CSCI 335 Software Design and Analysis III

Graph Algorithms

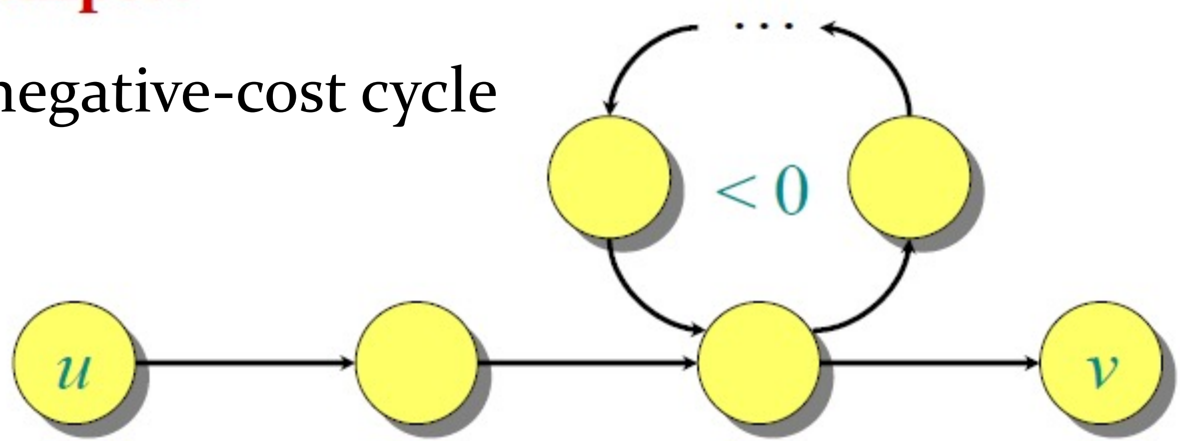
(Negative costs, Acyclic Graphs, Event-node graphs)

Negative edge costs

- Dijkstra may not work
- If there are no negative-cost cycles solution exists, but is expensive: O(|V|*|E|).

Example:

WITH negative-cost cycle



```
void Graph::WeightedNegative(Vertex s) {
    Queue<Vertex> q;

    for each Vertex v
        v.distance_ = kInfinity;

    s.distance_ = 0;
    q.enqueue(s);

    while (!q.isEmpty()) {
        Vertex v = q.dequeue();

        for each Vertex w adjacent to v
            if (v.distance_ + cvw < w.distance_ ) {
                w.distance_ = v.distance_ + cvw;
                w.previous_ = v;
                if (w is not already in q)
                    q.enqueue(w);
            }
    }
}
```

