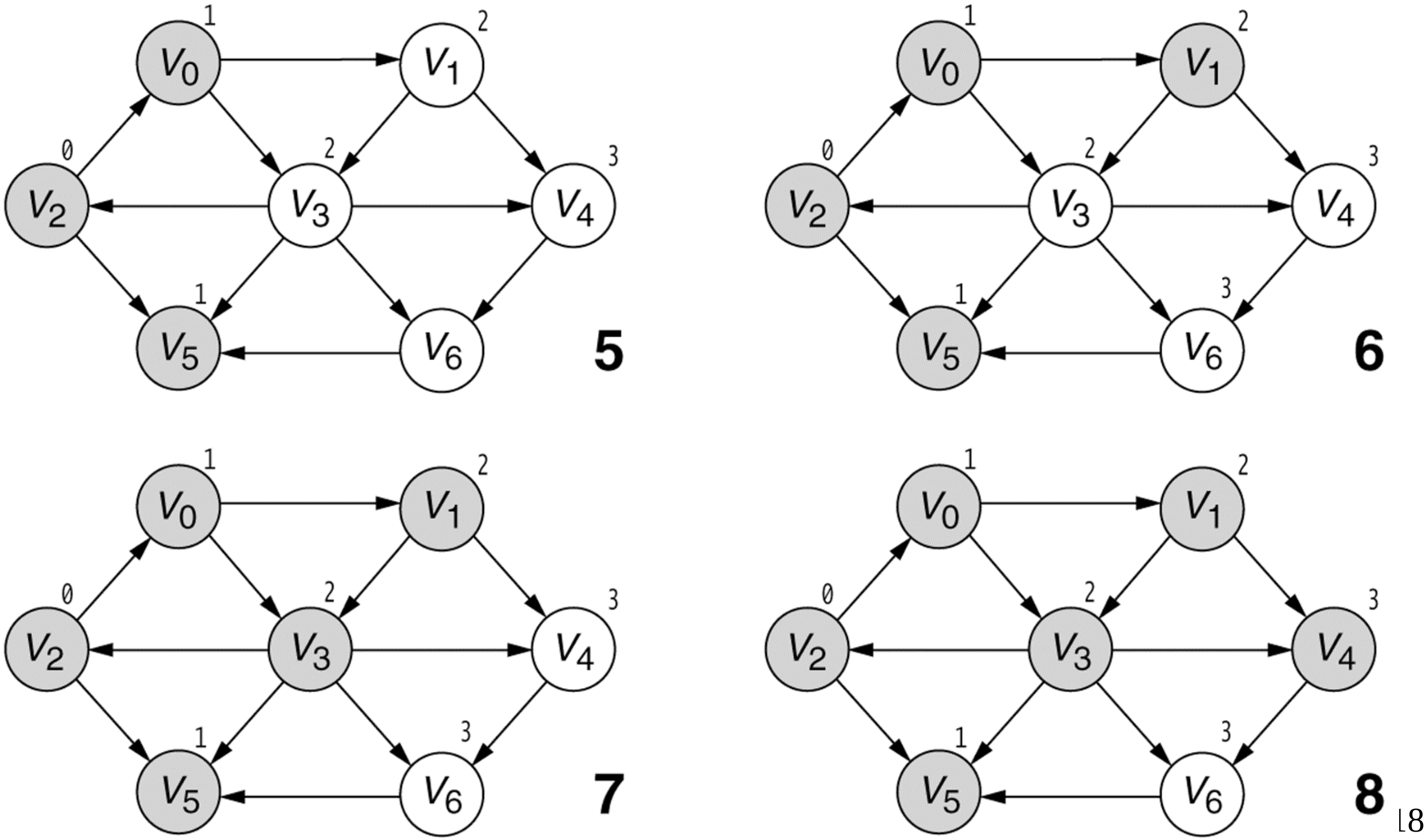


Figure 14.23

The eyeball is at *v* and *w* is adjacent, so D_w should be lowered to 6.

