Longest Paths in DAGs

Madhavan Mukund

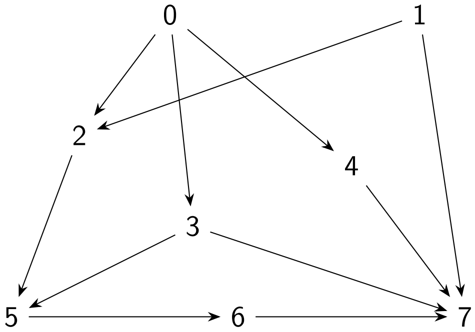
https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 4

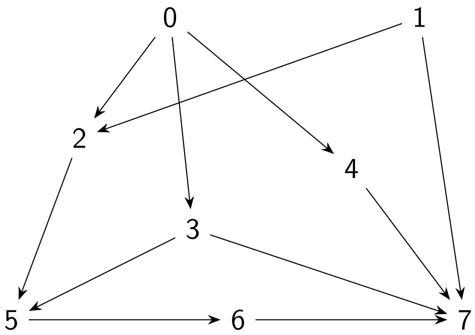
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles



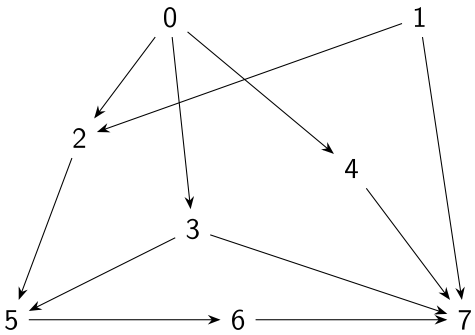
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \ldots, n-1\}$ such that for any $(i, j) \in E$, i appears before j
  - Feasible schedule



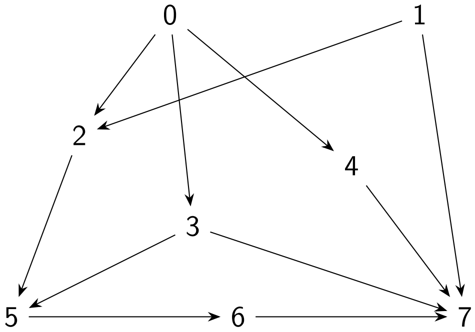
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \dots, n-1\}$ such that for any $(i, j) \in E$, i appears before j
  - Feasible schedule
- Imagine the DAG represents prerequisites between courses



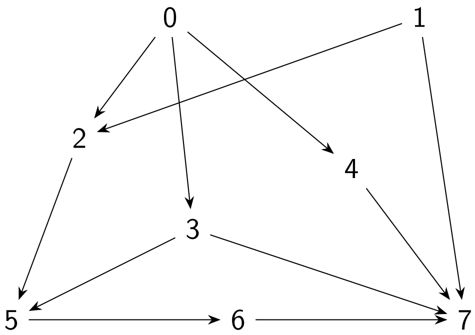
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \dots, n-1\}$ such that for any $(i, j) \in E$, i appears before j
  - Feasible schedule
- Imagine the DAG represents prerequisites between courses
- Each course takes a semester



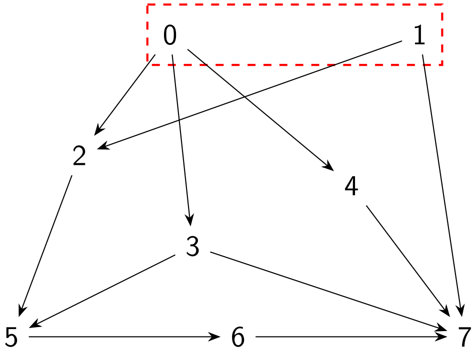
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \ldots, n-1\}$ such that for any $(i, j) \in E$, i appears before j
  - Feasible schedule
- Imagine the DAG represents prerequisites between courses
- Each course takes a semester
- Minimum number of semesters to complete the programme?