No KNOWN vertices

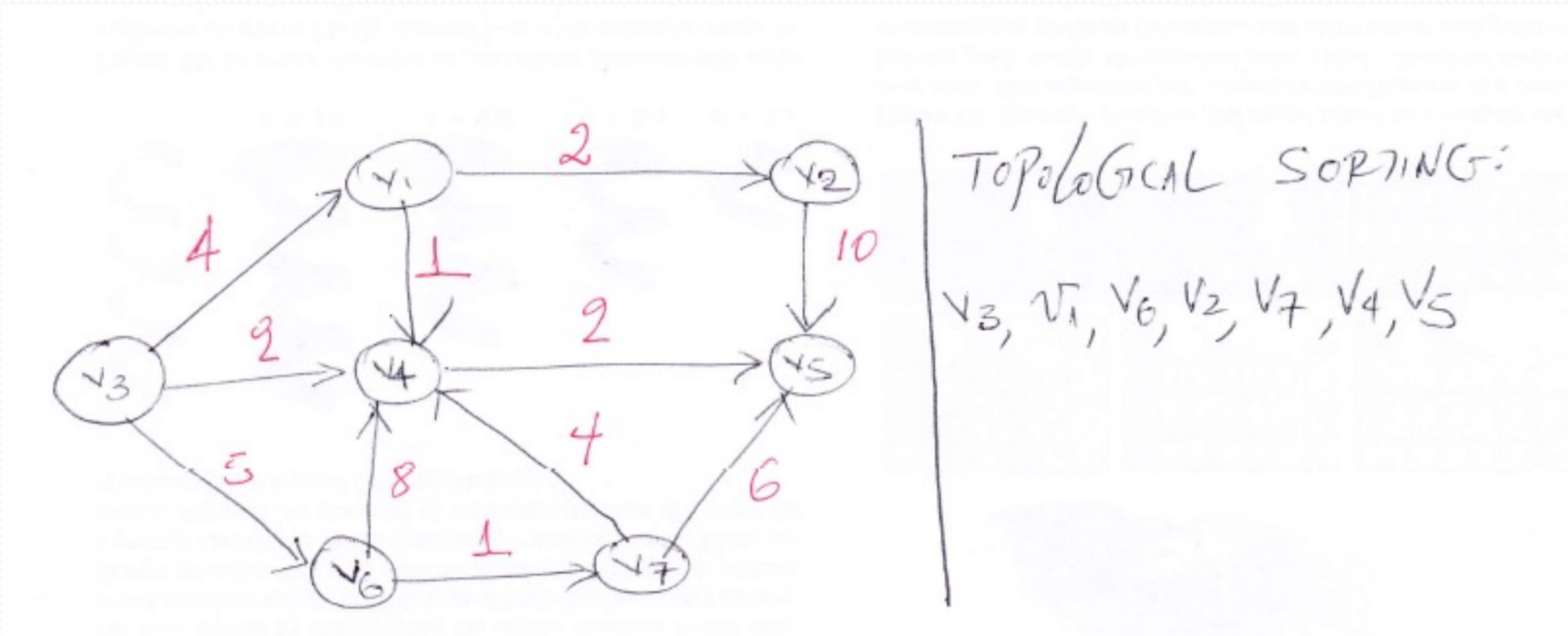
Vertex can be enqueued/dequeued multiple times

No priority queue

Acyclic graphs

- Improvement ?

