NPTEL MOOC,JAN-FEB 2015
Week 3, Module 7

DESIGN AND ANALYSIS OF ALGORITHMS

DAGs: Longest paths

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Directed Acyclic Graphs

- G = (V,E), a directed graph
- No cycles
  - No directed path from any v in V back to itself
- Such graphs are also called DAGs

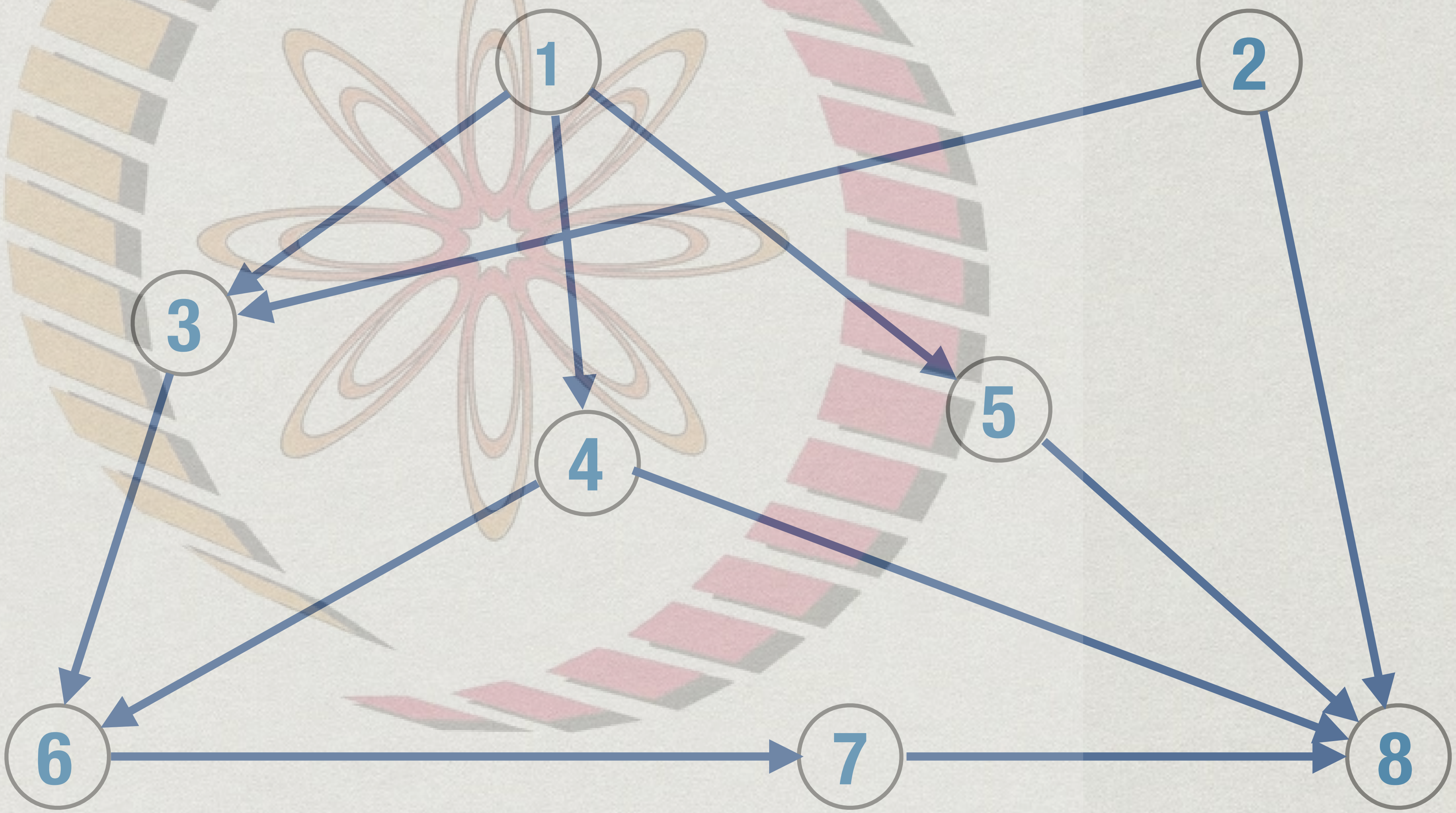
Topological ordering

- Given a DAG $G = (V,E)$, $V = \{1,2,\ldots,n\}$
- Enumerate the vertices as $\{i_1,i_2,\ldots,i_n\}$ so that
  - For any edge (j,k) in E,

