Topological Sorting

By Ralph McDougall

IOI Camp 2

(4/5 February 2017)

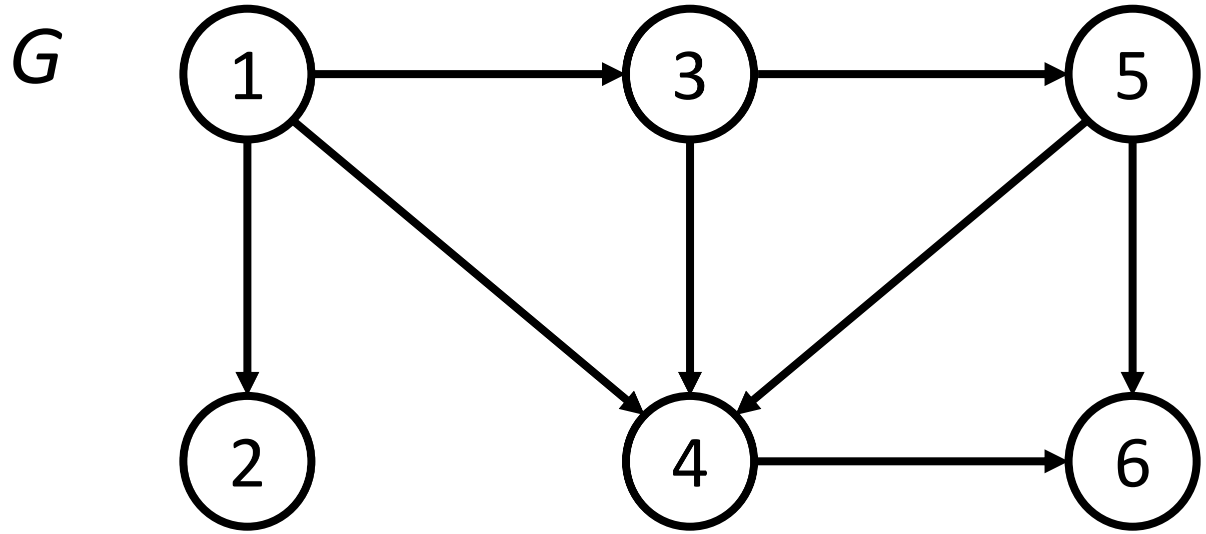
Definition

Given a **directed graph** G with n vertices (1, 2, …, n) and some edges, the n-tuple T is a topological ordering of the vertices of G if and only if:

I[a] < I[b] where ab is an edge from a to b in G and $v \in T$ for all $v \in (1, 2, \ldots, n)$

(T[I[k]] = k for all k in (1, 2, …, n))

Example



T = (1, 3, 2, 5, 4, 6)

Note: there may be multiple topological orderings.

T = (1, 2, 3, 5, 4, 6) is also valid.

Practical Example

A practical example of a topological sorting is a list of tasks that needs to be completed with some tasks having to be completed first. The tasks would be nodes in a graph.

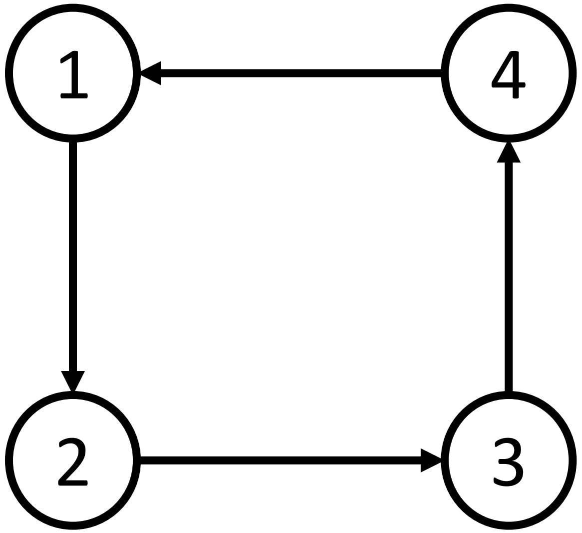
Practical example of practical example:

A cooking recipe.

You need to crack eggs before you can beat them, you must preheat the oven before you put food in it, etc.

Condition

A topological ordering of a graph is possible if and only if the graph does not contain any directed cycles.



(A topological ordering of this graph does not exist.)

Algorithms

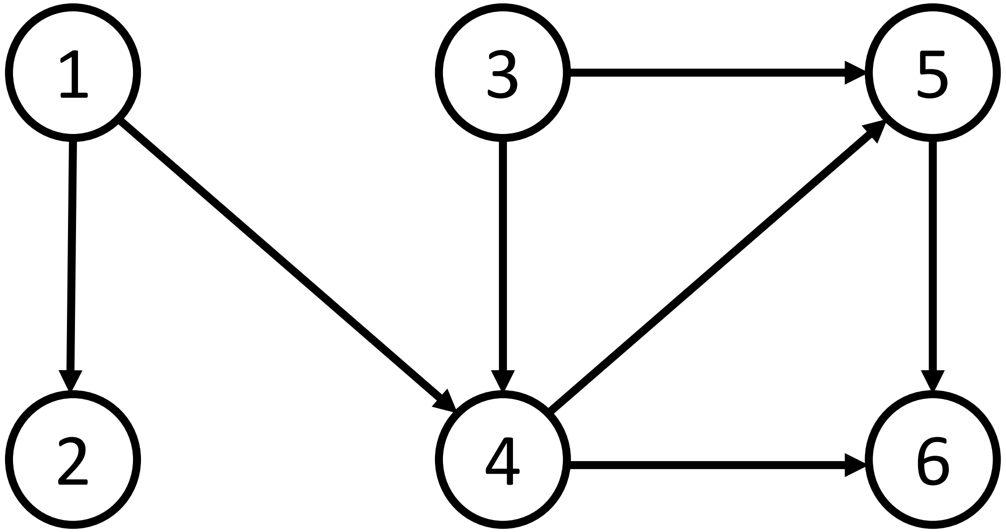
There are 2 main algorithms for finding the topological order of a graph:

1. Kahn’s Algorithm
2. DFS

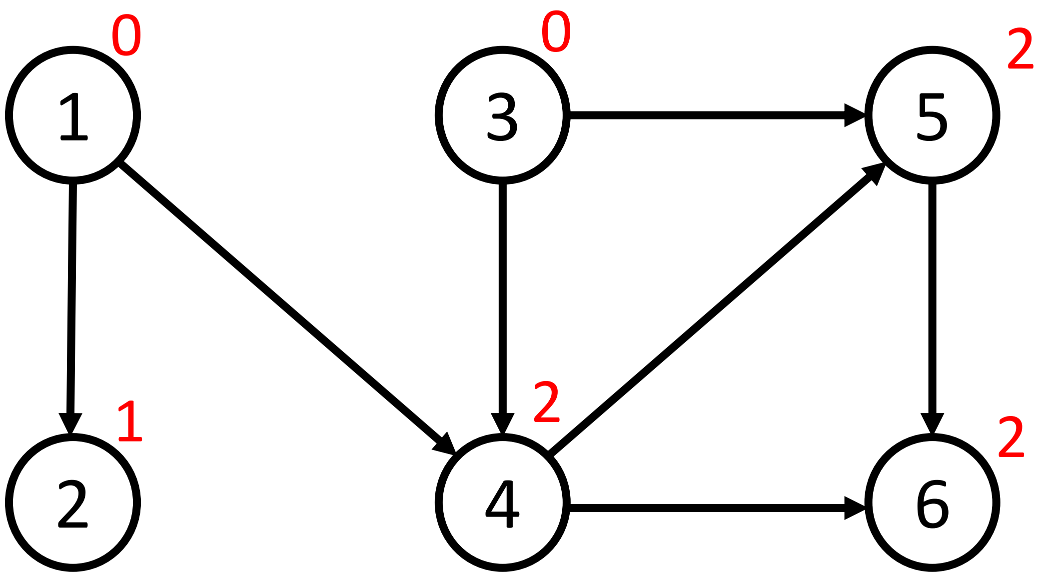
Kahn’s Algorithm

1. Compute the in-degree of each vertex
2. Add all of the vertices with an in-degree of 0 onto a queue *Q*
3. Remove a vertex *V* from *Q*
4. Increment the counter of visited nodes by 1
5. Add *V* onto *T* where *T* is the list that will contain the order of the nodes
6. Decrease the in-degree of all neighbours of *V* by 1
7. If a neighbour now has an in-degree of 0, add it to *Q*
8. If *Q* is not empty, go to step 3
9. If the vertex counter does not equal the total number of vertices, return ERROR (the topological ordering does not exist)
10. Return *T*

Visual

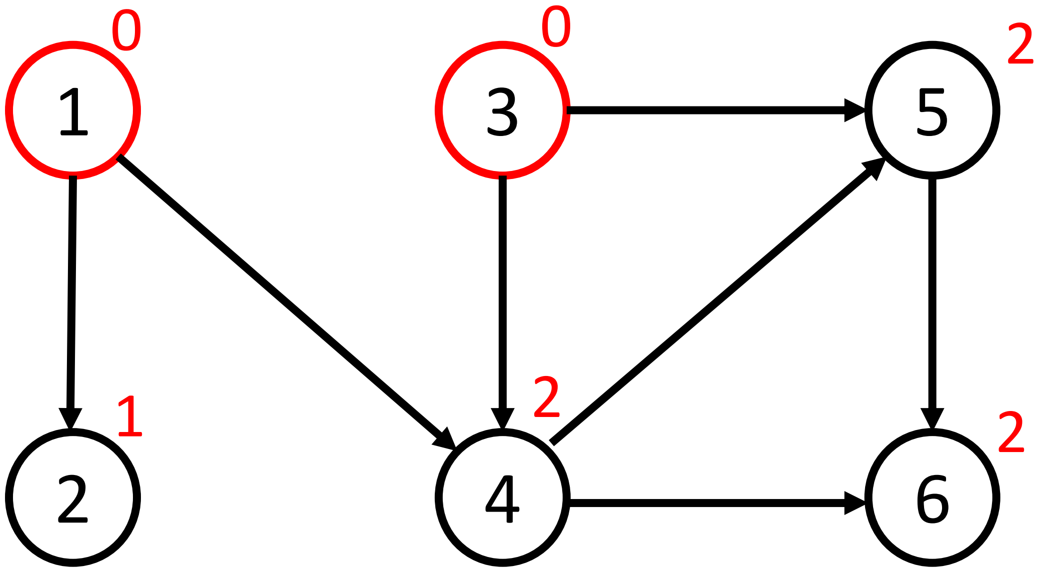


Visual



Compute the in-degree of each vertex.

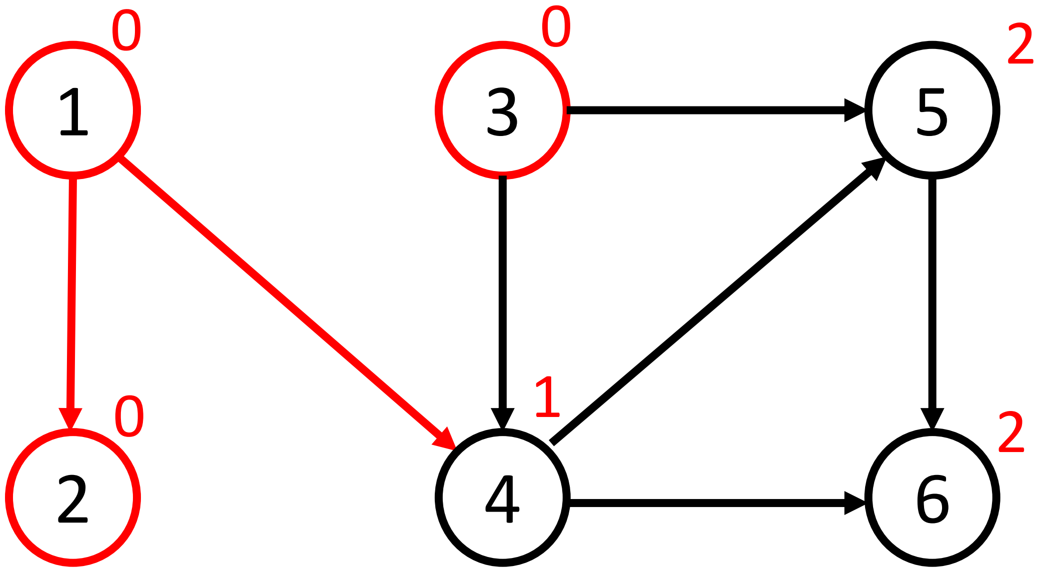
Visual



Q = (1, 3)
T = ()

Put the vertices with in-degree onto a queue.