Acyclic graph: example

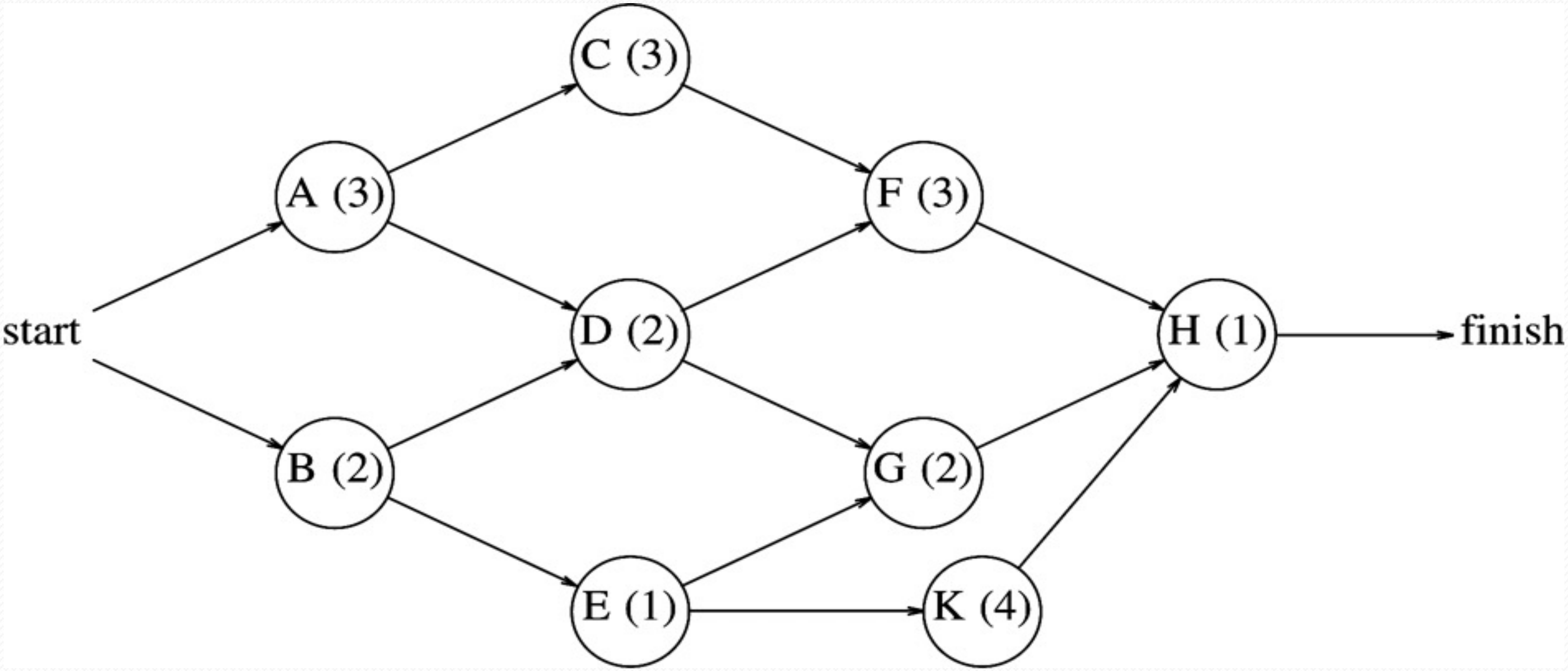


Acyclic graphs

- Dijkstra can be improved by selecting nodes in topological order
- Algorithm in one pass: select nodes and update distances

  => linear time $O(|V|+|E|)$

Activity-node graph



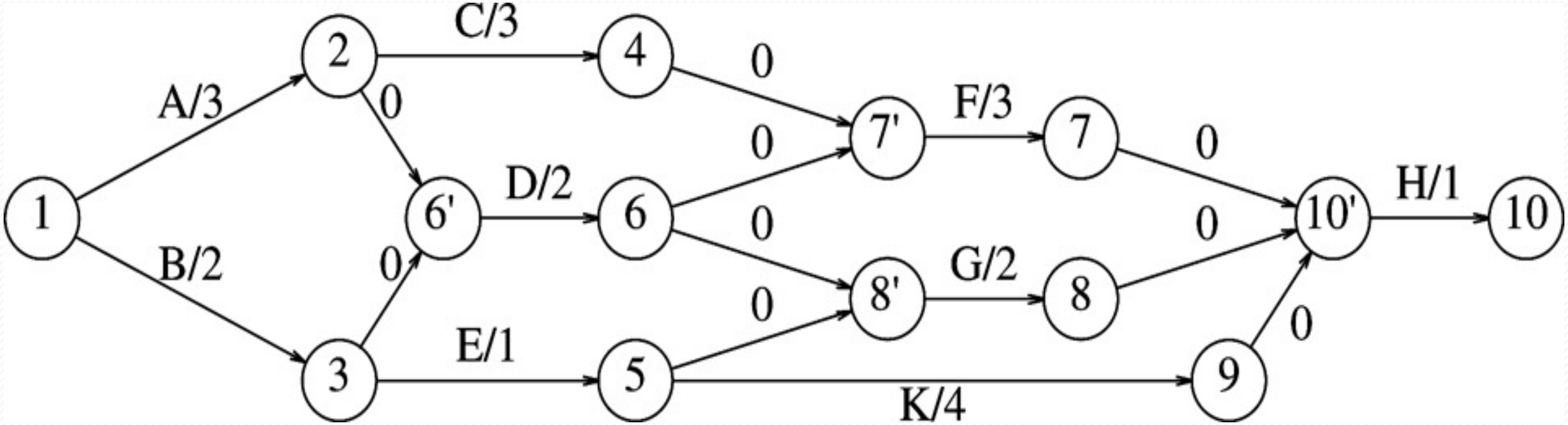
<u>Acyclic graph</u>

Nodes are tasks with their duration

In the above graph: D can start only after A and B are done, etc.

Can model construction projects etc.

Event-node graph



Edges: Task / Duration
Acyclic graph

Earliest-completion time

- Earliest-completion time of project = ?

