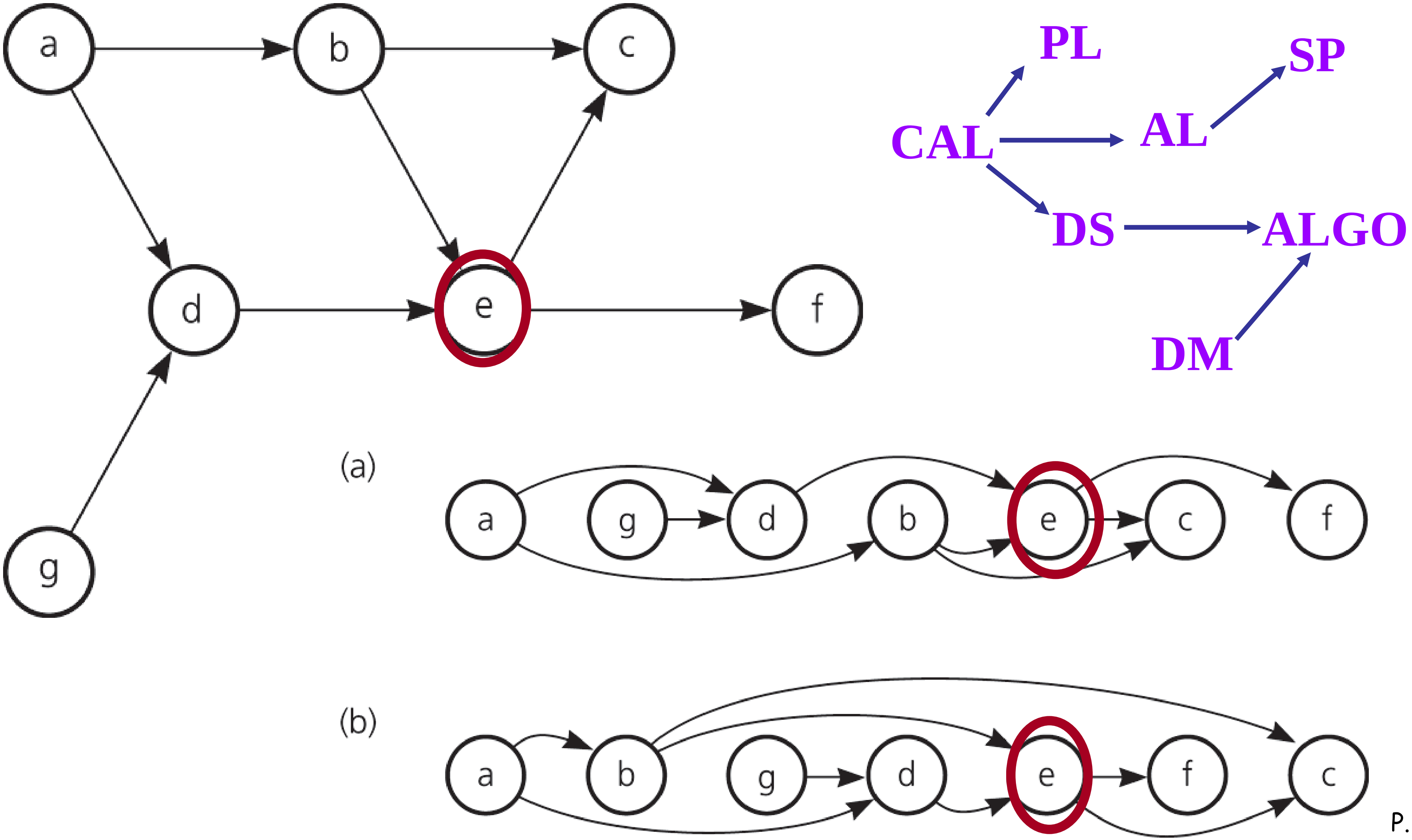
Graph Problems

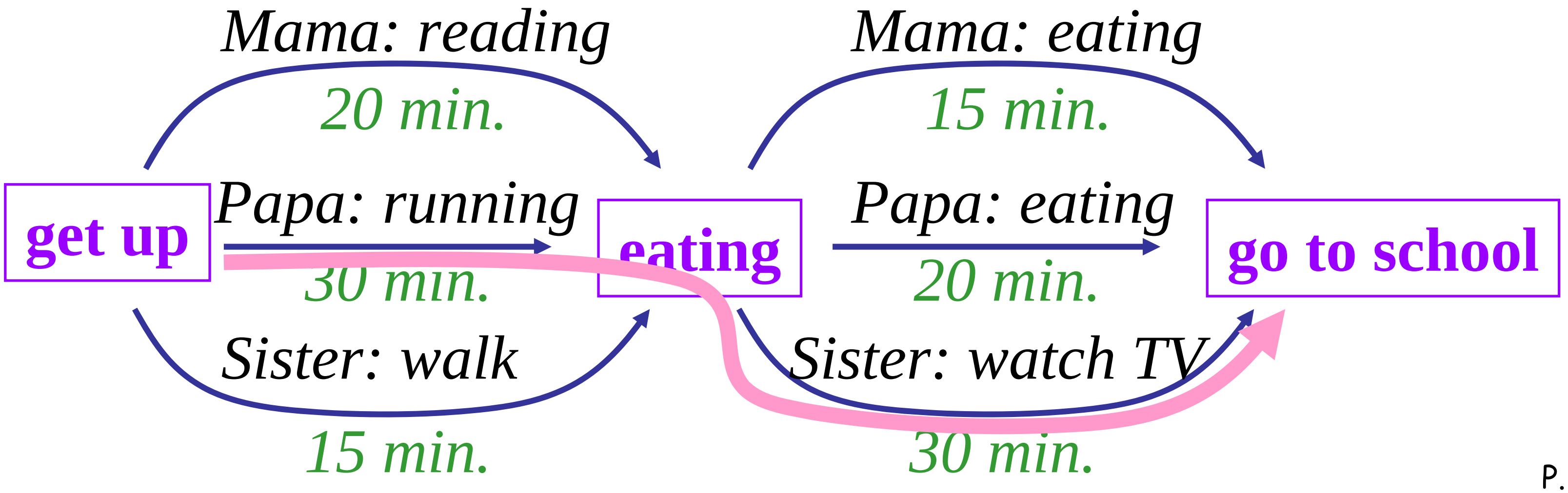
- □ Critical Path Analysis
- □ Maximum Flow Problem
- □ Other Difficult Problems

Activity-on-Vertex (AOV) Network



Activity-on-Edge (AOE) Network

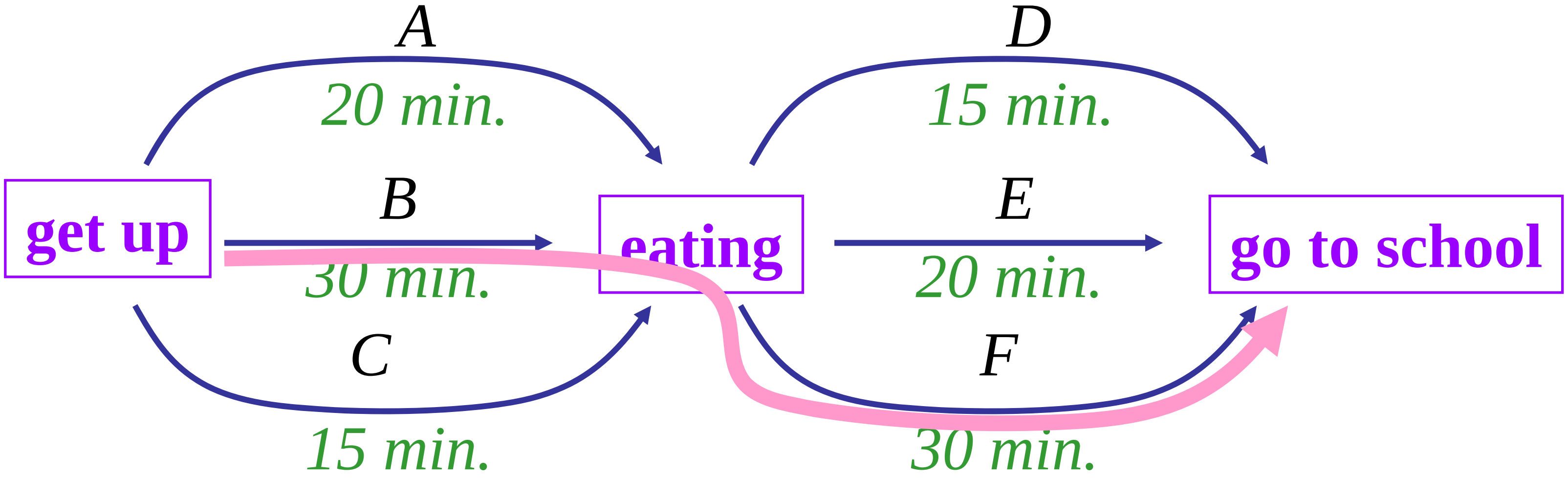
- □ Directed edge: activity (task) to be performed
- □ Vertex: event to signal the completion of certain activities
- □ Edge weight: the time required to perform an activity
- □ Path length: the total time from the start to the last event
- □ Critical Path: a path with the longest length
  - – *the minimum time required to complete the project*



