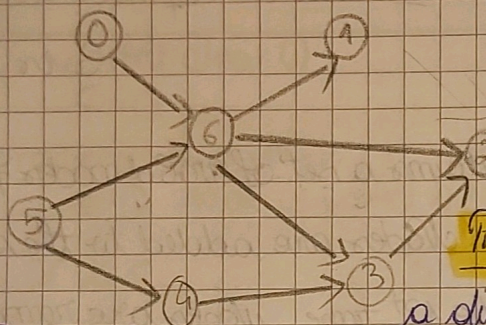
```
if idx[node] == -1
    i = len(stack) - 1
    while i >= 0:
        node = stack[i]
        stack.pop()
        onStack[node] = False
        low[node] = idx[node_id]
        if node == node_id:
            break
        i -= 1
```

## DAG

- TOPOLOGICAL SORTING Complexity: $O(V+E)$
  - ↳ put the vertices in a list s.t. whenever there is an edge $(x,y)$, then $x$ comes before $y$ in that list
  - ↳ the solution is not generally unique




Possible topological sortings:

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

**Property** Topological sorting is possible, for a directed graph, if and only if there are no cycles in the graph.

### PREDECESSOR COUNTING ALGORITHM

↳ we take a vertex with no predecessors, we put it on the sorted list and we eliminate it from the graph. Then, we take a vertex with no predecessors from the remaining graph and continue in the same way.

Finally, we either process all vertices and end up with the topologically sorted list, or we cannot get a vertex with no predecessors, which means we have a cycle.

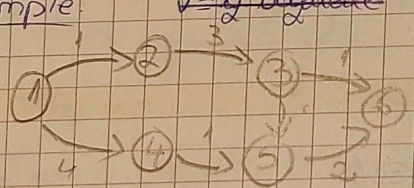
Main idea: don't actually remove vertices, but keep for each vertex, a counter of predecessors still in the graph

Input:
G: directed graph
Output:
sorted: a list of vertices in topological sorting order, or null if G has cycles

Algorithm

```
sorted = emptyList
Queue q
Dictionary count
for x in X do
    count[x] = inDegree(x) // how many predecessors x has
    if count[x] == 0 then // node with no predecessor
        q.enqueue(x)
while not q.isEmpty() do
    x = q.dequeue()
    sorted.append(x)
    for y in Nout(x) do
        count[y] = count[y] - 1
        if count[y] == 0 then
            q.enqueue(y)
if sorted.size() < X.size() then
    sorted = null
else
    ~~for x in X~~
        ~~x = q.dequeue~~
```

Example




Topological sorting orders:

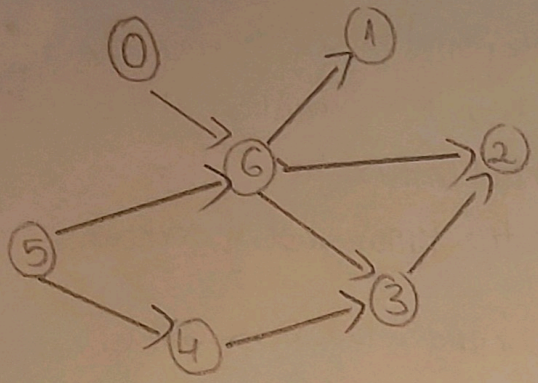
1 2 3 4 5 6
1 2 4 3 5 6
1 4 2 3 5 6

| | x,y | count: dict | q: queue | sorted: list |
|---|---|---|---|---|
| init | | 1 2 3 4 5 6<br>0 1 1 1 2 2 | ← 1 ← | [] |
| it 1 | x=1<br>y=2<br>y=4 | 0 0 1 1 2 2<br>0 0 1 0 2 2 | ← 2 ←<br>← 4 ← | [1] |
| it 2 | x=2<br>y=3 | 0 0 0 0 2 2 | ← 4 3 ← | [1,2] |
| it 3 | x=4<br>y=5 | 0 0 0 0 1 2 | ← 3 ← | [1,2,4] |
| it 4 | x=3<br>y=5<br>y=6 | 0 0 0 0 0 2<br>0 0 0 0 0 1 | ← 5 ←<br>← 5 ← | [1,2,4,3] |
| it 5 | x=5<br>y=6 | 0 0 0 0 0 0 | ← 6 ← | [1,2,4,3,5] |
| it 6 | x=6 | 0 0 0 0 0 0 | ← ← | [1,2,4,3,5,6] |

sizeof(sorted) = 6

TOPOSORT PREDECESSOR COUNT

# Topological sorting using predecessor count




Predecessor count for each vertex

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 2 | 1 | 0 | 2 |

Queue (vertices with no predecessors)

← 0 | 5 ←

| | x, y | count . dict | q: queue | sorted list |
|---|---|---|---|---|
| initialization | | 0 1 2 3 4 5 6<br>0 1 2 2 1 0 2 | ← 0 \| 5 ← | [ ] |
| it 1. | x = 0<br>y = 6 | 0 1 2 3 4 5 6<br>0 1 2 2 1 0 1 | ← 5 ← | [0] |
| it. 2. | x = 5<br>y = 4<br>y = 6 | 0 1 2 3 4 5 6<br>0 1 2 2 0 0 1<br>0 1 2 2 0 0 0 | ← ←<br>← 4 ←<br>← 4 \| 6 ← | [0, 5] |
| it. 3 | x = 4<br>y = 3 | 0 1 2 3 4 5 6<br>0 1 2 1 0 0 0 | ← 6 ← | [0, 5, 4] |
| it 4 | x = 6<br>y = 1<br>y = 2<br>y = 3 | 0 1 2 3 4 5 6<br>0 0 2 1 0 0 0<br>0 0 1 1 0 0 0<br>0 0 1 0 0 0 0 | ← ←<br>← 1 ←<br>← 1 \| 3 ← | [0, 5, 4, 6] |
| it. 5 | x = 1 | | ← 3 ← | [0, 5, 4, 6, 1] |
| it 6 | x = 3<br>y = 2 | 0 1 2 3 4 5 6<br>0 0 0 0 0 0 0 | ← ←<br>← 2 ← | [0, 5, 4, 6, 1, 3] |
| it 7 | x = 2 | | ← ← | [0, 5, 4, 6, 1, 3, 2] |

↳ sizeof (sorted) = 7 = vertexNumber

⇒ A valid topological sorting is: 0 5 4 6 1 3 2