Earliest-completion time

- Earliest-completion time of project =
  length of longest path from start to finish
- Be aware of positive-cost cycles
- Acyclic graphs do not pose such a problem

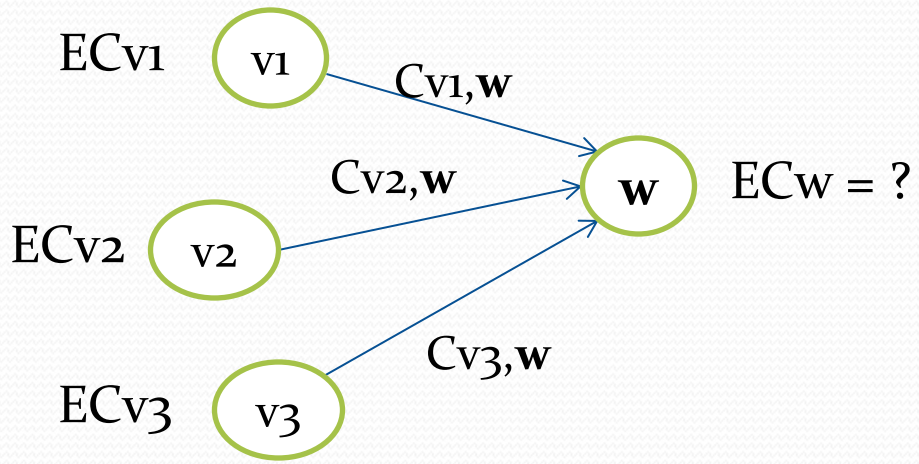
Earliest-completion time

EC_i : earliest completion time of node i

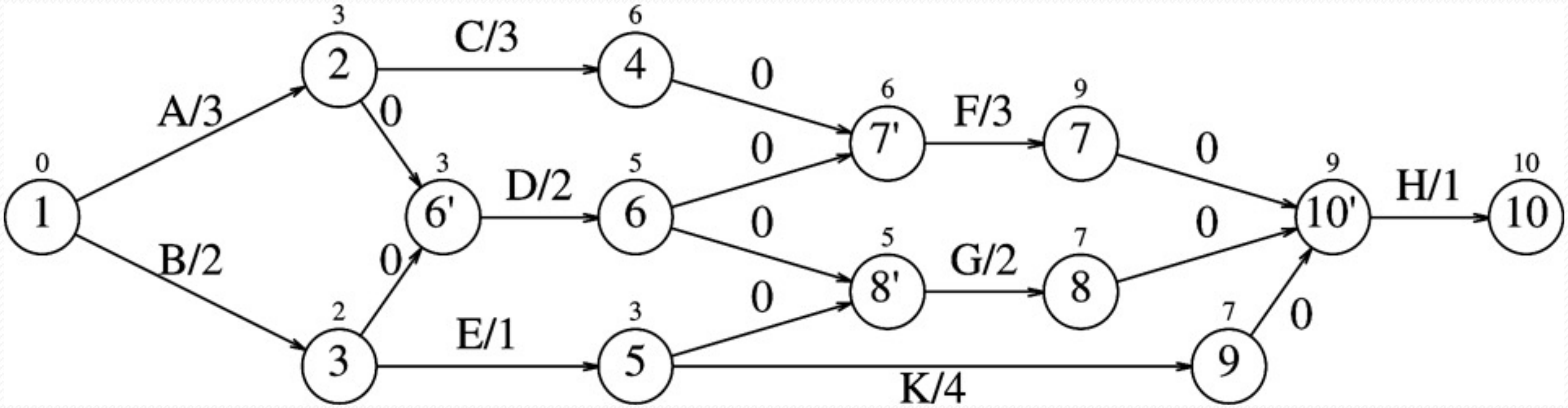
Then

$EC_1 = 0$

$EC_w = \max(EC_v + c_{v,w})$ over all (v,w) in E



Earliest completion time



Algorithm for acyclic graphs provides solution in linear time.

How?

Critical paths

- **Latest completion time** of each task:

  latest time a task can be accomplished without affecting final completion time.