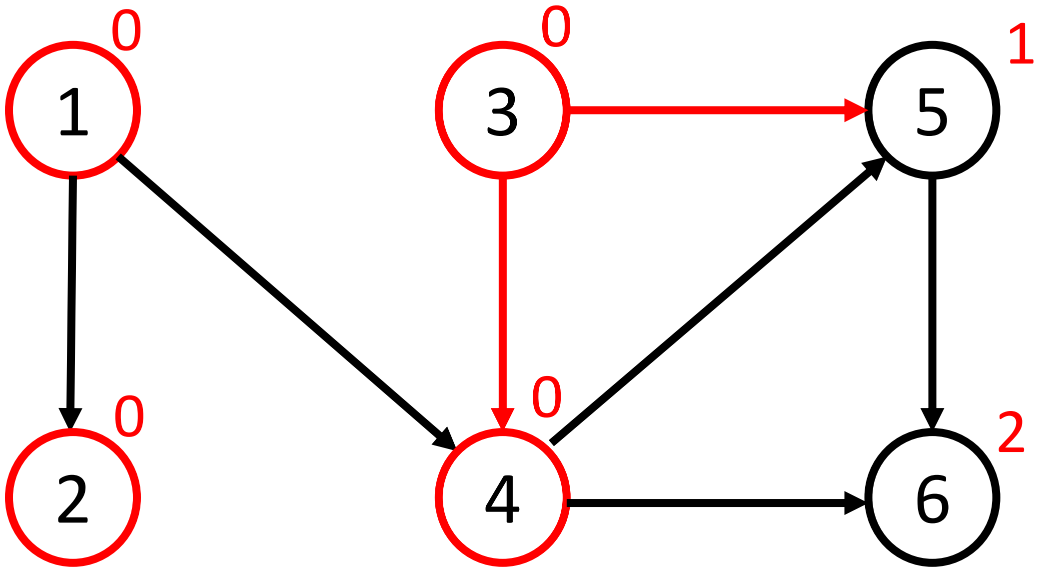
Visual



Add a vertex to T and reduce the in-degree of its neighbours by 1.

Q = (3, 2)
T = (1)

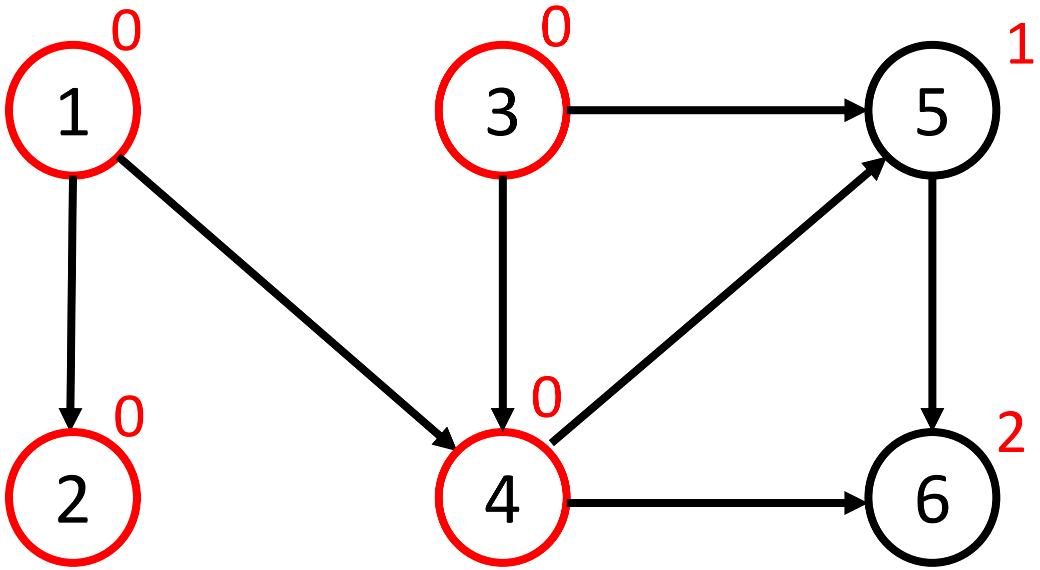
Visual



Add a vertex to T and reduce the in-degree of its neighbours by 1.

Q = (2, 4)
T = (1, 3)

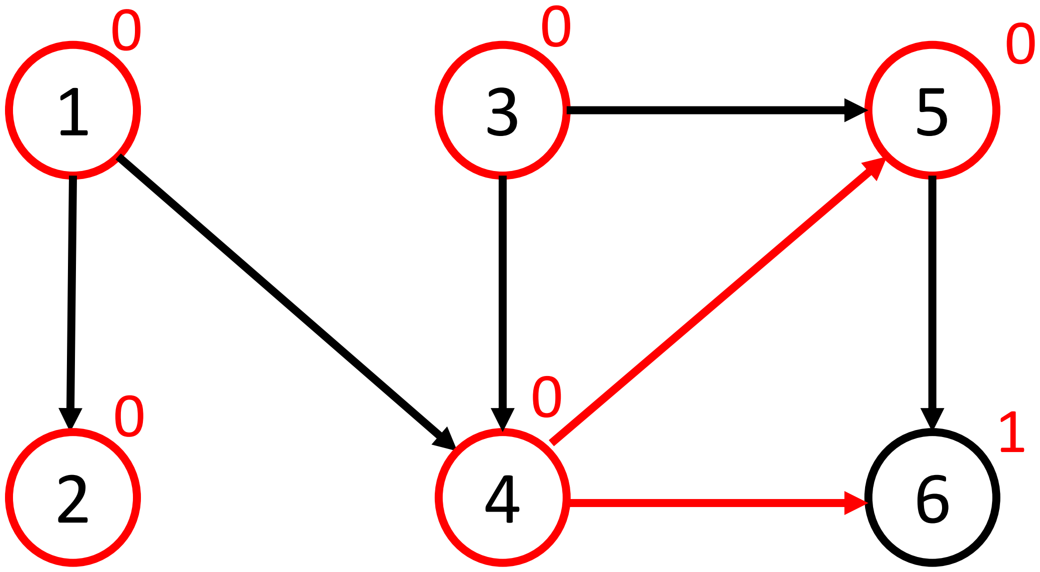
Visual



Add a vertex to T and reduce the in-degree of its neighbours by 1.

Q = (4)
T = (1, 3, 2)

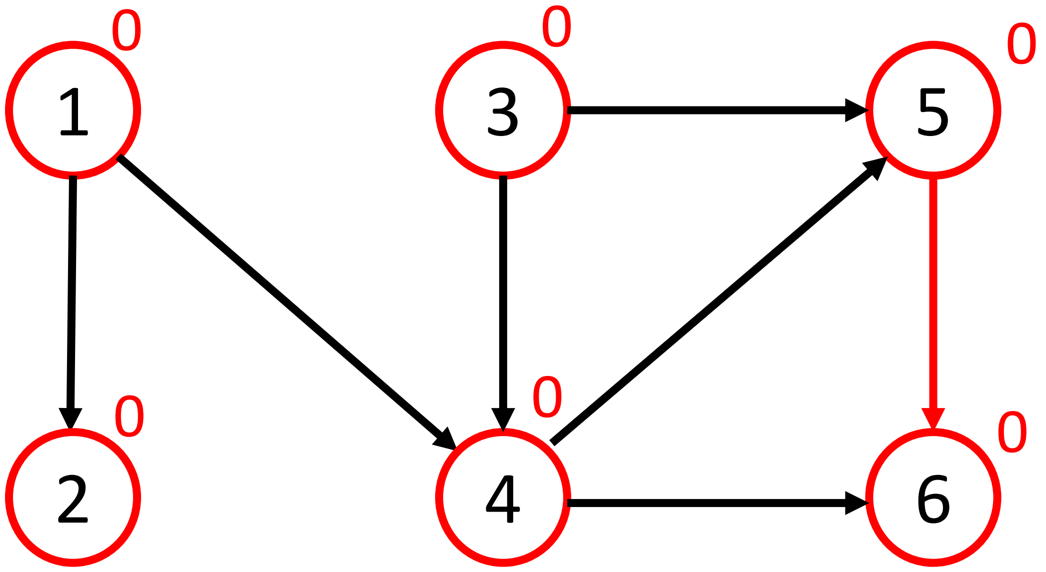
Visual



Add a vertex to T and reduce the in-degree of its neighbours by 1.

