Topological Sorting

Madhavan Mukund

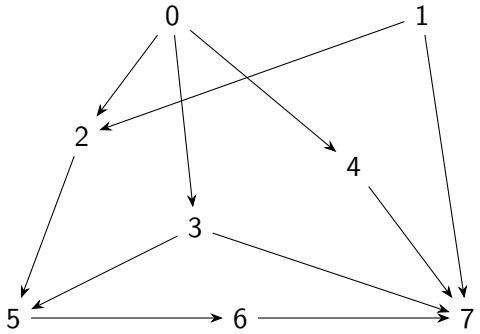
https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 4

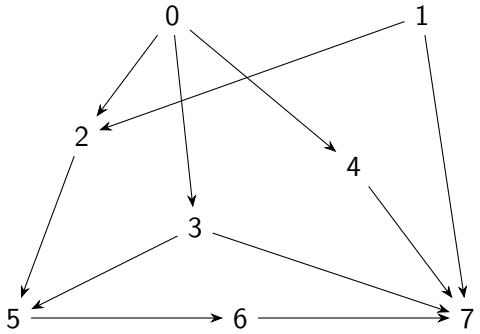
Directed Acyclic Graphs

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



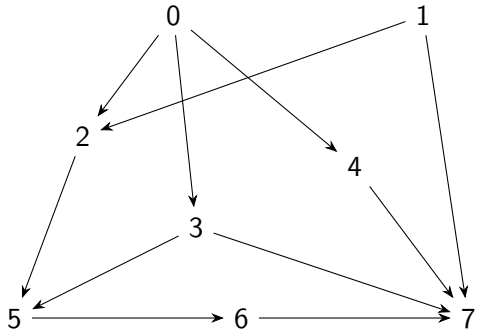
Topological Sort

- A graph with directed cycles cannot be sorted topologically
- Path $i \rightsquigarrow j$ means i must be listed before j
- Cycle $\Rightarrow$ vertices i, j such that there are paths $i \rightsquigarrow j$ and $j \rightsquigarrow i$
- i must appear before j, and j must appear before i, impossible!



Topological Sort

- A graph with directed cycles cannot be sorted topologically
- Path $i \rightsquigarrow j$ means i must be listed before j
- Cycle $\Rightarrow$ vertices i, j such that there are paths $i \rightsquigarrow j$ and $j \rightsquigarrow i$
- i must appear before j, and j must appear before i, impossible!



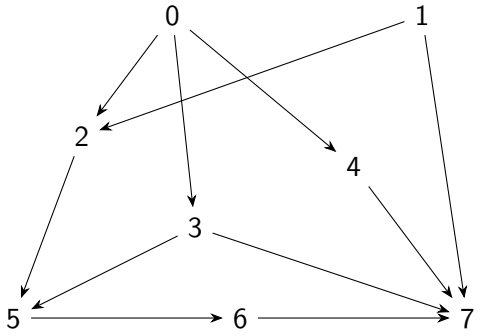
Claim

Every DAG can be topologically sorted

How to topologically sort a DAG?

Strategy

- First list vertices with no dependencies
- As we proceed, list vertices whose dependencies have already been listed
- ...



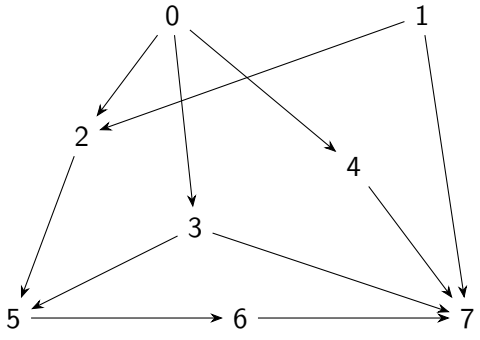
How to topologically sort a DAG?

Strategy

- First list vertices with no dependencies
- As we proceed, list vertices whose dependencies have already been listed
- ...

