Advanced Programming

Graphs - Part2

Ramaseshan Ramachandran

OVERVIEW

- $G = (V, E)$
- Set of nodes V Set of edges E
- E is a subset of pairs (u, v):
- Undirected graph: The edges (u, v) and (v, u) are the same
- Directed graph:
- (u, v) is an edge from u to v. It is does not represent the edge (v, u)

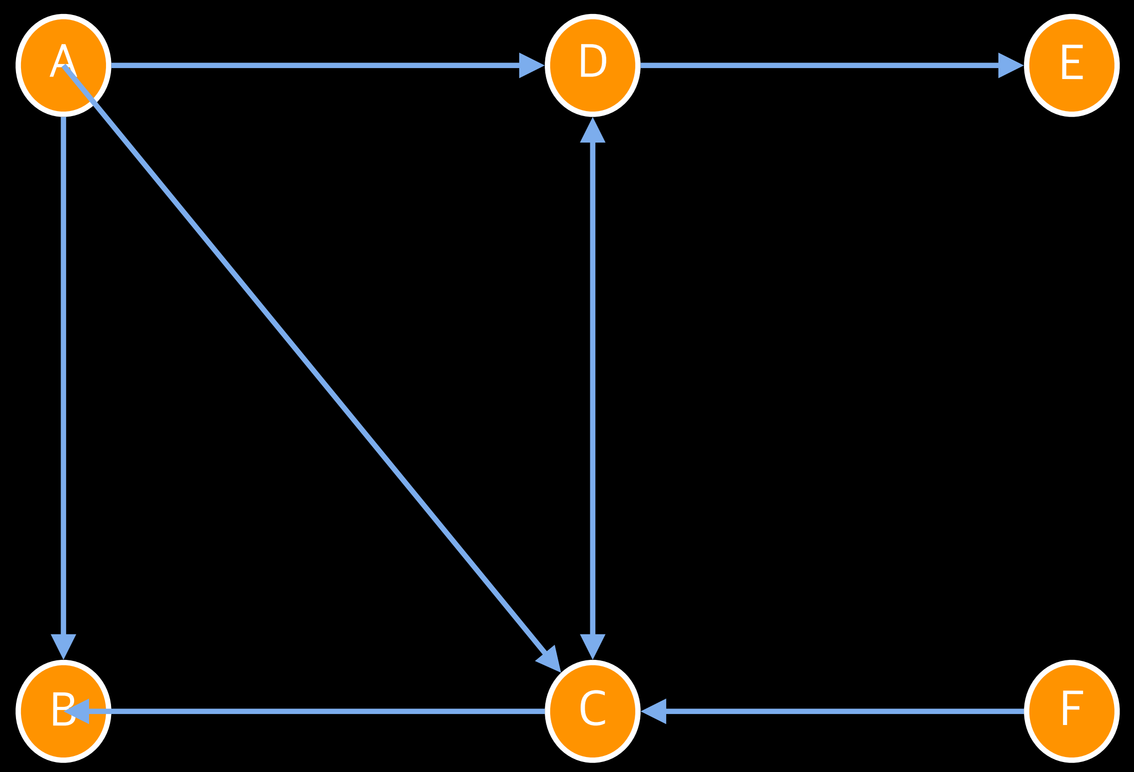
- BFS
  - Explore nodes in the ascending order of levels
- DFS
  - Explore nodes as soon as they are visited
  - Directed graph:
- (u, v) is an edge from u to v. It is does not represent the edge (v, u)

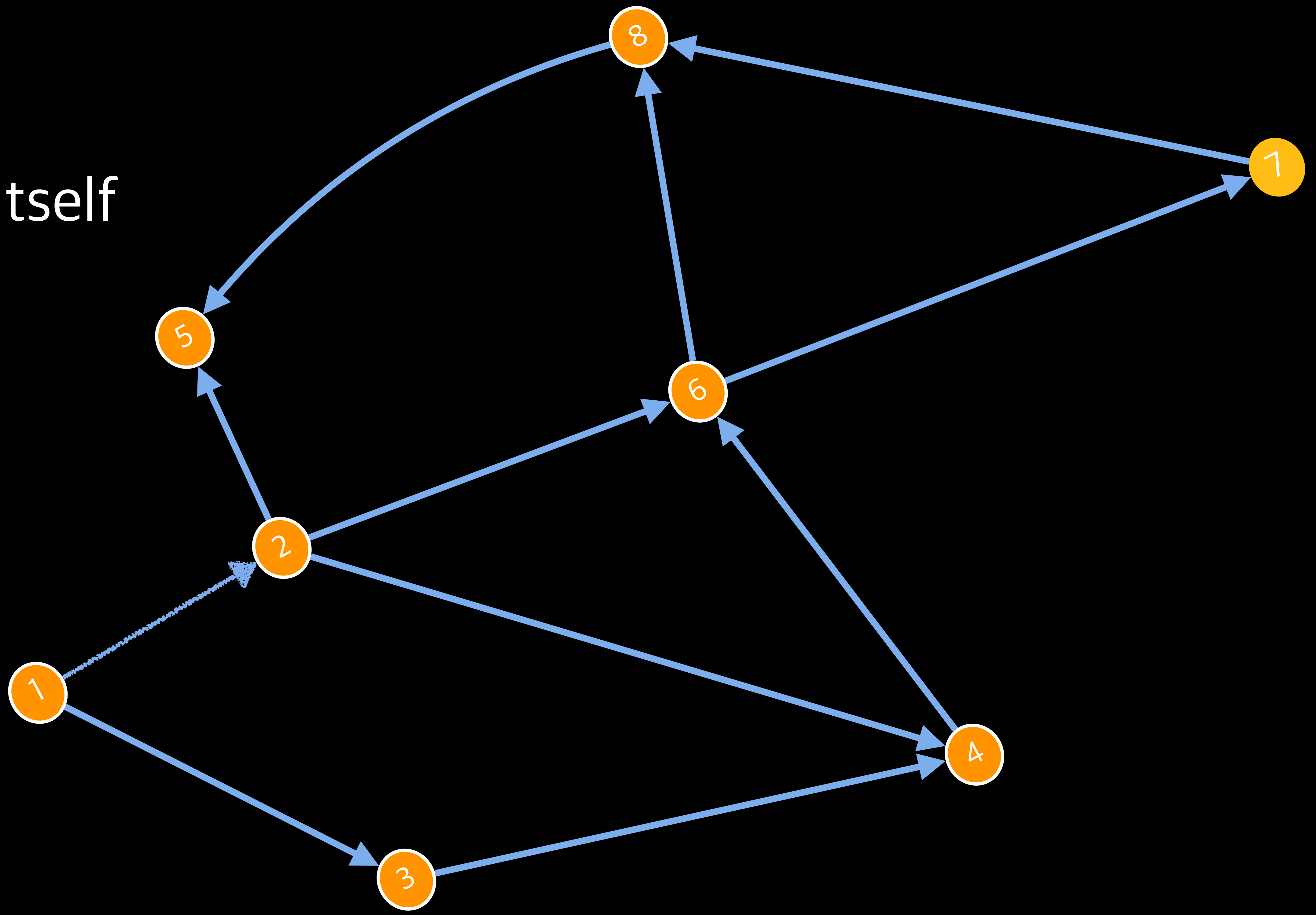
COMPLEXITY OF DFS

- Each vertex marked and explored exactly once
- DFS(j) need to examine all neighbours of j
- If there are n entries in the adjacency matrix, every row corresponding to a node has to be examined
- Overall $O(n^2)$
- With adjacency list, scanning takes $O(m)$ time across all vertices
- Total time is $O(m + n)$

DIRECTED ACYCLIC GRAPHS

- Let $G = (V, E)$ a directed graph
- No cycles
  - No directed path from any v in V back to itself