Q = (5)
T = (1, 3, 2, 4)

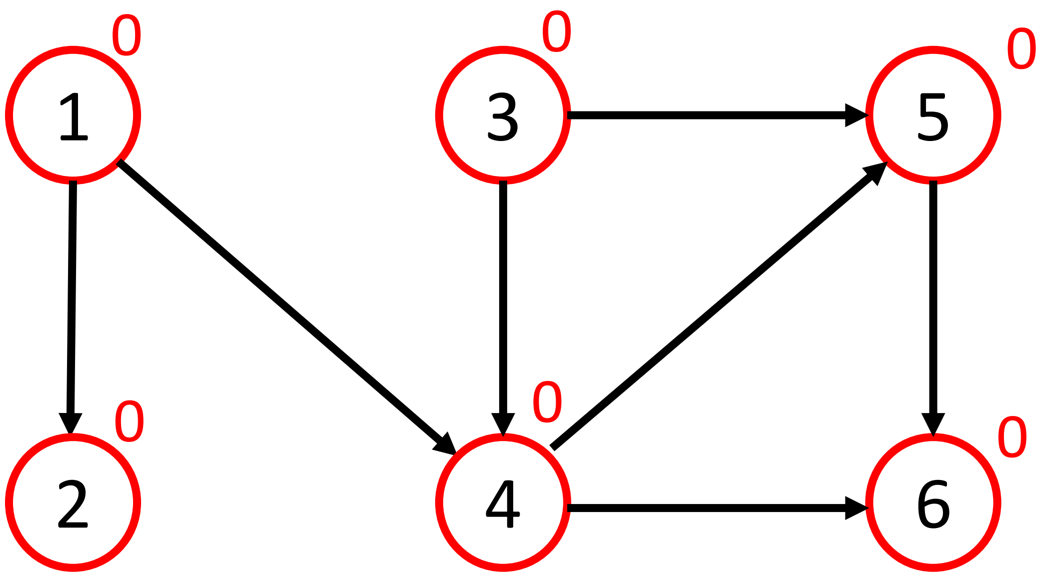
Visual



Add a vertex to T and reduce the in-degree of its neighbours by 1.

Q = (6)
T = (1, 3, 2, 4, 5)

Visual



Return T.

Q = ()
T = (1, 3, 2, 4, 5, 6)

Pseudocode

```
def topological_order(G, n):
    T = []
    in_degree = [0 for i in range(n)]
    Q = Queue.Queue()

    for v in range(n):
        in_degree[v] = len(G[v])
        if in_degree[v] == 0:
            Q.put(v)
    vertex_counter = 0
```

(Continued on next slide)

Pseudocode Continued

```
while not Q.empty():
      v = Q.get()
      vertex_counter += 1

      for neighbour in G[v]:
         in_degree[neighbour] -= 1
         if in_degree[neigbour] == 0:
            Q.put(neighbour)
      T.append(v)

   if vertex_counter != n:
      return []

   return T
```

Kahn’s Algorithm

- Time complexity: O(V + E)

You go through each vertex once and you check each edge once.

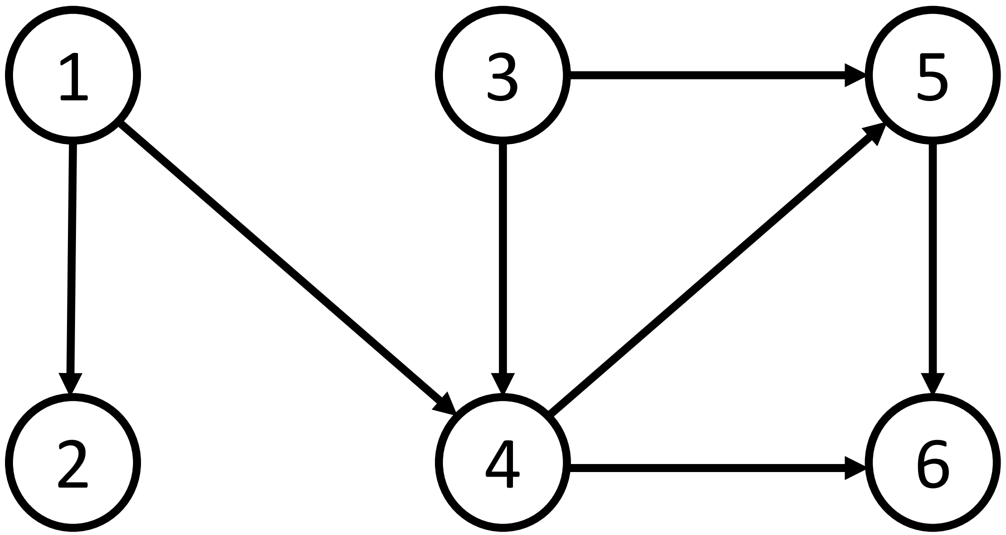
- Space complexity: O(V + E)

You only need a list containing the vertices and a list containing the edges.