Mama: reading
20 min.
Papa: running
30 min.
Sister: walk
15 min.
Mama: eating
15 min.
Papa: eating
20 min.
Sister: watch TV
30 min.
get up
eating
go to school

Critical Path Analysis

- □ Earliest time of an activity/event: early(E) = 30
- □ Latest time of an activity/event: late(E) = 60 – 20 = 40
- □ Critical activity: late(F) = early(F) = 30

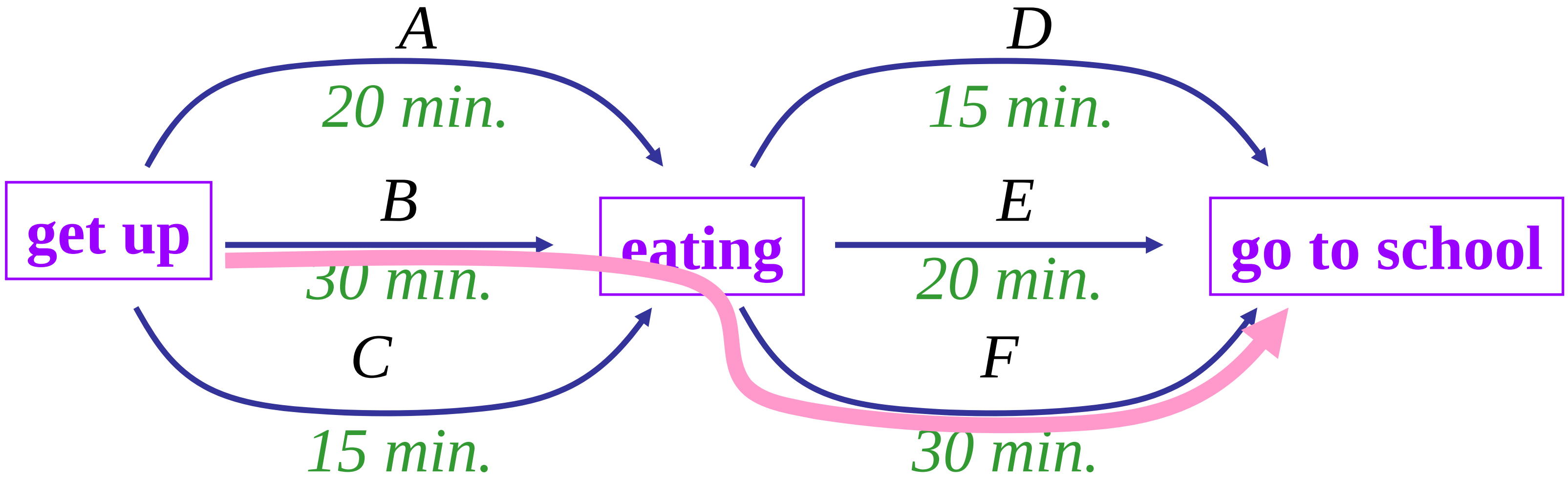


Critical Path Analysis

- **AOE network is very useful for evaluating the performance of many types of projects**
  - Project Evaluation and Review Techniques (PERT)

  Q1. What is *the least amount of time* in which the project may be complete (assuming there is no cycle in the network)?

  Q2. Which *activities* should be speeded to reduce project length?



Critical Path Analysis: *Background*

- **Developed in the 1950s by the US Navy**
  - Originally, it considered only *logical dependencies* among *project activities*. Since then, it has been expanded to include the *resources* related to each activity, through the process called *resource leveling*.
- **John Fondahl**
  - US Marine Corps Sergeant
  - Stanford CE Professor Emeritus
  - 1961 Paper for the US Navy – "*Non-Computer Approach to the Critical Path Method for the Construction Industry*"

wikipedia ©

Critical Path Analysis: *Model*

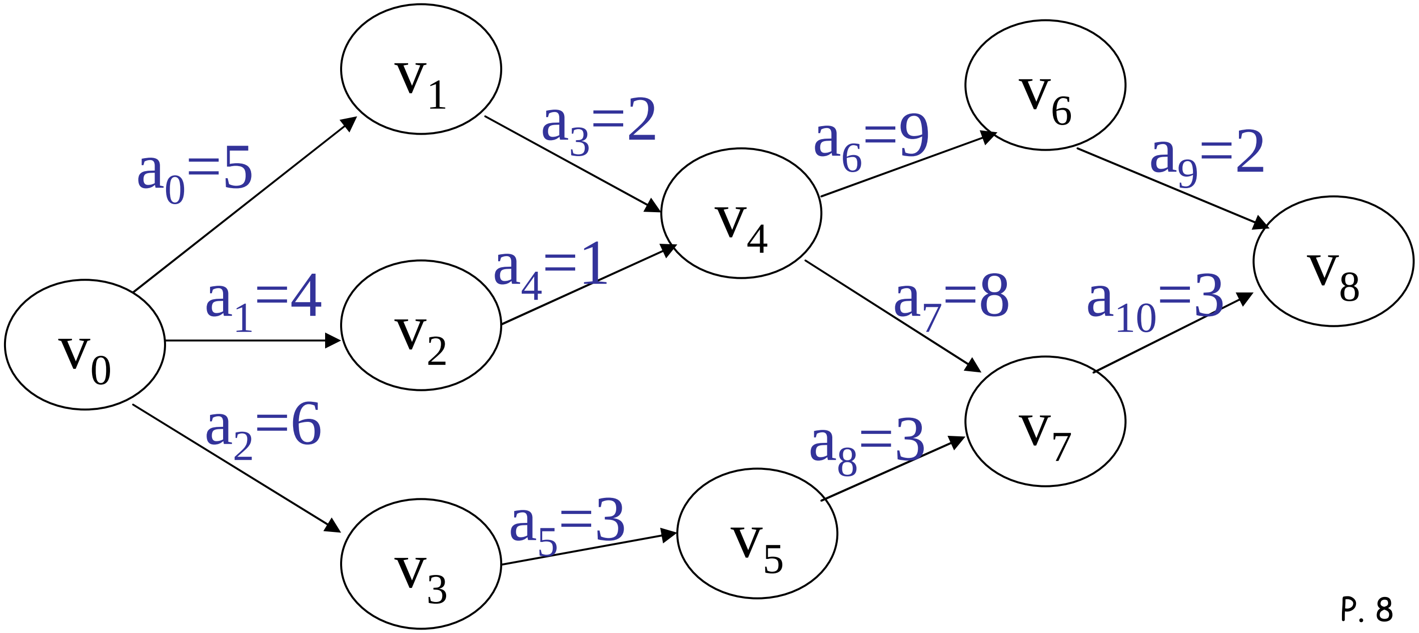
- □ **Input**
  - – A list of all *activities* required to complete the project
  - – The time (*duration*) that each activity will take to completion
  - – The *dependencies* between the activities.
- □ **Output**
  - – The *longest* path of planned activities to the end of the project
  - – The *earliest* and *latest* that each activity can start and finish without making the project longer
  - – Determines "*critical*" activities (on the longest path)
  - – Prioritize activities for the effective management and to shorten the critical path of a project

Critical Path Analysis: *Example*

- **Activities: a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 $a_9 a_{10}$**
- **Dependencies among Activities**

$a_0 \rightarrow a_3$, $a_1 \rightarrow a_4$, $a_2 \rightarrow a_5$, $a_5 \rightarrow a_8$, $a_6 \rightarrow a_9$

