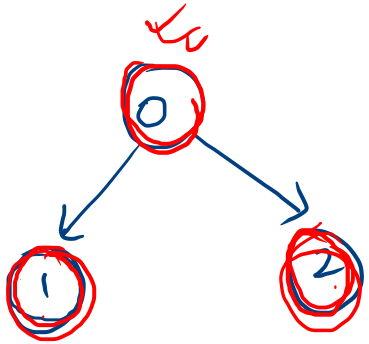
Topological Sort

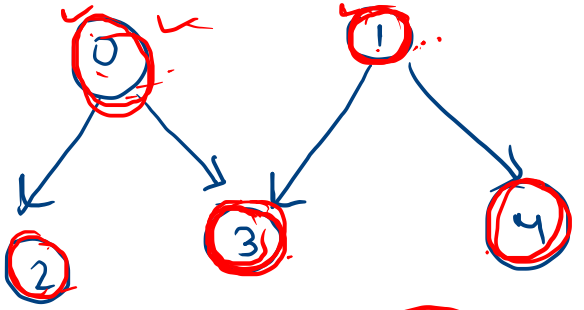
(Kahn's Algo

I/p:



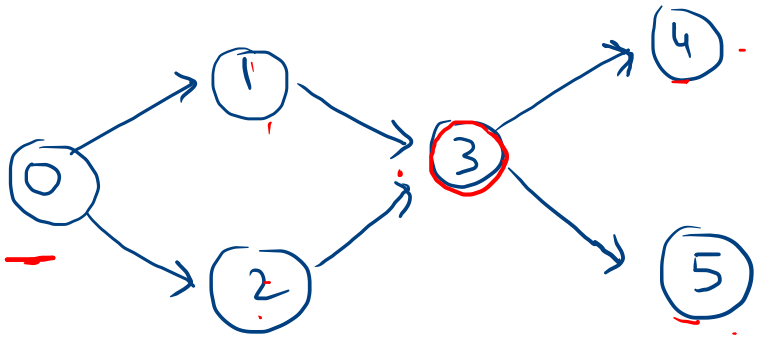
o/p: 0 1 2
0 2 1

I/p:



o/p: 0 1

I/p:

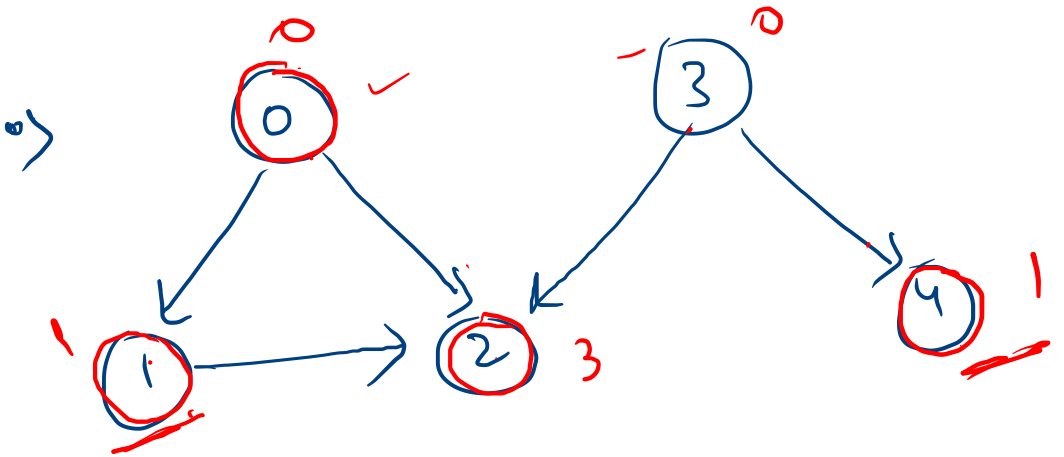


o/p: 0 1 2 3 4 5

⇒ Less dependent on vertex write first.] { conclusion }

⇒ [Less dependent → means → Indegree of vertex]