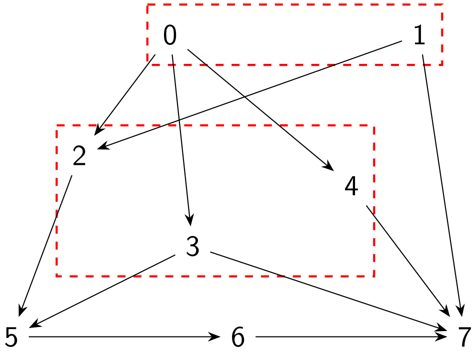
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \ldots, n-1\}$ such that for any $(i,j) \in E$, i appears before j
  - Feasible schedule
- Imagine the DAG represents prerequisites between courses
- Each course takes a semester
- Minimum number of semesters to complete the programme?



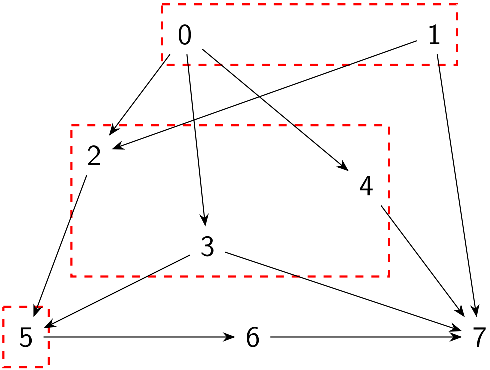
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \ldots, n-1\}$ such that for any $(i, j) \in E$, i appears before j
  - Feasible schedule
- Imagine the DAG represents prerequisites between courses
- Each course takes a semester
- Minimum number of semesters to complete the programme?



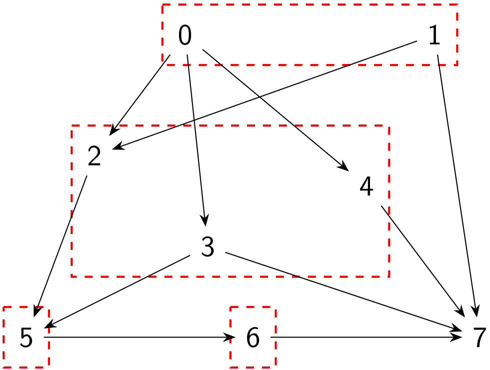
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \ldots, n-1\}$ such that for any $(i, j) \in E$, i appears before j
  - Feasible schedule
- Imagine the DAG represents prerequisites between courses
- Each course takes a semester
- Minimum number of semesters to complete the programme?



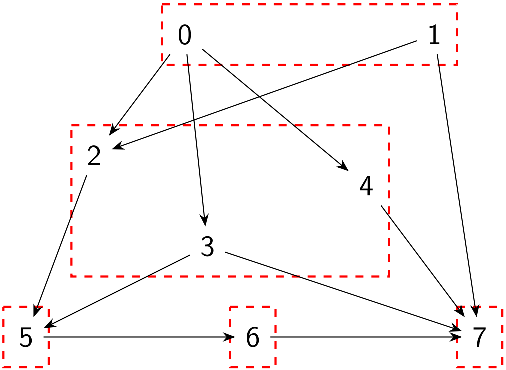
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \ldots, n-1\}$ such that for any $(i, j) \in E$, i appears before j
  - Feasible schedule
- Imagine the DAG represents prerequisites between courses
- Each course takes a semester
- Minimum number of semesters to complete the programme?



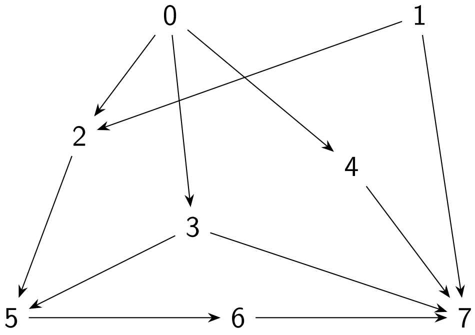
Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \ldots, n-1\}$ such that for any $(i, j) \in E$, i appears before j
  - Feasible schedule
- Imagine the DAG represents prerequisites between courses
- Each course takes a semester
- Minimum number of semesters to complete the programme?