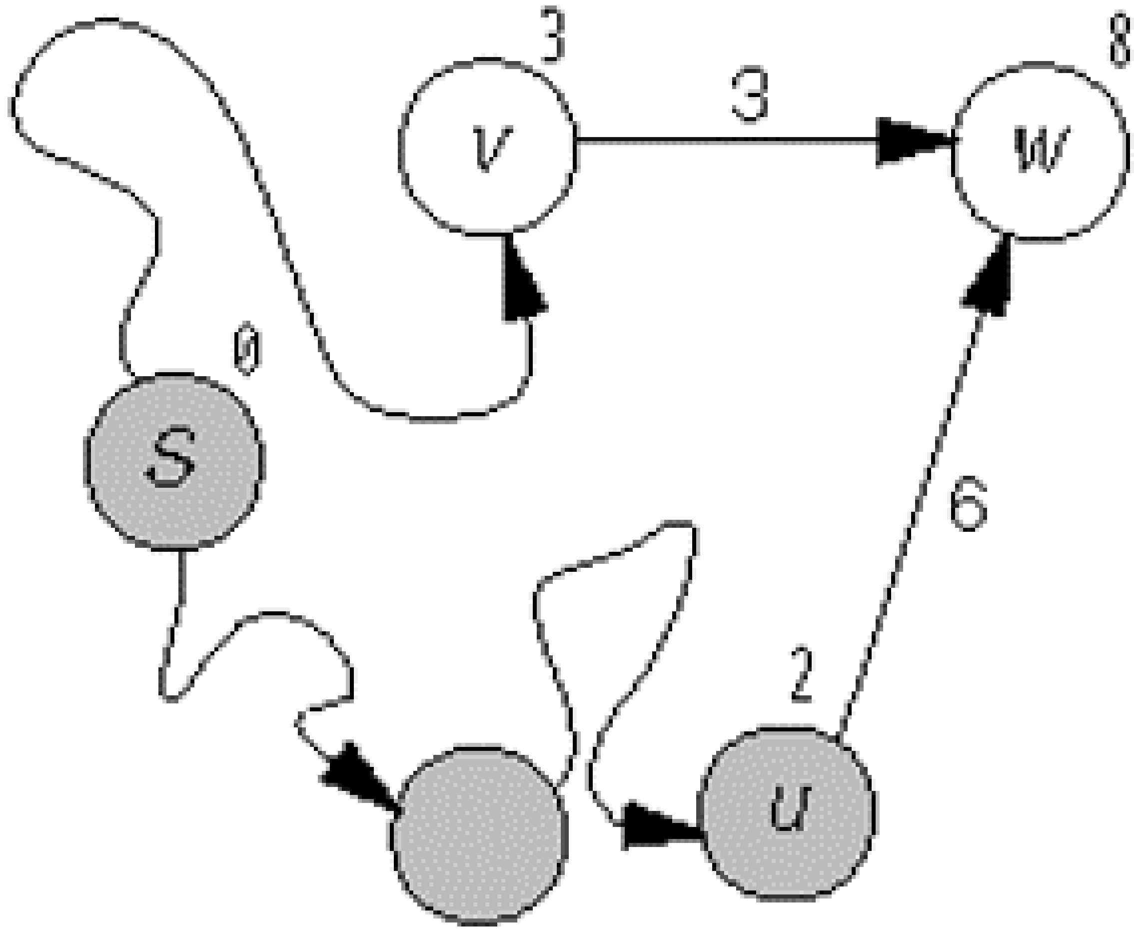


Figure 14.24

If D_v is minimal among all unseen vertices and if all edge costs are nonnegative, D_v represents the shortest path.

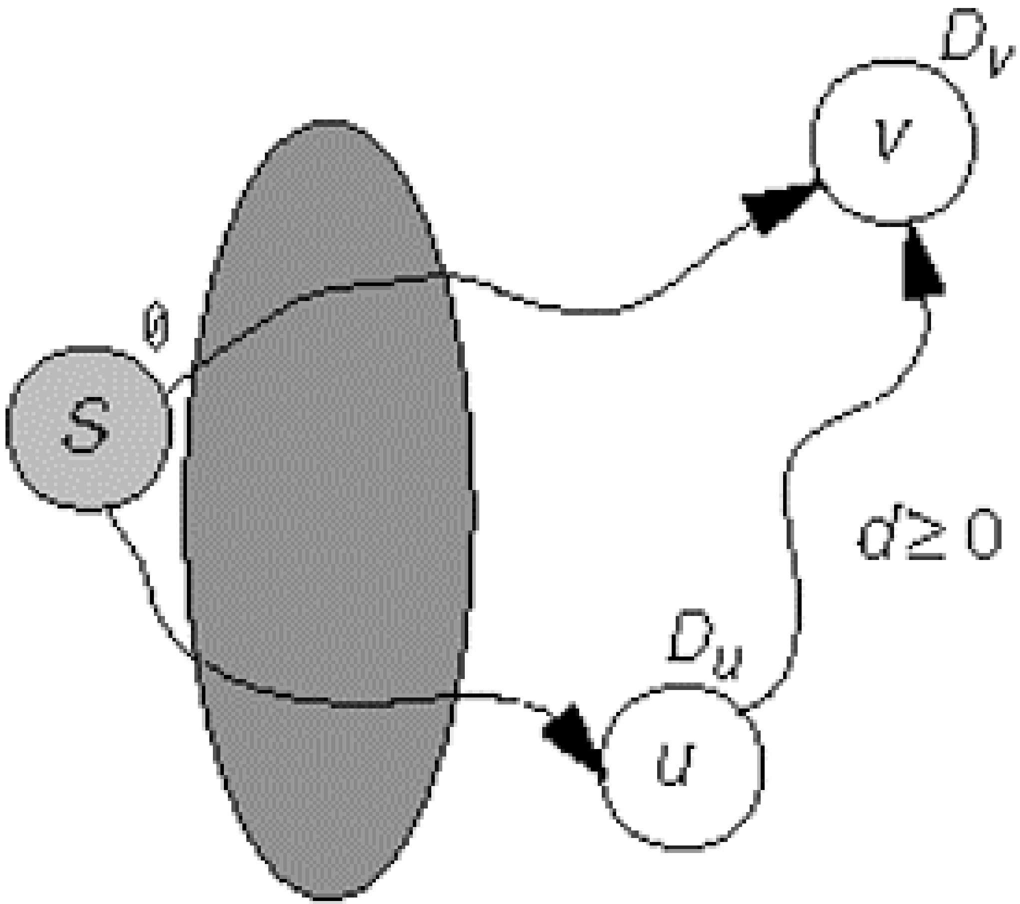


Figure 14.25A

Stages of Dijkstra’s algorithm. The conventions are the same as those in Figure 14.21 (*continued*).