$a_3\ a_4 \rightarrow a_6\ a_7$, $a_7\ a_8 \rightarrow a_{10}$



Critical Path Analysis: *Forward*

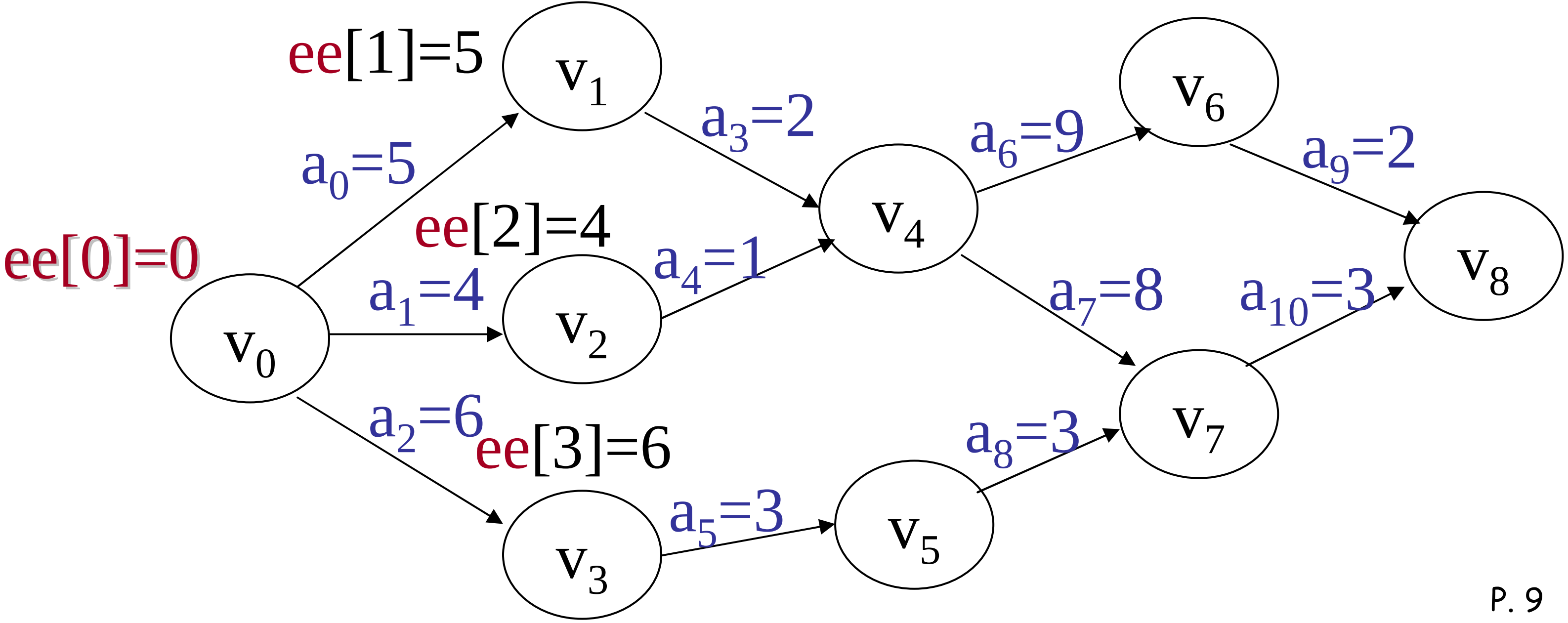
- **earliest time of an activity: ea[0..10]**
  - earlist time of an event: ee[0..8]
    - **ea[x] = ee[i] if a_x is on the edge $<v_i, v_j>$**
    - **ee[j] = *max*{ee[i] + *duration* of $<v_i, v_j>$} for every v_i that is an *immediate predecessor* of v_j**

ea

[0]: 0

[1]: 0

[2]: 0



Critical Path Analysis: *Forward*

- **ea[x] = ee[i] if a_x is on the edge $<v_i, v_j>$**
- **ee[j] = *max*{ee[i] + *duration* of $<v_i, v_j>$} for every v_i that is an *immediate predecessor* of v_j**

ea

[0]: 0

[1]: 0

[2]: 0

[3]: 5

[4]: 4

[5]: 6

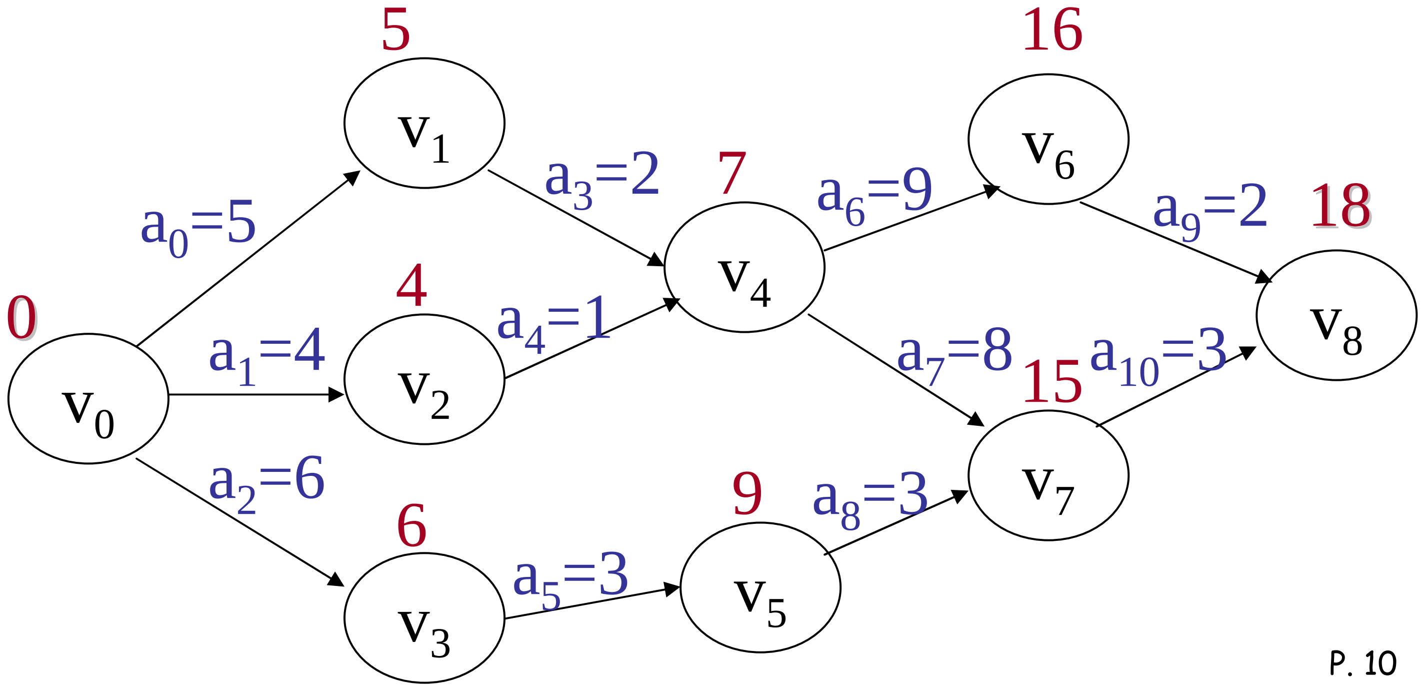
[6]: 7

[7]: 7

[8]: 9

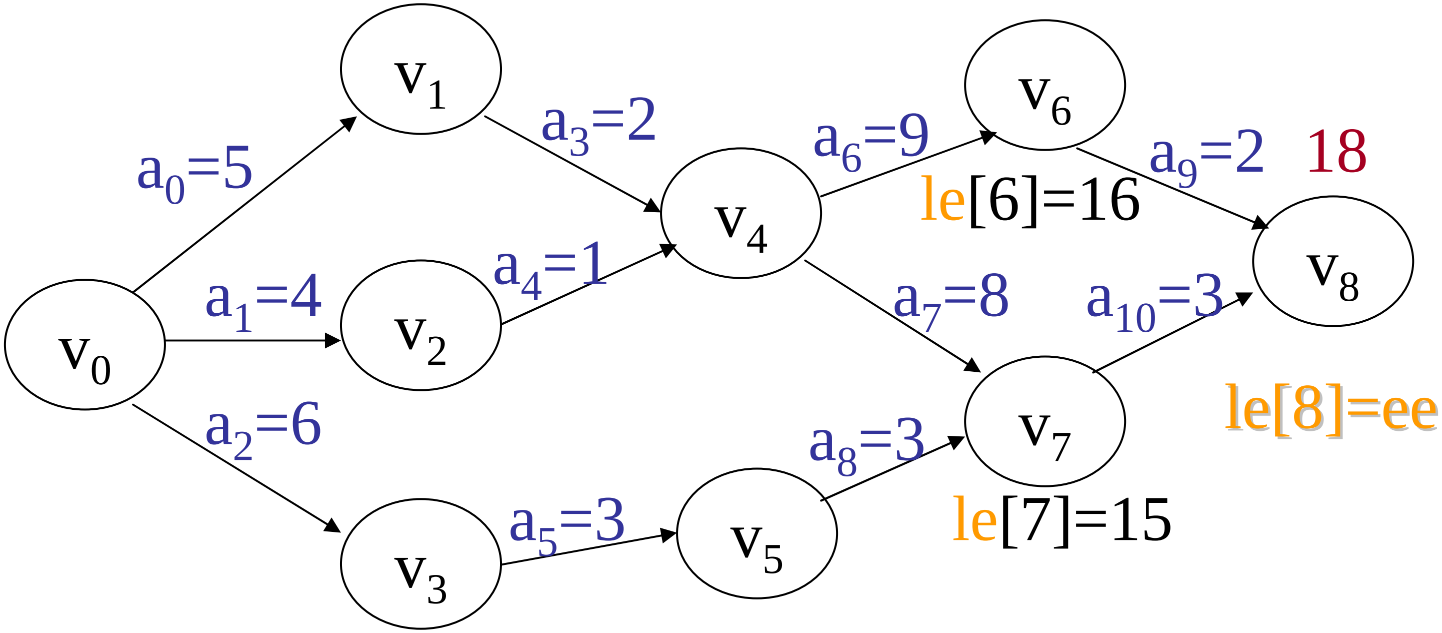
[9]: 16

[10]: 15



Critical Path Analysis: *Backward*

- **latest time of an activity: la[0..10]**
  - latest time of an event: le[0..8]
    - **la[x] = le[j] - *duration* of <v_i, v_j>, where a_x is on <v_i, v_j>**
    - **le[i] = *min*{le[j] - *duration* of <v_i, v_j>} for every v_j that is an *immediate successor* of v_i**