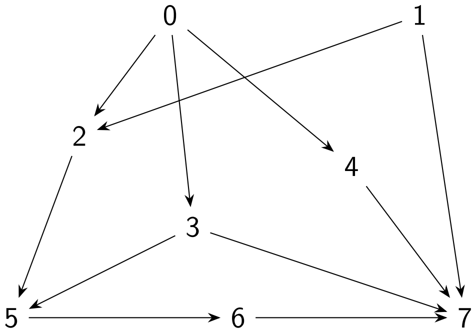
Longest Path

- Find the longest path in a DAG



Longest Path

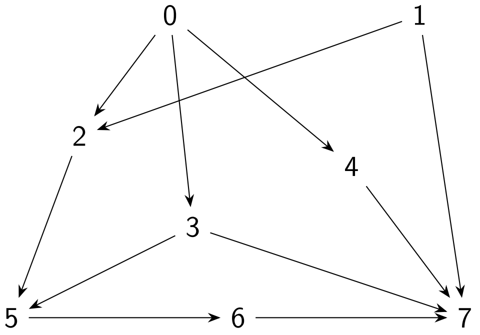
- Find the longest path in a DAG
- If indegree(i) $= 0$,
  longest-path-to(i) $= 0$



Longest Path

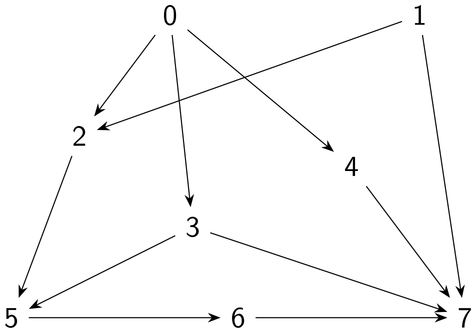
- Find the longest path in a DAG
- If $\text{indegree}(i) = 0$,
  $\text{longest-path-to}(i) = 0$
- If $\text{indegree}(i) > 0$, longest path to i is 1 more than longest path to its incoming neighbours

  $\text{longest-path-to}(i) =$
  $1 + \max\{\text{longest-path-to}(j) \mid (j, i) \in E\}$



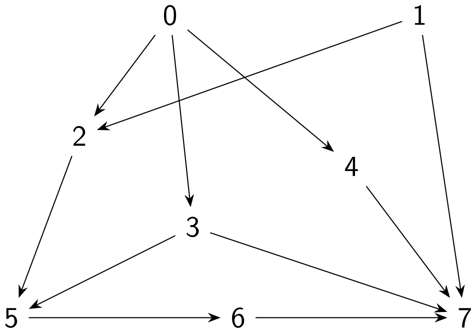
Longest Path

- longest-path-to(i) $=$
  $1 + \max\{\text{longest-path-to}(j) \mid (j, i) \in E\}$



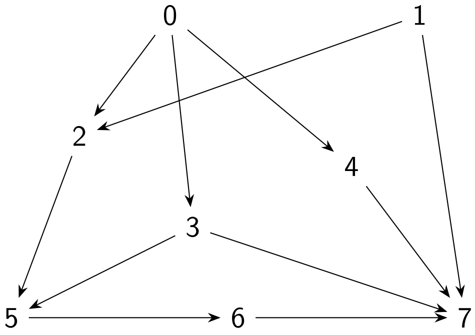
Longest Path

- longest-path-to(i) $=$
  $1 + \max\{\text{longest-path-to}(j) \mid (j,i) \in E\}$
- To compute longest-path-to(i), need longest-path-to(k), for each incoming neighbour k



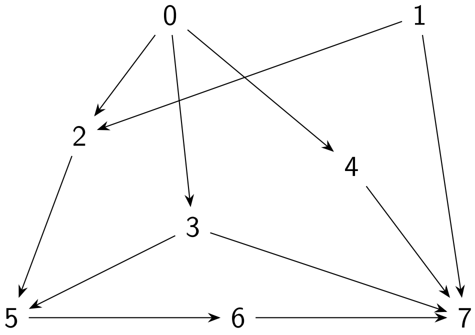
Longest Path

- longest-path-to(i) $=$
  $1 + \max\{$longest-path-to(j) $\mid (j, i) \in E\}$
- To compute longest-path-to(i), need longest-path-to(k), for each incoming neighbour k
- If graph is topologically sorted, k is listed before i