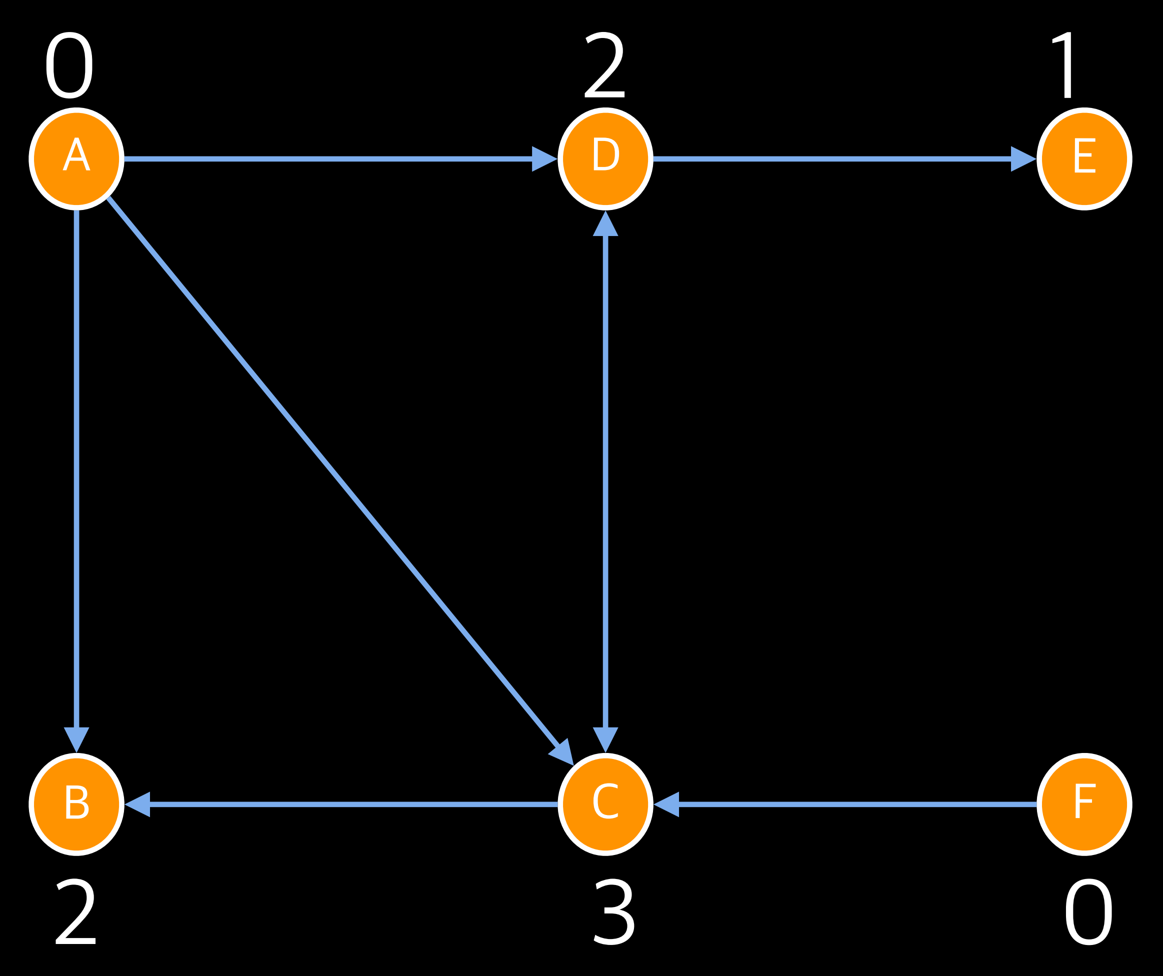


TOPOLOGICAL ORDERING

Given a Directed Acyclic Graph (DAG), the problem of topological sorting involves finding an ordering of the vertices in such a way that all edges go forward in the ordering

If we start with the node set $\{1,2,\ldots,n\}$ and include an edge (i,j) whenever $i < j$, then the resulting directed graph has $\binom{n}{2}$ edges but no cycles

TOPOLOGICAL ORDERING



- $indegree(v)$: number of edges into v
- $outdegree(v)$: number of edges out of v
- Every DAG has at least one vertex with $indegree = 0$
- Start with any v such that $indegree(v) > 0$
- Walk backwards to a predecessor so long as $indegree(v) > 0$
- If no vertex has $indegree = 0$, within n steps we will complete a cycle!
- There should be no cycles
  - No directed path from any v in V back to itself
- Topological order can be non-unique - more than one topological order possible

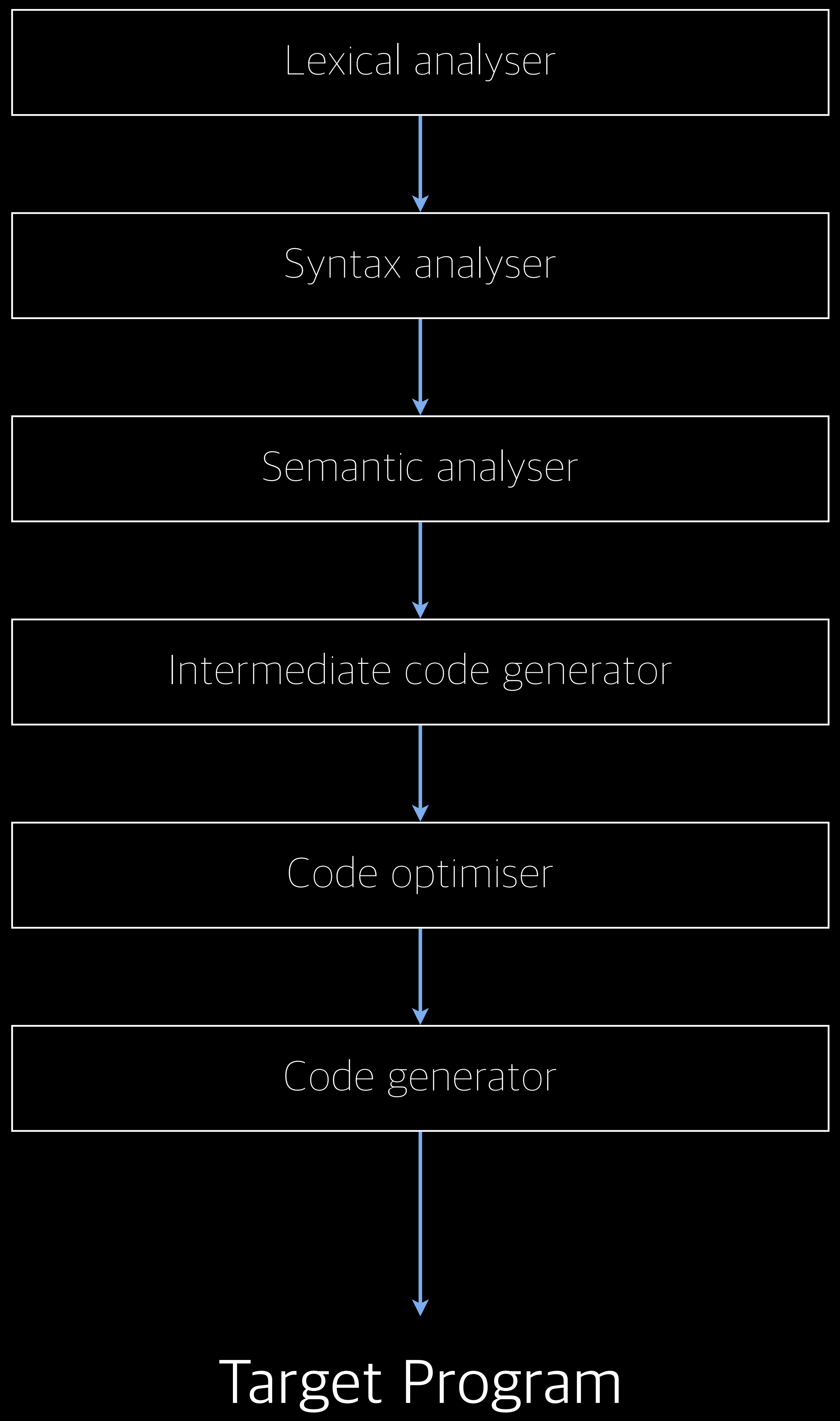
TOPOLOGICAL ORDERING

- Where do we start?
- Pick a vertex with indegree = 0 – no dependencies
- Enumerate it and delete from graph
    - The dependent vertices indegree will be reduced by 1
- Repeat the step until there is no nodes left and DAG is empty