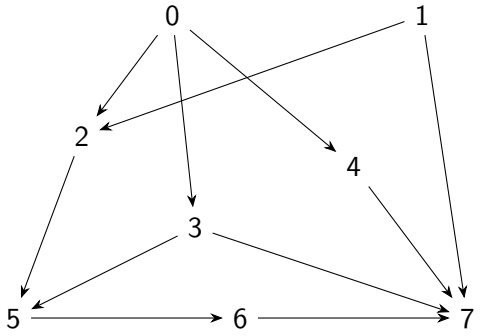
Questions

- Why will there be a starting vertex with no dependencies?
- How do we guarantee we can keep progressing with the listing?



Algorithm for topological sort

- A vertex with no dependencies has no incoming edges, indegree(v) $= 0$

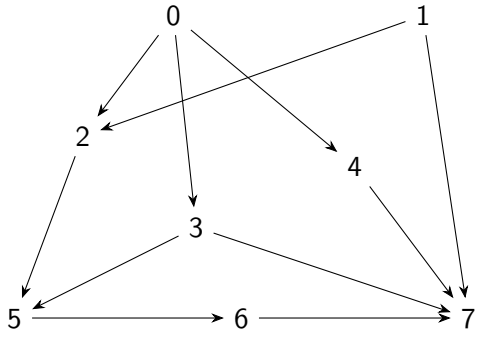


Algorithm for topological sort

- A vertex with no dependencies has no incoming edges, indegree(v) $= 0$

Claim

Every DAG has a vertex with indegree 0



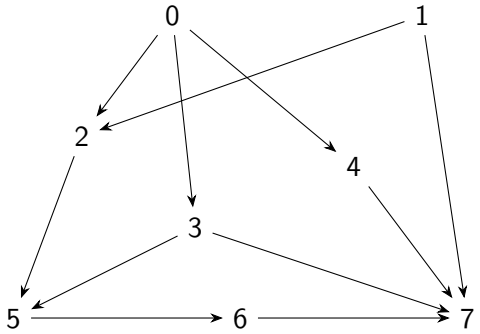
Algorithm for topological sort

- A vertex with no dependencies has no incoming edges, indegree(v) $= 0$

Claim

Every DAG has a vertex with indegree 0

- Start with any vertex with indegree > 0
- Follow edge back to one of its predecessors
- Repeat so long as indegree > 0
- If we repeat n times, we must have a cycle, which is impossible in a DAG

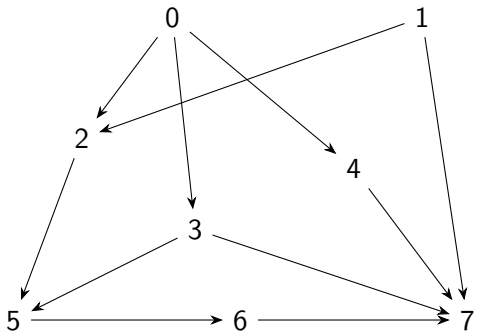


Topological sort algorithm

Fact

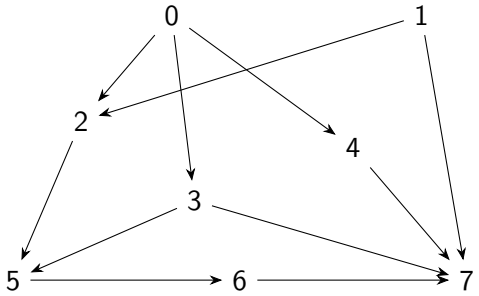
Every DAG has a vertex with indegree 0

- List out a vertex j with indegree $= 0$
- Delete j and all edges from j
- What remains is again a DAG!
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed



