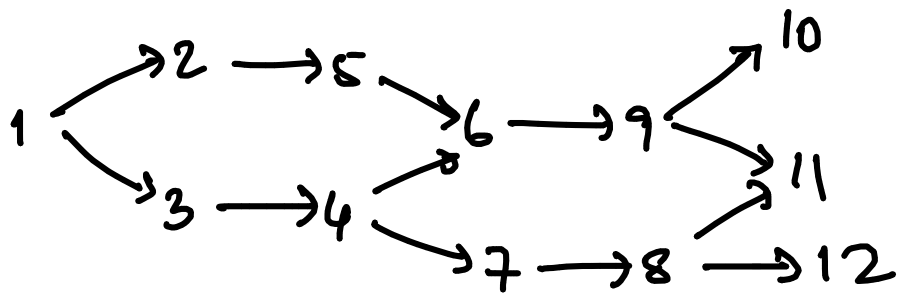
Directed Acyclic Graphs (DAGs)



Topological Sort:

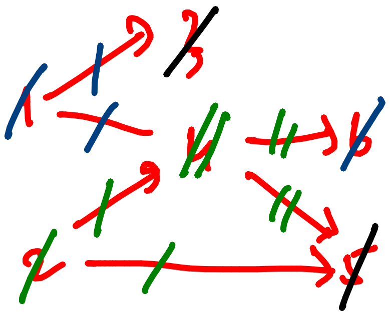
Enumerate (i.e. list out) vertices in an order compatible with edge relation

- if $i \to j$ in G, i appears before j

Can always find a vertex with indegree(0) – can start with such a vertex

Algorithm for topological sort

while G is not empty
  find v with indegree(v) = 0
  enumerate v
  remove v and all edges (v, w)

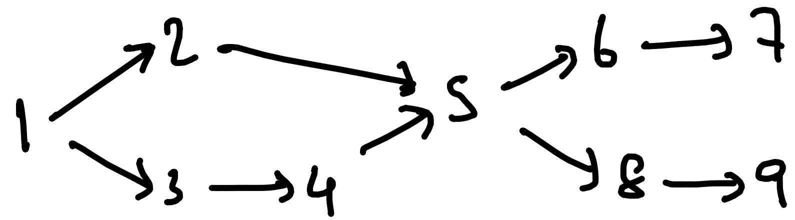


⇒ 2 1 3 4 6 5

Also

1 3 2 4 5 6 etc

Counting the # of legal topological ordering



Start with 1

Orderings of {2,3,4} $\binom{3}{1} = 3$

Then 5

$\times$

Orderings of {6,7,8,9} $\binom{4}{2} = 6$

$= 18$

Hard to compute in general

Efficient implementation:

How to compute indegree(v)?

How to eliminate v & edges (v,w) from G?

Adjacency Matrix

indegree(j) - # 1's in column j

$O(n^2)$ to initialize indegree[1..n]

O(outdegree(j)) to update O(m) overall

delete(j) & edges (j,k)

Make row j 0, Mark[j] = 1

topological-sort (G)

for i in 1..n compute indegree[i] $O(n^2)$
mark[i] = 0

$O(n^2)$

for j in 1..n

find smallest k s.t. mark[k]==0, indegree[k]==0

$O(n)$

print(k)
mark[k] = 1
for l = 1..n $O(n)$

if A[k][l]==1 , A[k][l] = 0
indegree[l] = indegree[l] -1

Can we do better?

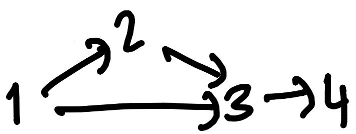
Adjacency list

Compute & update indegree?

Invert the list - $O(m+n)$ time

enough to keep indegree

$1 \to [\cancel{2}, \cancel{3}]$	$1 \leftarrow [\,]$	0
$2 \to [\cancel{3}]$	$2 \leftarrow [1]$	1
$3 \to [\cancel{4}]$	$3 \leftarrow [1,2]$	2
$4 \to [\,]$	$4 \leftarrow [3]$	1



Indegree 0? inlist[i] = []

Explicitly scanning for this costs $O(n)$

Suppose

$1 \to [2,3]$, indegree[2] = 1

After enumerating 1 , indegree[2] = 0 – Make a note of this now!

Keep a queue of all pending indegree 0 nodes

From adjacency list, compute indegree[1..n]

Add each i with indegree[i] == 0 to queue Q

While Q is not empty

  remove & enumerate head

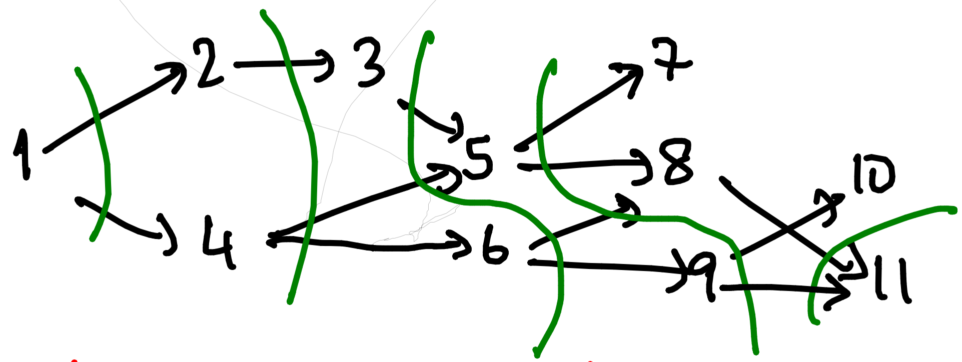
  update indegree & Q

```
better_topo_sort(G)
  for i in 1..n    indegree[i] = 0                            O(n)
  for i in 1..n                                               O(n+m)
      for each (i,j) ∈ E, indegree[j] = indegree[j]+1
  for i in 1..n                                               O(n)
      if indegree[i]==0 ,   Q.append(i)

  While not(Q.is_empty)
      j = Q.extract_head()
      for each (j,k) ∈ E , indegree[k] = indegree[k]-1
                        if indegree[k]==0 , Q.append(k)
```

O(n+m)

Topological sort – sequential order to process tasks



Parallel execution – can do as many tasks as possible in parallel

1 {2,4} {3,6} {5,9} {7,8,10} 11

Compute earliest[j] = earliest step when j can be done

Inductive definition of earliest[i] ?

$$\text{earliest}[j] = \left(\max_{(i,j) \in E} \text{earliest}[i] \right) + 1$$

Can compute earliest[j] if earliest[i] for all its incoming edges is known

Topological sort — when enumerating j,
look up earliest[i] for i in inlist[j]
Across all j, $\sum$ indegree(j) = $O(m)$

Alternatively

Set earliest[i] = 1 for all i with indegree[i] == 0 initially

When we enumerate j

for each (j,k) update earliest[k] to

max (earliest[k], earliest[j]+1)

Avoids inlist, integrate into topo sort

Computing earliest is equivalent to finding longest path to i

Highest value of earliest[ ] gives length of longest path in a DAG

- Path = no repeated vertices

In general graphs – NP-complete