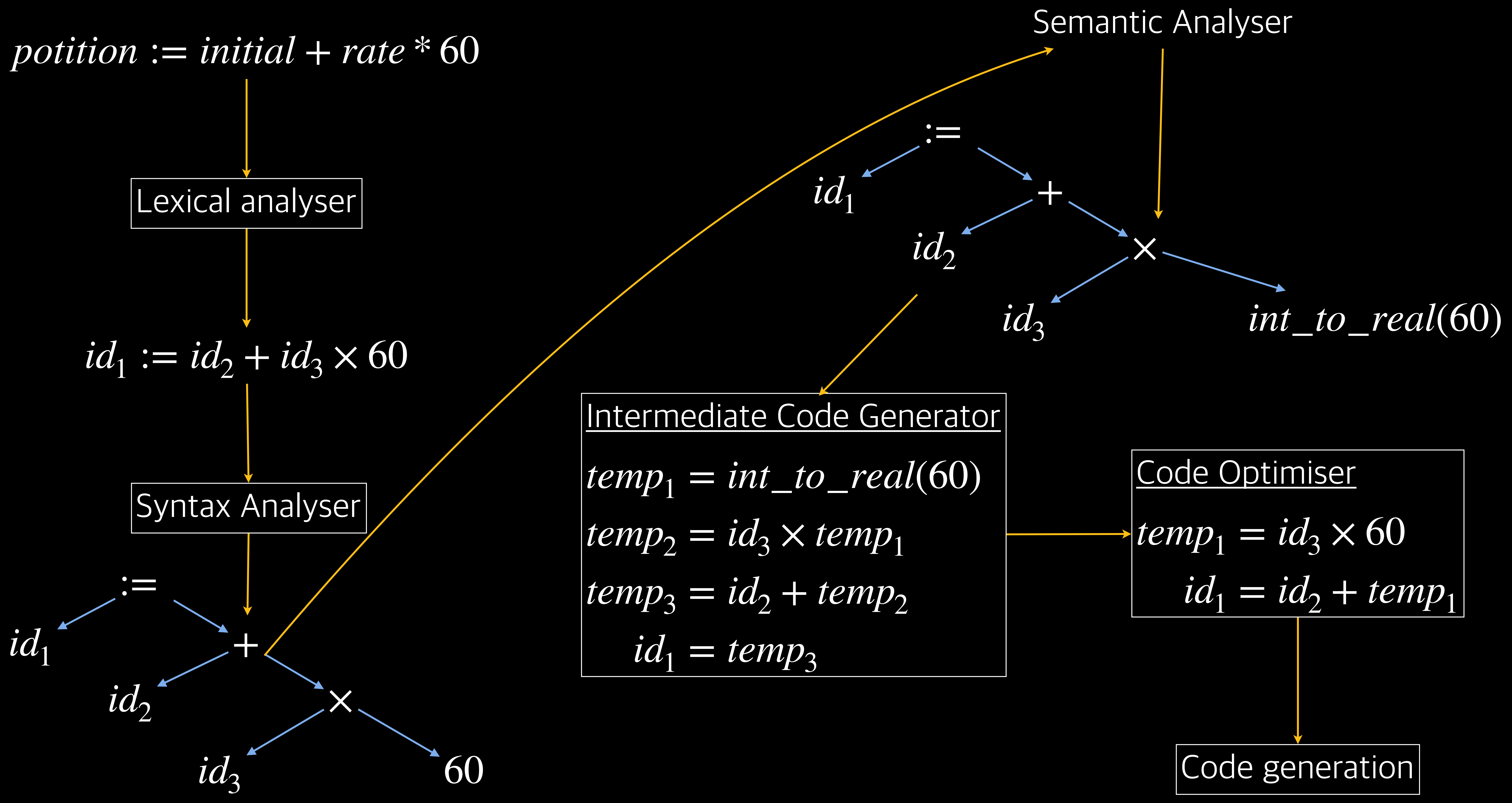
TOPOLOGICAL ORDERING - EXAMPLES

- Courses, with prerequisite requirements
- Computing jobs – A pipeline of computing jobs with dependencies