Topological sort algorithm

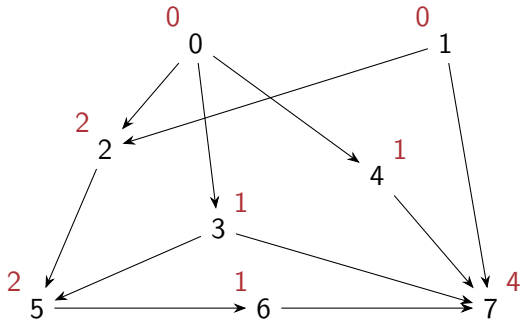
- Compute indegree of each vertex



Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix

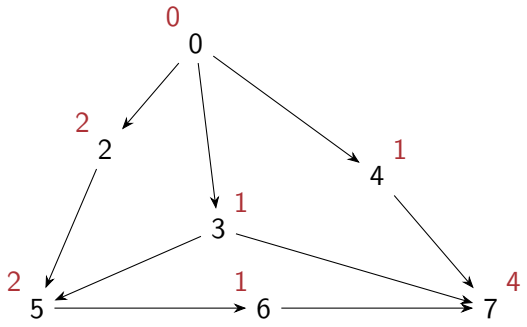
Indegree



Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG

Indegree



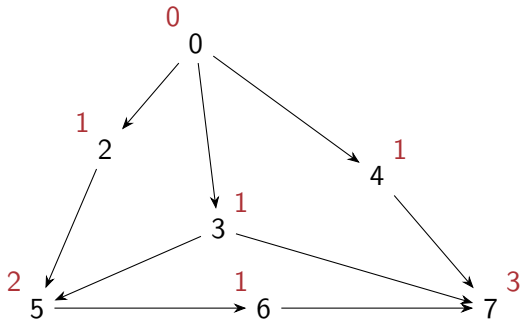
Topologically sorted sequence

1,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees

Indegree



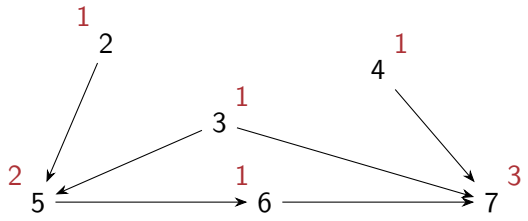
Topologically sorted sequence

1,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate

Indegree



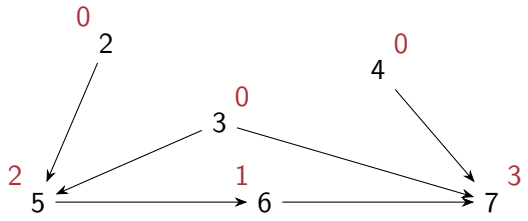
Topologically sorted sequence

1, 0,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate

Indegree



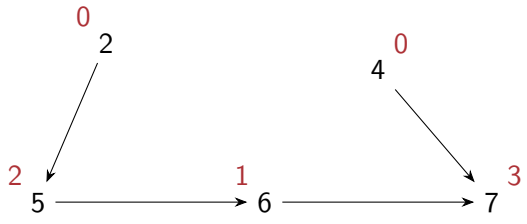
Topologically sorted sequence

1, 0,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree



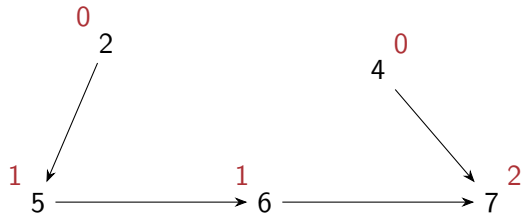
Topologically sorted sequence

1, 0, 3,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree



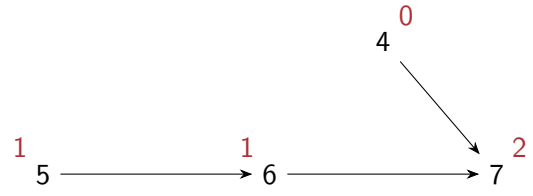
Topologically sorted sequence

1, 0, 3,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree



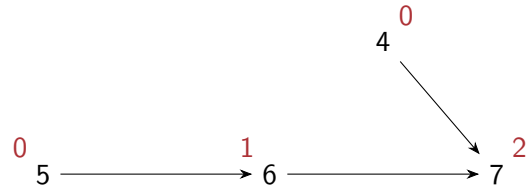
Topologically sorted sequence

1, 0, 3, 2,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree



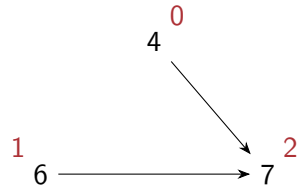
Topologically sorted sequence

1, 0, 3, 2,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree



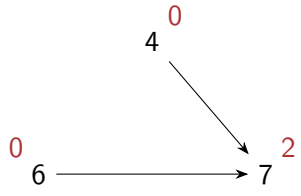
Topologically sorted sequence

1, 0, 3, 2, 5,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree



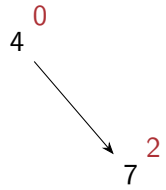
Topologically sorted sequence

1, 0, 3, 2, 5,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree



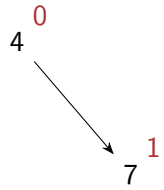
Topologically sorted sequence

1, 0, 3, 2, 5, 6,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree



Topologically sorted sequence

1, 0, 3, 2, 5, 6,

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree

Topologically sorted sequence

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

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed

Indegree

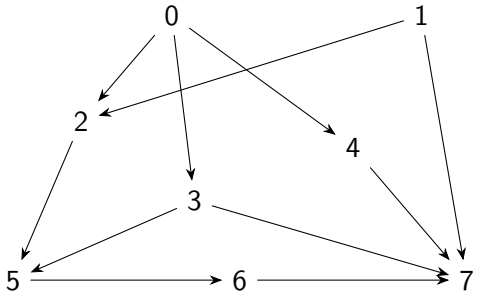
0
7

Topologically sorted sequence

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

Topological sort algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Can find another vertex with indegree $= 0$ to list and eliminate
- Repeat till all vertices are listed



Topologically sorted sequence

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

An implementation of topological sort

- Compute indegrees by scanning columns of adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Repeat till all vertices are listed

```