    j appears before k in the enumeration
- Also known as **topological sorting**

Questions about DAGs

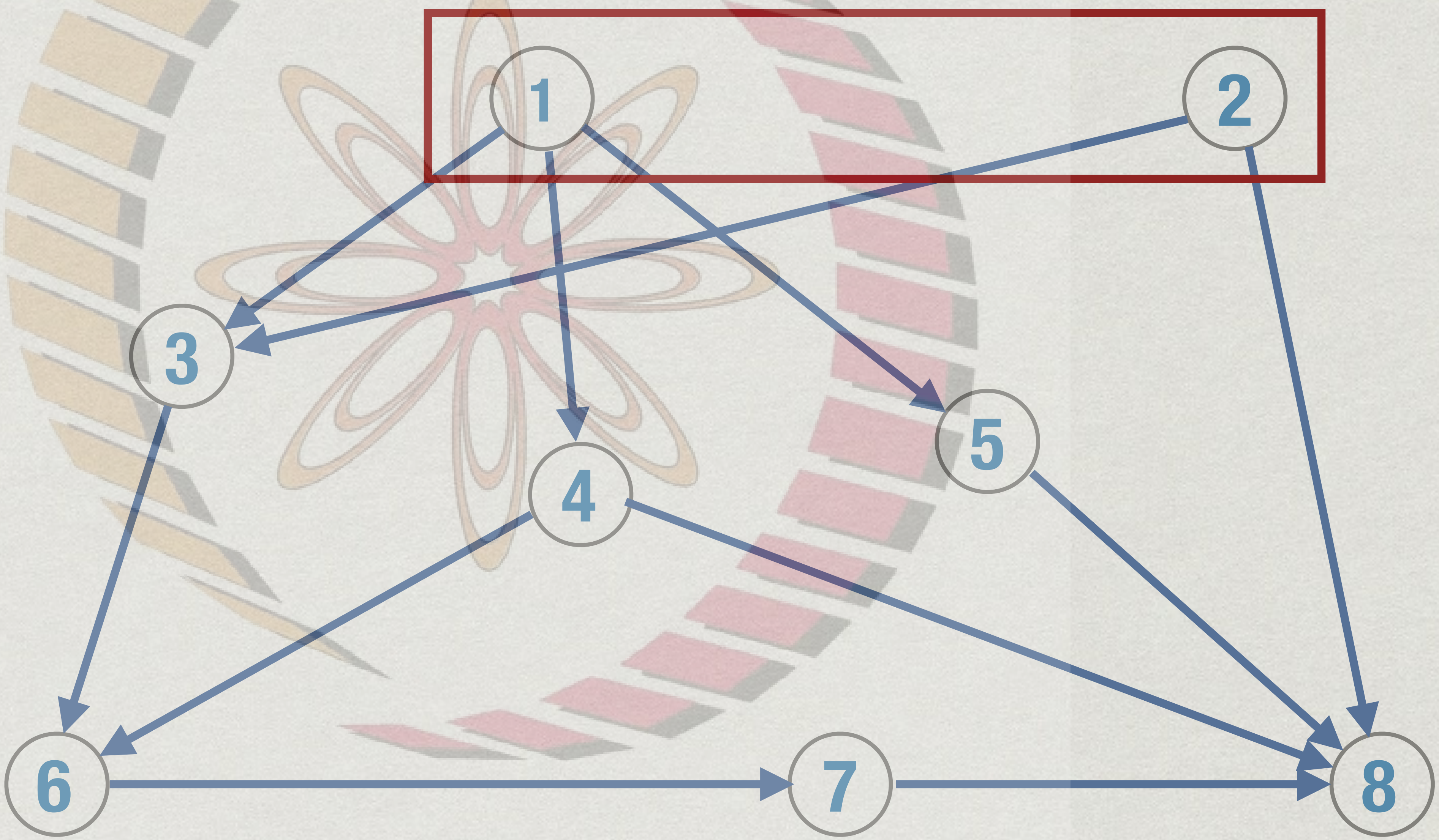
- Imagine these are courses
- Edges are pre-requisites



- What is the minimum number of semesters to complete the programme?

Questions about DAGs

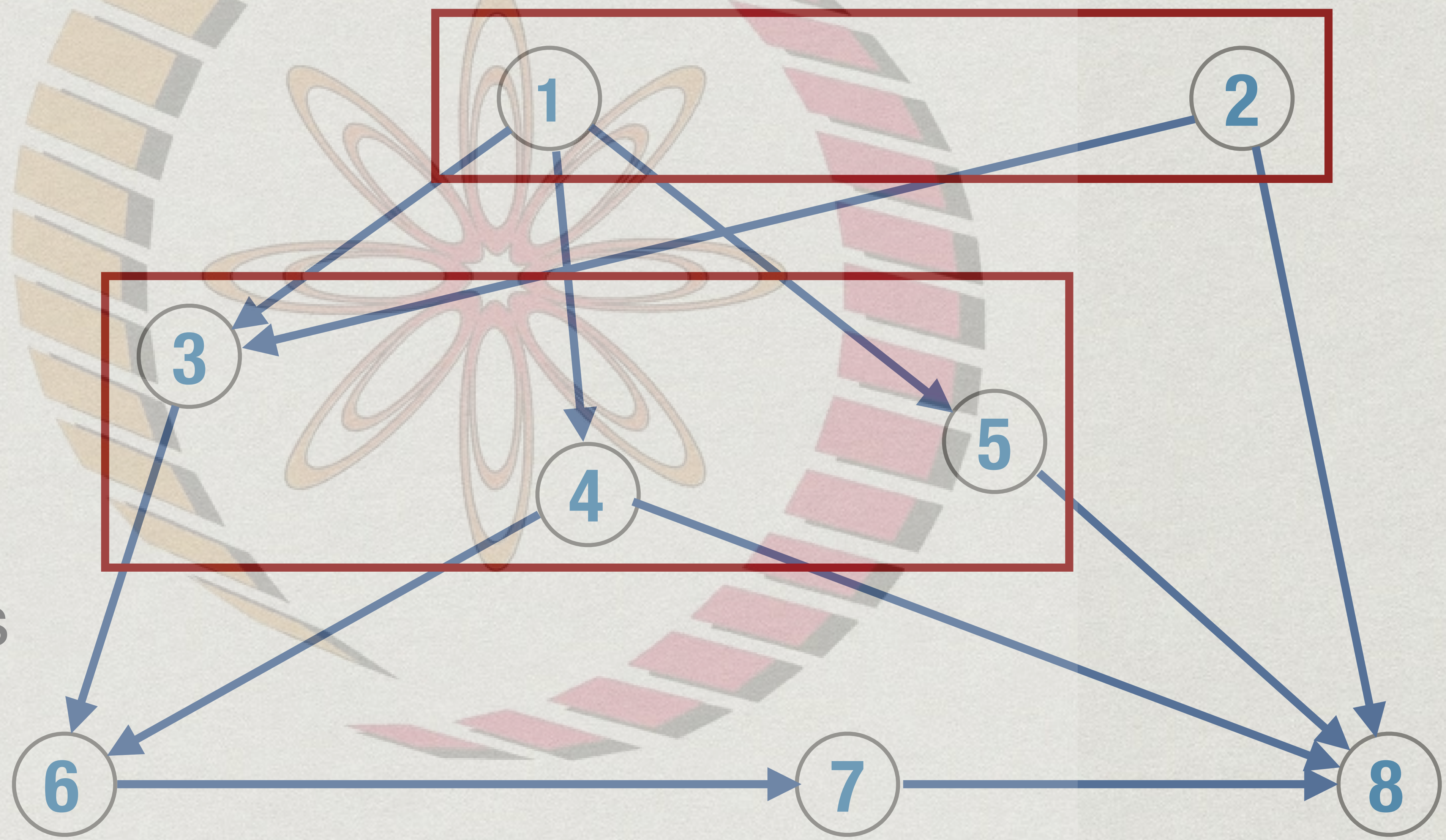
- Imagine these are courses
- Edges are pre-requisites



- What is the minimum number of semesters to complete the programme?

