Monday, April 18, 2022 6:32 AM

Topological Sorting

- Difficulty Level : Medium
- Last Updated : 18 Jan, 2022

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

For example, a topological sorting of the following graph is “5 4 2 3 1 0”. There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is “4 5 2 3 1 0”. The first vertex in topological sorting is always a vertex with in-degree as 0 (a vertex with no incoming edges).

Topological Sorting vs Depth First Traversal (DFS):

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. For example, in the given graph, the vertex ‘5’ should be printed before vertex ‘0’, but unlike DFS, the vertex ‘4’ should also be printed before vertex ‘0’. So Topological sorting is different from DFS. For example, a DFS of the shown graph is “5 2 3 1 0 4”, but it is not a topological sorting.

Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.

Algorithm to find Topological Sorting:

We recommend to first see the implementation of DFS. We can modify DFS to find Topological Sorting of a graph. In DFS, we start from a vertex, we first print it and then recursively call DFS for its adjacent vertices. 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 contents of the stack. Note that 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.

Below image is an illustration of the above approach:

Following are the implementations of topological sorting. Please see the code for Depth First Traversal for a disconnected Graph and note the differences between the second code given there and the below code.

- C++
- Java
- Python3
- C#

```java
// A Java program to print topological
// sorting of a DAG
import java.io.*;
import java.util.*;

// This class represents a directed graph
// using adjacency list representation
class Graph {
    // No. of vertices
    private int V;

    // Adjacency List as ArrayList of ArrayList's
    private ArrayList<ArrayList<Integer> > adj;

    // Constructor
    Graph(int v)
    {
        V = v;
        adj = new ArrayList<ArrayList<Integer> >(v);
        for (int i = 0; i < v; ++i)
            adj.add(new ArrayList<Integer>());
    }

    // Function to add an edge into the graph
    void addEdge(int v, int w) { adj.get(v).add(w); }

    // A recursive function used by topologicalSort
    void topologicalSortUtil(int v, boolean visited[],
                             Stack<Integer> stack)
    {
        // Mark the current node as visited.
        visited[v] = true;
        Integer i;

        // Recur for all the vertices adjacent
        // to thisvertex
        Iterator<Integer> it = adj.get(v).iterator();
        while (it.hasNext()) {
            i = it.next();
            if (!visited[i])
                topologicalSortUtil(i, visited, stack);
        }

        // Push current vertex to stack
        // which stores result
        stack.push(new Integer(v));
    }

    // The function to do Topological Sort.
    // It uses recursive topologicalSortUtil()
    void topologicalSort()
    {
        Stack<Integer> stack = new Stack<Integer>();

        // Mark all the vertices as not visited
        boolean visited[] = new boolean[V];
```

```
        for (int i = 0; i < V; i++)
            visited[i] = false;

        // Call the recursive helper
        // function to store
        // Topological Sort starting
        // from all vertices one by one
        for (int i = 0; i < V; i++)
            if (visited[i] == false)
                topologicalSortUtil(i, visited, stack);

        // Print contents of stack
        while (stack.empty() == false)
            System.out.print(stack.pop() + " ");
    }

    // Driver code
    public static void main(String args[])
    {
        // Create a graph given in the above diagram
        Graph g = new Graph(6);
        g.addEdge(5, 2);
        g.addEdge(5, 0);
        g.addEdge(4, 0);
        g.addEdge(4, 1);
        g.addEdge(2, 3);
        g.addEdge(3, 1);

        System.out.println("Following is a Topological "
                           + "sort of the given graph");
        // Function Call
        g.topologicalSort();
    }
}
// This code is contributed by Aakash Hasija
```

Output

```
Following is a Topological Sort of the given graph
5 4 2 3 1 0
```

Complexity Analysis:

- **Time Complexity:** O(V+E).
  The above algorithm is simply DFS with an extra stack. So time complexity is the same as DFS which is.
- **Auxiliary space:** O(V).
  The extra space is needed for the stack.
  Note: Here, we can also use vector instead of the stack. If the vector is used then print the elements in reverse order to get the topological sorting.

Applications:

Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when recomputing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in make files, data serialization, and resolving symbol

dependencies in linkers [2].

From <https://www.geeksforgeeks.org/topological-sorting/>