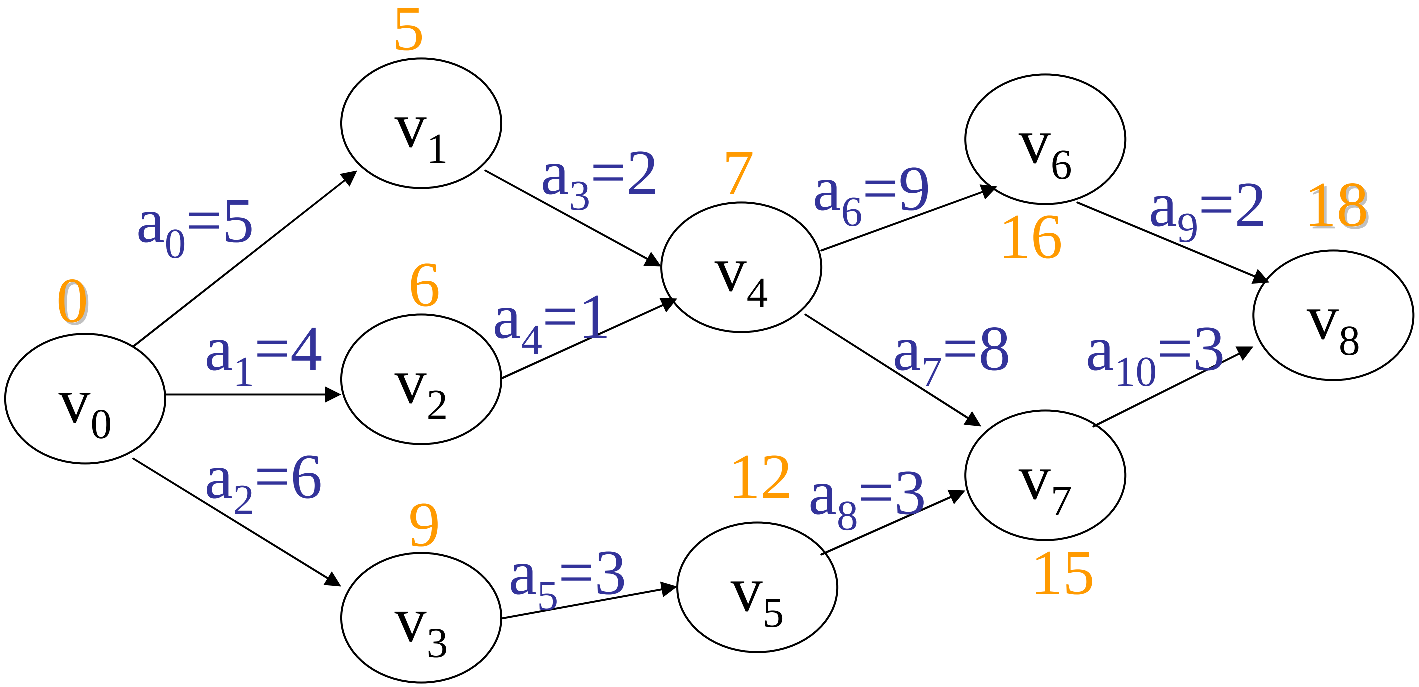
le[8]=ee[8]

[9]: 16
[10]: 15
la

Critical Path Analysis: *Backward*

- la[x] = le[j] - *duration* of $<v_i, v_j>$, where a_x is on $<v_i, v_j>$
- le[i] = *min*{le[j] - *duration* of $<v_i, v_j>$} for every v_j that is an *immediate successor* of v_i

[0]: 0
[1]: 2
[2]: 3
[3]: 5
[4]: 6
[5]: 9
[6]: 7
[7]: 7
[8]: 12
[9]: 16
[10]: 15
la

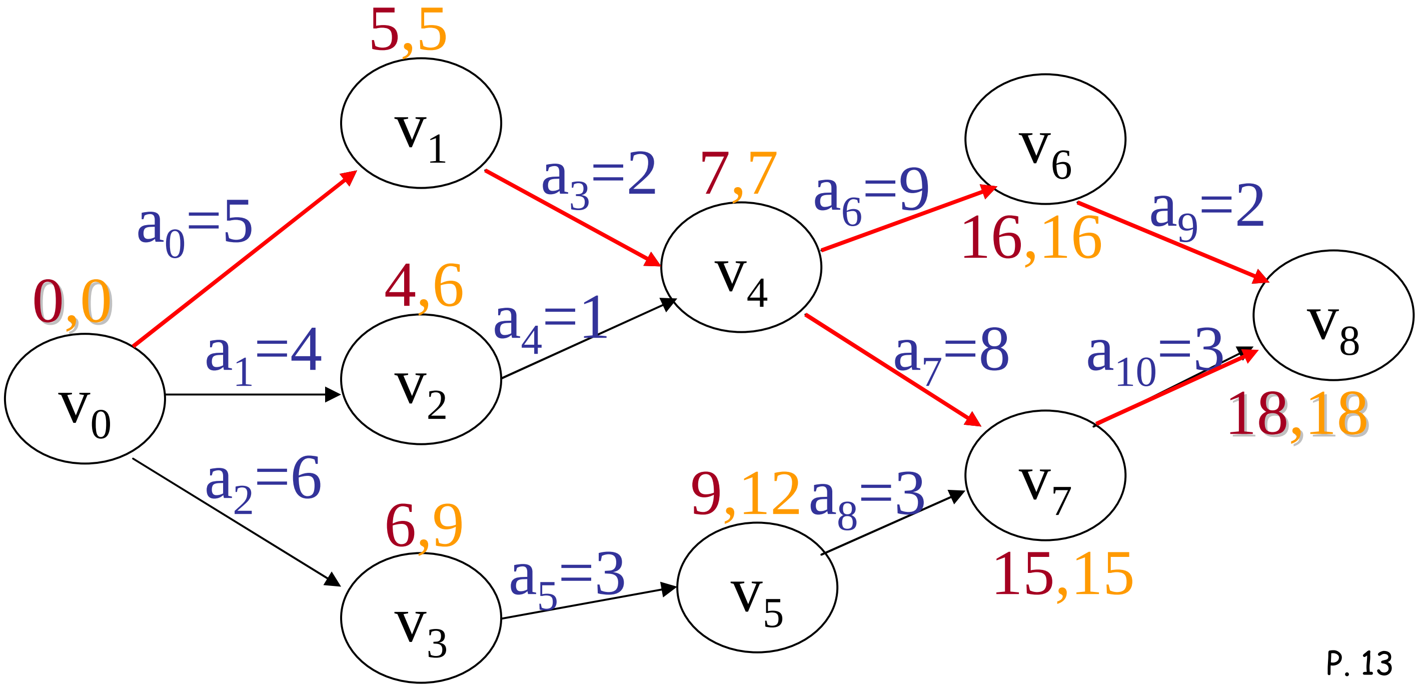


Critical Path Analysis: *Results*

ea	la	la-ea
[0]: 0	0	**0**
[1]: 0	2	**2**
[2]: 0	3	**3**
[3]: 5	5	**0**
[4]: 4	6	**2**
[5]: 6	9	**3**
[6]: 7	7	**0**
[7]: 7	7	**0**
[8]: 9	12	**3**
[9]: 16	16	**0**
[10]: 15	15	**0**