Questions about DAGs

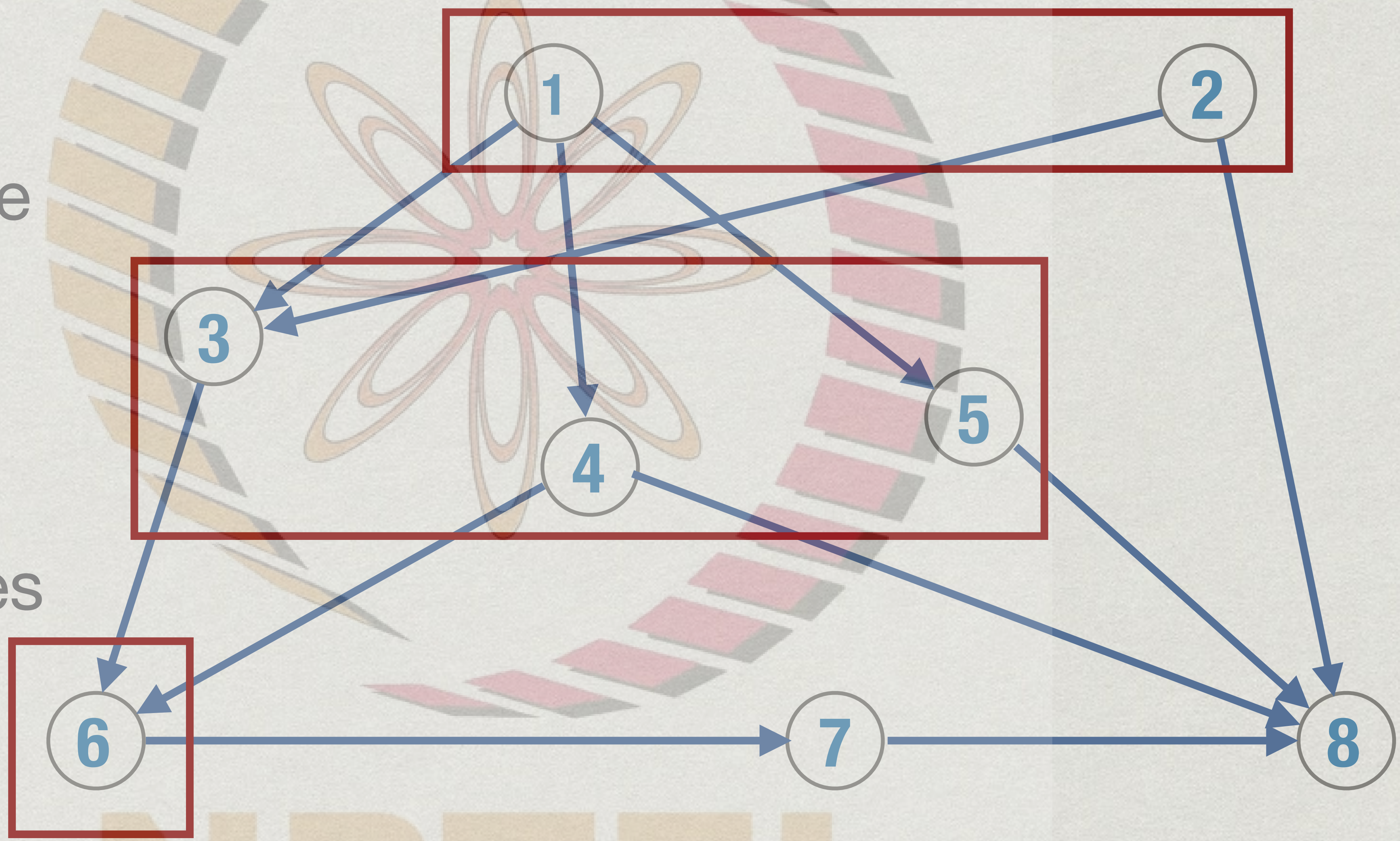
- Imagine these are courses
- Edges are pre-requisites



- What is the minimum number of semesters to complete the programme?

