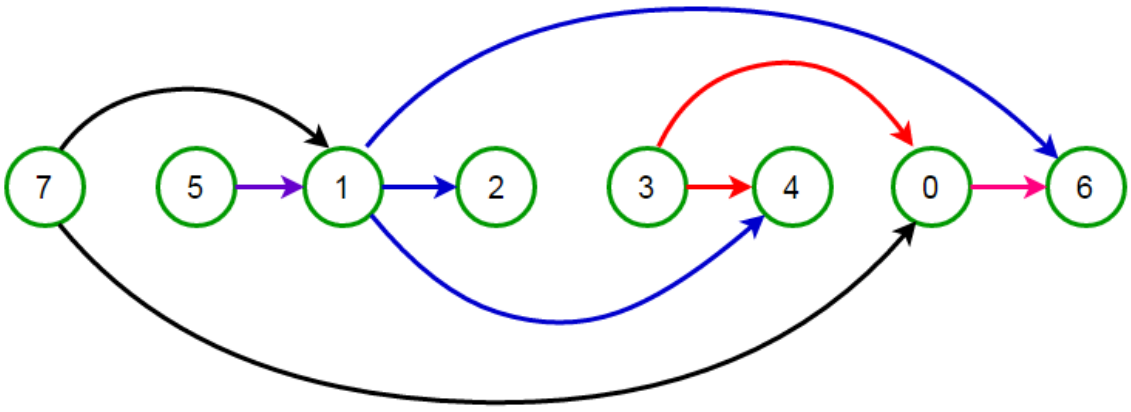
Topological Sort Algorithm for DAG

Given a Directed Acyclic Graph (DAG), print it in topological order using topological sort algorithm. If the graph has more than one topological ordering, output any of them. Assume valid Directed Acyclic Graph (DAG).

A **Topological sort** or Topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge `uv` from vertex `u` to vertex `v`, `u` comes before `v` in the ordering. A topological ordering is possible if and only if the graph has no directed cycles, i.e. if the graph is DAG.

For example, consider the following graph:



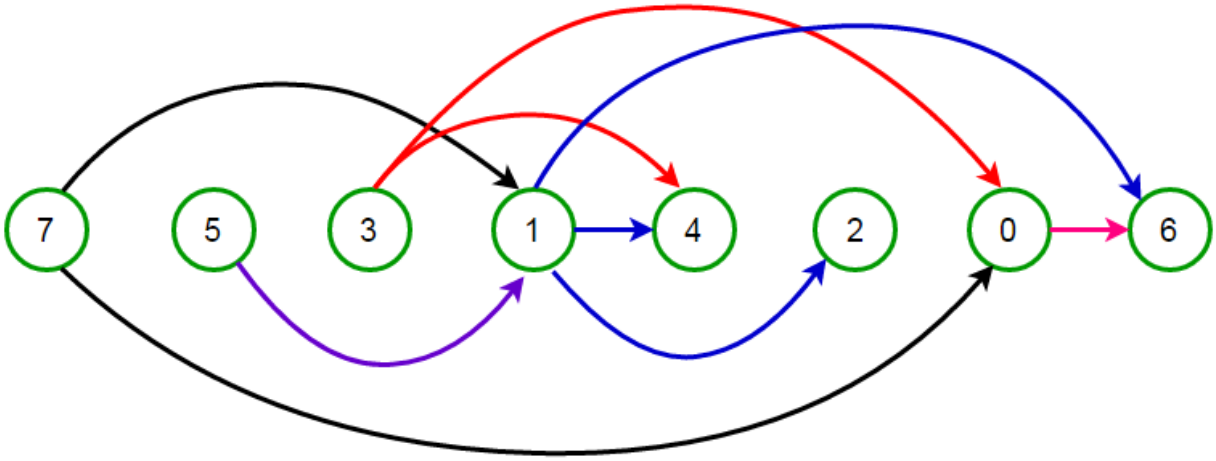
Topological Order

The graph has many valid topological ordering of vertices like,

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

```
3   7   0   5   1   4   2   6
…
etc.
```

Note that for every directed edge `u –> v`, `u` comes before `v` in the ordering.