def toposort(AMat):
  (rows,cols) = AMat.shape
  indegree = {}
  toposortlist = []

  for c in range(cols):
    indegree[c] = 0
    for r in range(rows):
      if AMat[r,c] == 1:
        indegree[c] = indegree[c] + 1

  for i in range(rows):
    j = min([k for k in range(cols)
             if indegree[k] == 0])
    toposortlist.append(j)
    indegree[j] = indegree[j]-1
    for k in range(cols):
      if AMat[j,k] == 1:
        indegree[k] = indegree[k] - 1

  return(toposortlist)
```

An implementation of topological sort

- Compute indegrees by scanning columns of adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Repeat till all vertices are listed

Analysis

```
def toposort(AMat):
  (rows,cols) = AMat.shape
  indegree = {}
  toposortlist = []

  for c in range(cols):
    indegree[c] = 0
    for r in range(rows):
      if AMat[r,c] == 1:
        indegree[c] = indegree[c] + 1

  for i in range(rows):
    j = min([k for k in range(cols)
             if indegree[k] == 0])
    toposortlist.append(j)
    indegree[j] = indegree[j]-1
    for k in range(cols):
      if AMat[j,k] == 1:
        indegree[k] = indegree[k] - 1

  return(toposortlist)
```

An implementation of topological sort

- Compute indegrees by scanning columns of adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Repeat till all vertices are listed

Analysis

- Initializing indegrees is $O(n^2)$

```
def toposort(AMat):
  (rows,cols) = AMat.shape
  indegree = {}
  toposortlist = []

  for c in range(cols):
    indegree[c] = 0
    for r in range(rows):
      if AMat[r,c] == 1:
        indegree[c] = indegree[c] + 1

  for i in range(rows):
    j = min([k for k in range(cols)
             if indegree[k] == 0])
    toposortlist.append(j)
    indegree[j] = indegree[j]-1
    for k in range(cols):
      if AMat[j,k] == 1:
        indegree[k] = indegree[k] - 1

  return(toposortlist)
```

An implementation of topological sort

- Compute indegrees by scanning columns of adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Repeat till all vertices are listed

Analysis

- Initializing indegrees is $O(n^2)$
- Loop to enumerate vertices runs n times
  - Identify next vertex to enumerate: $O(n)$
  - Updating indegrees: $O(n)$

```
def toposort(AMat):
  (rows,cols) = AMat.shape
  indegree = {}
  toposortlist = []

  for c in range(cols):
    indegree[c] = 0
    for r in range(rows):
      if AMat[r,c] == 1:
        indegree[c] = indegree[c] + 1

  for i in range(rows):
    j = min([k for k in range(cols)
             if indegree[k] == 0])
    toposortlist.append(j)
    indegree[j] = indegree[j]-1
    for k in range(cols):
      if AMat[j,k] == 1:
        indegree[k] = indegree[k] - 1

  return(toposortlist)
```