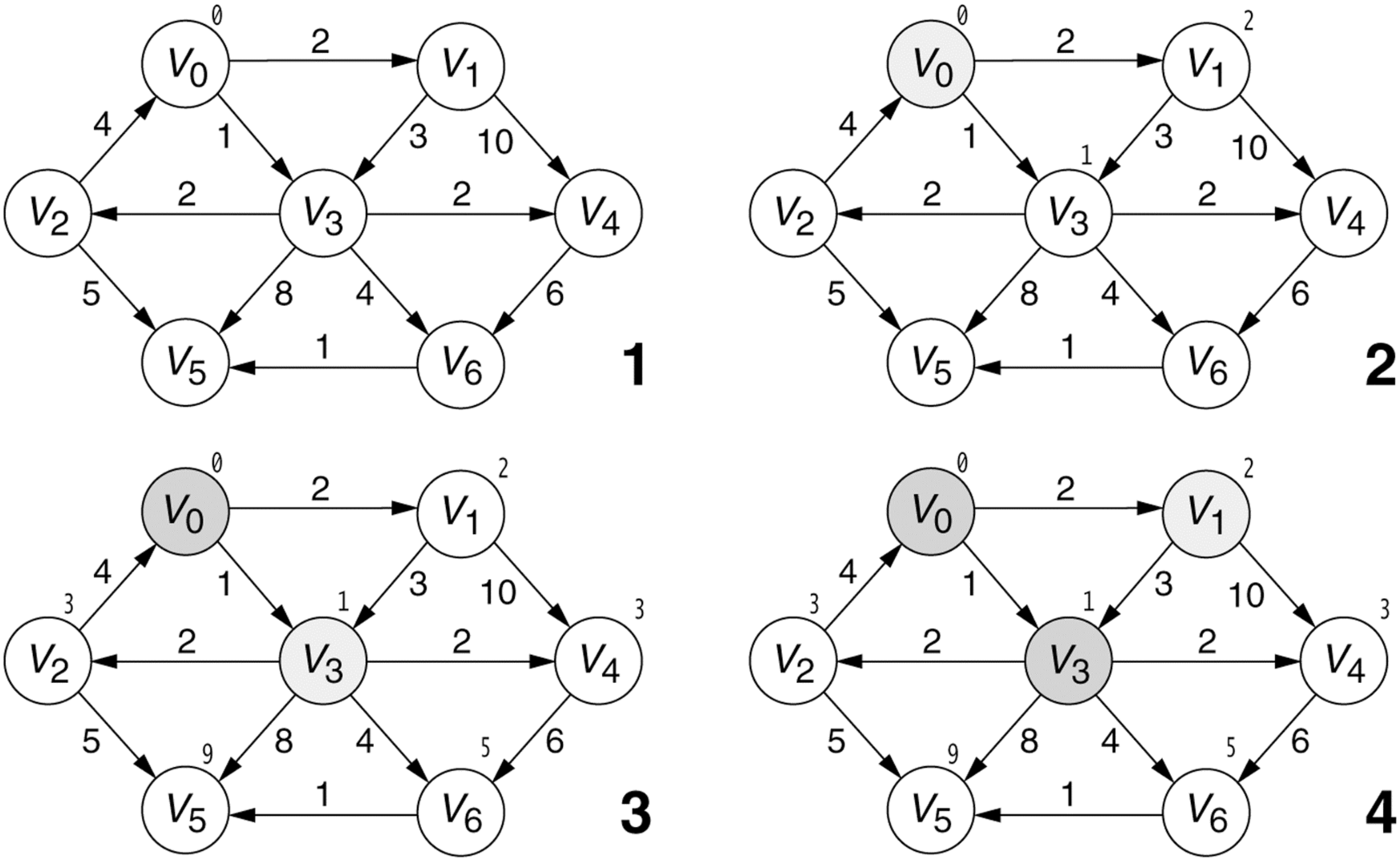


Figure 14.25B

Stages of Dijkstra’s algorithm. The conventions are the same as those in Figure 14.21.

