Topological Sort (medium)

We'll cover the following

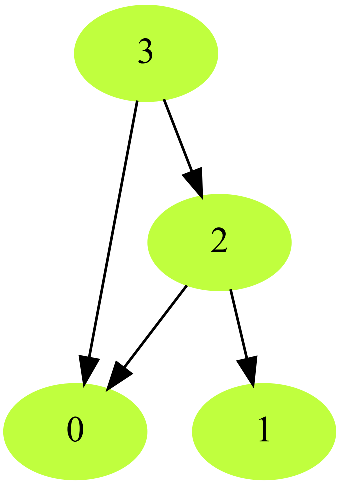
Problem Statement#

Topological Sort of a directed graph (a graph with unidirectional edges) is a linear ordering of its vertices such that for every directed edge (U, V) from vertex `U` to vertex `V`, `U` comes before `V` in the ordering.

Given a directed graph, find the topological ordering of its vertices.

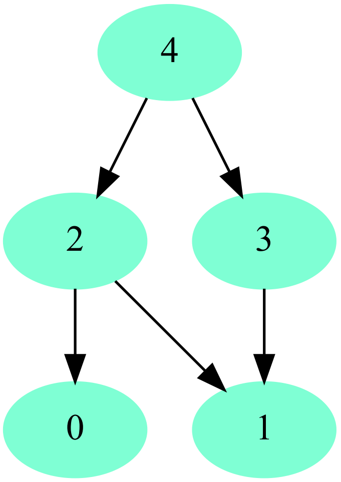
Example 1:

```
Input: Vertices=4, Edges=[3, 2], [3, 0], [2, 0], [2, 1]
Output: Following are the two valid topological sorts for the given grap
h:
1) 3, 2, 0, 1
2) 3, 2, 1, 0
```



Example 2:

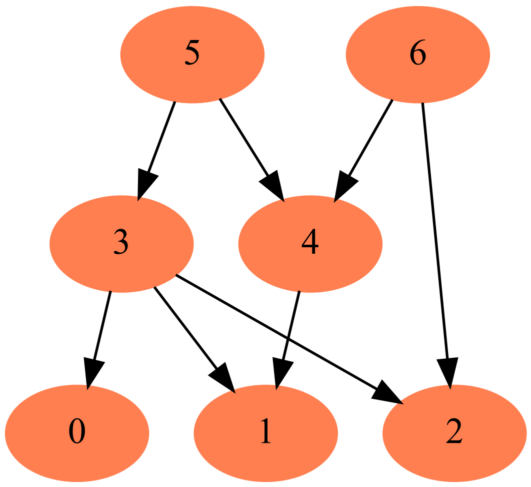
```
Input: Vertices=5, Edges=[4, 2], [4, 3], [2, 0], [2, 1], [3, 1]
Output: Following are all valid topological sorts for the given graph:
1) 4, 2, 3, 0, 1
2) 4, 3, 2, 0, 1
3) 4, 3, 2, 1, 0
4) 4, 2, 3, 1, 0
5) 4, 2, 0, 3, 1
```



Example 3:

```
Input: Vertices=7, Edges=[6, 4], [6, 2], [5, 3], [5, 4], [        ,        ]
 [3, 2], [4, 1]
Output: Following are all valid topological sorts for the given graph:
1) 5, 6, 3, 4, 0, 1, 2
2) 6, 5, 3, 4, 0, 1, 2
3) 5, 6, 4, 3, 0, 2, 1
4) 6, 5, 4, 3, 0, 1, 2
5) 5, 6, 3, 4, 0, 2, 1
6) 5, 6, 3, 4, 1, 2, 0

There are other valid topological ordering of the graph too.
```



$$\begin{tikzcd} & 5 \arrow[dl] \arrow[d] & 6 \arrow[dl] \arrow[ddr] & \\ 3 \arrow[d] \arrow[dr] \arrow[drr] & 4 \arrow[d] & & \\ 0 & 1 & 2 & \end{tikzcd}$$

Try it yourself#

Try solving this question here:

C++

```
    using namespace std;

    #include <iostream>
    #include <queue>
    #include <unordered_map>
    #include <vector>
```

```
class TopologicalSort {
 public:
  static vector<int> sort(int vertices, const vector<vector<int>>& edges) {
    vector<int> sortedOrder;
    // TODO: Write your code here
    return sortedOrder;
  }
};

int main(int argc, char* argv[]) {
  vector<int> result =
      TopologicalSort::sort(4, vector<vector<int>>{vector<int>{3, 2}, vector
                                                   vector<int>{2, 0}, vector
  for (auto num : result) {
    cout << num << " ";
  }
  cout << endl;

  result = TopologicalSort::sort(
      5, vector<vector<int>>{vector<int>{4, 2}, vector<int>{4, 3}, vector<in
```