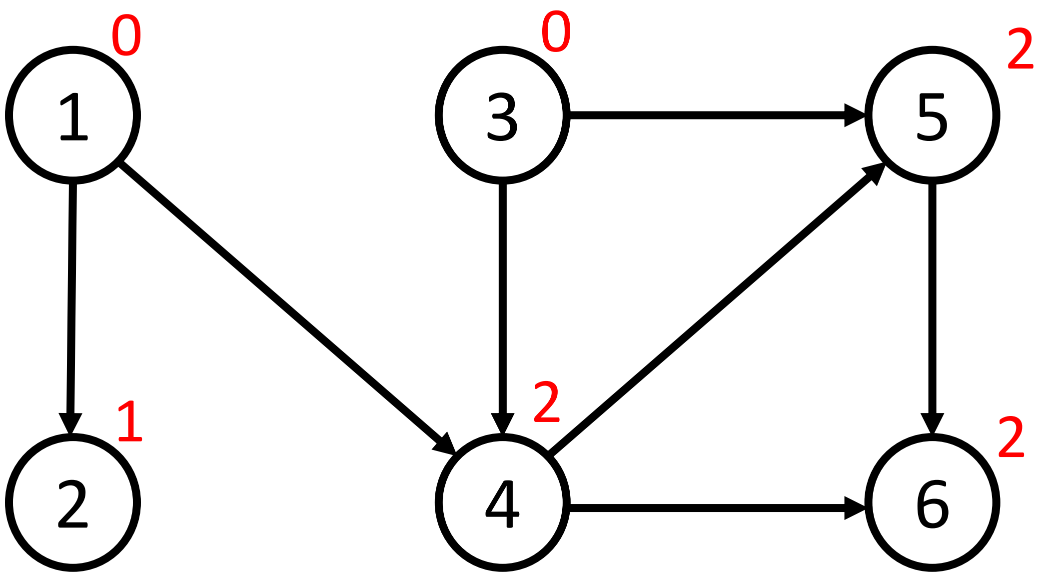
DFS

1. Add all vertices with an in-degree of 0 onto a list
2. Take each vertex in the list one at a time
3. Go to all of it’s neighbours
4. If it doesn’t have any neighbours that are unvisited, add it onto the front of T, mark it as visited and go back
5. Else, go to Step 3
6. When all vertices of the graph have been visited, return T

Visual



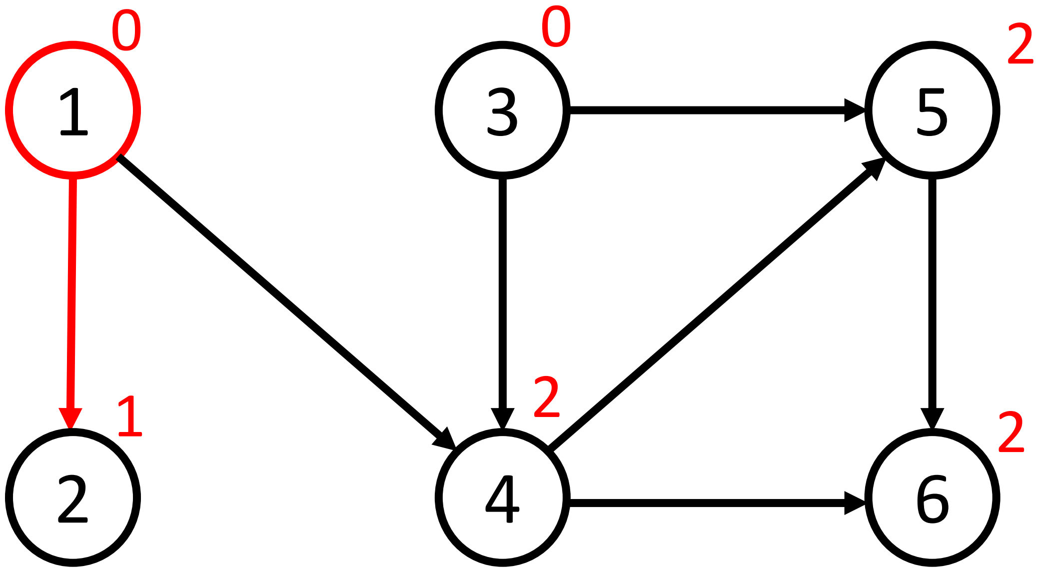
Visual



Compute the in-degree of each vertex.

T = ()
Q = (1, 3)

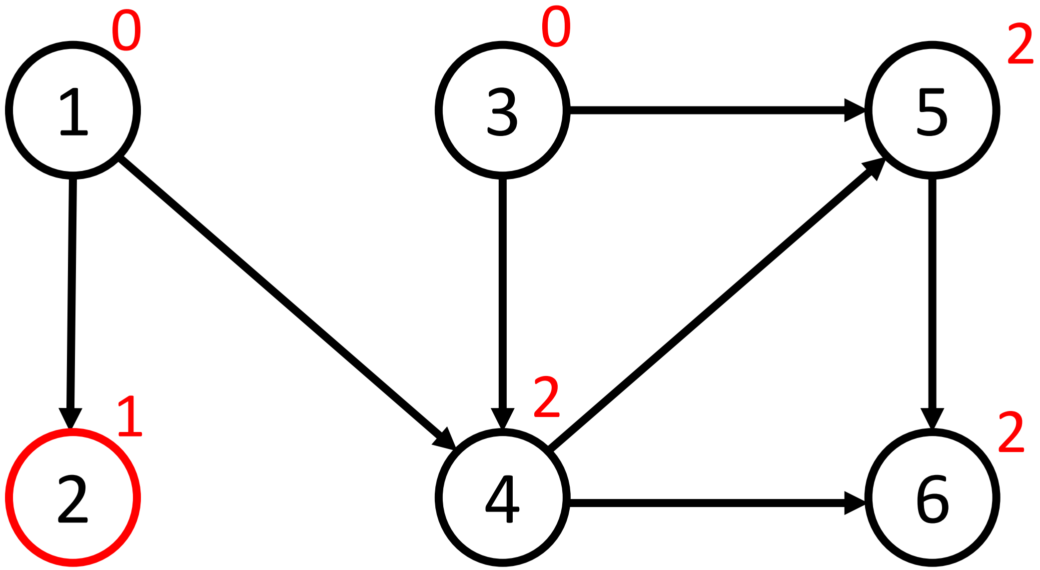
Visual



Go to a neighbour.

T = ()
Q = (1, 3)
Current = 1

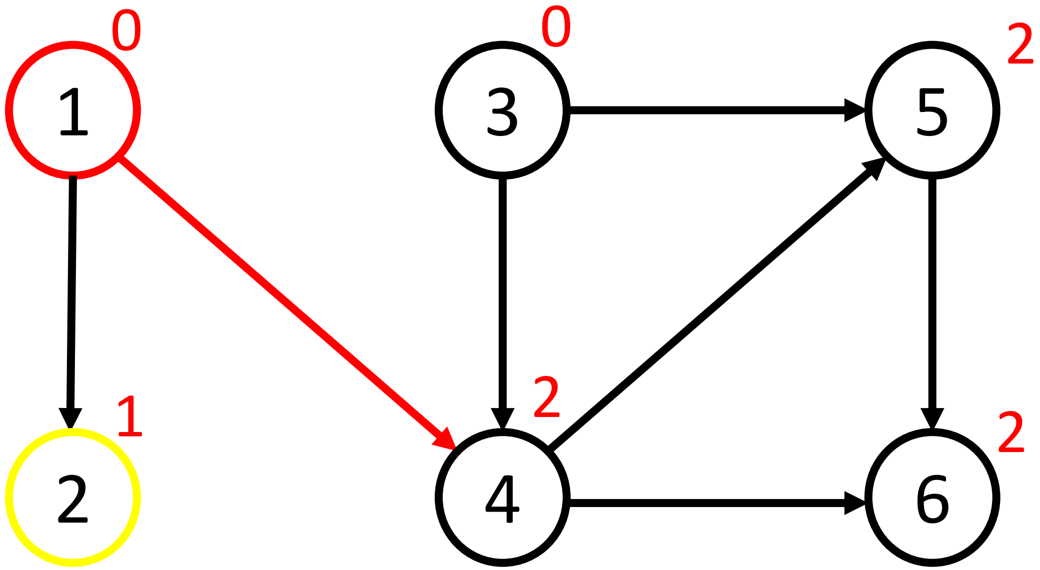
Visual



No unvisited neighbours.