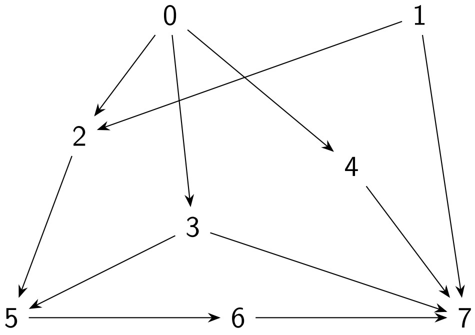
Longest Path

- longest-path-to(i) $=$ $1 + \max\{\text{longest-path-to}(j) \mid (j,i) \in E\}$
- To compute longest-path-to(i), need longest-path-to(k), for each incoming neighbour k
- If graph is topologically sorted, k is listed before i
- Hence compute longest-path-to() in topological order



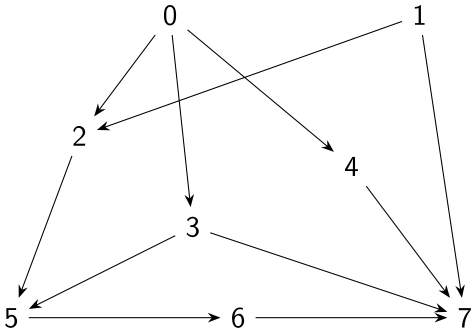
Longest Path

- Let $i_0, i_1, \ldots, i_{n-1}$ be a topological ordering of V



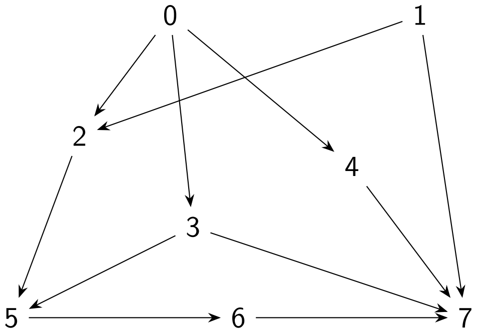
Longest Path

- Let $i_0, i_1, \ldots, i_{n-1}$ be a topological ordering of V
- All neighbours of i_k appear before it in this list



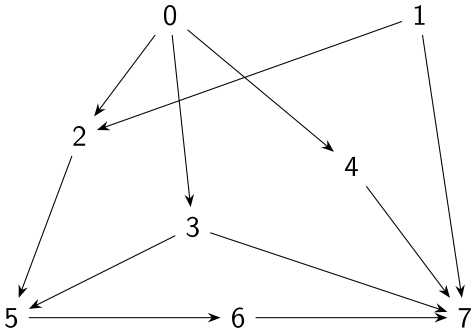
Longest Path

- Let $i_0, i_1, \ldots, i_{n-1}$ 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
  $1 + \max\{\text{longest-path-to}(i_j) \mid (i_j, i_k) \in E\}$



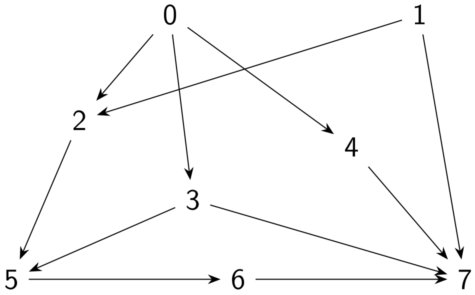
Longest Path

- Let $i_0, i_1, \ldots, i_{n-1}$ be a topological ordering of V
- All neighbours of i_k appear before it in this list
- From left to right, compute $\text{longest-path-to}(i_k)$ as $1 + \max\{\text{longest-path-to}(i_j) \mid (i_j, i_k) \in E\}$
- Overlap this computation with topological sorting