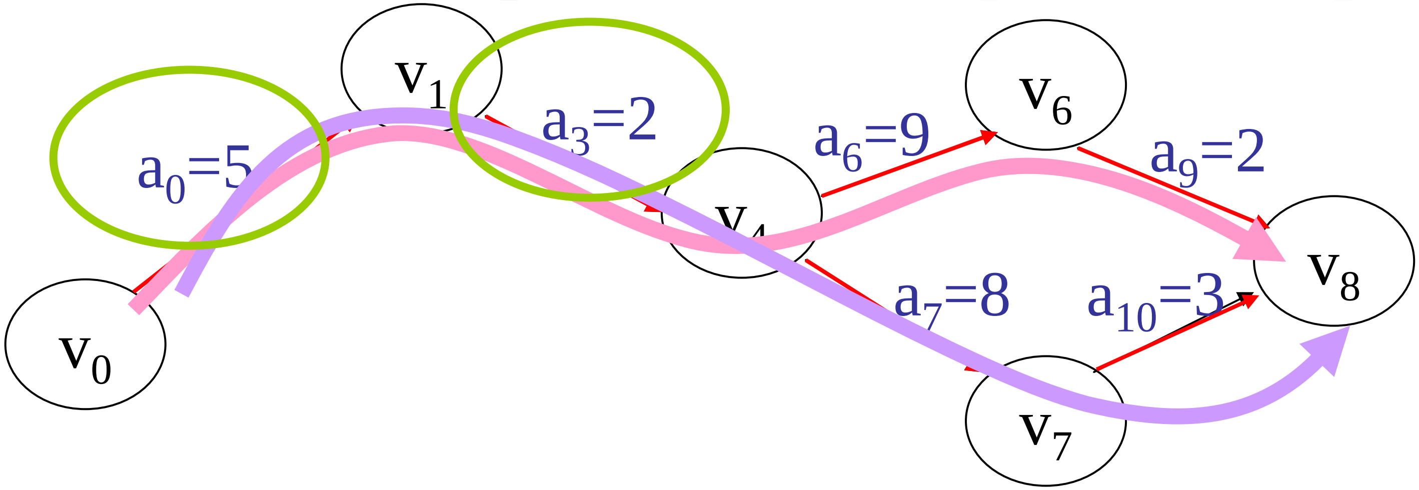
- **la-ea is called (total) *float* or *slack***
  - amount of time that a task can be delayed without causing a delay to project completion time

la-ea==0 means a *critical activity*



Critical Path Analysis: *Results*

- **Determine Critical Paths**
  - Delete all non-critical activities (*nonzero slack*)
  - Generate all the paths from the start to the end
- **Speed up the activities on all critical paths**
  - *resource* can be concentrated on these activities in an attempt to reduce the project completion time



Critical Path Method: *Forward Phase*

- **Like *topSort1***

Initialize: ee[v_0]=0

1. Find vertex *v* that has no predecessor (in-degree=0)
2. For each immediate successor *u*, do the following:
   - **Set ea[*x*] = ee[*v*], where *x* is the *activity* on <*v*,*u*>**
   - **Set ee[*u*] = *max*{ee[*u*], ee[*v*] + *duration* of <*v*,*u*>}**
   - **Decrease the in-degree of *u***

Put *u* into queue or stack

3. Repeat the steps until *all vertices are visited*
   - **For the vertex *w* that has no successor, le[*w*]=ee[*w*]!**

Critical Path Method: *Forward Phase*

	in-degree	activity	duration	activity	duration	activity	duration
v_0	0	a_0	5	a_1	4	a_2	6
v_1	0	a_3	2				
v_2	0	a_4	1				
v_3	0	a_5	3				
v_4	0	a_6	9	a_7	8		
v_5	1	a_8	3				
v_6	1	a_9	2				
v_7	2	a_{10}	3				
v_8	2						

queue: v_3 v_4

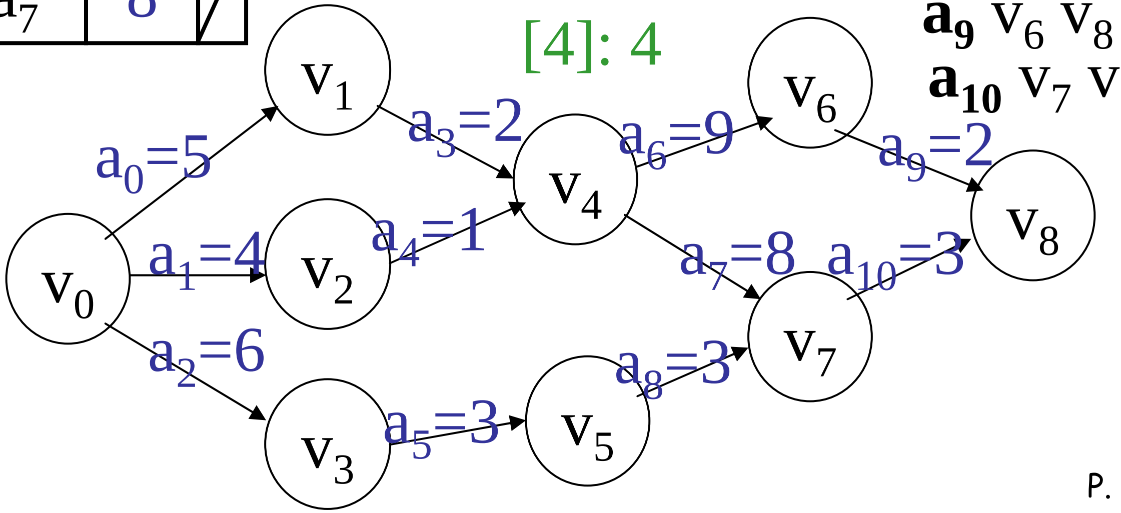
ea

[0]: 0
[1]: 0
[2]: 0
[3]: 5
[4]: 4

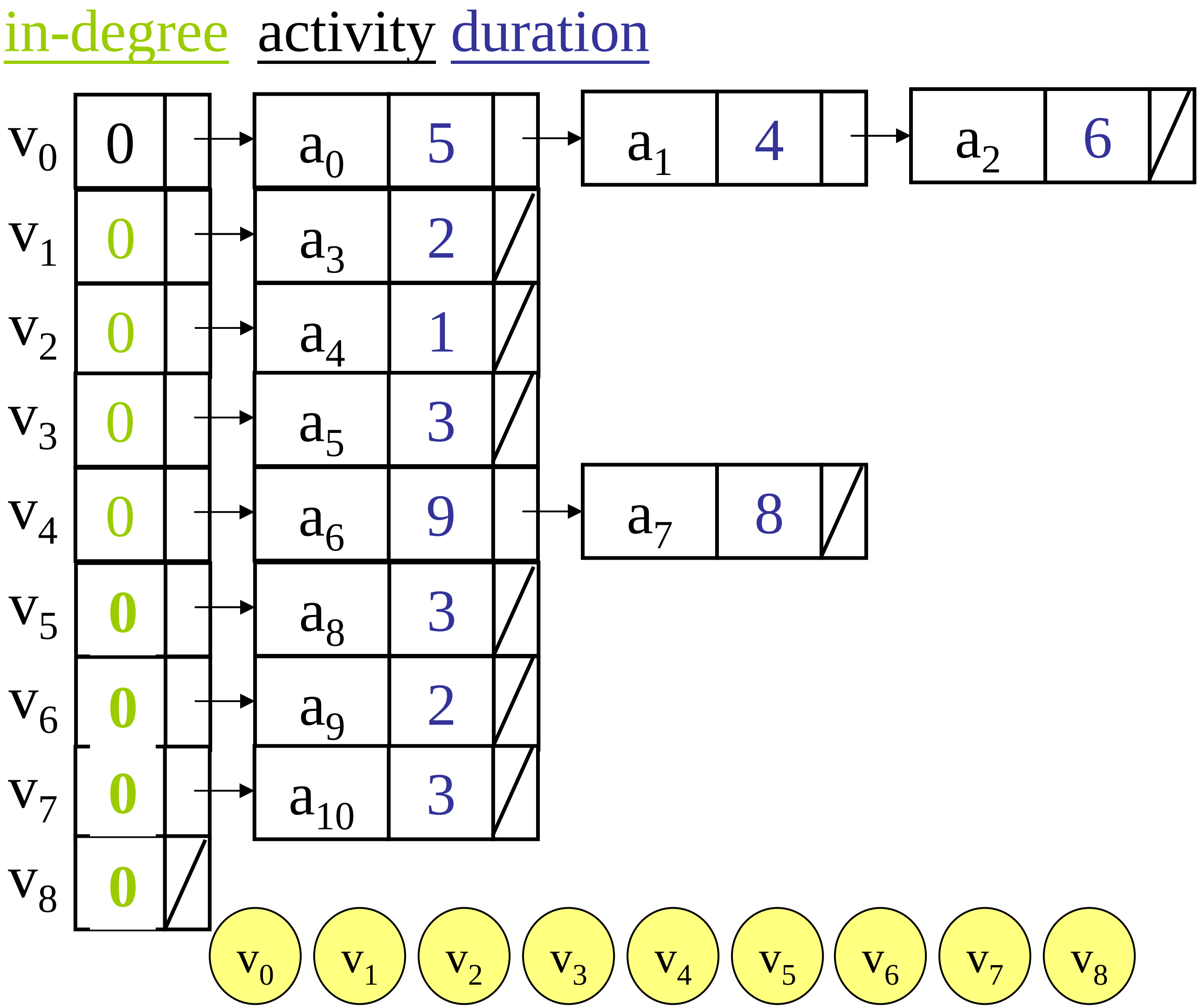
ee

[0]: 0
[1]: 5
[2]: 4
[3]: 6
[4]: 7

a_0	v_0	v_1
a_1	v_0	v_2
a_2	v_0	v_3
a_3	v_1	v_4
a_4	v_2	v_4
a_5	v_3	v_5
a_6	v_4	v_6
a_7	v_4	v_7
a_8	v_5	v_7
a_9	v_6	v_8
a_{10}	v_7	v_8