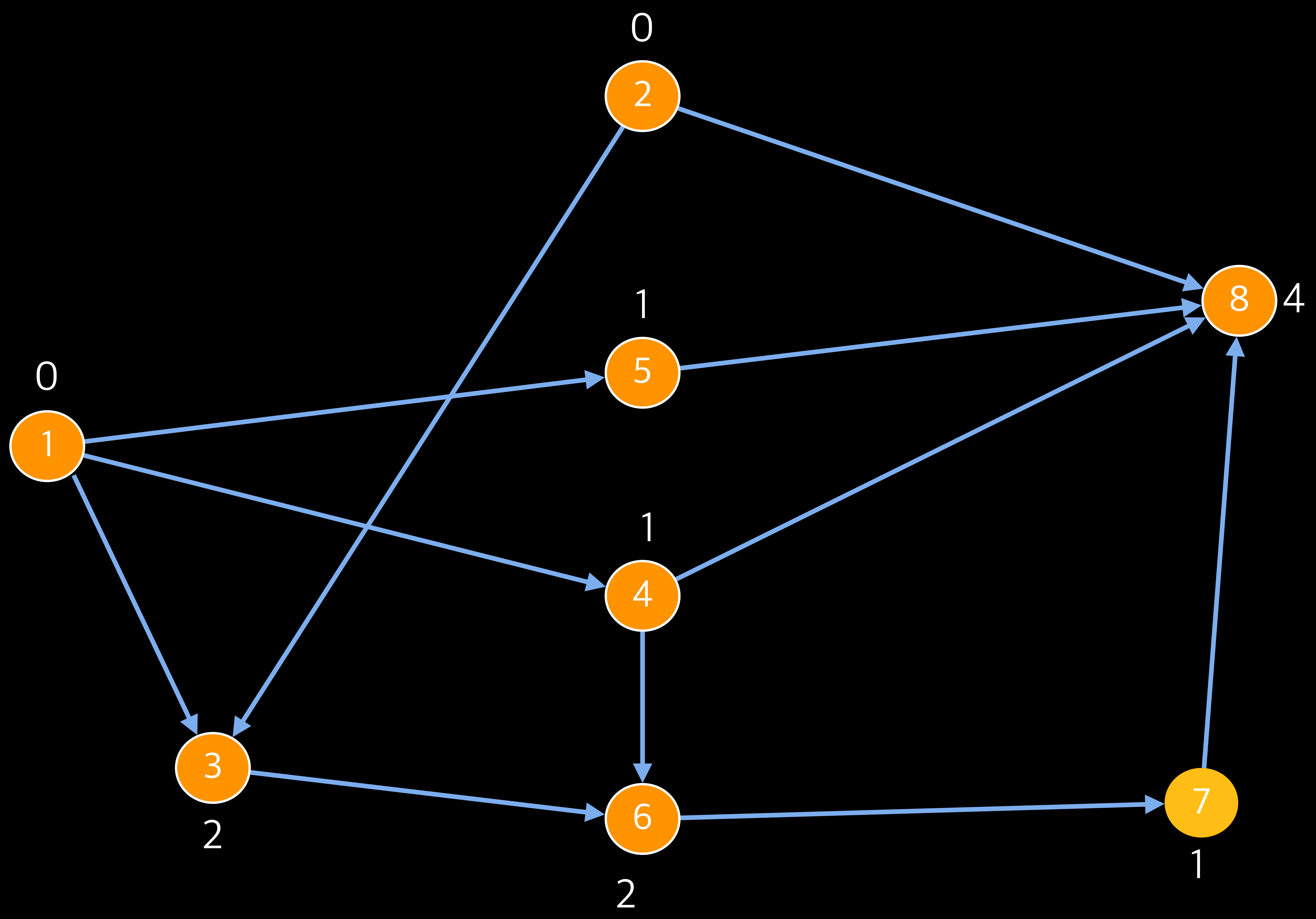
PHASES OF A COMPILER



TRANSLATION OF A STATEMENT

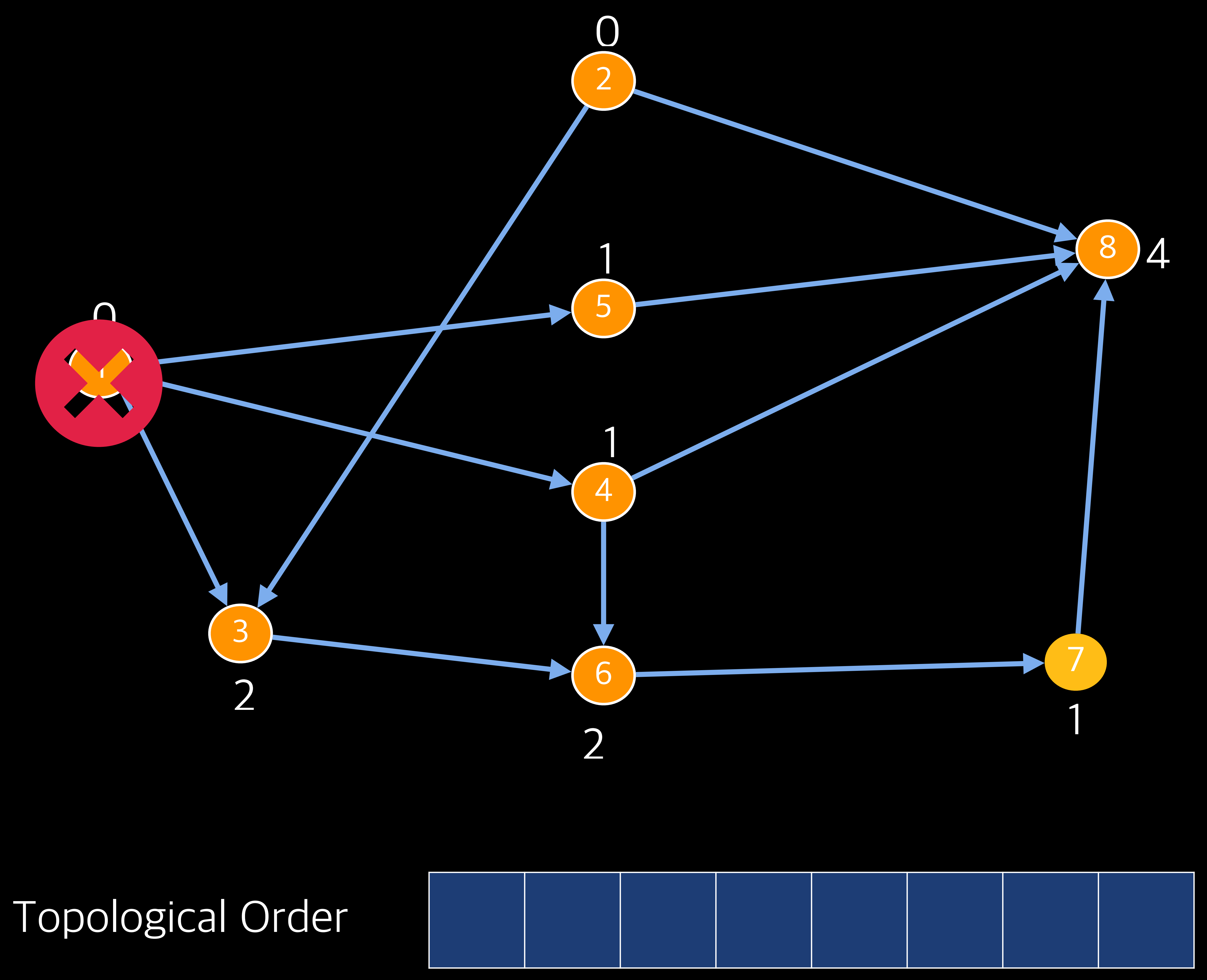


TOPOLOGICAL ORDERING

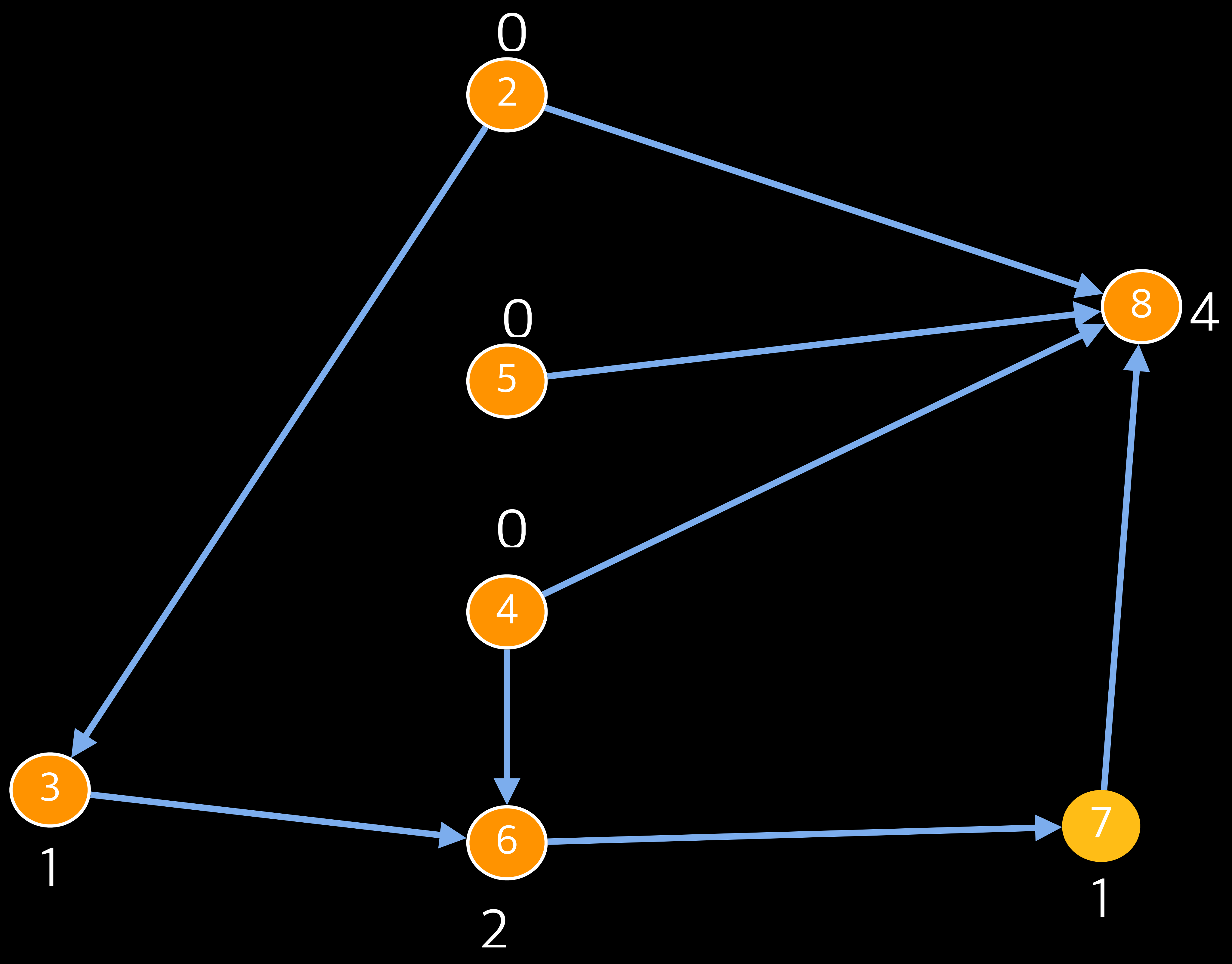


In every DAG G, there is at least one node v with no incoming edges