Longest path algorithm

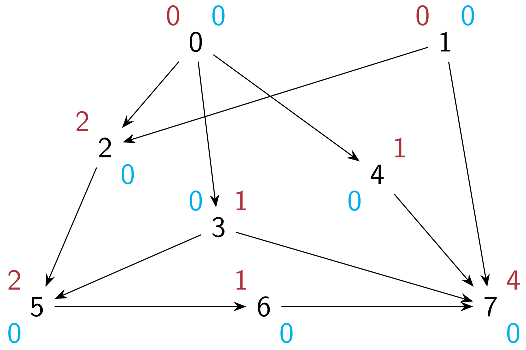
- Compute indegree of each vertex



Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices

Indegree, Longest path



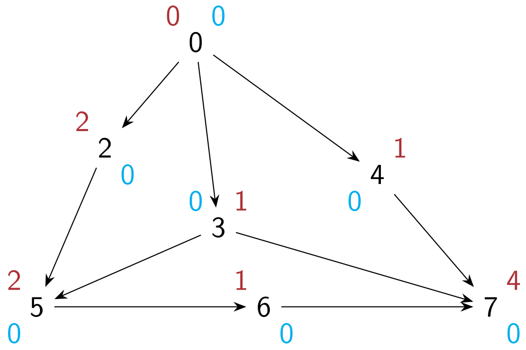
Topological order

Longest path to

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG

Indegree, Longest path

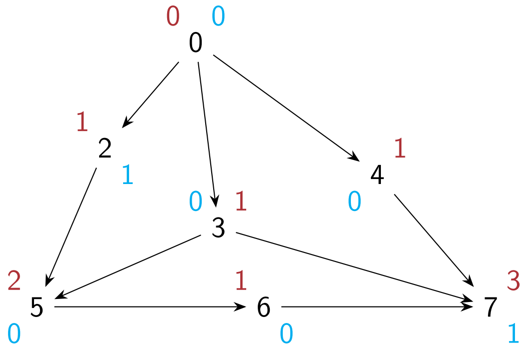


Topological order	1
Longest path to	0

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $\textit{text}longest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path

Indegree, Longest path

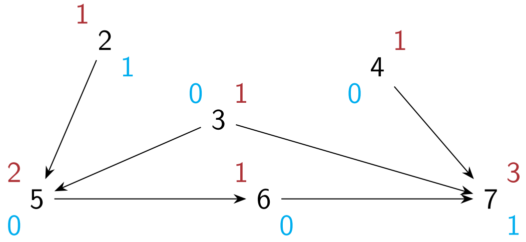


Topological order 1
Longest path to 0

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $text longest-path-to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

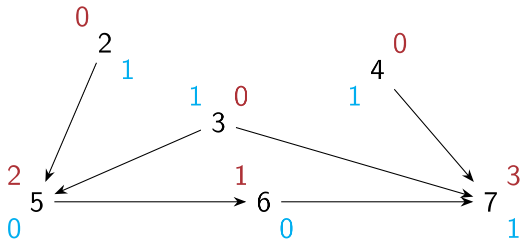


Topological order	1	0
Longest path to	0	0

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest-path-to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

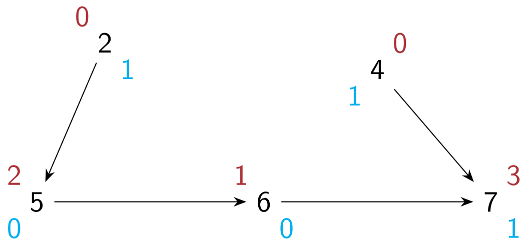


Topological order	1	0
Longest path to	0	0

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $text longest-path-to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