Critical paths

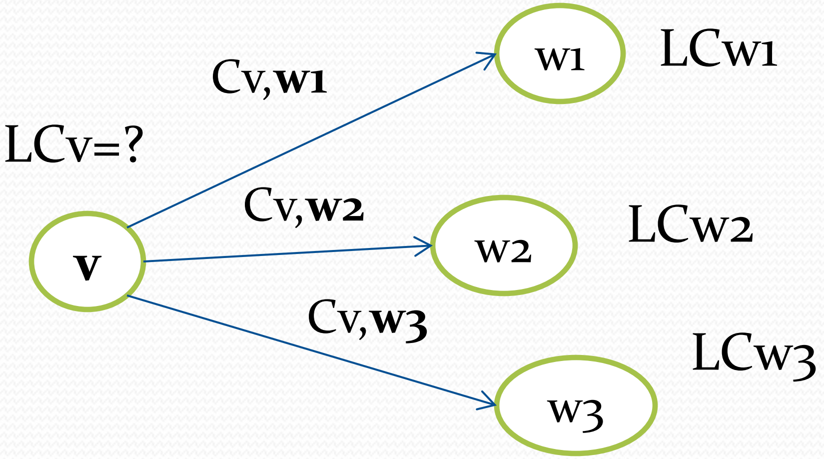
- **Latest completion** time of each task:

latest time a task can be accomplished without affecting final completion time.

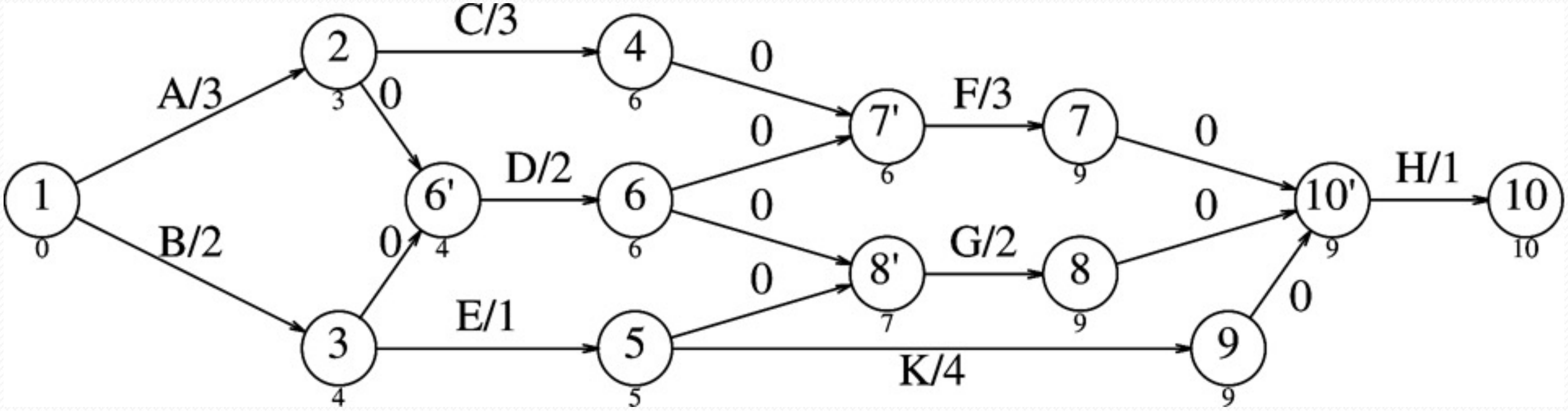
- Go from finish to start:

$LC_n = EC_n$ (last node)