Questions about DAGs

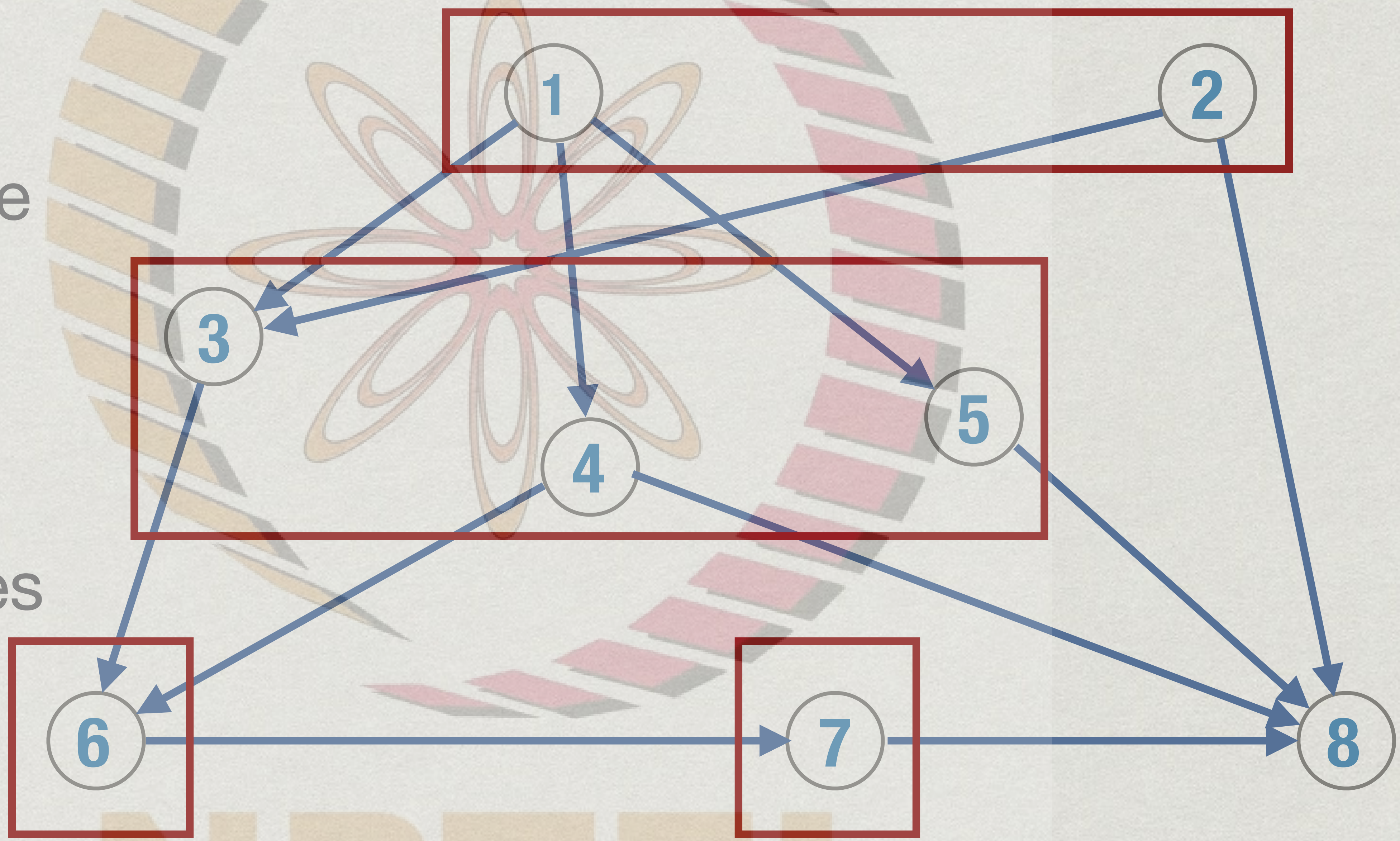
* Imagine these are courses
* Edges are pre-requisites



* What is the minimum number of semesters to complete the programme?

Questions about DAGs

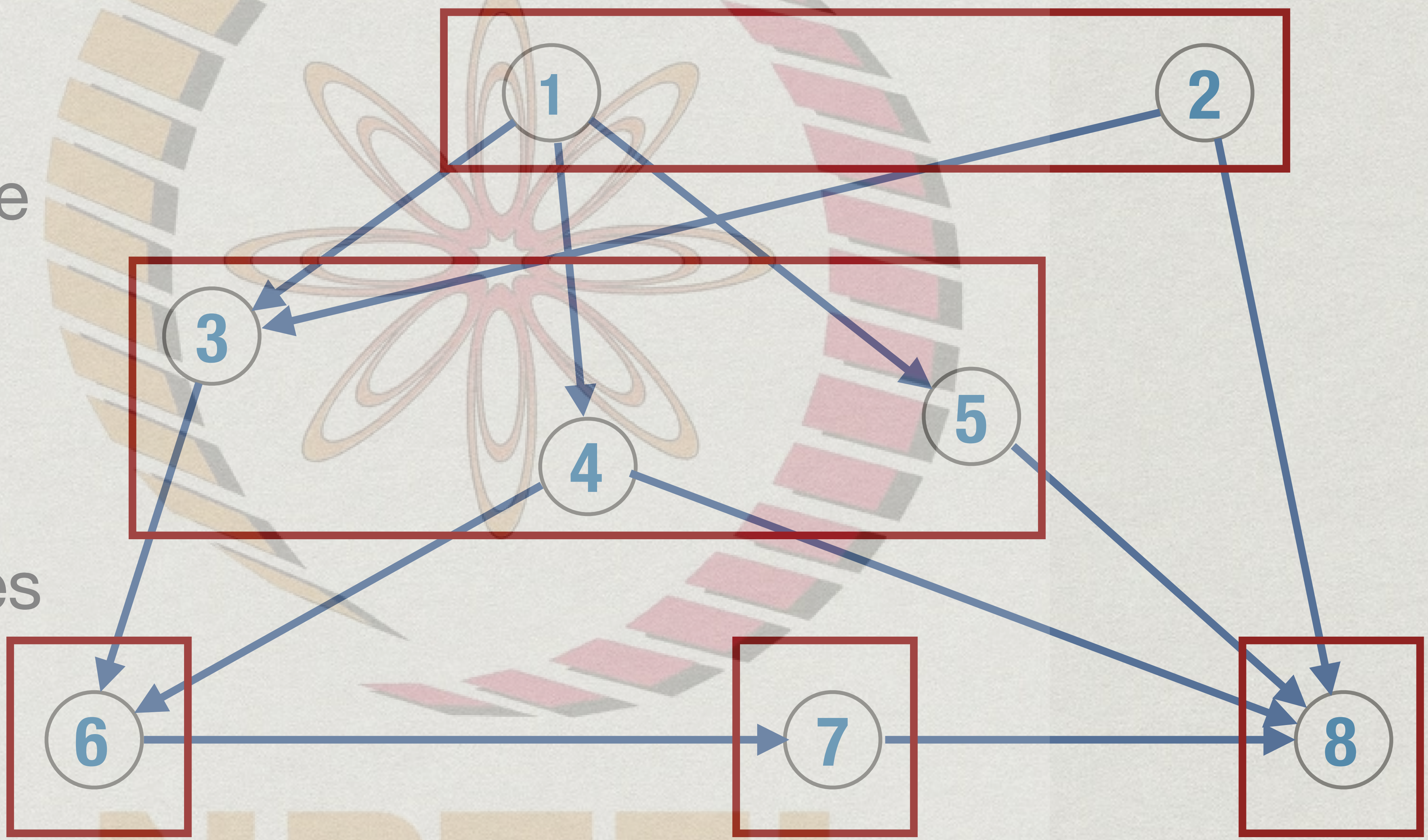
- Imagine these are courses
- Edges are pre-requisites



- What is the minimum number of semesters to complete the programme?