T = (2)
Q = (1, 3)
Current = 2

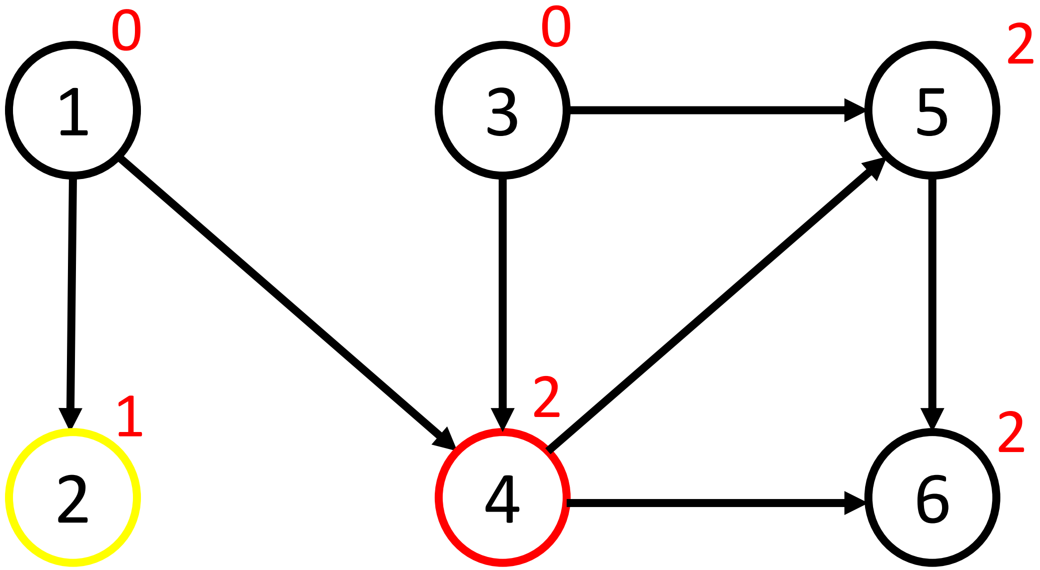
Visual



Go back.

T = (2)
Q = (1, 3)
Current = 1

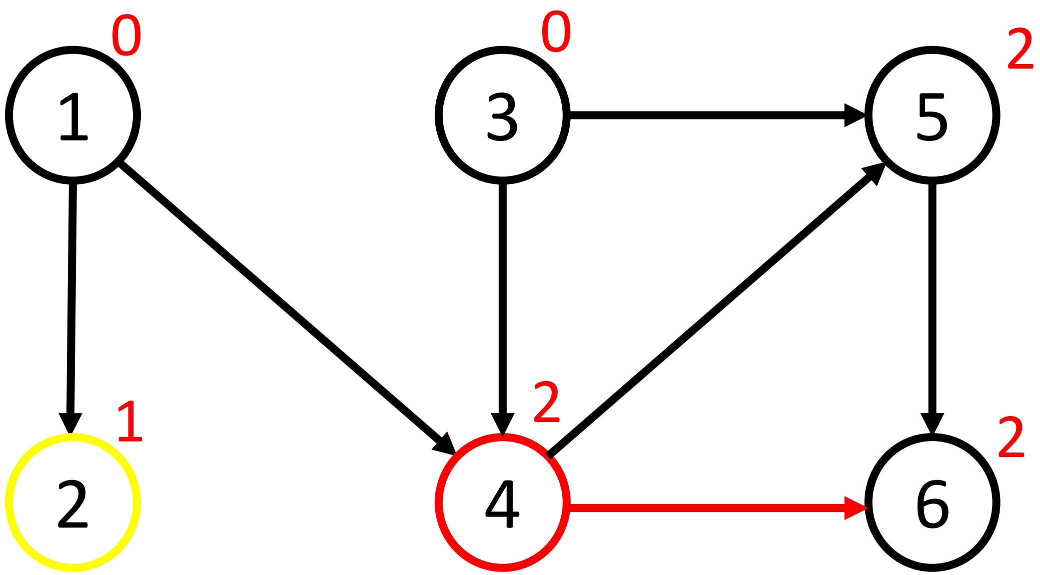
Visual



Go to a neighbour.

T = (2)
Q = (1, 3)
Current = 4

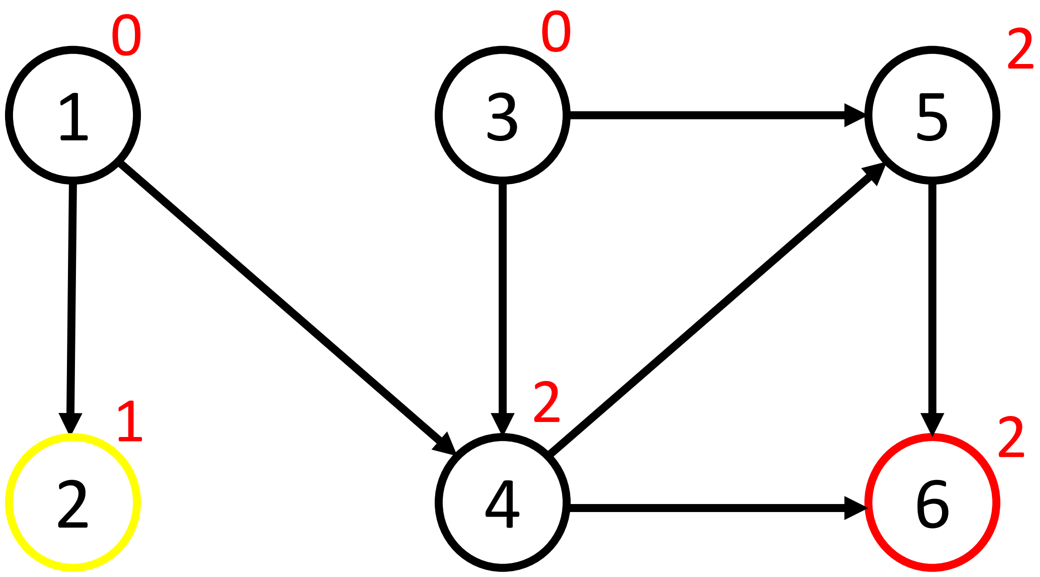
Visual



Go to a neighbour.