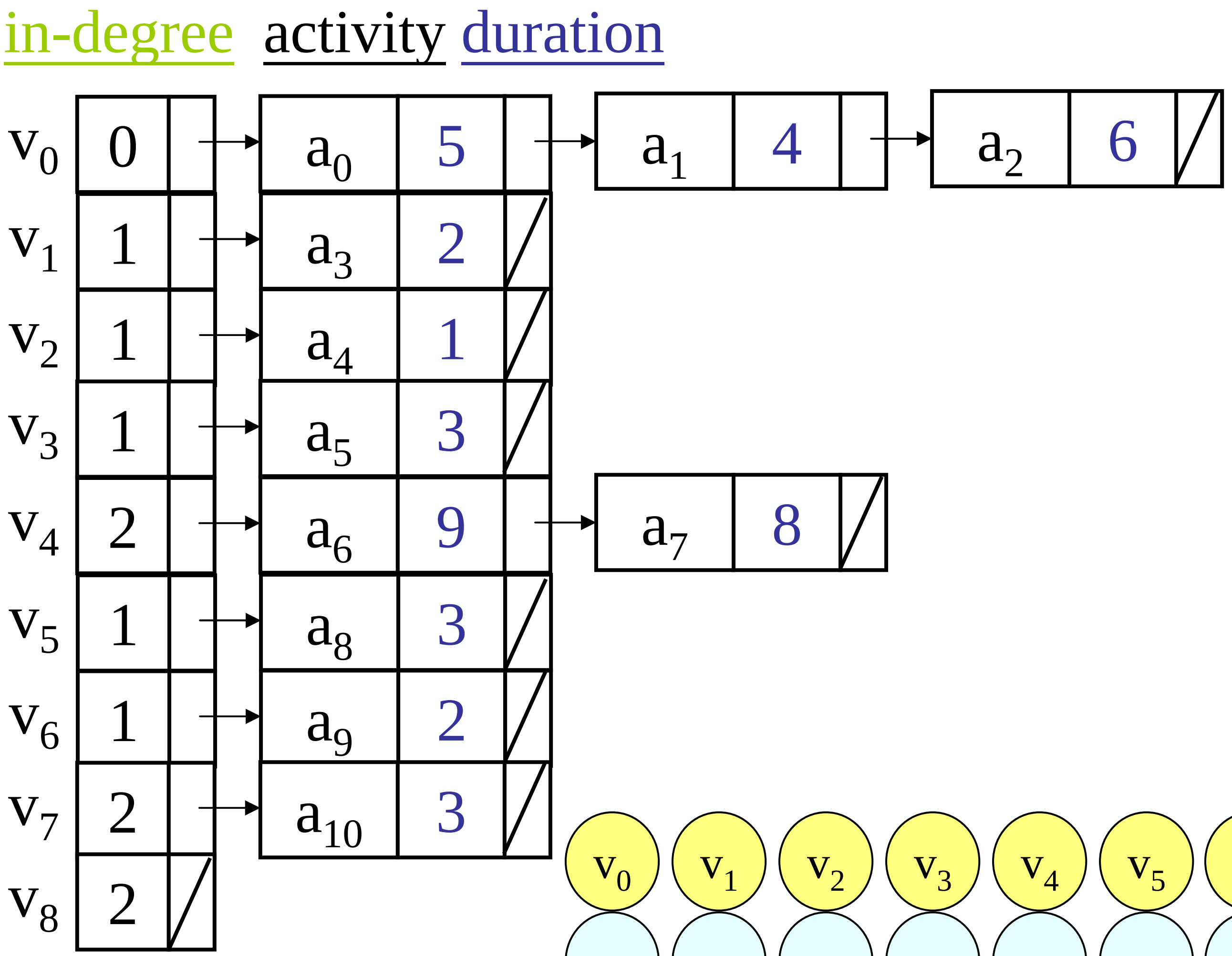
Critical Path Method: *Forward Phase*



ea	ee
[0]: 0	[0]: 0
[1]: 0	[1]: 5
[2]: 0	[2]: 4
[3]: 5	[3]: 6
[4]: 4	[4]: 7
[5]: 6	[5]: 9
[6]: 7	[6]: 16
[7]: 7	[7]: 15

$\mathbf{a_0}$	v_0	v_1
$\mathbf{a_1}$	v_0	v_2
$\mathbf{a_2}$	v_0	v_3
$\mathbf{a_3}$	v_1	v_4
$\mathbf{a_4}$	v_2	v_4
$\mathbf{a_5}$	v_3	v_5
$\mathbf{a_6}$	v_4	v_6
$\mathbf{a_7}$	v_4	v_7
$\mathbf{a_8}$	v_5	v_7
$\mathbf{a_9}$	v_6	v_8
$\mathbf{a_{10}}$	v_7	v_8

Critical Path Method: *Forward Phase*



in-degree activity duration

	in-degree			
v_0	0	a_0 5	a_1 4	a_2 6
v_1	1	a_3 2		
v_2	1	a_4 1		
v_3	1	a_5 3		
v_4	2	a_6 9	a_7 8	
v_5	1	a_8 3		
v_6	1	a_9 2		
v_7	2	a_{10} 3		
v_8	2			

$\mathbf{a_0}$	v_0	v_1
$\mathbf{a_1}$	v_0	v_2
$\mathbf{a_2}$	v_0	v_3
$\mathbf{a_3}$	v_1	v_4
$\mathbf{a_4}$	v_2	v_4
$\mathbf{a_5}$	v_3	v_5
$\mathbf{a_6}$	v_4	v_6
$\mathbf{a_7}$	v_4	v_7
$\mathbf{a_8}$	v_5	v_7
$\mathbf{a_9}$	v_6	v_8
$\mathbf{a_{10}}$	v_7	v_8

v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8

v_0 v_3 v_5 v_2 v_1 v_4 v_7 v_6 v_8

Critical Path Method: *Backward Phase*

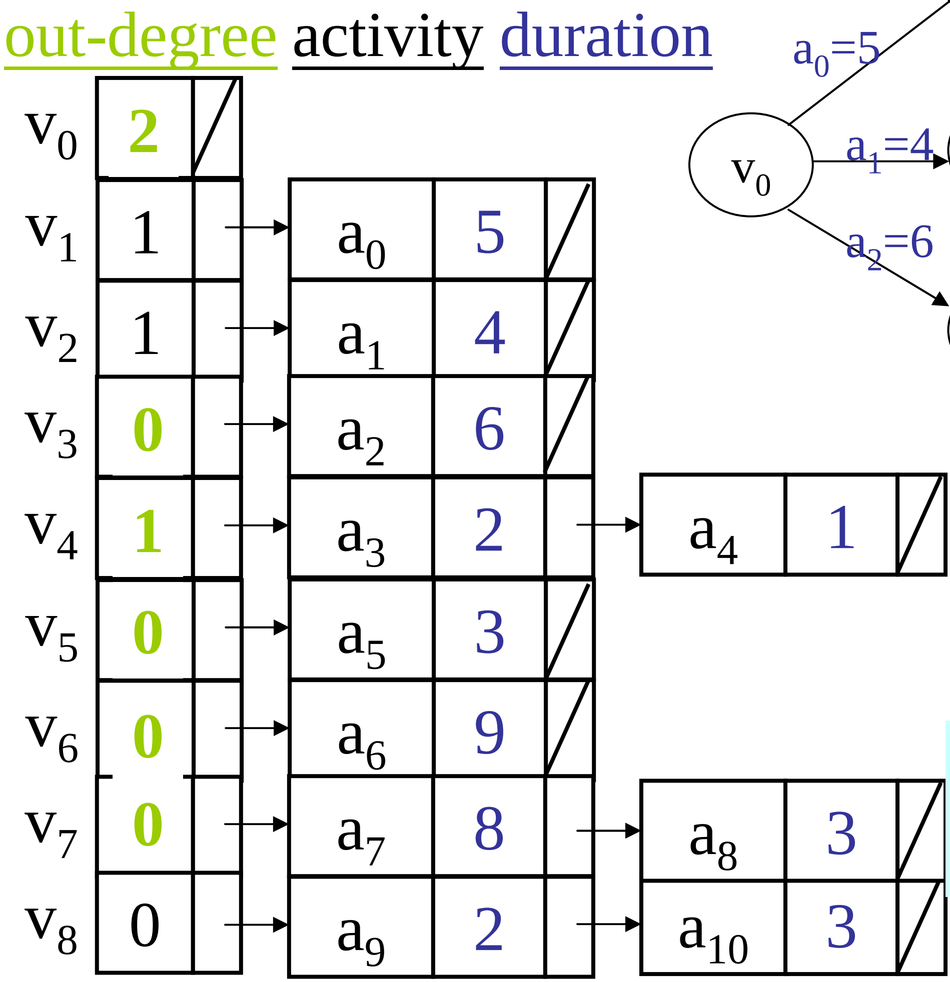
Initialize: le[*u*]=ee[*u*]

