
Topological
Sort

01

02

03

Topological Sorting

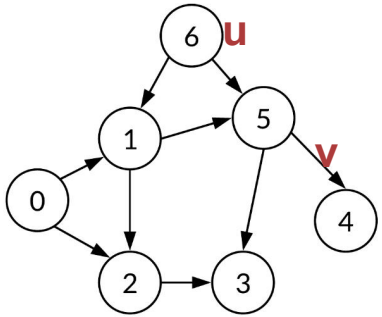
Topological Sort

- It is a Linear ordering of its vertices such that for every directed edge uv for Vertex u to v , u comes before vertex v in the ordering.
- Graph Should be Directed Acyclic Graph (DAG)
- Every DAG will have atleast one Topological Ordering

Vertex: Consider all nodes or vertices as Jobs

Edges: Consider all directed edges as dependencies

Rule: Parent Job must finish before children jobs



›

‹

01

02

03

Topological Sort

What is Directed Acyclic Graph(DAG)

- All rooted trees have a topological ordering since the do not contain any cycle.
- Topological Ordering
    - Picking from the bottom once root of subtree has create out children the procedure continues so that we have no node”

- In DFS, we print a vertex and then recursively call DFS for its adjacent vertices.
- In topological sorting, we need to print a vertex before its adjacent vertices.

›

‹

01

02

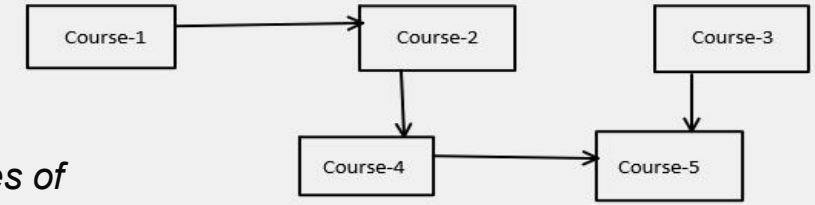
03

Real Life Example

Topological Sort

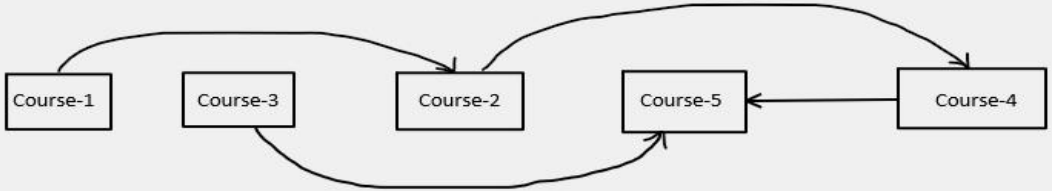
Course Schedule problem

Prerequisites of course:



There are some courses and they may have some prerequisite courses. One can finish courses in some order.

A possible topological ordering:



iq.opengenus.org

Some other applications of the topological sort are manufacturing workflows, project management technique, context-free grammar

›

‹

01

02

03

Algorithm of Topological Sort

Topological Sort

In topological sorting,

- We use a temporary stack.
- We don't print the vertex immediately,
- We first recursively call topological sorting for all its adjacent vertices, then push it to a stack.
- Finally, print the contents of the stack.

Note:
A vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in the stack

›

‹

01

02

03

Pseudocode of Topological Sort

Topological Sort

```
topoSort(int u, bool visited[], stack<int>&stk) {
   visited[u] = true;                //set as the node v is visited
   for(int v = 0; v<NODE; v++) {
      if(graph[u][v]) {              //for all vertices v adjacent to u
         if(!visited[v])
            topoSort(v, visited, stk);
      }
   }
   stk.push(u);     //push starting vertex into the stack
}
```

›

‹

01

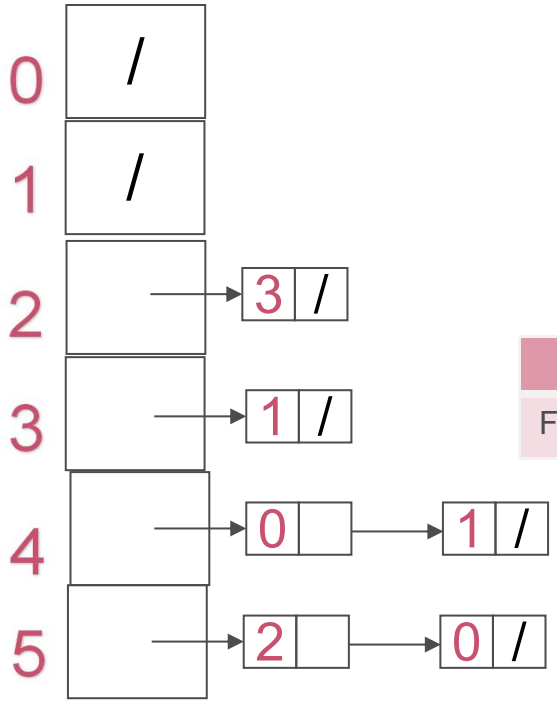
02

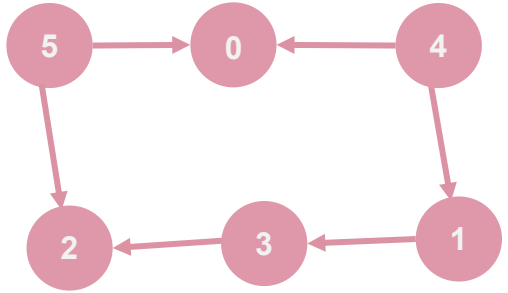
03

Working

Topological Sort

An adjacency list (based on In-degree)





0	1	2	3	4	5
False	False	False	False	False	False

An Empty Stack

01

02

03

Topological Sort

Step -1

Topological Sort (0) , Visited [0] = true
Adjacent List is Empty No More recursion CALL

Step -2

Topological Sort (1) , Visited [1] = true
Adjacent List is Empty No More recursion call

Step -3

Topological Sort (2) , Visited [2] = true

Topological Sort (3) , Visited [3] = true
“1” is already Visited No More recursion call

›

‹

01

02

03

Topological Sort

Step -4

Topological Sort (4) , Visited [4] = true
“0” ,“1” are already Visited No More recursion call

0	1	3	2	4	

Step -5

Topological Sort (5) , Visited [5] = true
“2” ,“0” are already Visited No More recursion call

0	1	3	2	4	5

Step -6

Topological Sort of the given graph

↓

Print all elements of stack from top to bottom

5	4	2	3	1	0

›

‹

01

02

03

Topological Sort

Complexities

Time Complexity: O(V + E) where V = Vertices, E = Edges.

- To determine the in-degree of each node, we will have to iterate through all the edges of the graph. So the time complexity of this step is **O(E)**.
- Next, look for nodes with in-degrees equal to 0. This will require us to iterate through the entire array that stores the in-degrees of each node. The size of this array is equal to V. So, the time complexity of this step is **O(V)**.

Auxiliary space: O(V), We have to create one array to store the in-degrees of all the nodes. This will require **O(V)** space.

›

‹

01

02

03

Topological Sort

Merits:

- Requires linear time and linear space to perform.
- Effective in detecting cyclic dependencies.
- Can efficiently find feedback loops that should not exist in a combinational circuit.
- Can be used to find the shortest path between two nodes in a DAG in linear time.

Demerits:

- Topological sort is not possible for a graph that is not directed and acyclic.

›

‹


Thank You