TOPOLOGICAL ORDERING

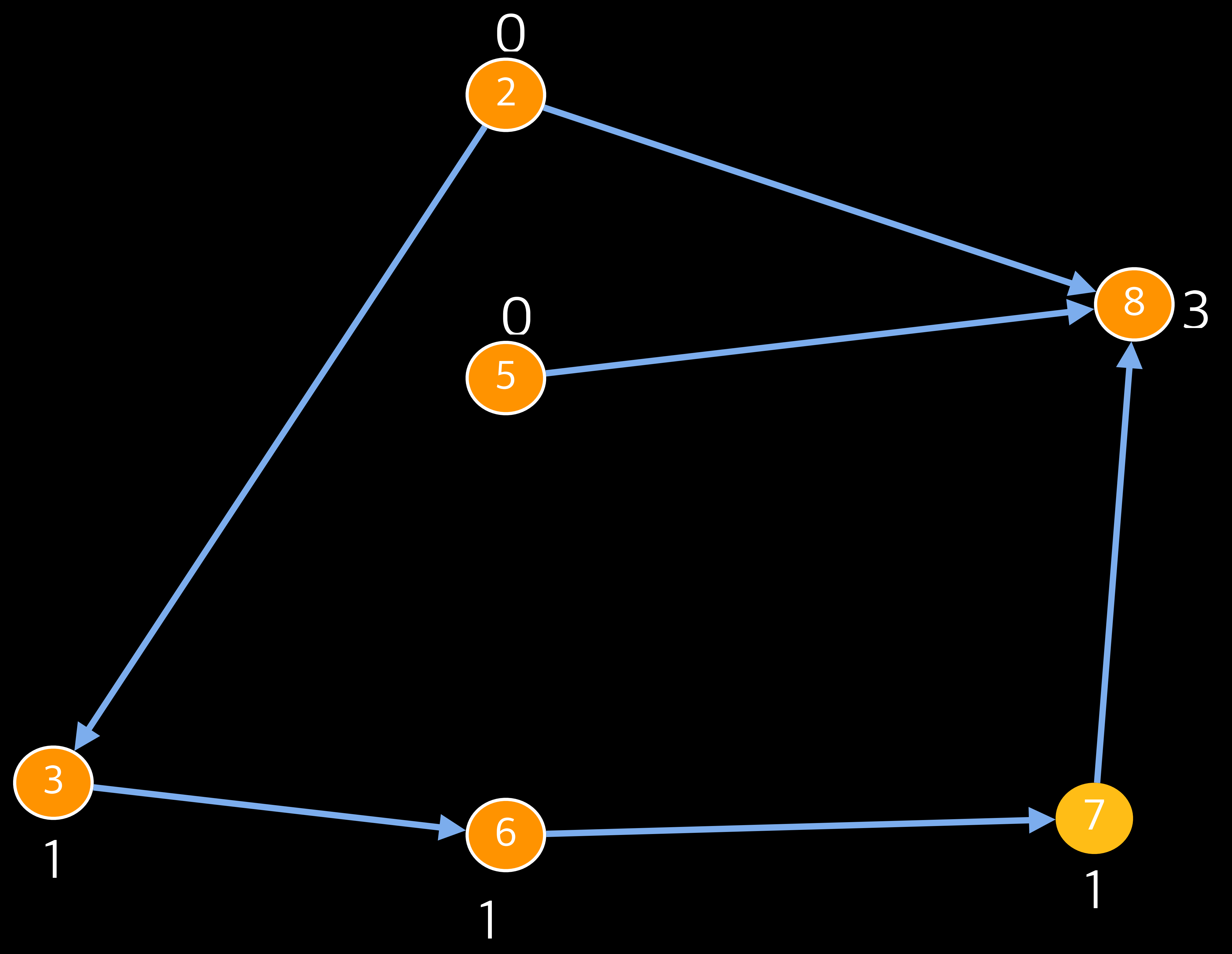


TOPOLOGICAL ORDERING



Topological Order	1							

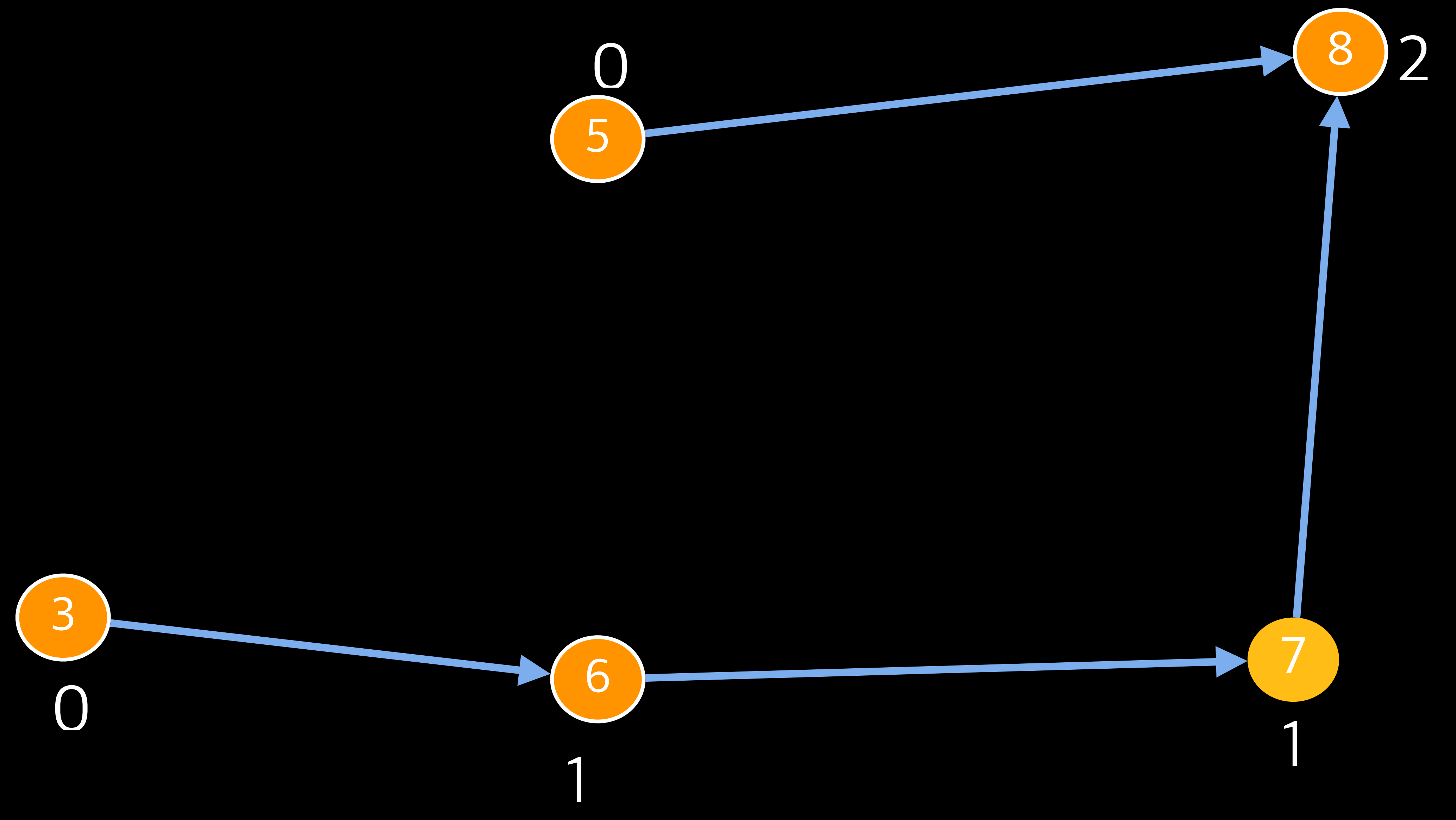
TOPOLOGICAL ORDERING



Topological Order

1	4						

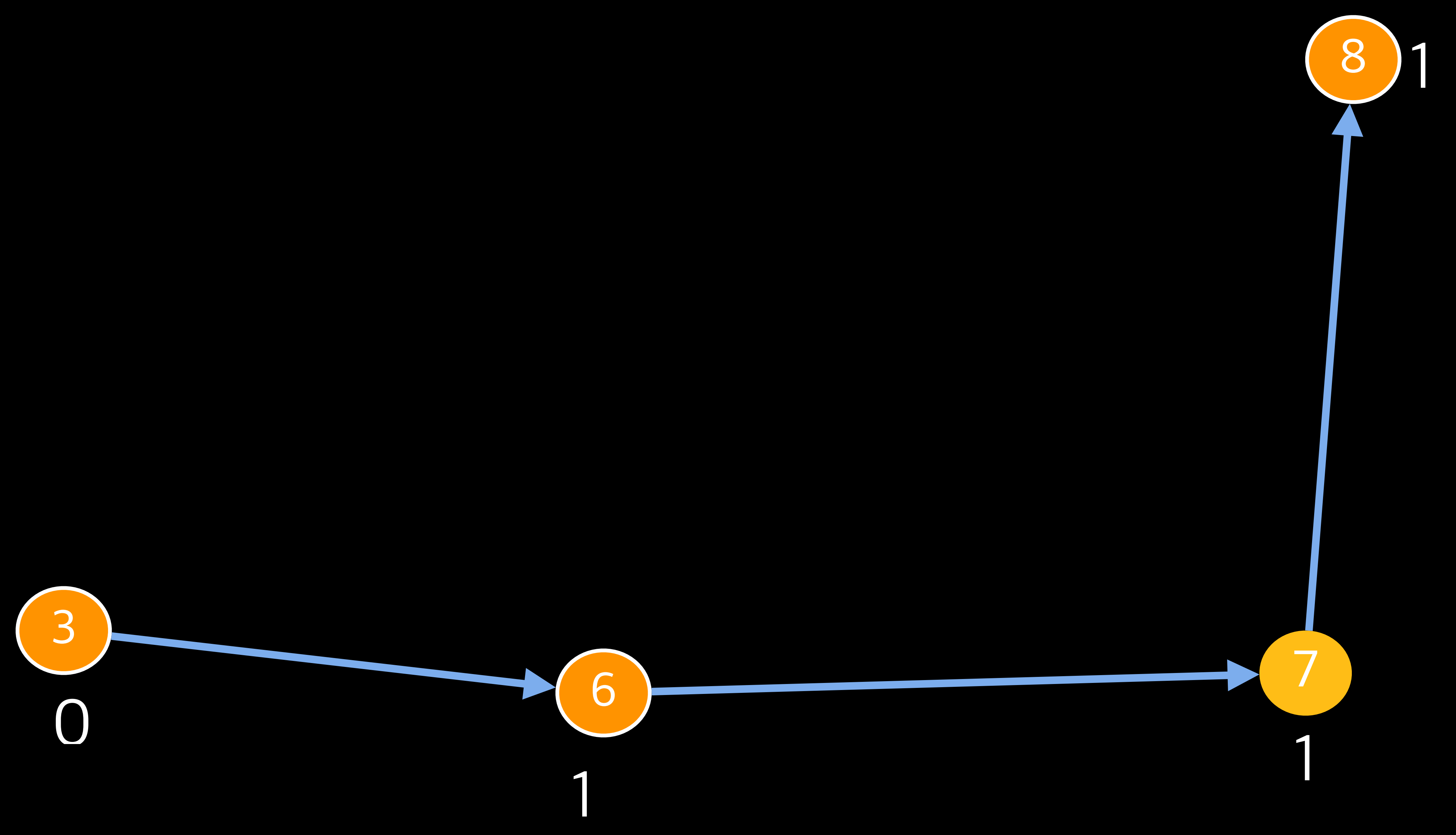
