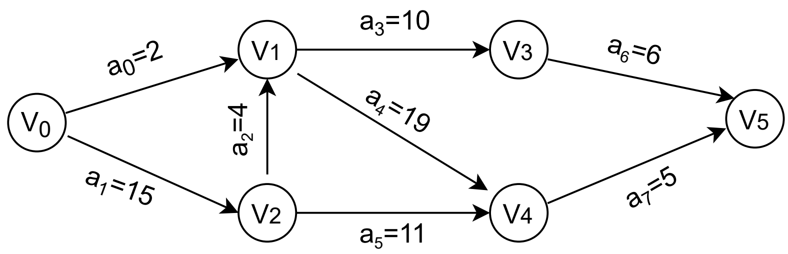
Critical Path

AOV Network reflects the relationship of before-after constraint among activities. In a real project, besides before-after order, activities also have a time of duration which it should go before finished.

On this situation, it needs another type of network--AOE(Activity On Edge), which edges represent the activities of network. AOE network is a weighted DAG(Directed Acyclic Graph),in which, its vertexes are events, its arcs are activites and its weights of arcs are the time of duration of activites.

For example, the following graph has 6 events and 8 activites. V_0 and V_5 are the source and sink respectively, V_1 could be starting only after a_0 and a_2 are finished. a_3 and a_4 could be starting after V_1 is finished.



In real application, **two problems** should be resolved.

1. evaluate the time the whole project spends
2. check which activities are critical ones that influence the progress of the project

Critical Path: the longest weighted path in an AOE network
Critical Activities: the activities in the critical path

How to resolve the critial path?

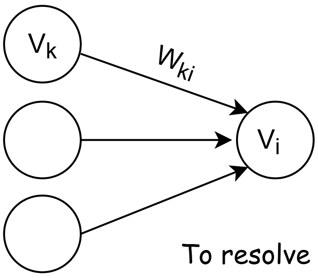
1. the earliest time of an event
2. the latest time of an event
3. the earliest time of an activity
4. the latest time of an activity

Critical Path

Earliest time of Event V_i: ve[i]

It's the longest path from the source to the event V_i because it should wait until all the before activities are finished. That's the activities coming into V_i should be finished before Event V_i starts. So, to resolve ve[i], let's start from the source, along the topologic order, move forward to Event V_i.

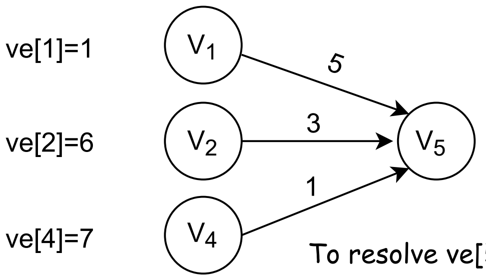
1. the earliest time of the source is 0, that's ve[0] = 0;
2. check the in-edges of V_i, the earliest time of V_i is relevant to the sum of the earliest time of the arc tail and its weights.



$$ve[i] = \max_k \{ve[k] + w_{ki}\}$$

To resolve ve[i], check in-edges of V_i

For example, in an AOE network, we have got ve[]s of V_1, V_2, V_4, it's easy to resolve ve of V_5



ve[5] = max {1+5, 6+3, 7+1} = 9

To resolve ve[5], check in-edges of V_5

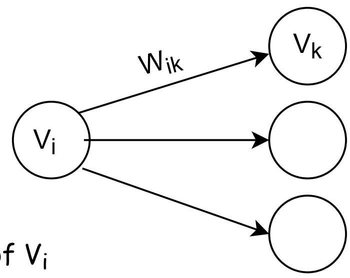
Critical Path

Latest time of Event V_i: vl[i]

The latest time of event V_i cannot delay the one of its successor. The lates time of event V_i cannot be greater than the difference between the one of its successor and the time of duration of the two events. To resolve vl[i], let's start from the sink, move backward to V_i along the topologic order.

1. initialize the latest time of the sink to its earliest time, that's vl[n-1] = ve[n-1]
2. check the out-going edges of V_i, the latest time of V_i is relevant to the differences between the latest time of the arc head and its weights.

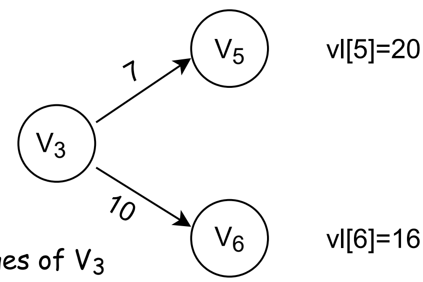
$$vl[i] = \min_k \{vl[k] - w_{ik}\}$$



To resolve vl[i], check out-going edges of V_i

For example, in an AOE network, we have got vl[]s of V_5, V_6, it's easy to resolve vl of V_3

vl[3] = max {20-7, 16-10} = 6

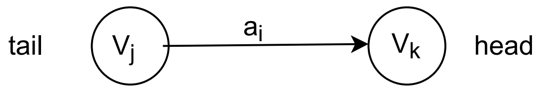


To resolve vl[3], check out-going edges of V_3

Critical Path

Earliest time of Activity a_i=<Vj, Vk>: e[i]

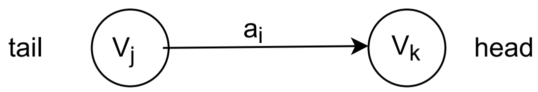
Once the event V_j happens, a_i can start. So, the earliest time of a_i is the earliest time of V_j. That's the earliest time of a_i is the earliest time of its arc tail.



e[i] = ve[j]

Latest time of Activity a_i=<Vj, Vk>: l[i]

The latest time of activity a_i cannot delay the one of V_k. So, the latest time of activity a_i is equal to the difference between the latest time of V_k and the time duration of a_i.



l[i] = vl[k]-w_{jk}

Critical Path

To resolve the critical path

1. resolve the topologic order of vertexes
2. resolve ve[i] along the topologic order
3. resolve vl[i] along the inversion of the order
4. resolve e[i] and l[i] and check whether it is a critical activity

Critical Activity

e[i] == l[i]