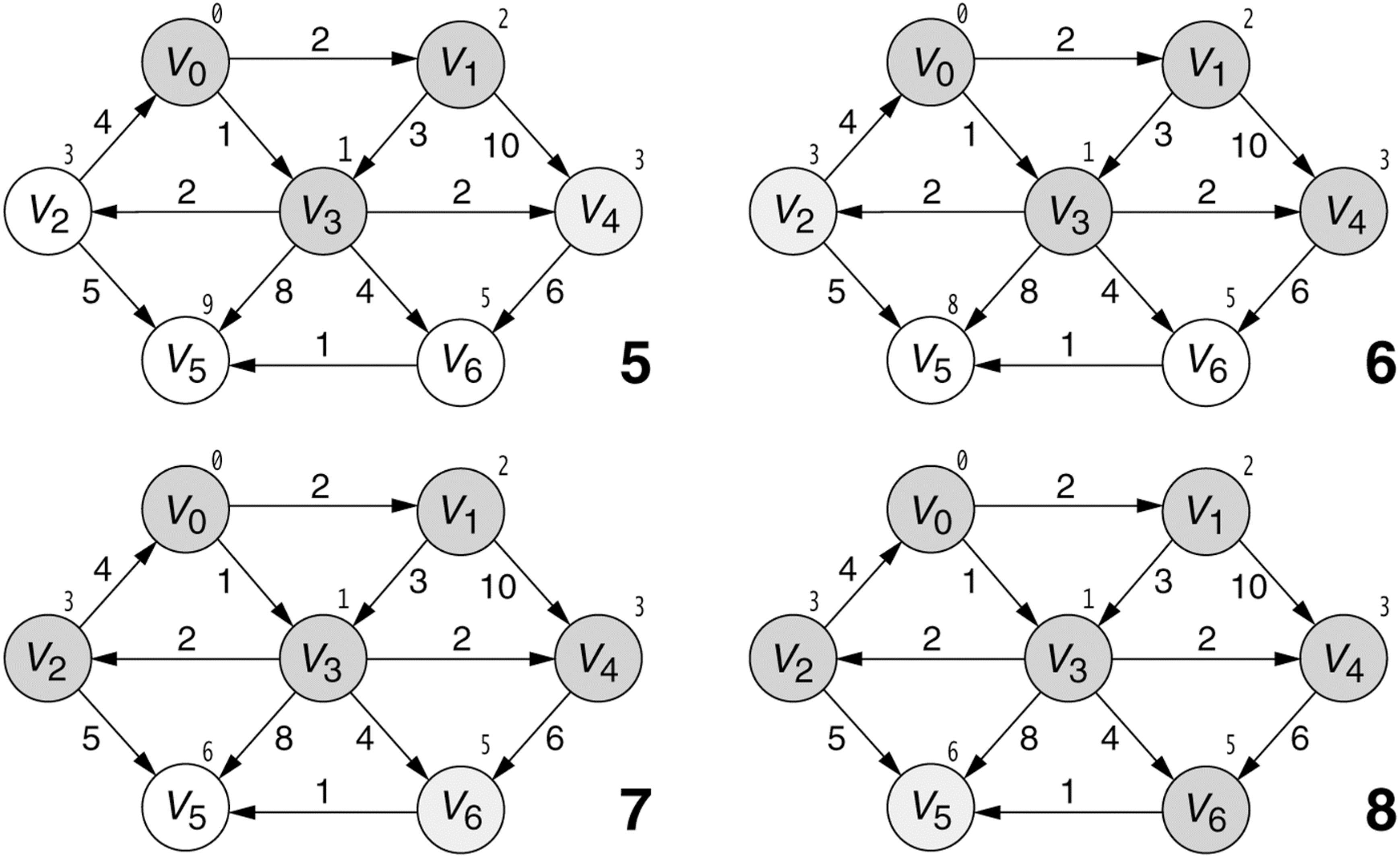


Figure 14.28
A graph with a negative-cost cycle

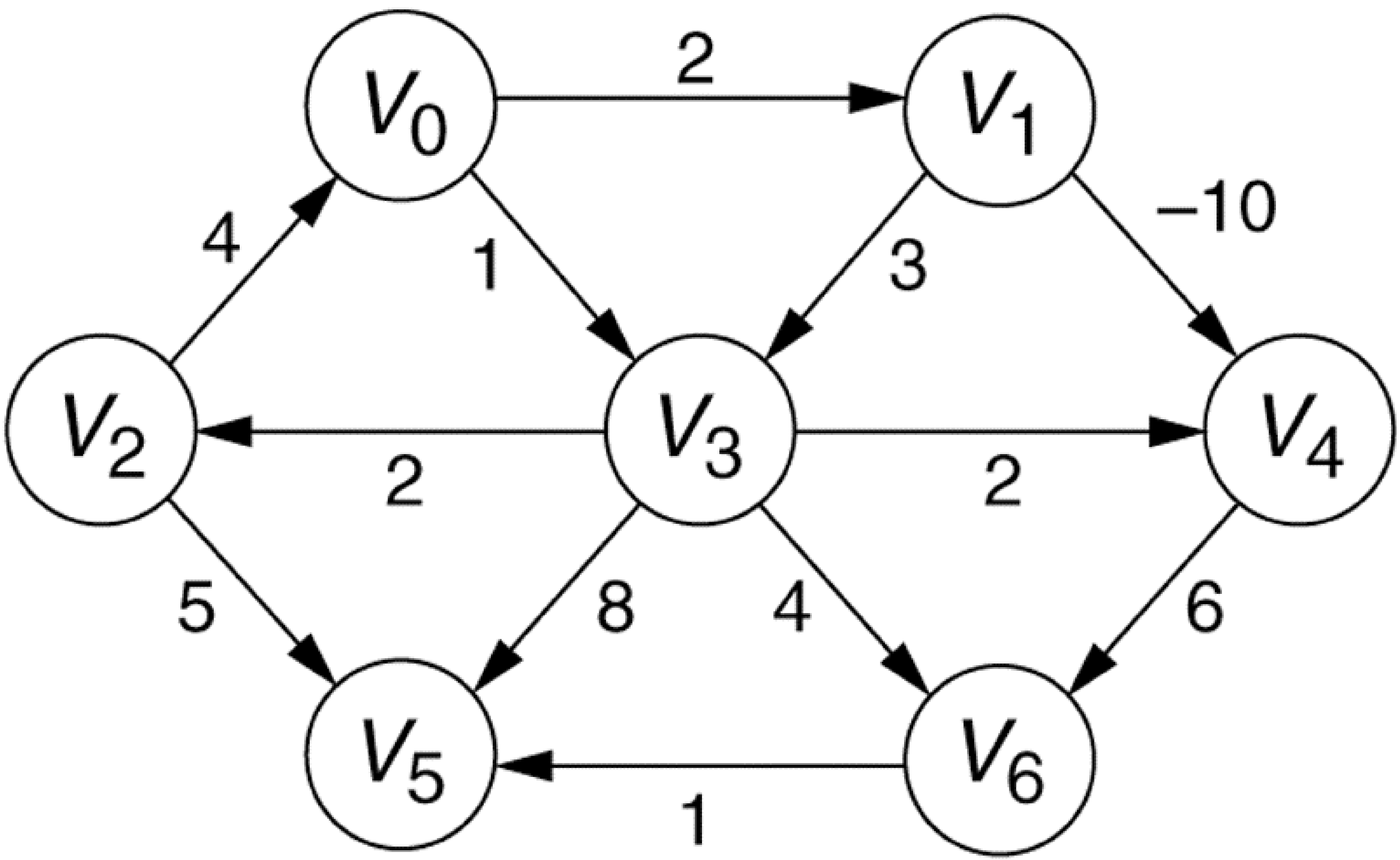


Figure 14.30A

A topological sort. The conventions are the same as those in Figure 14.21 (continued).