1. Find vertex *u* that has no successor (out-degree=0)
2. For each immediate predecessor *v*, do the following:
   - **Set la[*x*] = le[*u*] - *duration* of <*v*,*u*>}, where *x* is the *activity* on <*v*,*u*>**
   - **Set le[*v*] = *min*{le[*v*], le[*u*] - *duration* of <*v*,*u*>}**
   - **Decrease the out-degree of *v***
3. Repeat the steps until *all verti* *are visited*
   - **For the vertex *w* that has no pr ssor, le[*w*]=ee[*w*]!**

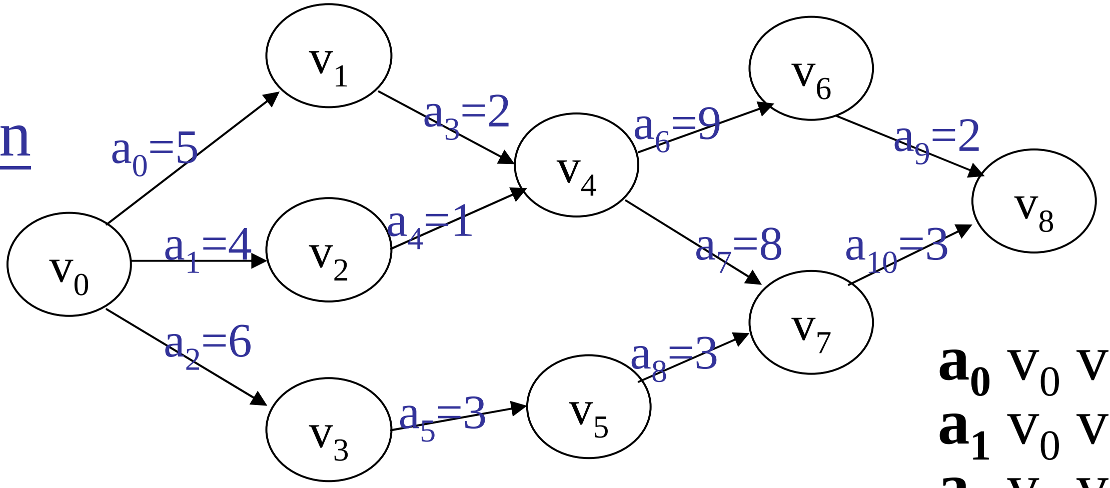
Put *v* into queue or stack

Critical Path Method: *Backward Phase*

Inverse adjacency list



	out-degree		activity	duration		activity	duration	
v_0	2							
v_1	1	→	a_0	5				
v_2	1	→	a_1	4				
v_3	0	→	a_2	6				
v_4	1	→	a_3	2	→	a_4	1	
v_5	0	→	a_5	3				
v_6	0	→	a_6	9				
v_7	0	→	a_7	8	→	a_8	3	
v_8	0	→	a_9	2	→	a_{10}	3	



v_6
stack

[2]: 3
[5]: 9
[7]: 7
[8]: 12
[9]:16
[10]: 15
la

[0]: 3
[3]: 9
[4]: 7
[5]: 12
[6]: 16
[7]: 15
[8]: 18
le

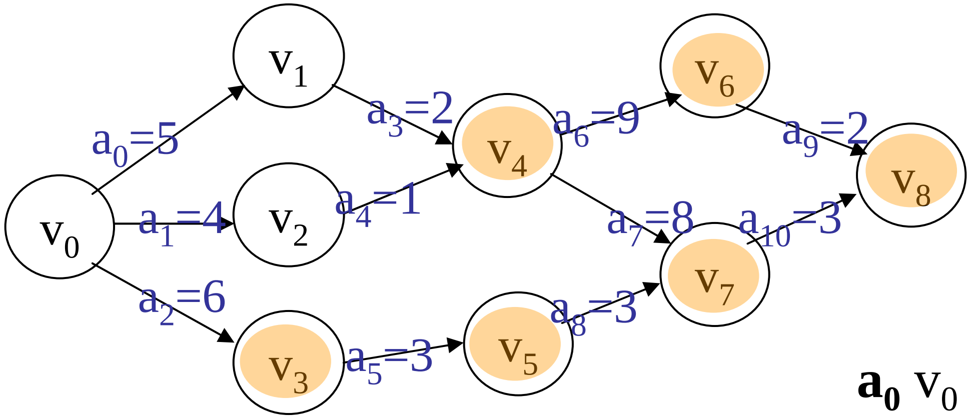
$\mathbf{a_0}$	v_0	v_1
$\mathbf{a_1}$	v_0	v_2
$\mathbf{a_2}$	v_0	v_3
$\mathbf{a_3}$	v_1	v_4
$\mathbf{a_4}$	v_2	v_4
$\mathbf{a_5}$	v_3	v_5
$\mathbf{a_6}$	v_4	v_6
$\mathbf{a_7}$	v_4	v_7
$\mathbf{a_8}$	v_5	v_7
$\mathbf{a_9}$	v_6	v_8
$\mathbf{a_{10}}$	v_7	v_8

Critical Path Method: *Backward Phase*

Inverse adjacency list

out-degree activity duration

	out-degree	activity	duration	activity	duration
v_0	2				
v_1	0	a_0	5		
v_2	0	a_1	4		
v_3	0	a_2	6		
v_4	0	a_3	2	a_4	1
v_5	0	a_5	3		
v_6	0	a_6	9		
v_7	0	a_7	8	a_8	3
v_8	0	a_9	2	a_{10}	3



v_2
v_1
stack

la

[2]: 3
[3]: 7-2=5
[4]: 7-1=6
[5]: 9
[6]: 16-9=7
[7]: 7
[8]: 12
[9]: 16
[10]: 15

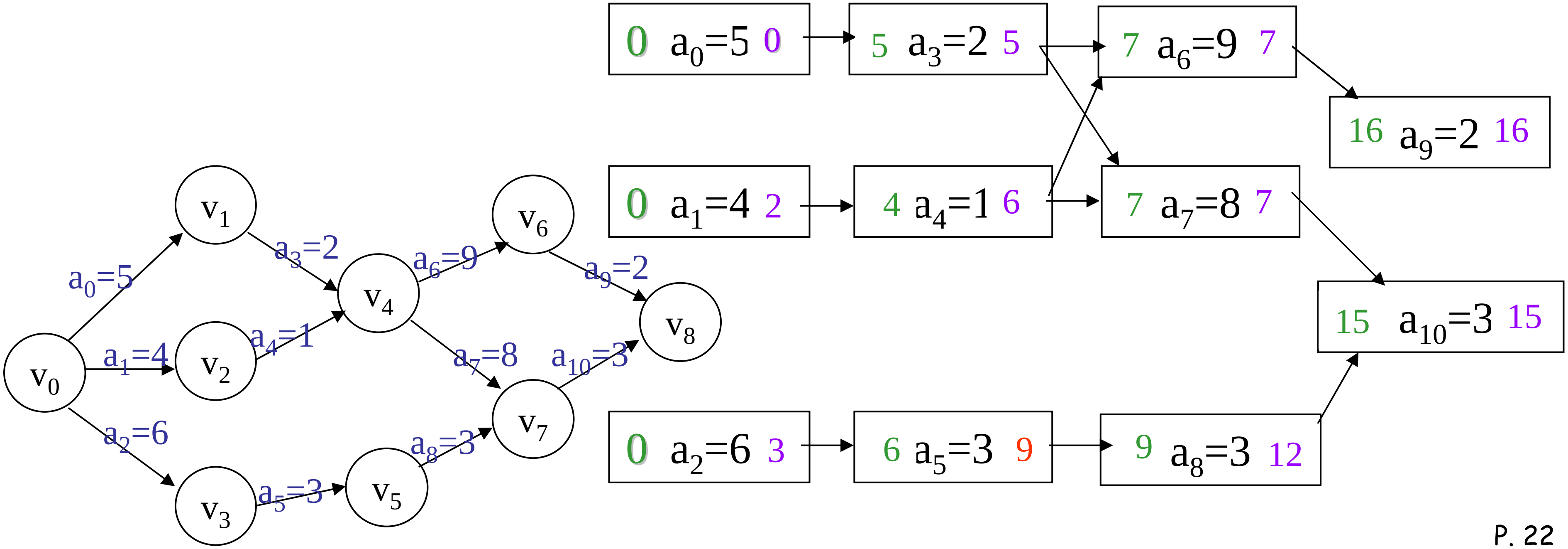
le

[0]: 3
[1]: 5
[2]: 6
[3]: 9
[4]: 7
[5]: 12
[6]: 16
[7]: 15
[8]: 18

$\mathbf{a_0}$	v_0	v_1
$\mathbf{a_1}$	v_0	v_2
$\mathbf{a_2}$	v_0	v_3
$\mathbf{a_3}$	v_1	v_4
$\mathbf{a_4}$	v_2	v_4
$\mathbf{a_5}$	v_3	v_5
$\mathbf{a_6}$	v_4	v_6
$\mathbf{a_7}$	v_4	v_7
$\mathbf{a_8}$	v_5	v_7
$\mathbf{a_9}$	v_6	v_8
$\mathbf{a_{10}}$	v_7	v_8