TOPOLOGICAL ORDERING



1	4	2					

Topological Order

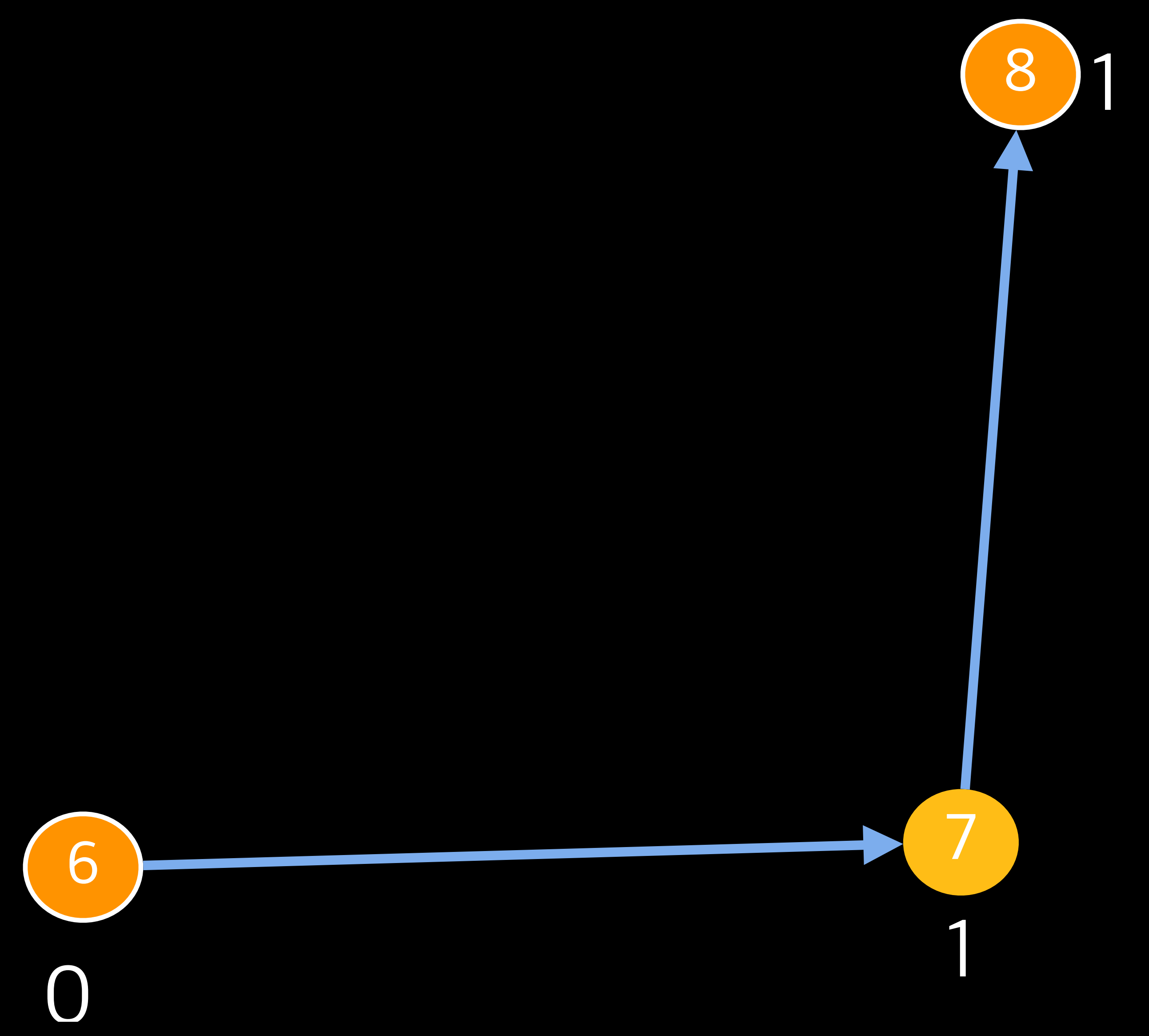
TOPOLOGICAL ORDERING



Topological Order

1	4	2	5				

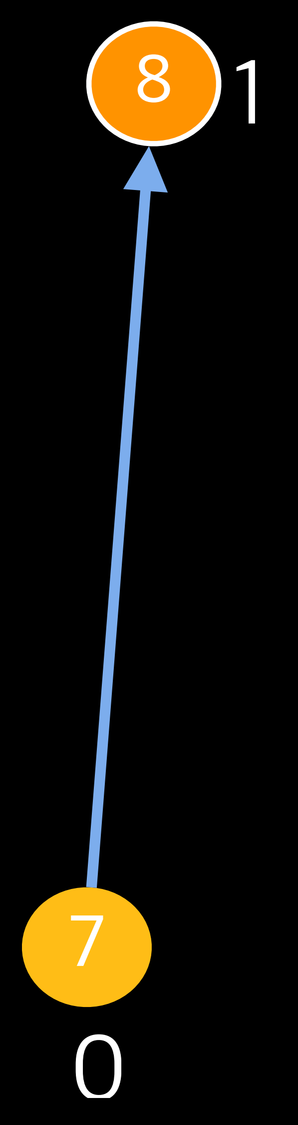
TOPOLOGICAL ORDERING



Topological Order