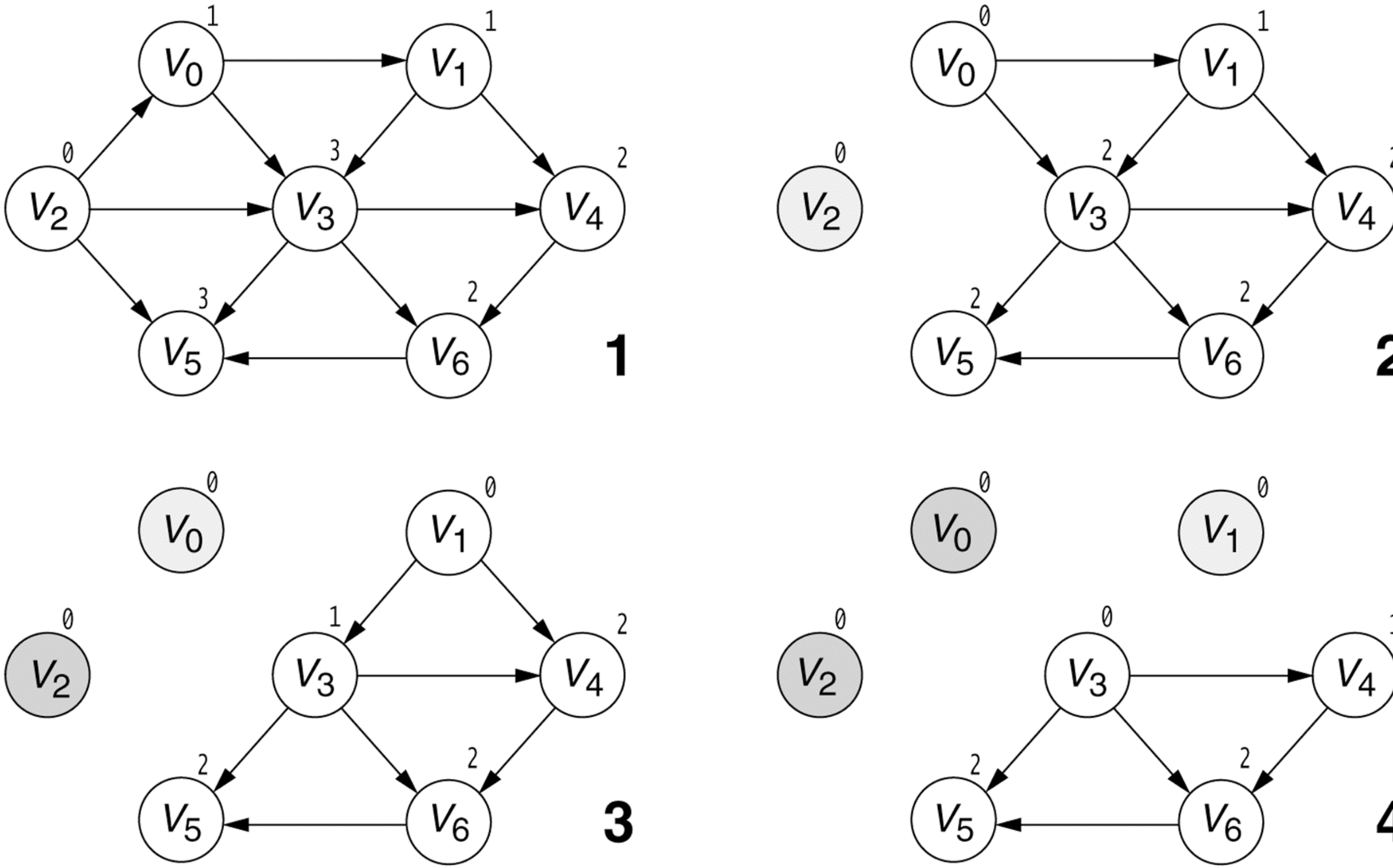


Figure 14.30B

A topological sort. The conventions are the same as those in Figure 14.21.

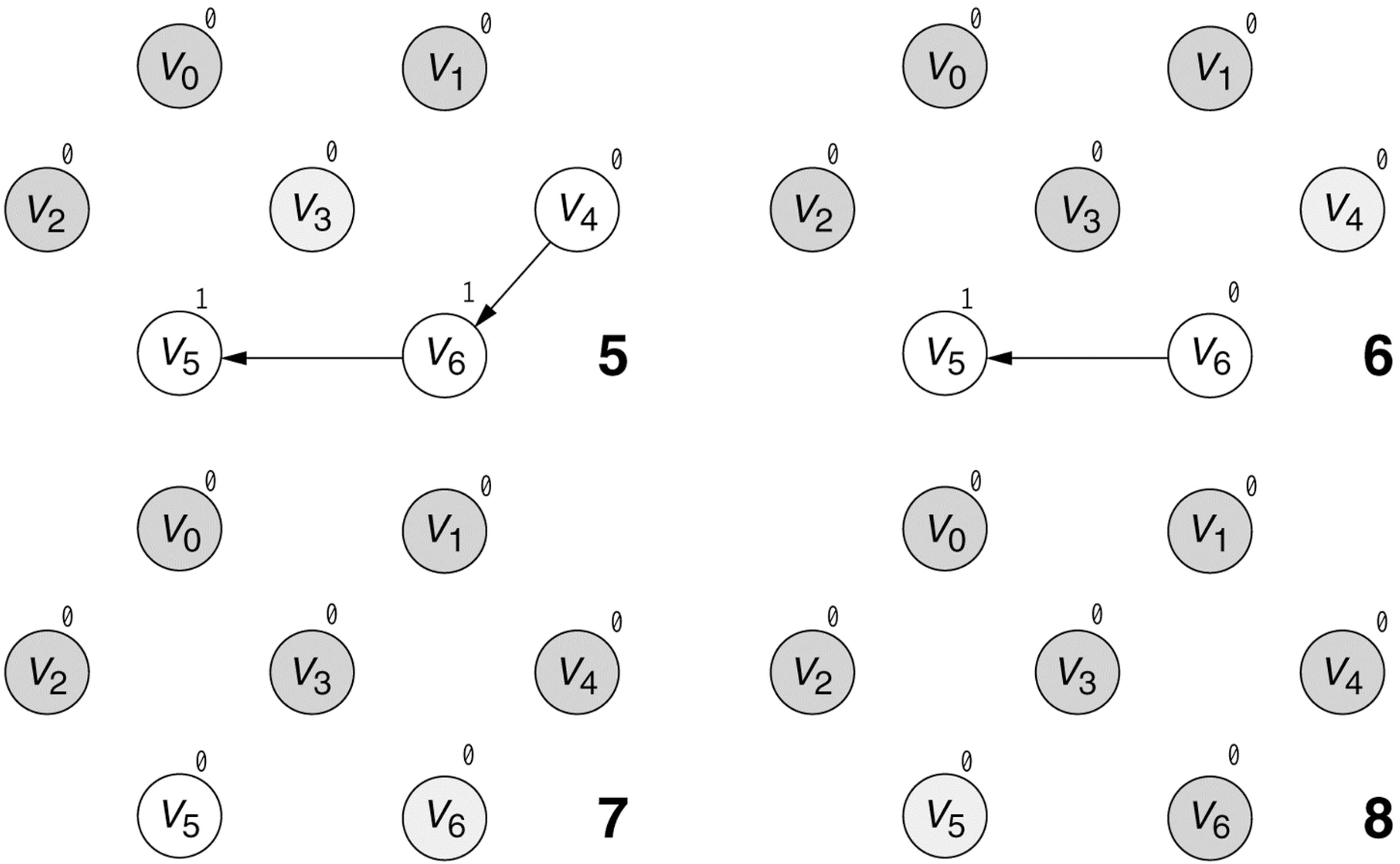


Figure 14.31A

The stages of acyclic graph algorithm. The conventions are the same as those in Figure 14.21 (*continued*).