⇒



0 3 4 1 2

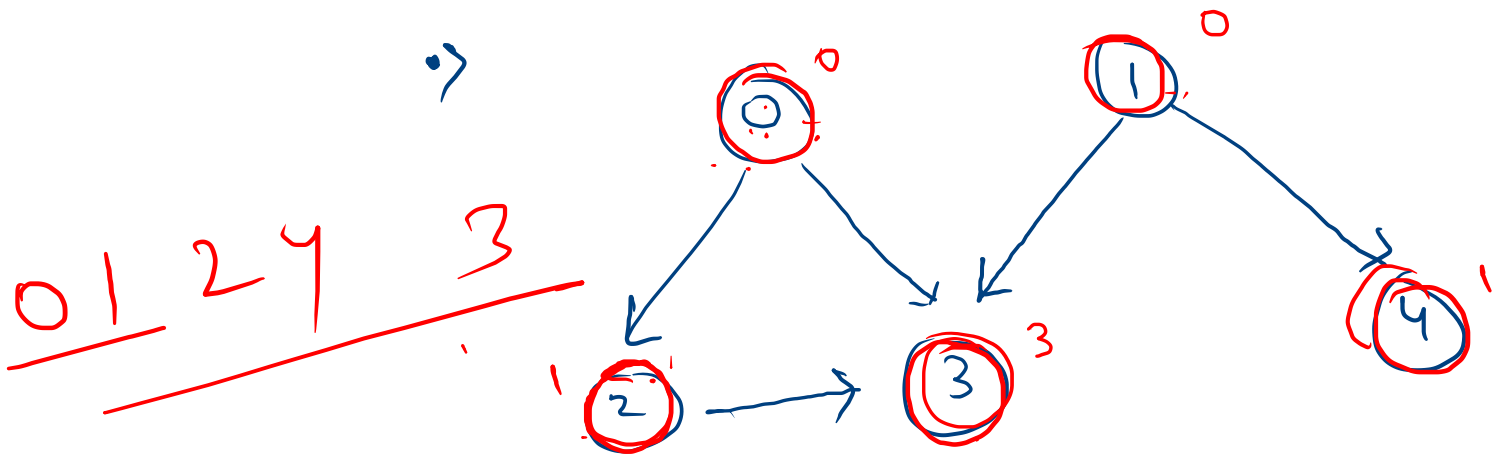
0 3 1 4 2

0 3 1 4 2

=> Simple, => Those who have lesser Indegree write first / choose first

=> Graph Traversal

=>

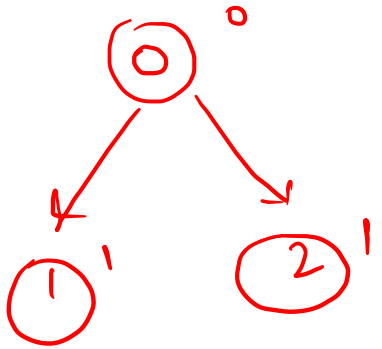


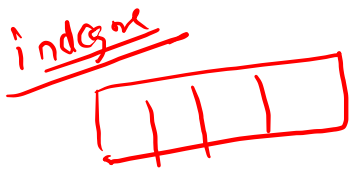
0 1 2 4 3

Ans: 0 1 2 4 3

2 / 0 1

ALGO OF TOPOLOGICAL SORT



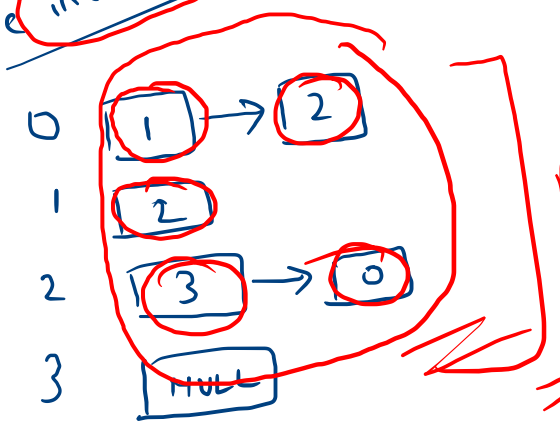


BFS (Algo)