1	4	2	5	3			

TOPOLOGICAL ORDERING



Topological Order								
	1	4	2	5	3	6		

TOPOLOGICAL ORDERING

Topological Order	1	4	2	5	3	6	7	

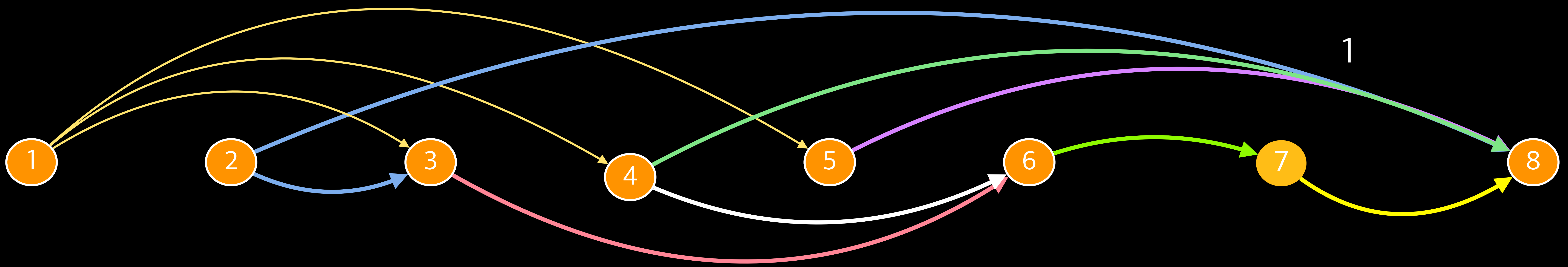
TOPOLOGICAL ORDERING

If G has a topological ordering, then G is a DAG.