Critical Path Analysis: *Extensions*

- **Critical-path analysis can be carried out with AOV network**
- **Free float: amount of time that a task can be delayed without causing a delay to the *earliest start time* of any *immediately following activities*.**
  - earilest finish time & latest finish time for each activity



0 $a_0=5$ 0

5 $a_3=2$ 5

7 $a_6=9$ 7

16 $a_9=2$ 16

0 $a_1=4$ 2

4 $a_4=1$ 6

7 $a_7=8$ 7

15 $a_{10}=3$ 15

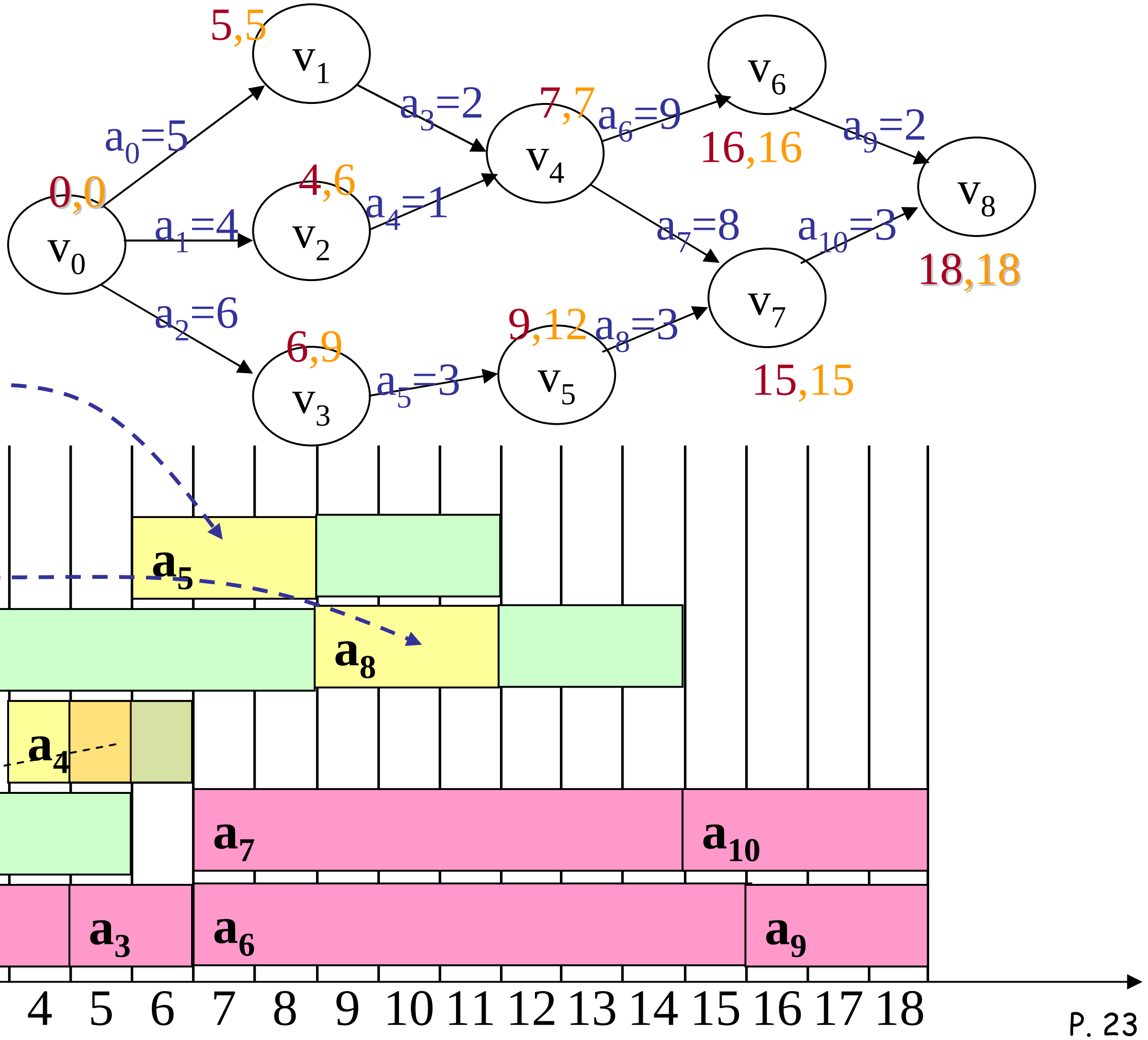
0 $a_2=6$ 3

6 $a_5=3$ 9

9 $a_8=3$ 12

Critical Path Method: *Gantt Chart*

	ea	la	la-ea
[0]:	0	0	**0**
[1]:	0	2	**2**
[2]:	0	3	**3**
[3]:	5	5	**0**
[4]:	4	6	**2**
[5]:	6	9	**3**
[6]:	7	7	**0**
[7]:	7	7	**0**
[8]:	9	12	**3**
[9]:	16	16	**0**
[10]:	15	15	**0**



free float =
7-(4+1) = 2

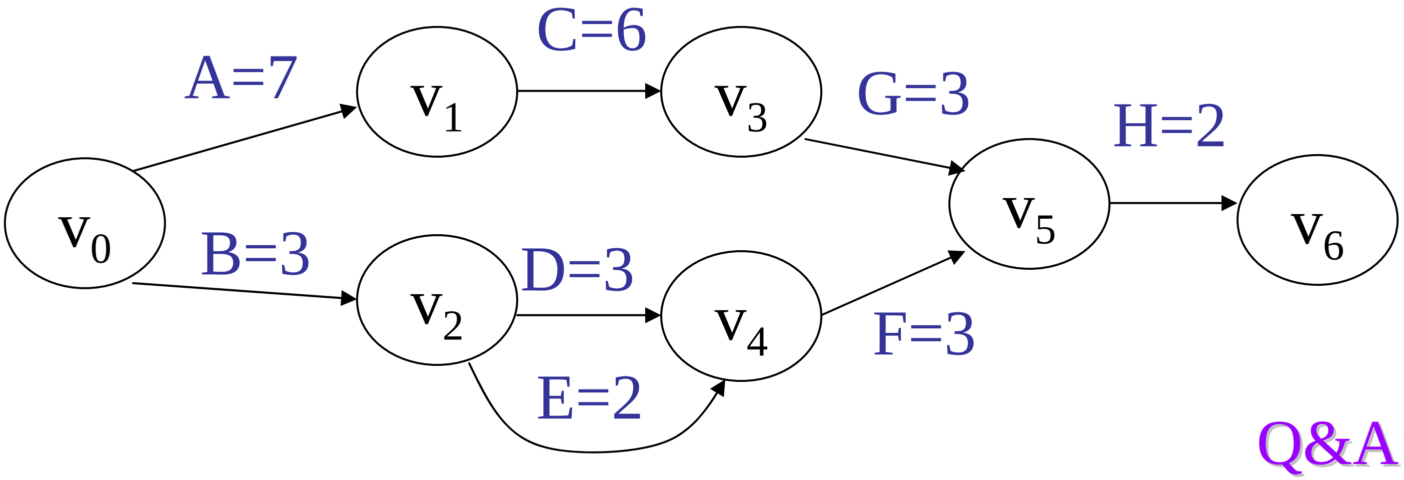
Practice 11: *Critical Path*

- **Find the critical path in the following network**

activity	duration	dependencies
A	7	
B	3	
C	6	A
D	3	B
E	2	B
F	3	D, E
G	3	C
H	2	F,G

Practice 11: *Solution*

- □ **Find the critical path in the following network**



Q&A: free float?

Self-exercise 7

1. Consider the activities and durations required to complete a project and dependencies among them.

(a) What is the latest time that the activity E can start?

(b) What is the minimum time required to complete the project?

(c) What are the critical activities?

activity	duration	predecessor
A	2	
B	3	
C	6	
D	3	A
E	2	B
F	4	D
G	3	C, E