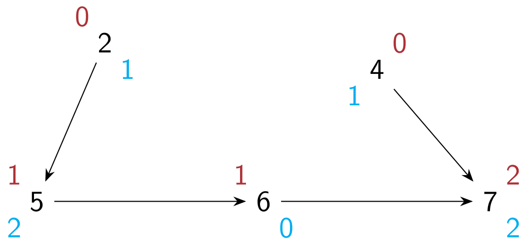


Topological order	1	0	3
Longest path to	0	0	1

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

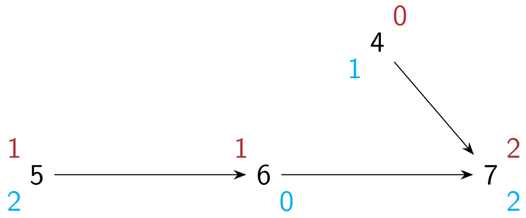


Topological order	1	0	3
Longest path to	0	0	1

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

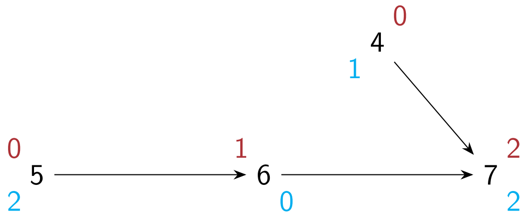


Topological order	1	0	3	2
Longest path to	0	0	1	1

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

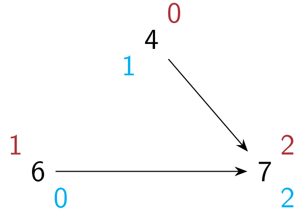


Topological order	1	0	3	2
Longest path to	0	0	1	1

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

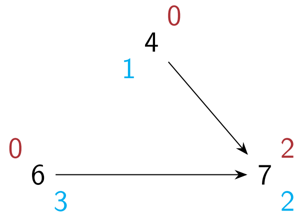


Topological order	1	0	3	2	5
Longest path to	0	0	1	1	2

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest-path-to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

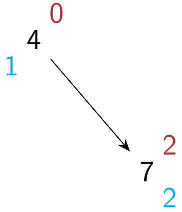


Topological order	1	0	3	2	5
Longest path to	0	0	1	1	2

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

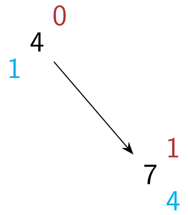


Topological order	1	0	3	2	5	6
Longest path to	0	0	1	1	2	3

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path



Topological order	1	0	3	2	5	6
Longest path to	0	0	1	1	2	3

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

1
7
4

Topological order	1	0	3	2	5	6	4
Longest path to	0	0	1	1	2	3	1

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest - path - to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

7 0 4

Topological order	1	0	3	2	5	6	4
Longest path to	0	0	1	1	2	3	1

Longest path algorithm

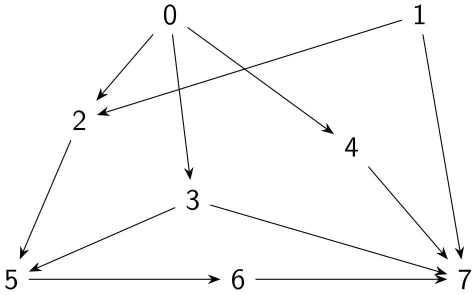
- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest-path-to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Indegree, Longest path

Topological order	1	0	3	2	5	6	4	7
Longest path to	0	0	1	1	2	3	1	4

Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize $textlongest-path-to$ to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed



Topological order	1	0	3	2	5	6	4	7
Longest path to	0	0	1	1	2	3	1	4

Longest path using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Process head of queue: update indegrees, update queue, update longest paths
- Repeat till queue is empty

```
def longestpathlist(AList):
  (indegree,lpath) = ({},{})
  for u in AList.keys():
    (indegree[u],lpath[u]) = (0,0)
  for u in AList.keys():
    for v in AList[u]:
      indegree[v] = indegree[v] + 1

  zerodegreeq = Queue()
  for u in AList.keys():
    if indegree[u] == 0:
      zerodegreeq.addq(u)

  while (not zerodegreeq.isempty()):
    j = zerodegreeq.delq()
    indegree[j] = indegree[j]-1
    for k in AList[j]:
      indegree[k] = indegree[k] - 1
      lpath[k] = max(lpath[k],lpath[j]+1)
      if indegree[k] == 0:
        zerodegreeq.addq(k)
  return(lpath)
```

Longest path using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Process head of queue: update indegrees, update queue, update longest paths
- Repeat till queue is empty

Analysis

```
def longestpathlist(AList):
  (indegree,lpath) = ({},{})
  for u in AList.keys():
    (indegree[u],lpath[u]) = (0,0)
  for u in AList.keys():
    for v in AList[u]:
      indegree[v] = indegree[v] + 1

  zerodegreeq = Queue()
  for u in AList.keys():
    if indegree[u] == 0:
      zerodegreeq.addq(u)

  while (not zerodegreeq.isempty()):
    j = zerodegreeq.delq()
    indegree[j] = indegree[j]-1
    for k in AList[j]:
      indegree[k] = indegree[k] - 1
      lpath[k] = max(lpath[k],lpath[j]+1)
      if indegree[k] == 0:
        zerodegreeq.addq(k)
  return(lpath)
```