T = (2)
Q = (1, 3)
Current = 4

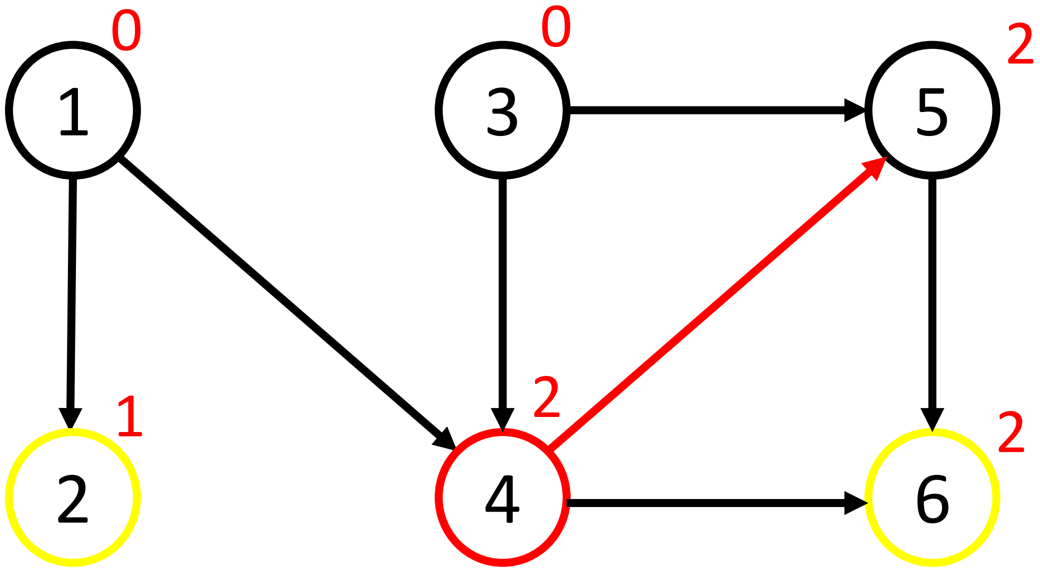
Visual



No unvisited neighbours.

T = (2)
Q = (1, 3)
Current = 6

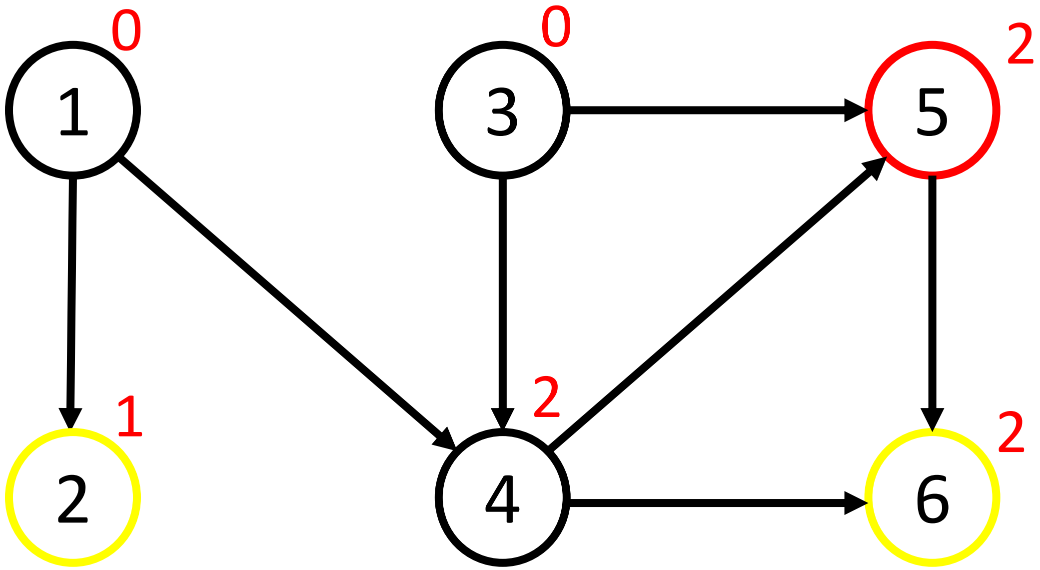
Visual



Go back and go to a neighbour.

T = (6, 2)
Q = (1, 3)
Current = 4

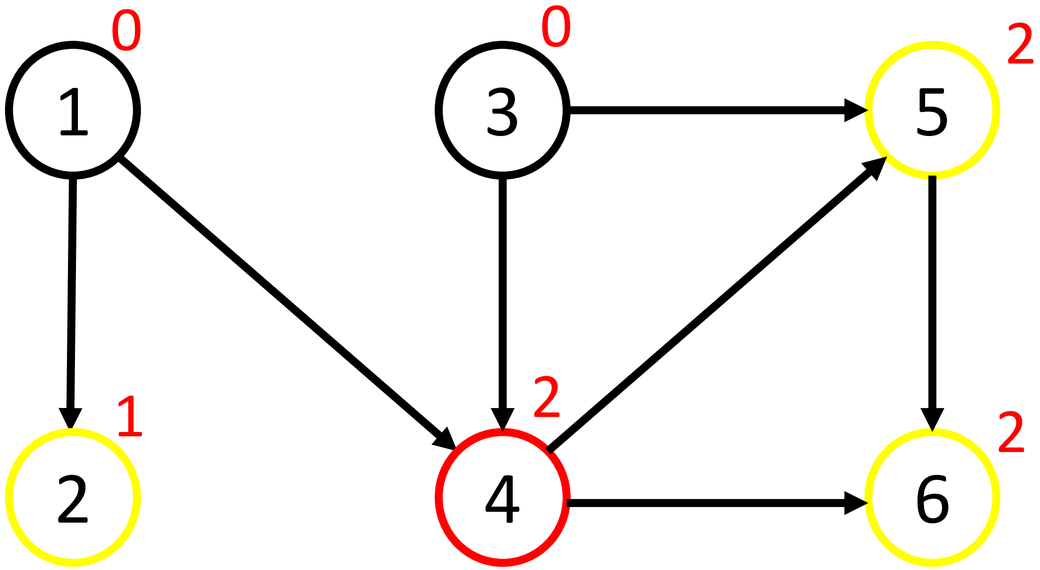
Visual



No unvisited neighbours.