Topological Order

Practice This Problem

We can use **Depth–first search (DFS)** to implement topological sort algorithm. *The idea is to order the vertices in order of their decreasing **departure time of vertices in DFS**, and we will get our desired topological sort.*

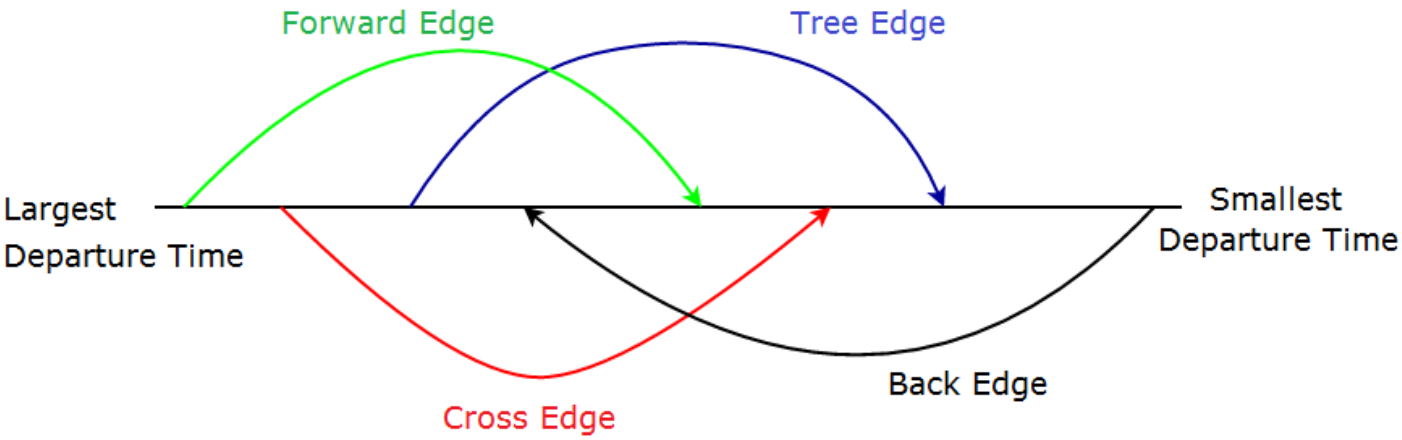
How does it work?

We have already discussed the relationship between all four types of edges involved in the DFS in the **previous post**. Following is the relationship we have seen between the departure time for different types of edges involved in a DFS of the directed graph:

```
Tree edge (u, v): departure[u] > departure[v]
Back edge (u, v): departure[u] < departure[v]
Forward edge (u, v): departure[u] > departure[v]
Cross edge (u, v): departure[u] > departure[v]
```

As we can see that for a tree edge, forward edge, or cross edge `(u, v)`, `departure[u]` is more than `departure[v]`. But only for the back edge, relationship `departure[u] < departure[v]` is true. So, it is guaranteed that if an edge `(u, v)` has `departure[u] > departure[v]`, it's not a back-edge.

We know that **in a DAG, no back-edge is present**. So if we order the vertices in order of their decreasing departure time, we will get the topological order of the graph (**every edge going from left to right**).



Following is the C++, Java, and Python implementation of the topological sort algorithm:

C++

Java

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

// A class to store a graph edge
class Edge
{
    int source, dest;

    public Edge(int source, int dest)
    {
        this.source = source;
        this.dest = dest;
    }
}

// A class to represent a graph object
class Graph
{
    // A list of lists to represent an adjacency list
    List<List<Integer>> adjList = null;

    // Constructor
    Graph(List<Edge> edges, int n)
```