Questions about DAGs

- Imagine these are courses
- Edges are pre-requisites



- What is the minimum number of semesters to complete the programme?

Longest path in a DAG

- Equivalent to finding longest path in the DAG
- If indegree(j) = 0, longest_path_to(j) = 0
- If indegree(k) > 0, longest_path_to(k) is

    1 + max{ longest_path_to(j) } among all

    incoming neighbours j of k

Longest path in a DAG

- To compute longest_path_to(k)
  - Need longest_path_to(j) for all incoming neighbours of k
- If j is an incoming neighbour, (j,k) in E
  - j is enumerated before k in topological order
- Hence, compute longest_path_to(i) in topological order

Longest path in a DAG

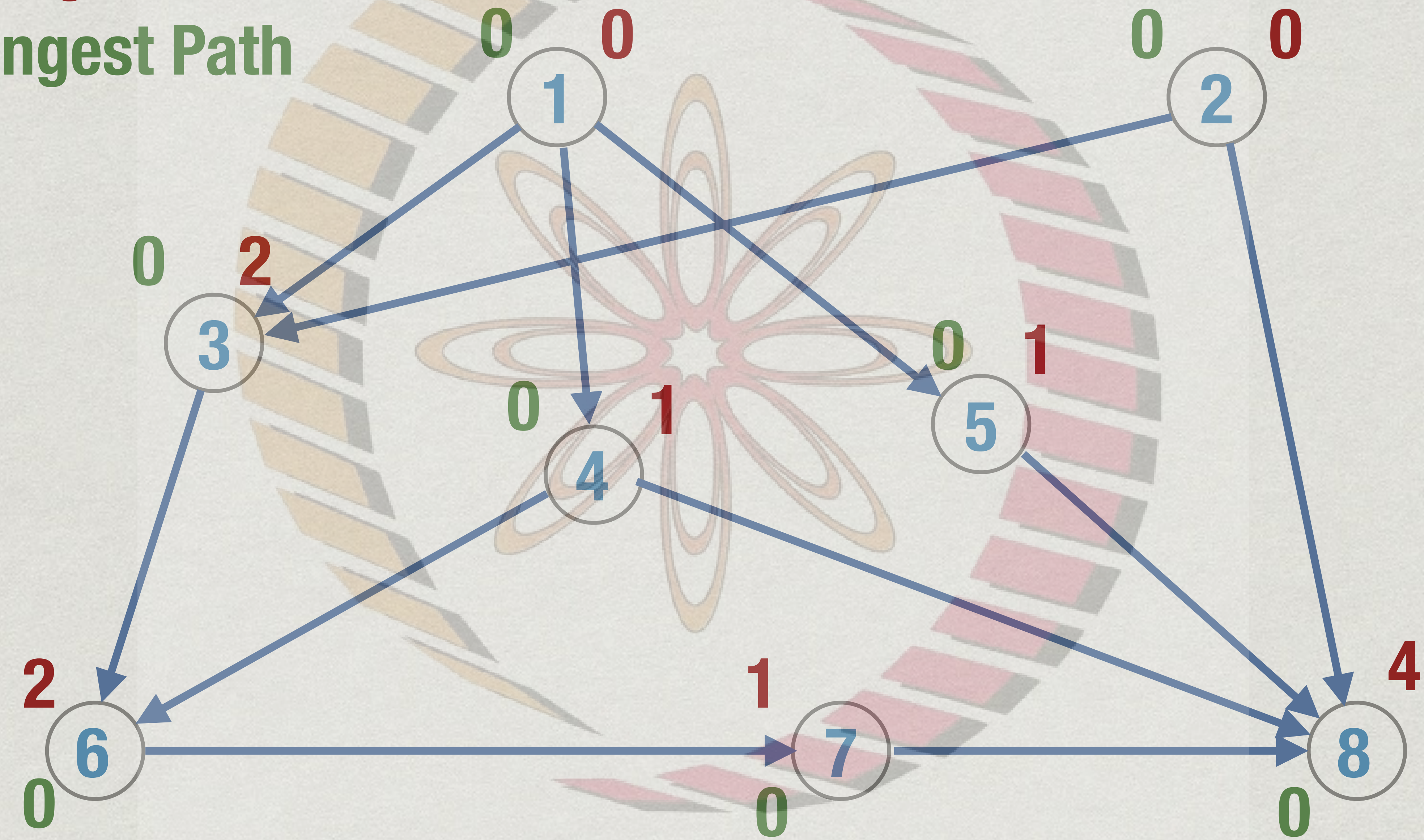
- Let $i_1, i_2, \ldots, i_n$ be a topological ordering of V
- All neighbours of i_k appear before it in this list
- From left to right, compute longest_path_to(i_k) as

  1 + max{ longest_path_to(i_j) } among all

  incoming neighbours i_j of i_k
- Can combine this calculation with topological sort