Topological Order	1	4	2	5	3	6	7	8

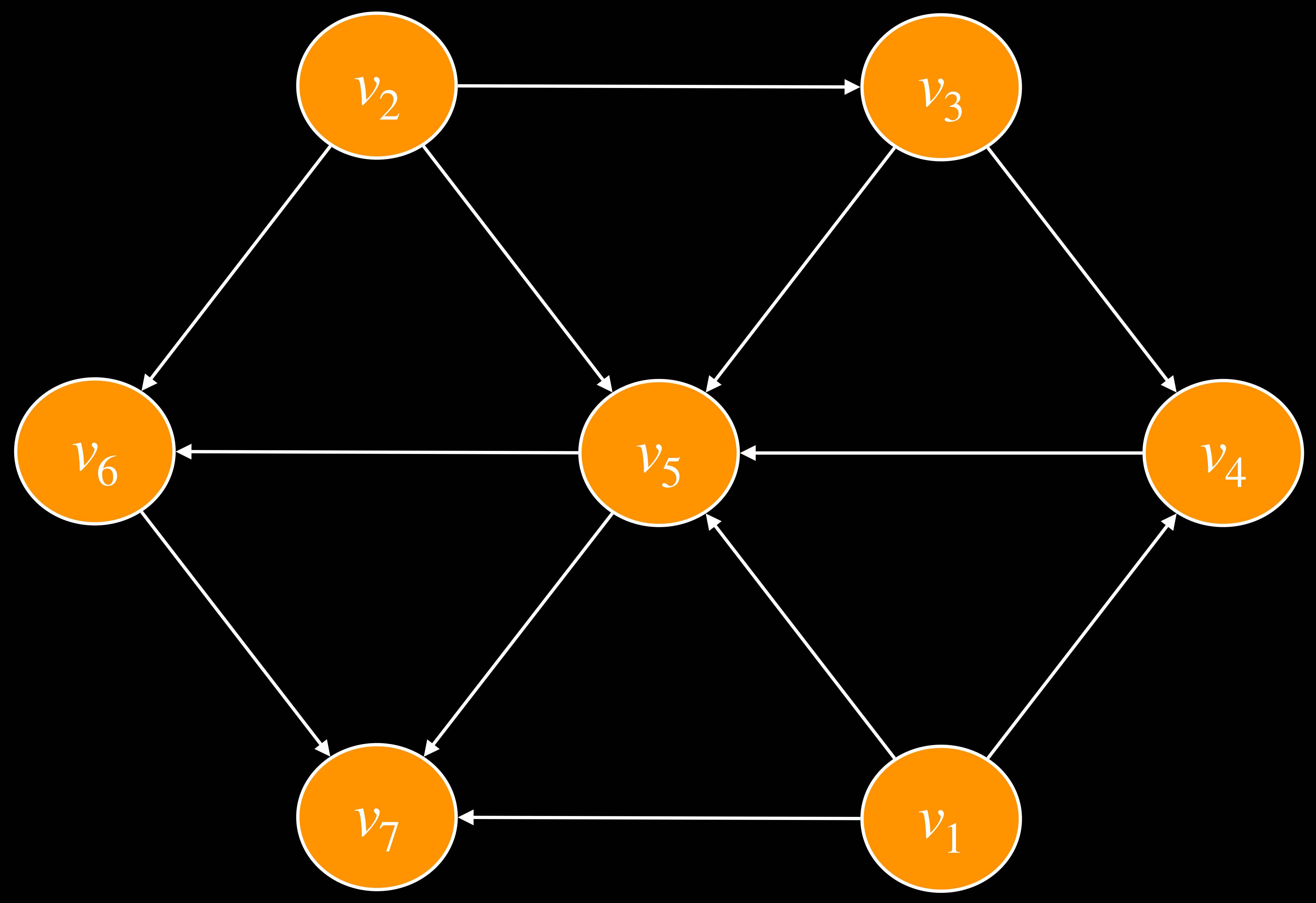
TOPOLOGICAL ORDERING

DAGs encode precedence relations or dependencies in a natural way



All edges point from left to right

TOPOLOGICAL ORDERING - EXERCISE

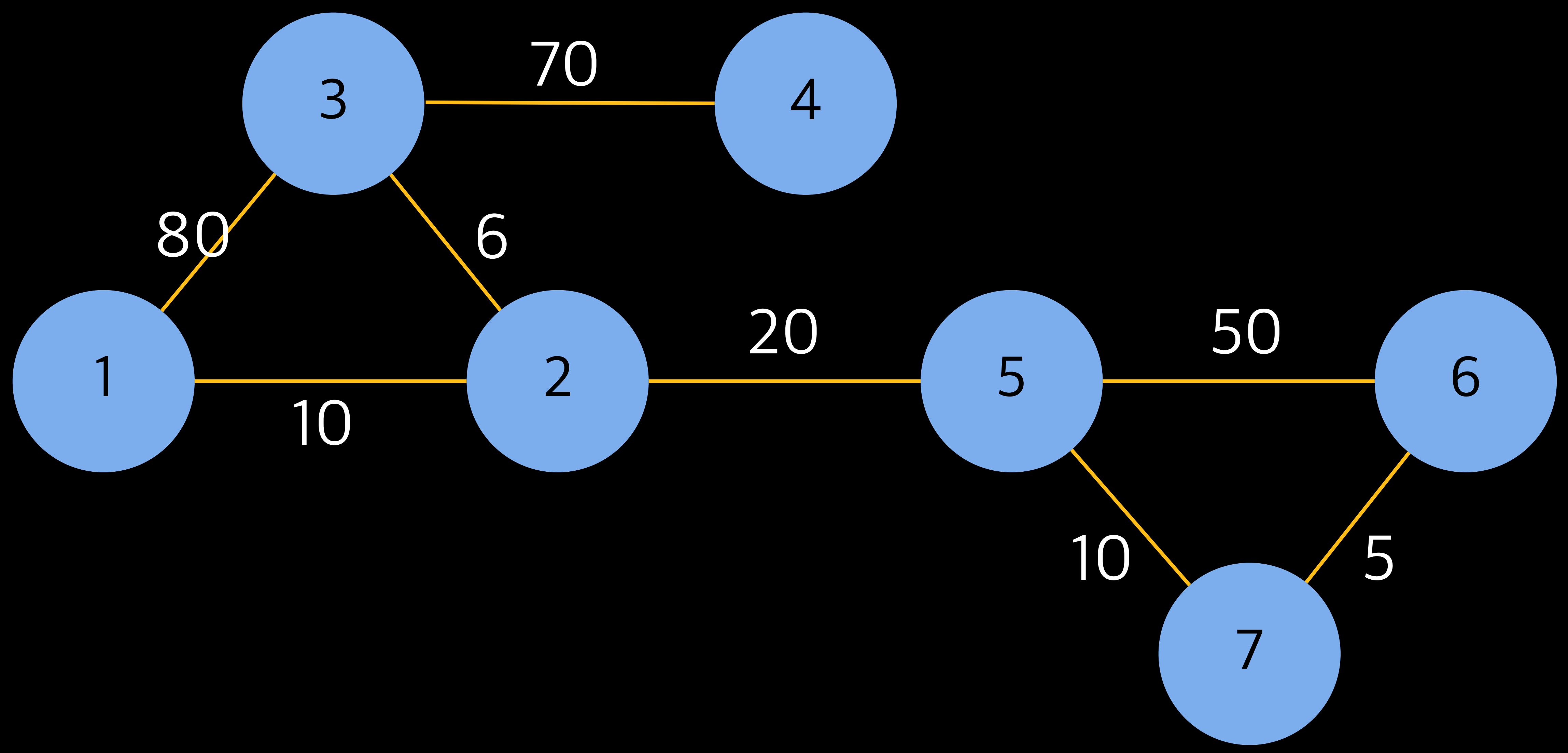


WEIGHTED GRAPH

- $G = (V, E)$ together with a weight function, $e : E \rightarrow Real$
- Let $e_1 = (v_0, v_1), e_2 = (v_1, v_2), \ldots, e_n = (v_{n-1}, v_n)$ be a path from v_0 to v_n
- Cost of the path is $w(e_1) + w(e_2) + \ldots + w(e_n)$
- Shortest path from v_0 to v_n will have minimum cost

- BFS finds path with fewest number of edges
- In a weighted graph, need not be the shortest path

WEIGHTED GRAPH - BFS COST



BFA - Shortest path: $1 \rightarrow 2 \rightarrow 5 \rightarrow 6$

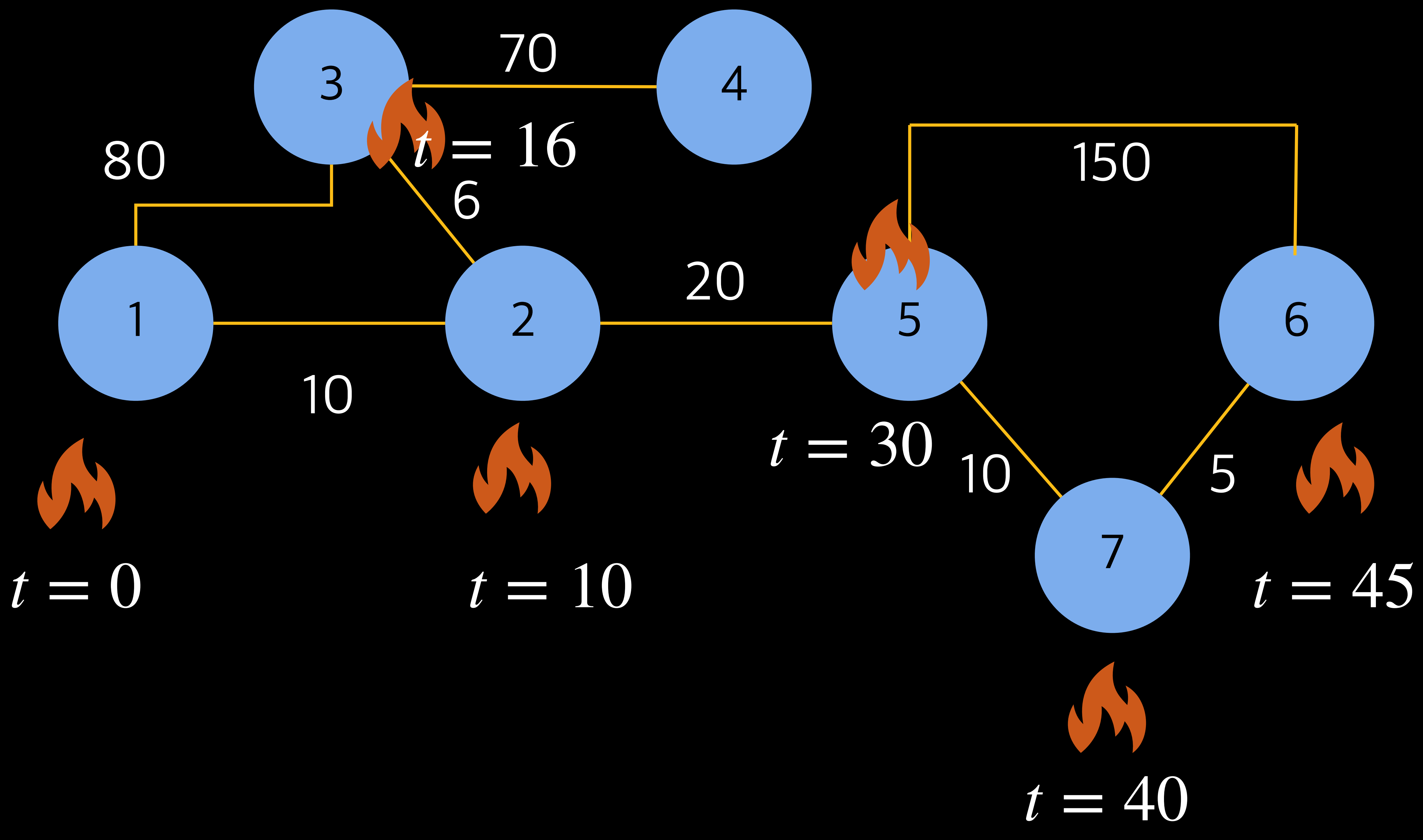
Cost = 80

BFA - Another path: $1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 6$

Cost = 35

SINGLE SOURCE SHORTEST PATH

- Imagine vertices are oil depots, edges are pipelines
- Set fire to oil depot at vertex 1
- Fire travels at uniform speed along each pipeline
- First oil depot to catch fire after 1 is nearest vertex Next oil depot is second nearest vertex



PSEUDO CODE

Maintain two arrays

$BurntVertices[]$, initially $False$ for all i

$ExpectedBurnTime[]$, initially ∞ for all i

Instead of ∞, you may use $\sum_i w(e_i) + 1$

$ExpectedBurnTime[1] \leftarrow 0$

Repeat until all vertices are burnt

Find j with $min(ExpectedBurnTime)$

$BurntVertices[j] \rightarrow True$

Recompute $ExpectedBurnTime[k]$ for each neighbour k of j

DIJKSTRA'S ALGORITHM

```
function ShortestPaths(s)
for  i = 1 to n do
    Visited[i] ←False
    Distance[i] ←infinity
end for
Distance[s] ←0
for i = 1ton do
    Choose u such that Visited[u] == False and Distance[u] is minimum
    Visited[u] ←True
    for all (u, v) and Visited[v] == False do
        if Distance[v] > Distance[u] + weight(u, v) then
            Distance[v] ← Distance[u] + weight(u, v)
        end if
    end for
end for
```