T = (6, 2)
Q = (1, 3)
Current = 5

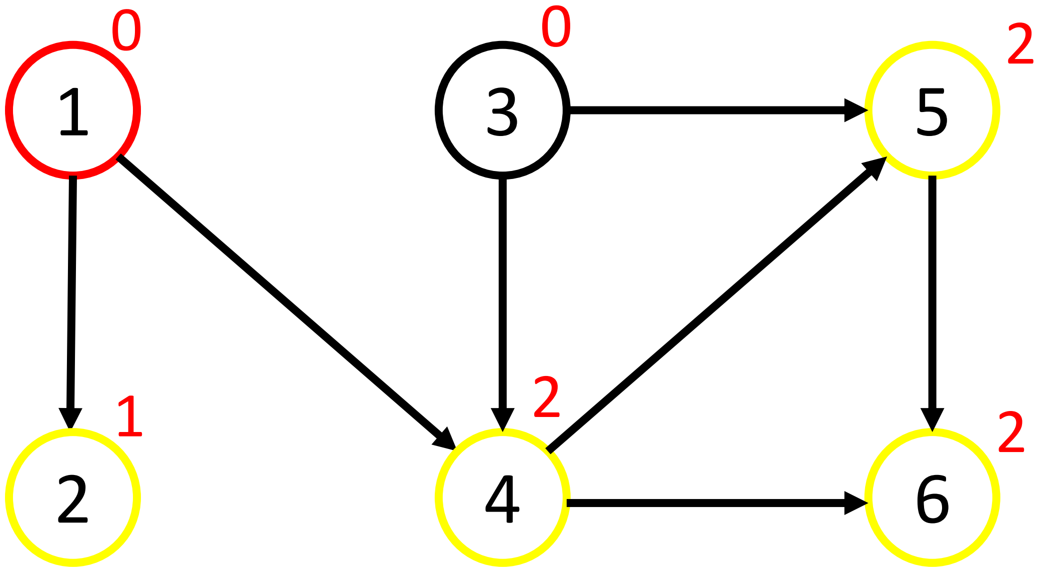
Visual



Go back.
No unvisited neighbours.

T = (5, 6, 2)
Q = (1, 3)
Current = 4

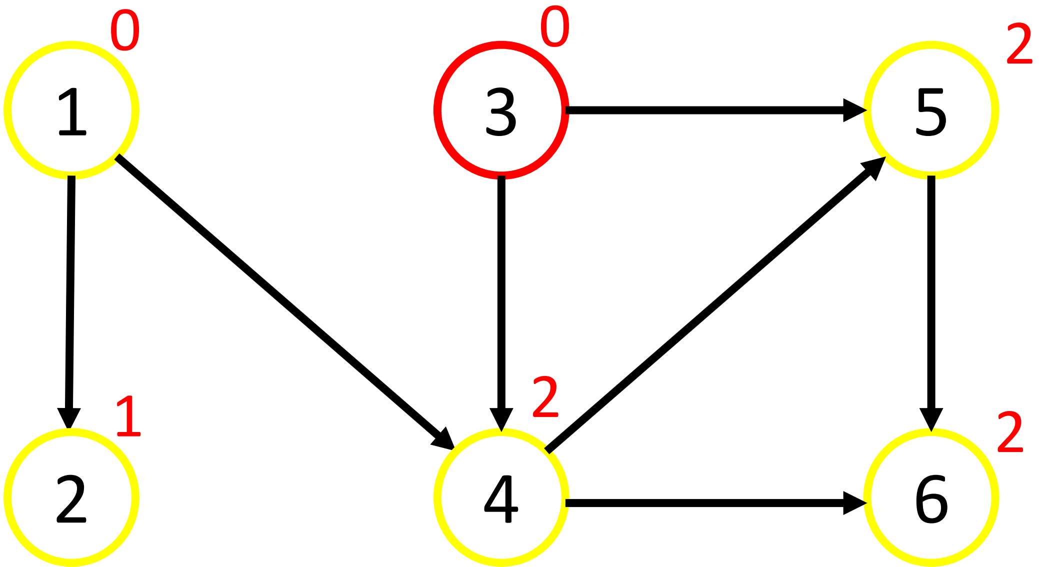
Visual



Go back.
No unvisited neighbours

T = (4, 5, 6, 2)
Q = (1, 3)
Current = 1

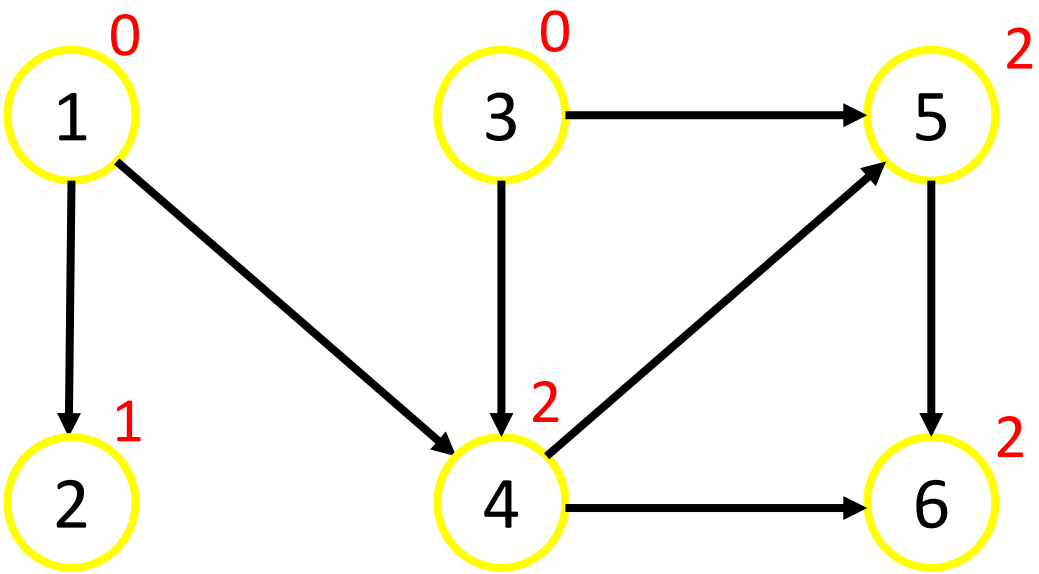
Visual



Go to next starting vertex.

T = (1, 4, 5, 6, 2)
Q = (1, 3)
Current = 3

Visual



No unvisited neighbours.

T = (3, 1, 4, 5, 6, 2)
Q = (1, 3)

Pseudocode

```
from collections import deque

def topological_order(G, T, visited, current):

    for neighbour in G[current]:
        if visited[neighbour]:
            continue

        topological_order(G, T, visited, neighbour)

    visited[current] = True
    T.appendleft(current)
```

(Continued on next slide)

Pseudocode Continued

```
T = deque()
G = [ CONNECTIONS GO HERE ]
visited = [False for i in range(num)]
for v in starter_vertices:
    topological_order(G, T, visited, v)
```

DFS

- Time complexity: O(V + E)

You traverse each edge once and you check each vertex once to find the in-degree of each vertex.

- Space complexity: O(V + E)

You need a list of edges and a few lists of the vertices (technically it’s 3V + E)

Example Problem

Given a list of lectures that John wants to attend and the prerequisite lectures for each lecture, construct a list of the order in which John should attend the lectures.

The lectures are vertices of a directed graph, and the prerequisites are directed edges of the graph. The topological sorting of the graph provides the list that is required.