$LC_v = \min(LC_w - c_{v,w})$ over all (v,w) in E



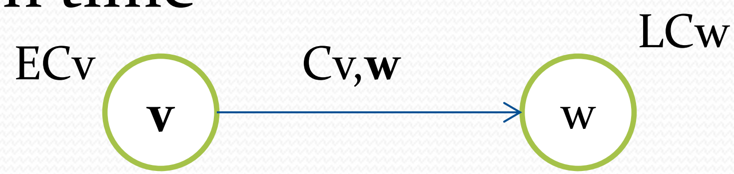
Latest completion times



Critical paths

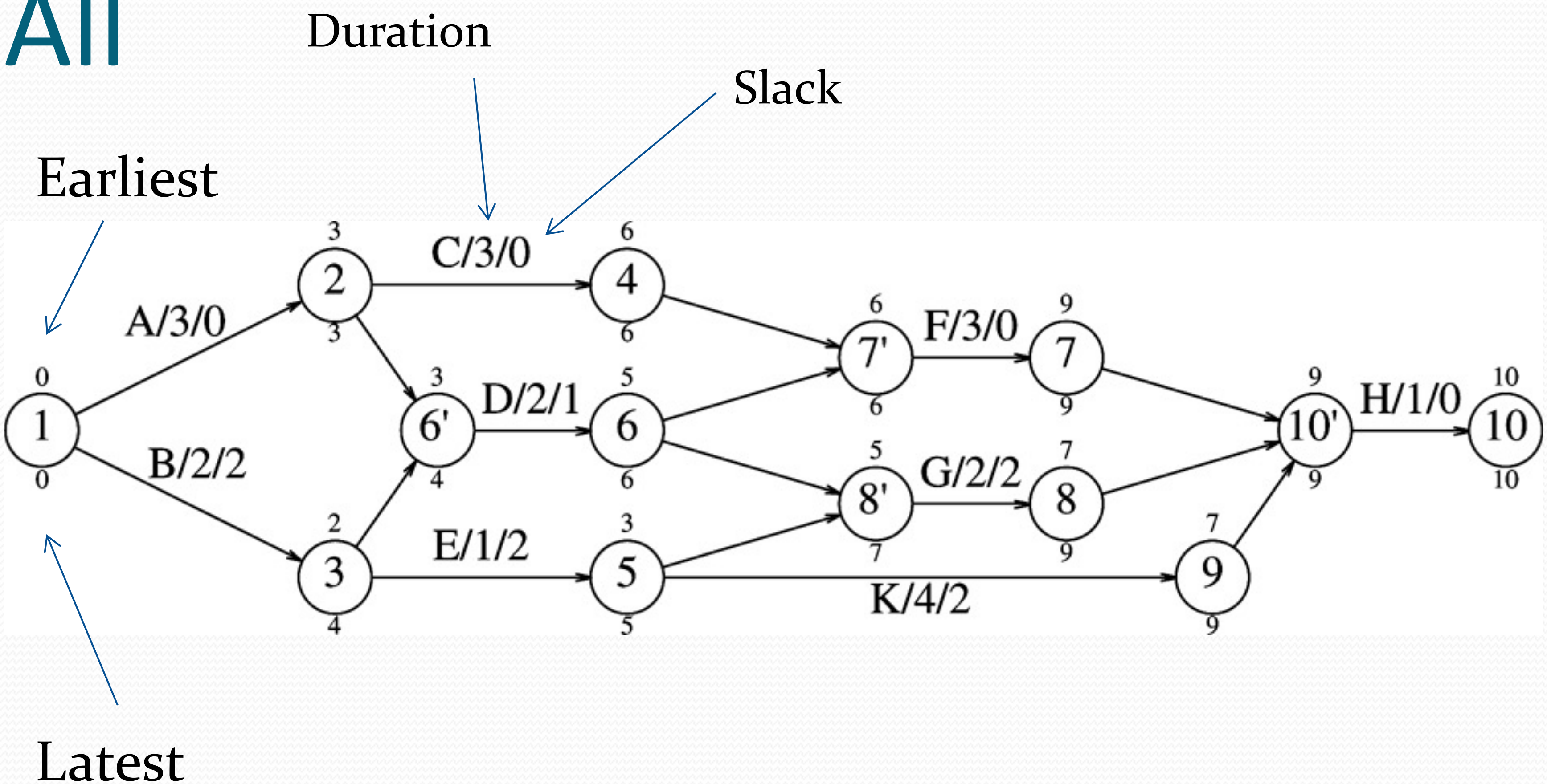
- Slack time of each edge: how much activity can be delayed without affecting completion time

$$Slack_{(v,w)} = LC_w - EC_v - c_{v,w}$$



- **Critical path:** Path from start->finish with zero-slack edges

All



Duration
Slack
Earliest
Latest
A/3/0
B/2/2
C/3/0
D/2/1
E/1/2
F/3/0
G/2/2
H/1/0
K/4/2
1
2
3
4
5
6'
6
7'
7
8'
8
9
10'
10