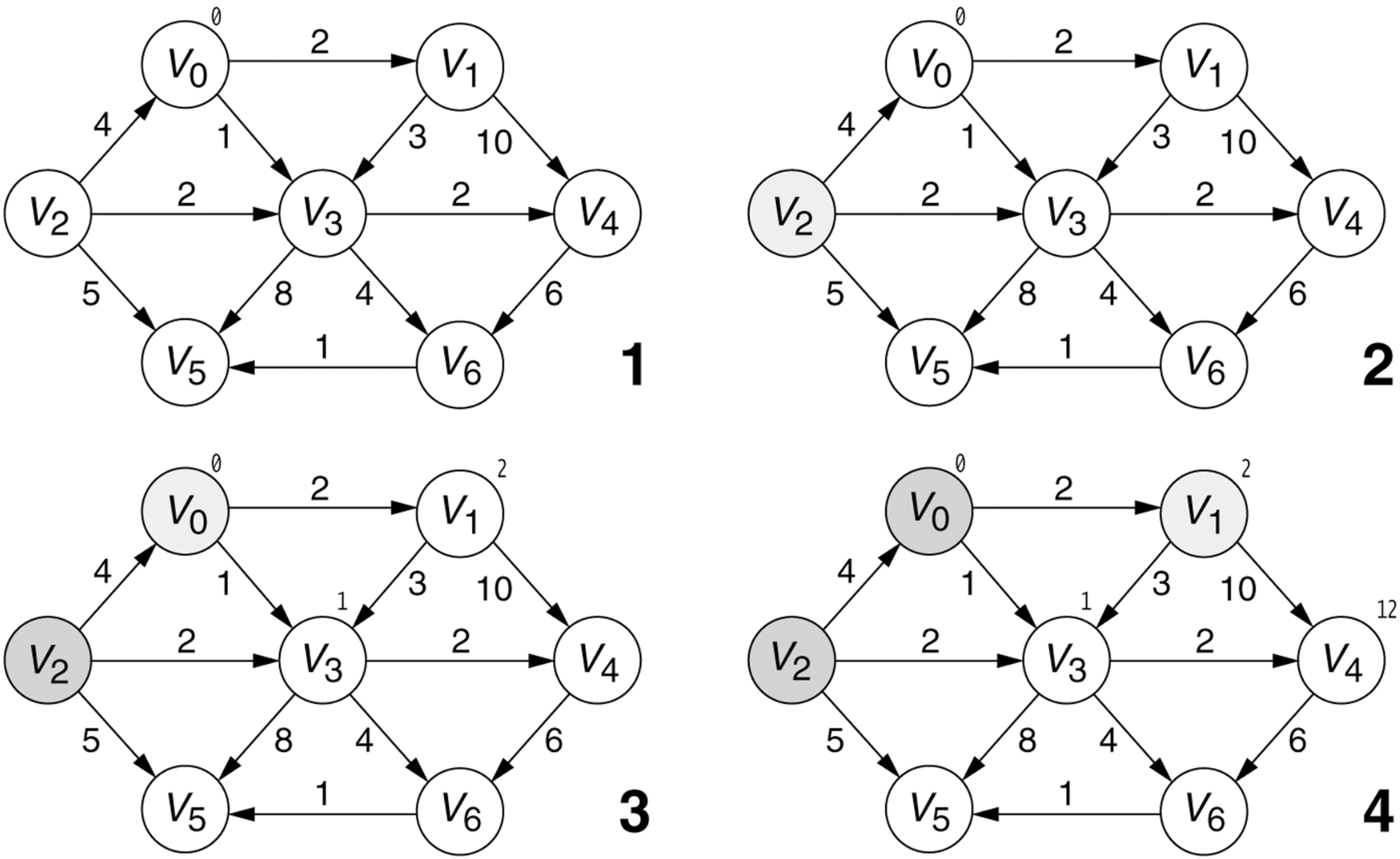


Figure 14.31B

The stages of acyclic graph algorithm. The conventions are the same as those in Figure 14.21.