An implementation of topological sort

- Compute indegrees by scanning columns of adjacency matrix
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees
- Repeat till all vertices are listed

Analysis

- Initializing indegrees is $O(n^2)$
- Loop to enumerate vertices runs n times
  - Identify next vertex to enumerate: $O(n)$
  - Updating indegrees: $O(n)$
- Overall, $O(n^2)$

```
def toposort(AMat):
  (rows,cols) = AMat.shape
  indegree = {}
  toposortlist = []

  for c in range(cols):
    indegree[c] = 0
    for r in range(rows):
      if AMat[r,c] == 1:
        indegree[c] = indegree[c] + 1

  for i in range(rows):
    j = min([k for k in range(cols)
             if indegree[k] == 0])
    toposortlist.append(j)
    indegree[j] = indegree[j]-1
    for k in range(cols):
      if AMat[j,k] == 1:
        indegree[k] = indegree[k] - 1

  return(toposortlist)
```

Using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Enumerate head of queue, update indegrees, add indegree 0 to queue
- Repeat till queue is empty

```
def toposortlist(AList):
    (indegree,toposortlist) = ({},[])
    for u in AList.keys():
        indegree[u] = 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()
        toposortlist.append(j)
        indegree[j] = indegree[j]-1
        for k in AList[j]:
            indegree[k] = indegree[k] - 1
            if indegree[k] == 0:
                zerodegreeq.addq(k)
    return(toposortlist)
```

Using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Enumerate head of queue, update indegrees, add indegree 0 to queue
- Repeat till queue is empty

Analysis

```
def toposortlist(AList):
    (indegree,toposortlist) = ({},[])
    for u in AList.keys():
        indegree[u] = 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()
        toposortlist.append(j)
        indegree[j] = indegree[j]-1
        for k in AList[j]:
            indegree[k] = indegree[k] - 1
            if indegree[k] == 0:
                zerodegreeq.addq(k)
    return(toposortlist)
```

Using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Enumerate head of queue, update indegrees, add indegree 0 to queue
- Repeat till queue is empty

Analysis

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

```
def toposortlist(AList):
    (indegree,toposortlist) = ({},[])
    for u in AList.keys():
        indegree[u] = 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()
        toposortlist.append(j)
        indegree[j] = indegree[j]-1
        for k in AList[j]:
            indegree[k] = indegree[k] - 1
            if indegree[k] == 0:
                zerodegreeq.addq(k)
    return(toposortlist)
```

Using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Enumerate head of queue, update indegrees, add indegree 0 to queue
- Repeat till queue is empty

Analysis

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

```
def toposortlist(AList):
    (indegree,toposortlist) = ({},[])
    for u in AList.keys():
        indegree[u] = 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()
        toposortlist.append(j)
        indegree[j] = indegree[j]-1
        for k in AList[j]:
            indegree[k] = indegree[k] - 1
            if indegree[k] == 0:
                zerodegreeq.addq(k)
    return(toposortlist)
```

Using adjacency lists

- Compute indegrees by scanning adjacency lists
- Maintain queue of vertices with indegree 0
- Enumerate head of queue, update indegrees, add indegree 0 to queue
- Repeat till queue is empty

Analysis

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

```
def toposortlist(AList):
    (indegree,toposortlist) = ({},[])
    for u in AList.keys():
        indegree[u] = 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()
        toposortlist.append(j)
        indegree[j] = indegree[j]-1
        for k in AList[j]:
            indegree[k] = indegree[k] - 1
            if indegree[k] == 0:
                zerodegreeq.addq(k)
    return(toposortlist)
```

Summary

- Directed acyclic graphs are a natural way to represent dependencies
- Topological sort gives a feasible schedule that represents dependencies
  - At least one vertex with no dependencies, indegree 0
  - Eliminating such a vertex retains DAG structure
  - Repeat the process till all vertices are listed
- Complexity
  - Using adjacency matrix takes $O(n^2)$
  - Using adjacency list takes $O(m+n)$
- More than one topological sort is possible
  - Choice of which vertex with indegree 0 to list next