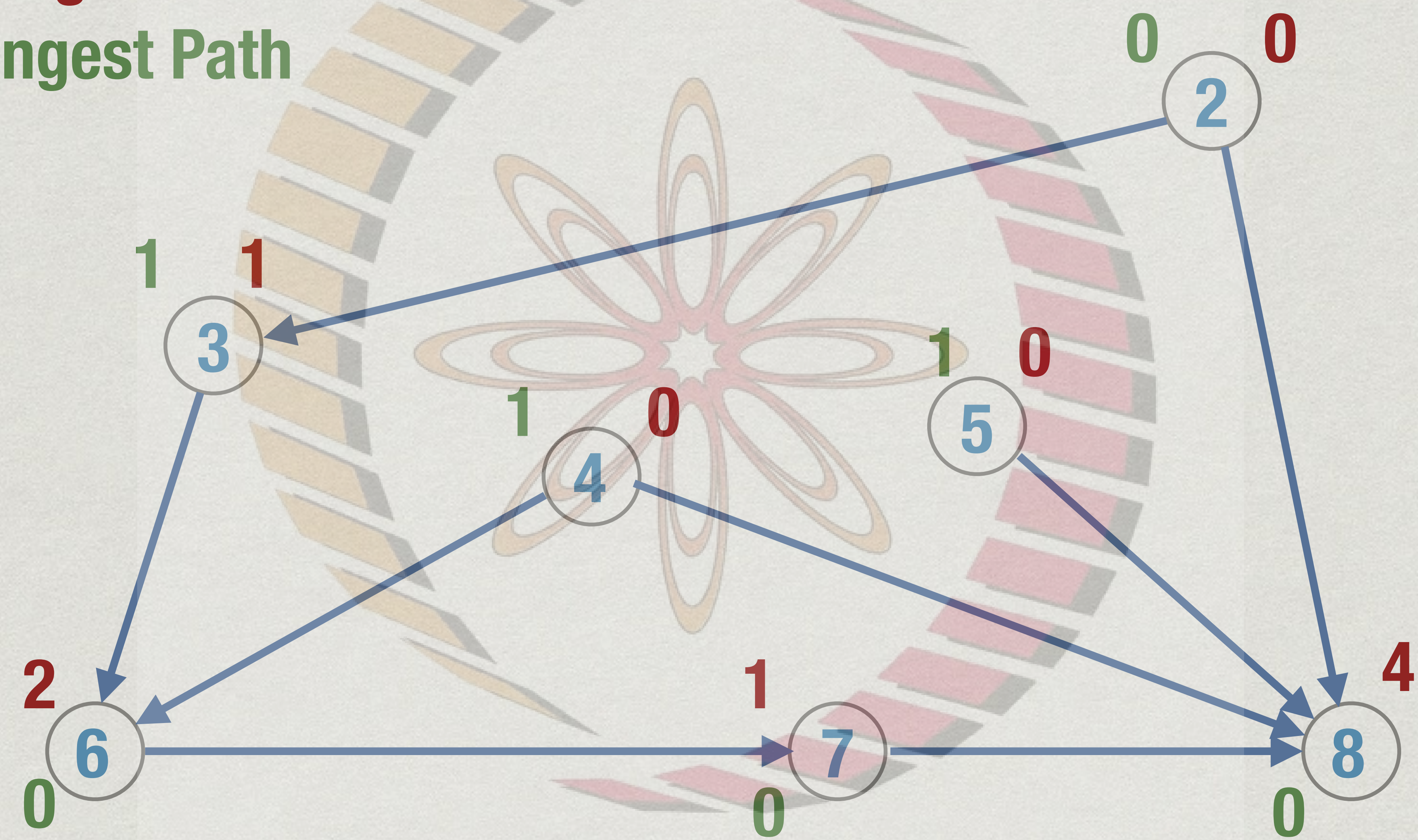
Indegree

Longest Path



Indegree

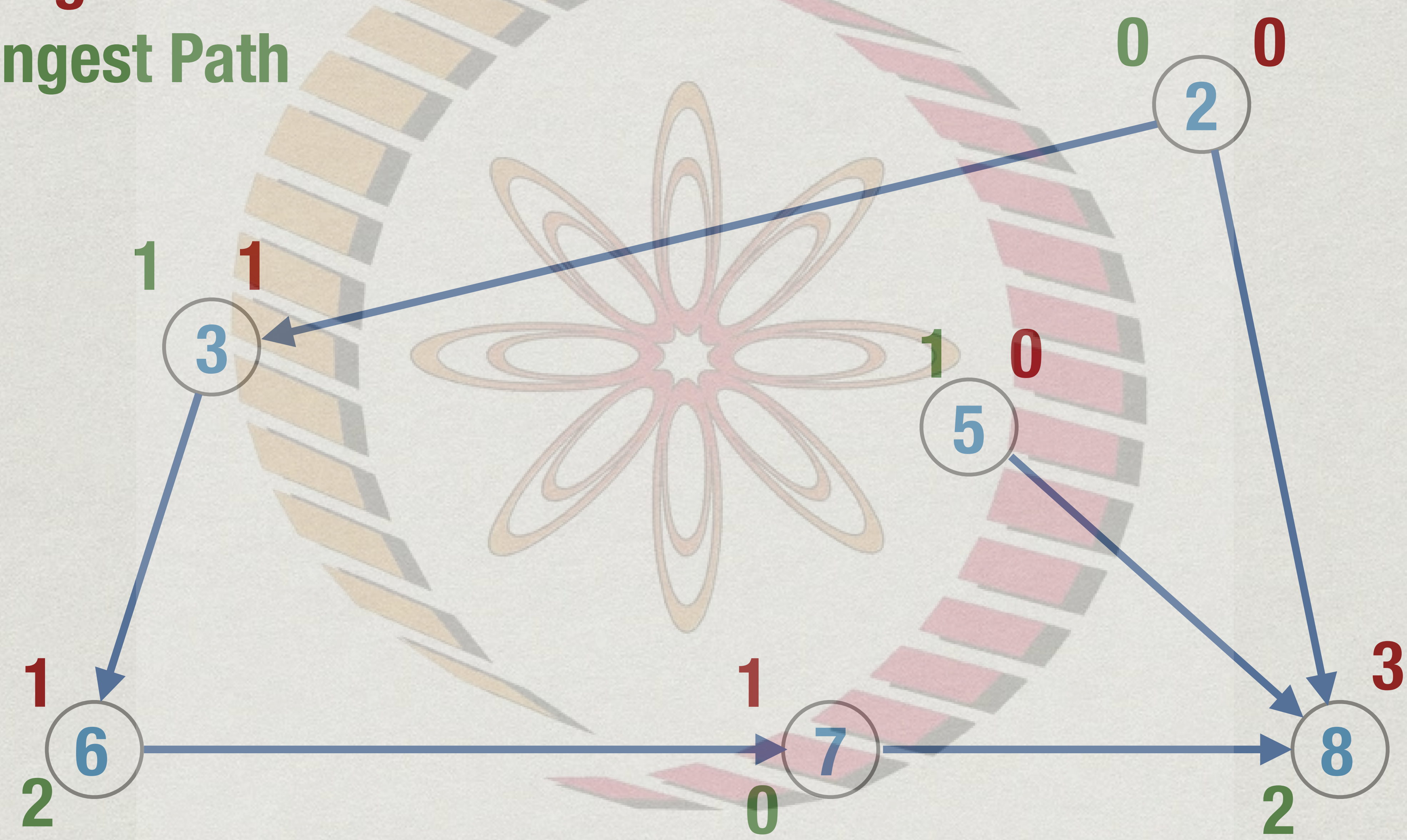
Longest Path



1							
0							

Indegree

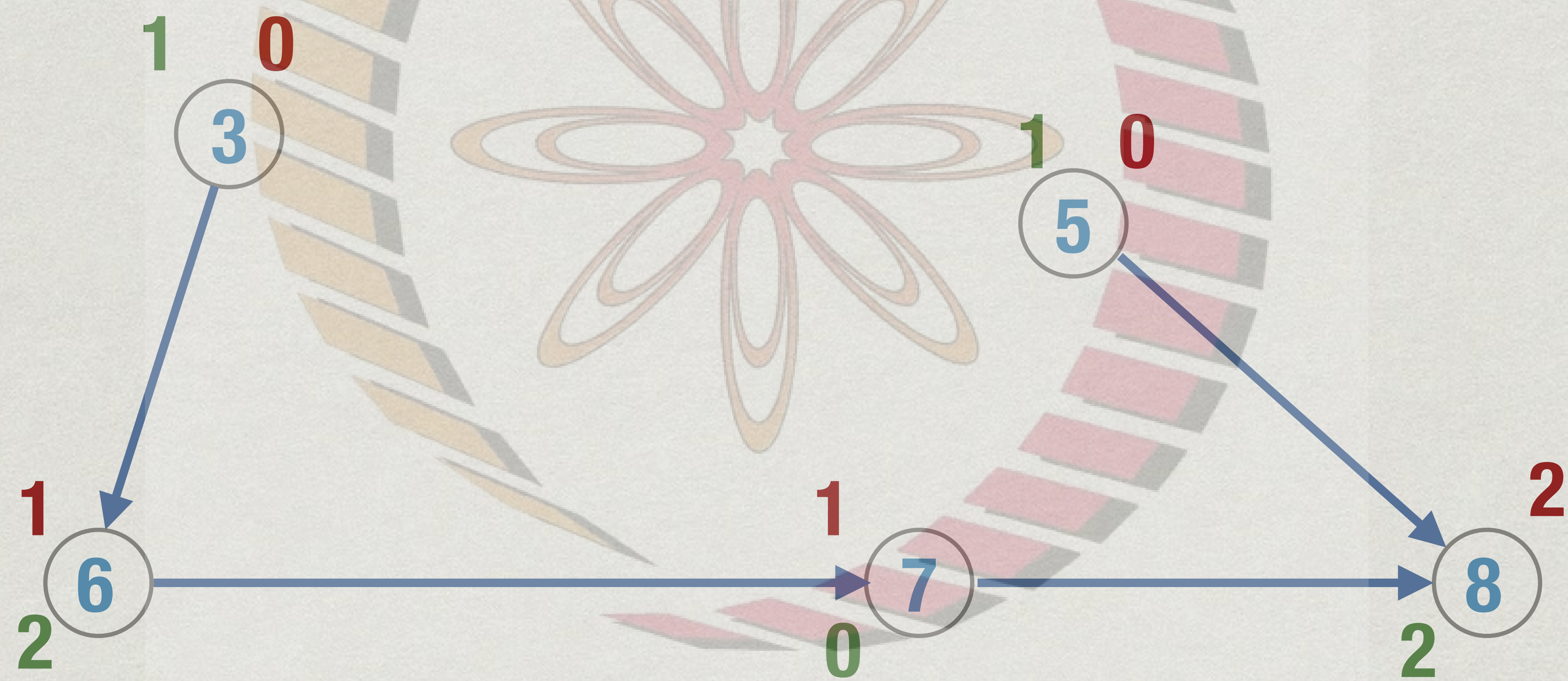
Longest Path



1	4						
0	1						

Indegree

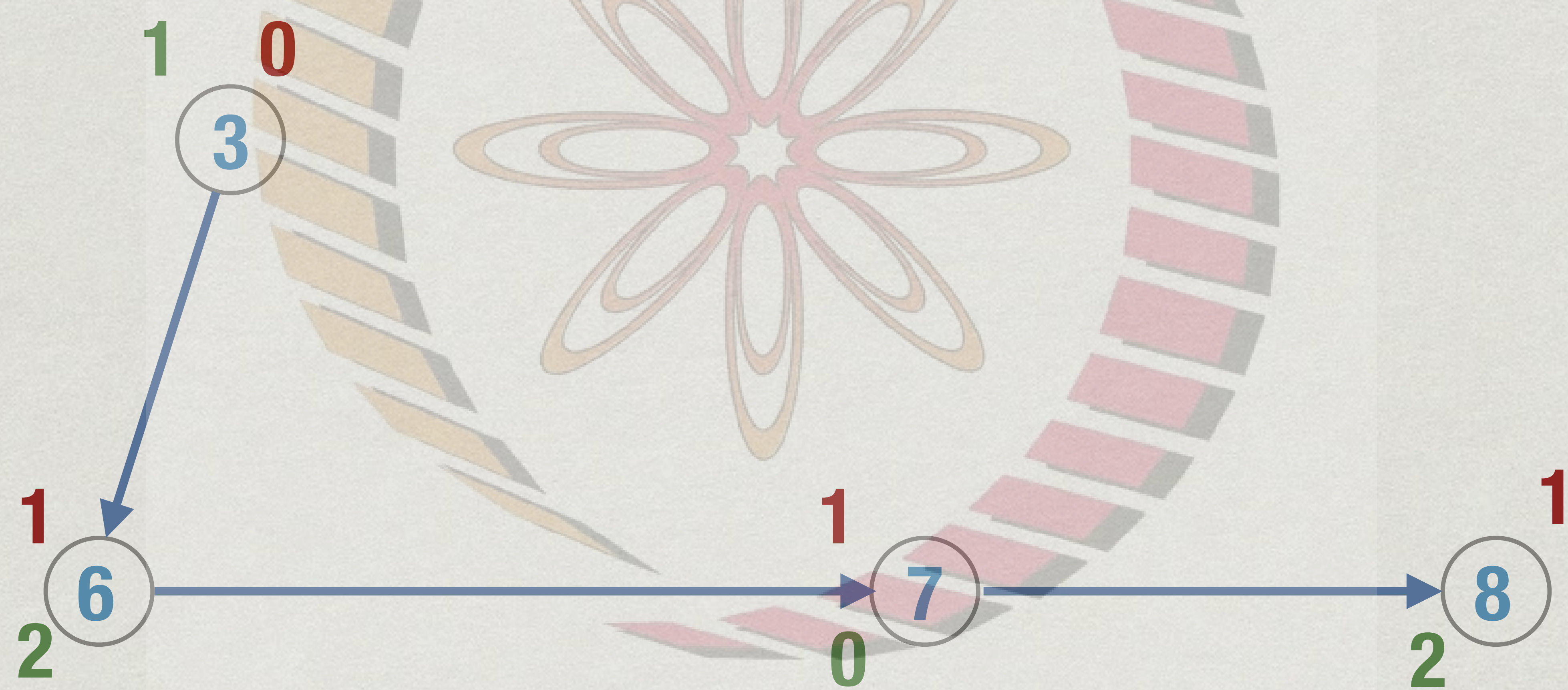
Longest Path



1	4	2					
0	1	0					

Indegree

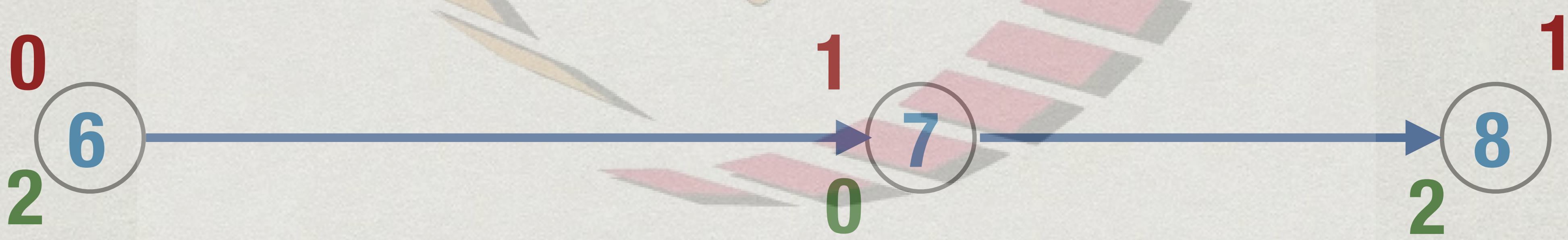
Longest Path



1	4	2	5				
0	1	0	1				

Indegree

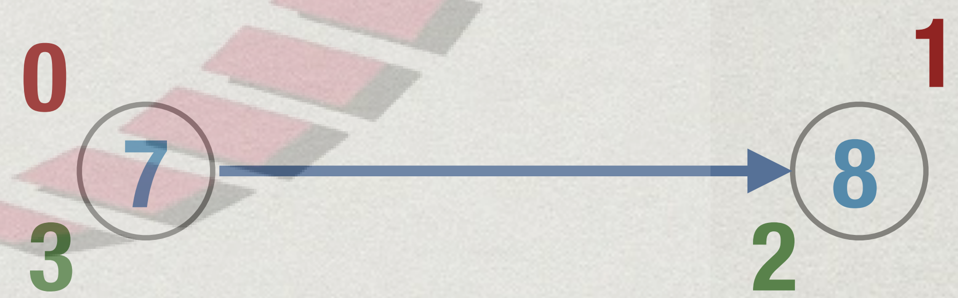
Longest Path



1	4	2	5	3			
0	1	0	1	1			

Indegree

Longest Path



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

Indegree

Longest Path

0

8

4

1	4	2	5	3	6	7	
0	1	0	1	1	2	3	

Indegree
Longest Path

1	4	2	5	3	6	7	8
0	1	0	1	1	2	3	4