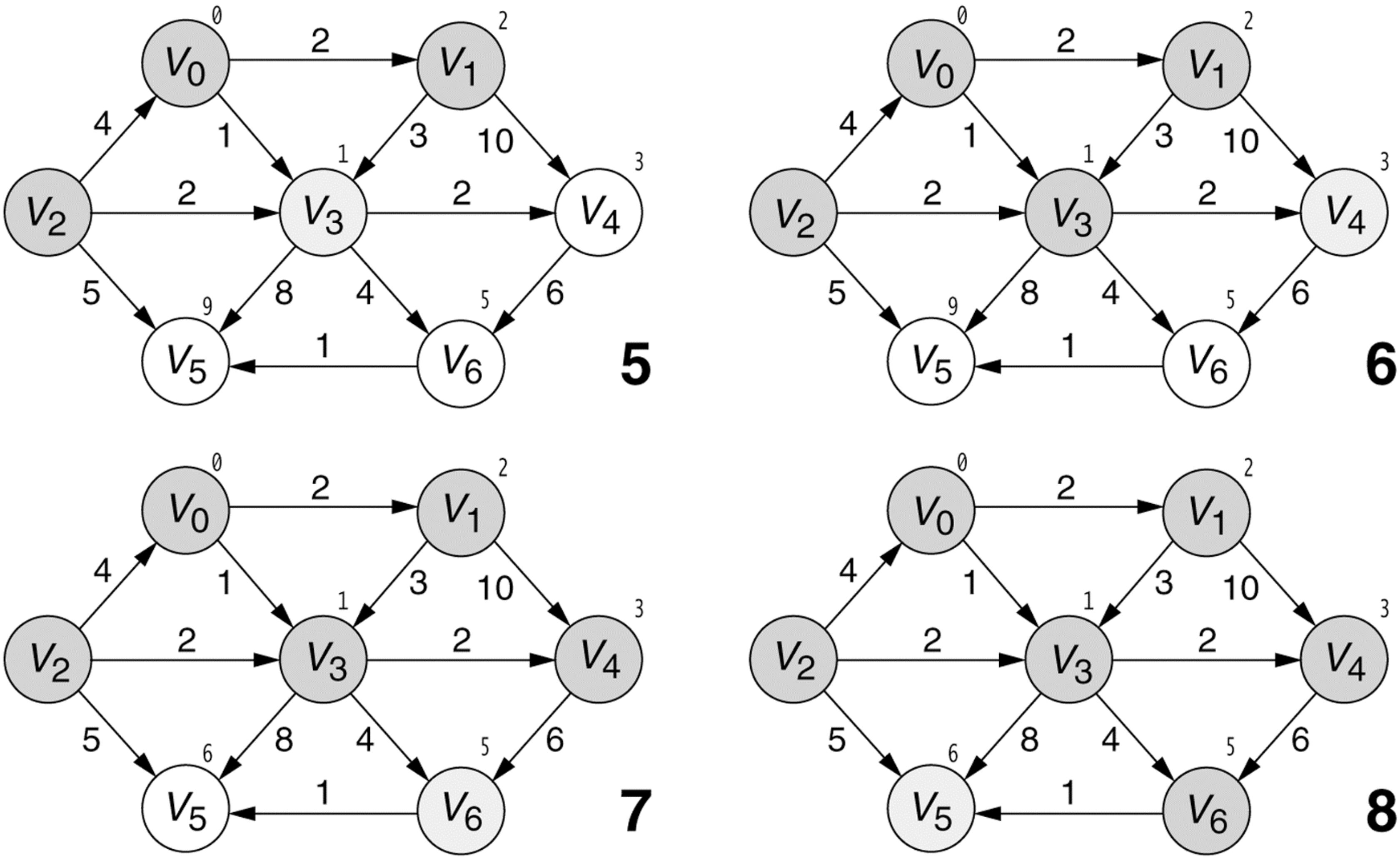


Figure 14.33

An activity-node graph

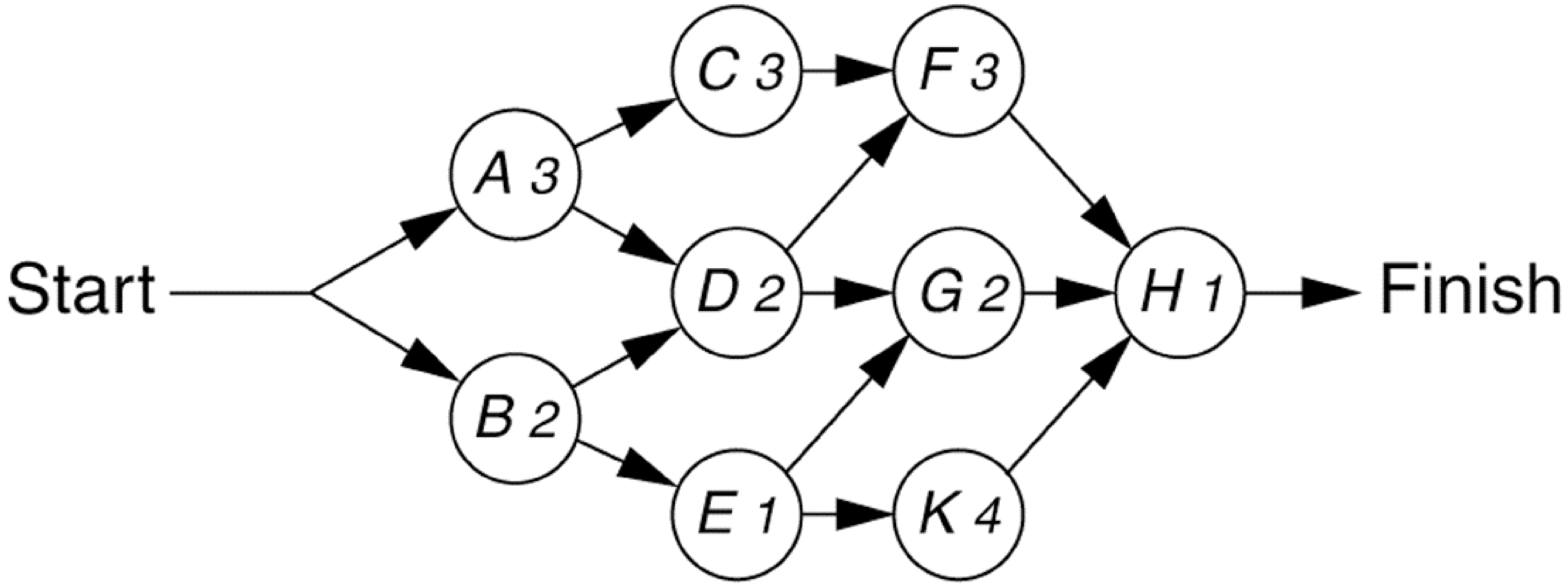


Figure 14.34
An event-node graph

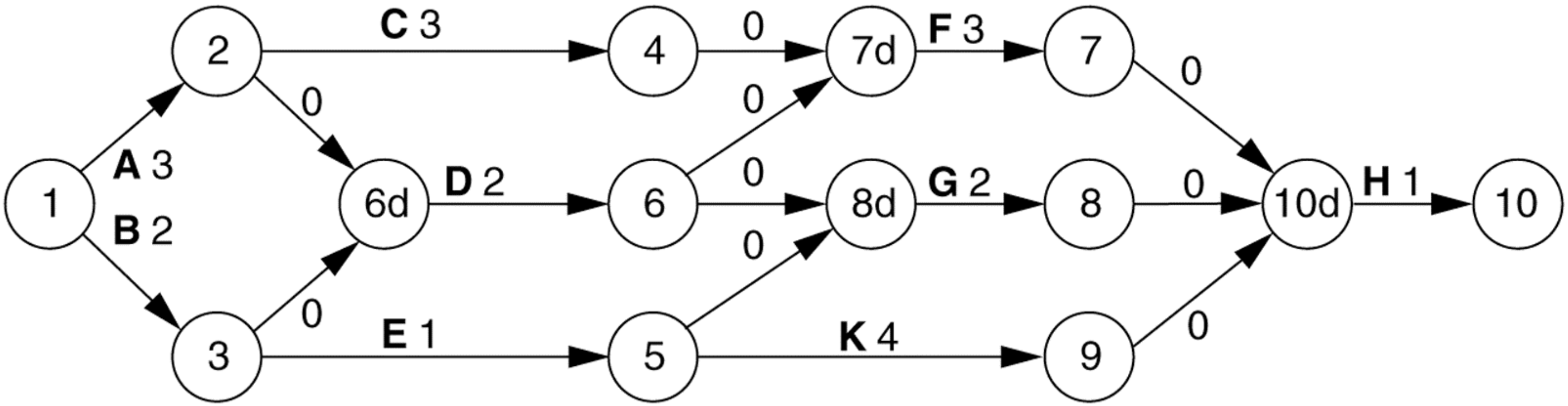


Figure 14.35
Earliest completion times

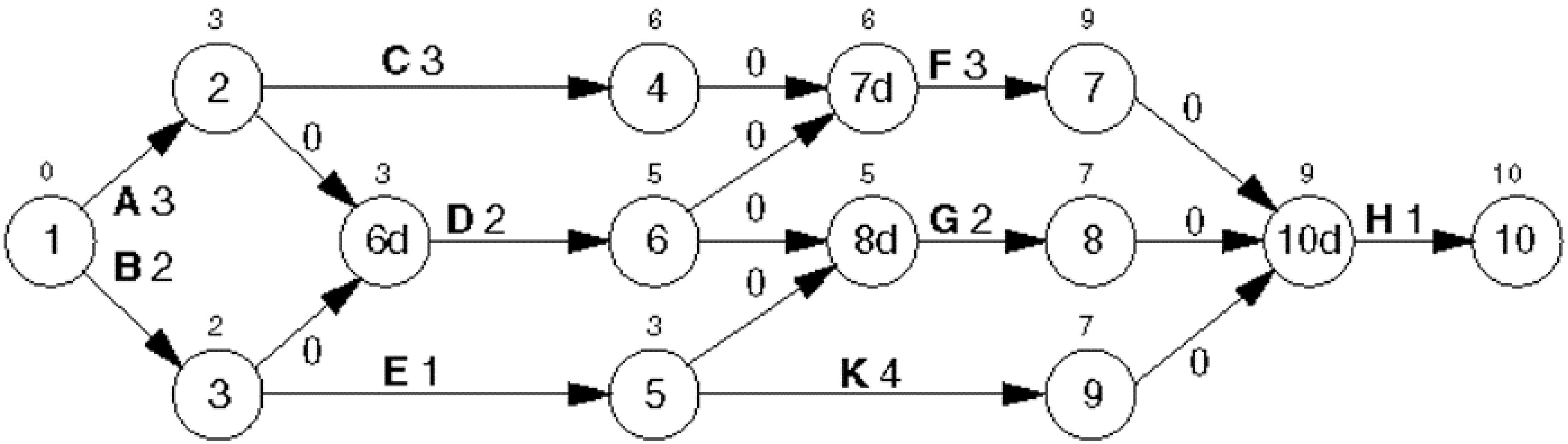


Figure 14.36
Latest completion times

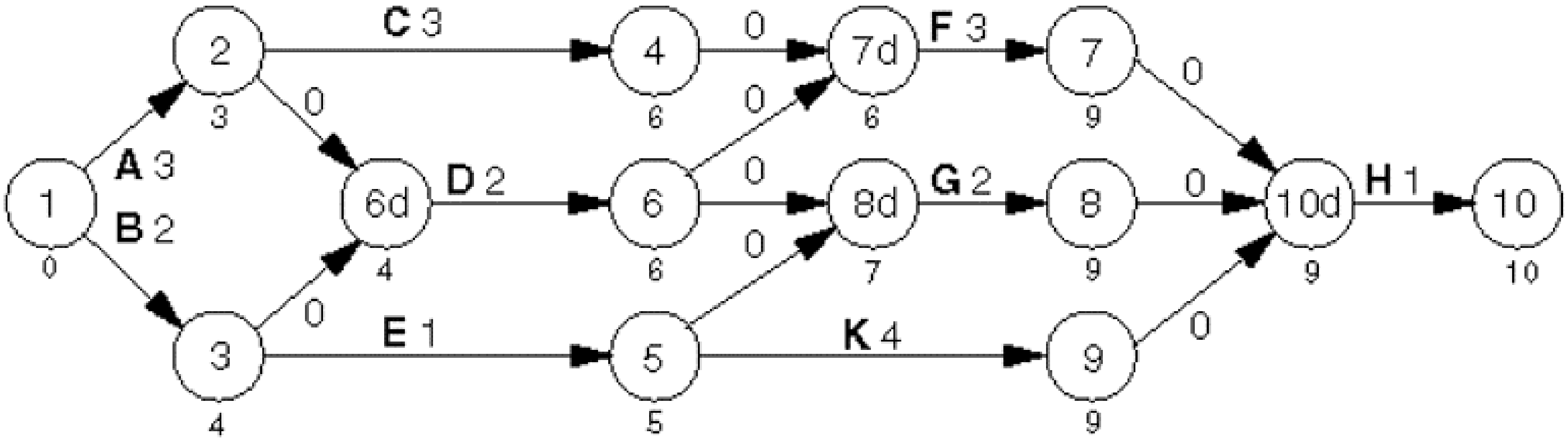


Figure 14.37

Earliest completion time, latest completion time, and slack (additional edge item)