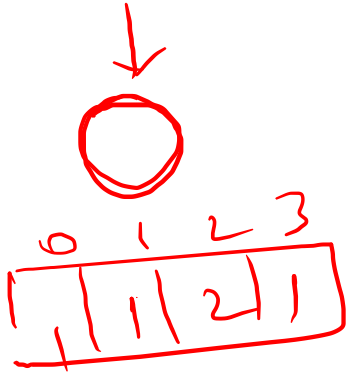
1. Store Indegree of Each Vertex
2. Create a queue, q
3. Store 0 indegree vertex to q
4. while (q is not empty)

```
{
    int u = q.pop()
    print 'u'
    for every adjacent V of u
        1) reduce indegree of V
        2) if indegree of V is 0
             add it to q
}
```

How to Find / Calculate indegree



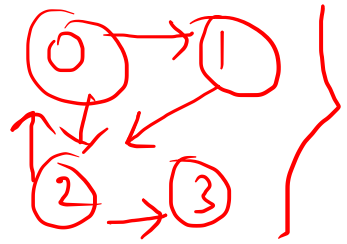
0→1 0→2 1→2 2→3 2→0



indgree[1]++
indgree[2]++
indgree[2]++

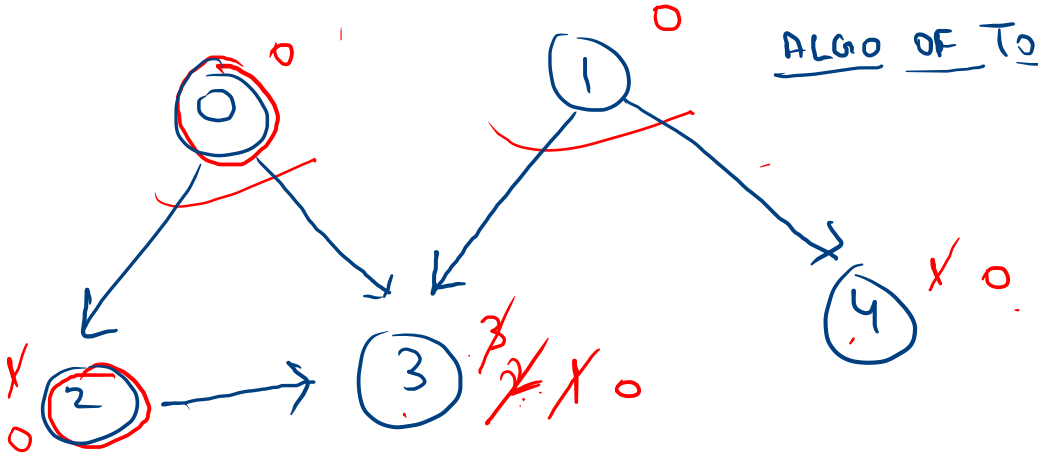
ALGO OF TOPOLOGICAL SORT

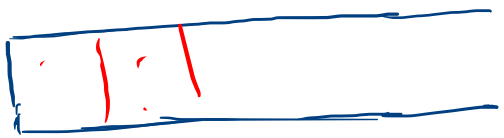
1. Store Indegree of Each Vertex
2. Create a queue, q
3. Store 0 indegree vertex to q
4. while (q is not empty)



{
- int u = q.pop()
- print 'u'
- for every adjacent V of u
  1) reduce indegree of V
  2) if indegree of V is 0
     add it to q

}



o/p: 0 1 2 4 3 | q = 

ALGO OF TOPOLOGICAL SORT

1. store Indegree of Each Vertex
2. Create a queue, q
3. Store 0 indegree vertex to q
4. while (q is not empty)

{

: int u = q.pop()

: print 'u'

: for every adjacent V of u

[1) reduce indegree of V]

[2) if indegree of V is 0 add it to q]

}