Topological ordering with longest path

```
function TopologicalOrderWithLongestPath(G)
  for i = 1 to n
    indegree[i] = 0; LPT[i] = 0
    for j = 1 to n
      indegree[i] = indegree[i] + A[j][i]

  for i = 1 to n
    choose j with indegree[j] = 0
      enumerate j
      indegree[j] = -1
      for k = 1 to n
        if A[j][k] == 1
          indegree[k] = indegree[k]-1
          LPT[k] = max(LPT[k], 1 + LPT[j])
```

Topological ordering with longest path

- This implementation has complexity is $O(n^2)$
- As before, we can use adjacency lists to improve the complexity to $O(m+n)$

Topological ordering with longest path 2

```
function TopologicalOrder(G) //Edges are in adjacency list
  for i = 1 to n { indegree[i] = 0; LPT[i] = 0}

  for i = 1 to n
    for (i,j) in E //proportional to outdegree(i)
      indegree[j] = indegree[j] + 1

  for i = 1 to n
    if indegree[i] == 0 { add i to Queue }

  while Queue is not empty
    j = remove_head(Queue)
    for (j,k) in E //proportional to outdegree(j)
      indegree[k] = indegree[k] - 1
      LPT[k] = max(LPT[k], 1 + LPT[j])
      if indegree[k] == 0 { add k to Queue }
```

Summary

- Dependencies are naturally modelled using DAGs
- Topological ordering lists vertices without violating dependencies
- Longest path in a DAG represents minimum number of steps to list all vertices in groups
- Note: Computing the longest path with no duplicate vertices in an arbitrary graph is not known to have any efficient algorithm!