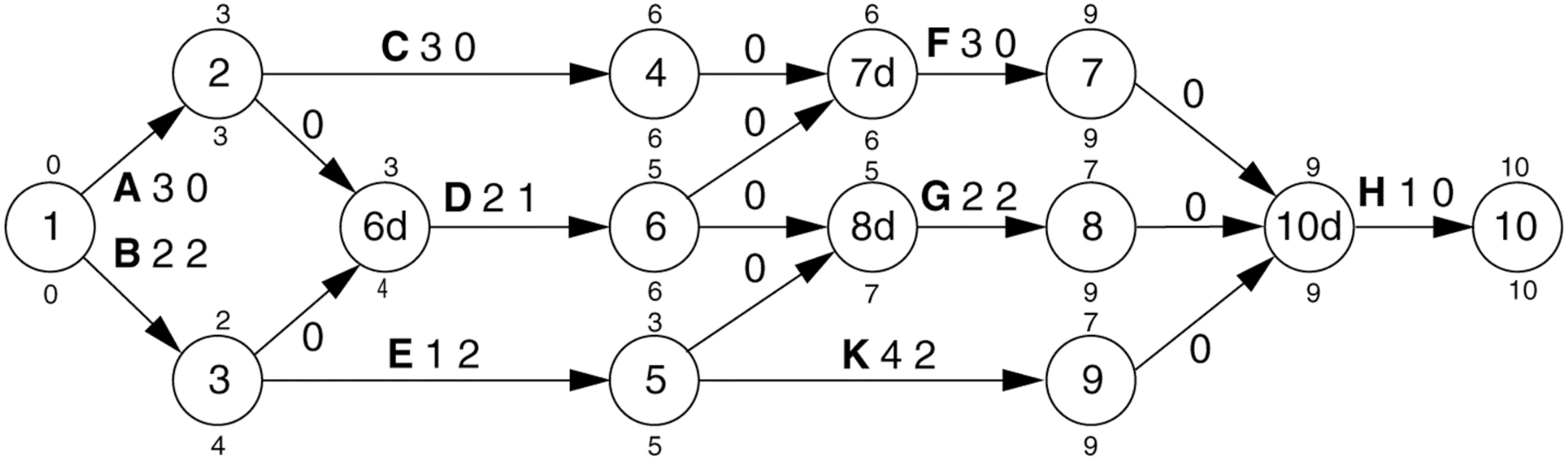


Figure 14.38
Worst-case running times of various graph algorithms

Type of Graph Problem	Running Time	Comments
Unweighted	$O(\|E\|)$	Breadth-first search
Weighted, no negative edges	$O(\|E\| \log \|V\|)$	Dijkstra's algorithm
Weighted, negative edges	$O(\|E\| \cdot \|V\|)$	Bellman–Ford algorithm
Weighted, acyclic	$O(\|E\|)$	Uses topological sort