Solution#

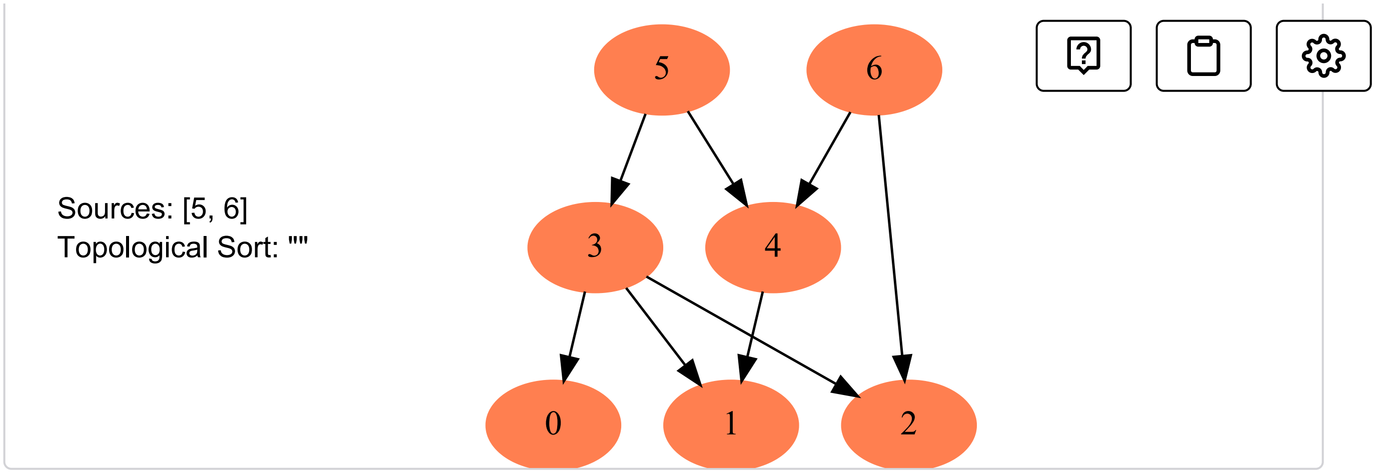
The basic idea behind the topological sort is to provide a partial ordering among the vertices of the graph such that if there is an edge from `U` to `V` then U≤V i.e., `U` comes before `V` in the ordering. Here are a few fundamental concepts related to topological sort:

1. **Source:** Any node that has no incoming edge and has only outgoing edges is called a source.
2. **Sink:** Any node that has only incoming edges and no outgoing edge is called a sink.
3. So, we can say that a topological ordering starts with one of the sources and ends at one of the sinks.

4. A topological ordering is possible only when the graph h[illegible]ed cycles, i.e. if the graph is a **Directed Acyclic Graph (DAG)**. If the graph has a cycle, some vertices will have cyclic dependencies which makes it impossible to find a linear ordering among vertices.

To find the topological sort of a graph we can traverse the graph in a **Breadth First Search (BFS)** way. We will start with all the sources, and in a stepwise fashion, save all sources to a sorted list. We will then remove all sources and their edges from the graph. After the removal of the edges, we will have new sources, so we will repeat the above process until all vertices are visited.

Here is the visual representation of this algorithm for Example-3:



This is how we can implement this algorithm:

a. Initialization

1. We will store the graph in **Adjacency Lists**, which means each parent vertex will have a list containing all of its children. We will do this using a **HashMap** where the 'key' will be the parent vertex number and the value will be a **List** containing children vertices.
2. To find the sources, we will keep a **HashMap** to count the in-degrees i.e., count of incoming edges of each vertex. Any vertex with '0' in-degree will be a source.

b. Build the graph and find in-degrees of all vertices

1. We will build the graph from the input and populate the in-degrees **HashMap**.

c. Find all sources

1. All vertices with '0' in-degrees will be our sources and we will store them in a **Queue**.

d. Sort

1. For each source, do the following things:
    - Add it to the sorted list.

   - Get all of its children from the graph.
   - Decrement the in-degree of each child by 1.
   - If a child's in-degree becomes '0', add it to the sources **Queue**.
2. Repeat step 1, until the source **Queue** is empty.

Code#

Here is what our algorithm will look like:

C++

```
using namespace std;

#include <iostream>
#include <queue>
#include <unordered_map>
#include <vector>

class TopologicalSort {
 public:
  static vector<int> sort(int vertices, const vector<vector<int>>& edges) {
    vector<int> sortedOrder;
    if (vertices <= 0) {
      return sortedOrder;
    }

    // a. Initialize the graph
    unordered_map<int, int> inDegree;       // count of incoming edges for e
    unordered_map<int, vector<int>> graph;  // adjacency list graph
    for (int i = 0; i < vertices; i++) {
      inDegree[i] = 0;
      graph[i] = vector<int>();
    }

    // b. Build the graph
    for (int i = 0; i < edges.size(); i++) {
      int parent = edges[i][0], child = edges[i][1];
      graph[parent].push_back(child);  // put the child into it's parent's l
```

Time Complexity#

In step ‘d’, each vertex will become a source only once and each edge will be accessed and removed once. Therefore, the time complexity of the above algorithm will be $O(V + E)$, where ‘V’ is the total number of vertices and ‘E’ is the total number of edges in the graph.

Space Complexity#

The space complexity will be $O(V + E)$, since we are storing all of the edges for each vertex in an adjacency list.

Similar Problems#

Problem 1: Find if a given **Directed Graph** has a cycle in it or not.

Solution: If we can’t determine the topological ordering of all the vertices of a directed graph, the graph has a cycle in it. This was also referred to in the above code:

```
    if (sortedOrder.size() != vertices) // topological sort is not possi
ble as the graph has a cycle
      return new ArrayList<>();
```

← Back

Introduction

Next →

Tasks Scheduling (medium)

✓ Completed

Report an Is