Longest path using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Process head of queue: update indegrees, update queue, update longest paths
- Repeat till queue is empty

Analysis

- Initializing indegrees is $O(m+n)$

```
def longestpathlist(AList):
  (indegree,lpath) = ({},{})
  for u in AList.keys():
    (indegree[u],lpath[u]) = (0,0)
  for u in AList.keys():
    for v in AList[u]:
      indegree[v] = indegree[v] + 1

  zerodegreeq = Queue()
  for u in AList.keys():
    if indegree[u] == 0:
      zerodegreeq.addq(u)

  while (not zerodegreeq.isempty()):
    j = zerodegreeq.delq()
    indegree[j] = indegree[j]-1
    for k in AList[j]:
      indegree[k] = indegree[k] - 1
      lpath[k] = max(lpath[k],lpath[j]+1)
      if indegree[k] == 0:
        zerodegreeq.addq(k)
  return(lpath)
```

Longest path using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Process head of queue: update indegrees, update queue, update longest paths
- Repeat till queue is empty

Analysis

- Initializing indegrees is $O(m+n)$
- Loop to enumerate vertices runs n times
  - Updating indegrees, longest path: amortised $O(m)$

```python
def longestpathlist(AList):
  (indegree,lpath) = ({},{})
  for u in AList.keys():
    (indegree[u],lpath[u]) = (0,0)
  for u in AList.keys():
    for v in AList[u]:
      indegree[v] = indegree[v] + 1

  zerodegreeq = Queue()
  for u in AList.keys():
    if indegree[u] == 0:
      zerodegreeq.addq(u)

  while (not zerodegreeq.isempty()):
    j = zerodegreeq.delq()
    indegree[j] = indegree[j]-1
    for k in AList[j]:
      indegree[k] = indegree[k] - 1
      lpath[k] = max(lpath[k],lpath[j]+1)
      if indegree[k] == 0:
        zerodegreeq.addq(k)
  return(lpath)
```

Longest path using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Process head of queue: update indegrees, update queue, update longest paths
- Repeat till queue is empty

Analysis

- Initializing indegrees is $O(m+n)$
- Loop to enumerate vertices runs n times
  - Updating indegrees, longest path: amortised $O(m)$
- Overall, $O(m+n)$

```
def longestpathlist(AList):
  (indegree,lpath) = ({},{})
  for u in AList.keys():
    (indegree[u],lpath[u]) = (0,0)
  for u in AList.keys():
    for v in AList[u]:
      indegree[v] = indegree[v] + 1

  zerodegreeq = Queue()
  for u in AList.keys():
    if indegree[u] == 0:
      zerodegreeq.addq(u)

  while (not zerodegreeq.isempty()):
    j = zerodegreeq.delq()
    indegree[j] = indegree[j]-1
    for k in AList[j]:
      indegree[k] = indegree[k] - 1
      lpath[k] = max(lpath[k],lpath[j]+1)
      if indegree[k] == 0:
        zerodegreeq.addq(k)
  return(lpath)
```

Summary

- Directed acyclic graphs are a natural way to represent dependencies
- Topological sort gives a feasible schedule that represents dependencies
- In parallel with topological sort, we can compute the longest path

Summary

- Directed acyclic graphs are a natural way to represent dependencies
- Topological sort gives a feasible schedule that represents dependencies
- In parallel with topological sort, we can compute the longest path
- Notion of longest path makes sense even for graphs with cycles
  - No repeated vertices in a path, so path has at most $n-1$ edges

Summary

- Directed acyclic graphs are a natural way to represent dependencies
- Topological sort gives a feasible schedule that represents dependencies
- In parallel with topological sort, we can compute the longest path
- Notion of longest path makes sense even for graphs with cycles
  - No repeated vertices in a path, so path has at most $n-1$ edges
- However, computing longest paths in arbitrary graphs is much harder than for DAGs
  - No better strategy known than exhaustively enumerating paths