Topological Sorting

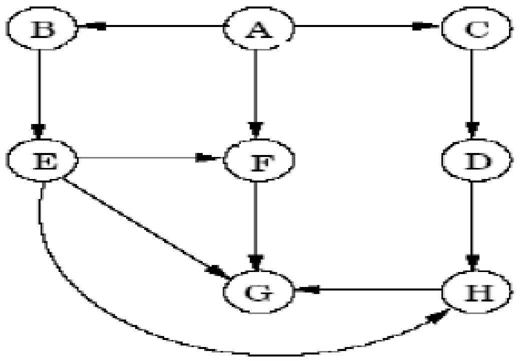
Preet Raj
CS 560
Computer Science Department
Illinois Institute of Technology
Chicago, Illinois 60616
Praj2@iit.edu

***Abstract* – This paper includes information about topological sorting (basic definition and implementation examples). It explains the algorithm in detail and also explains it using data sets. The paper highlights various scenarios in which these algorithms are implemented and then a detailed analysis of the same is performed to understand the differences in results obtained. It analysis three different algorithms that are used to perform topological sort. The paper explains the advantages and disadvantages of each algorithm. It quotes examples from other papers explaining the difference in techniques used to sort tasks. Finally it analysis topological sort and further explains the practical applications of the same.**

Index Terms – DFS (Depth for Search), SRL (Source Removal Algorithm)

INTRODUCTION

Topological sorting of a graph is the linear ordering of its vertices in a specific way. Consider an edge ab where a comes before b. These are set of tasks in which some of the tasks have to done before others. It is similar to taking a course only after taking its prerequisites. In other words it is a systematic way of linearly arranging tasks in an order in which they should be performed. There are certain guidelines that have to be followed in order for it to work correctly, such as there should be no directed cycles. It should always be a directed acyclic graph (DAG). A DAG is a directed graph without cycles.



[1]

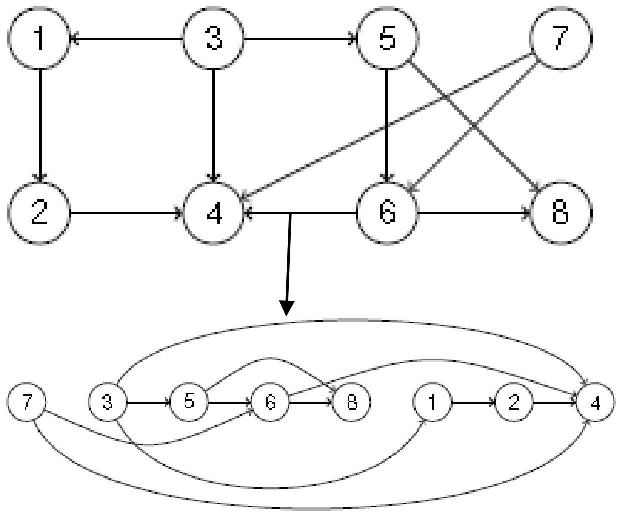
DAGs are useful in modeling scenarios that involve prerequisite constraints.

Properties of DAGs[1]:

- Every DAG must have at least one vertex with in-degree zero and at least one vertex with out-degree zero
- A vertex with in-degree zero is called a source vertex, a vertex with out-degree zero is called a sink vertex.
- G is a DAG iff each vertex in G is in its own strongly connected component
- Every edge (v, w) in a DAG G has finishingTime[w] < finishingTime[v] in a DFS traversal of G

The above mentioned properties are very essential for achieving topological sort. Consider a directed graph G=(V,E) a topological sort of G is an arrangement of V such that for any edge (u,v). u is always before v.

Example:



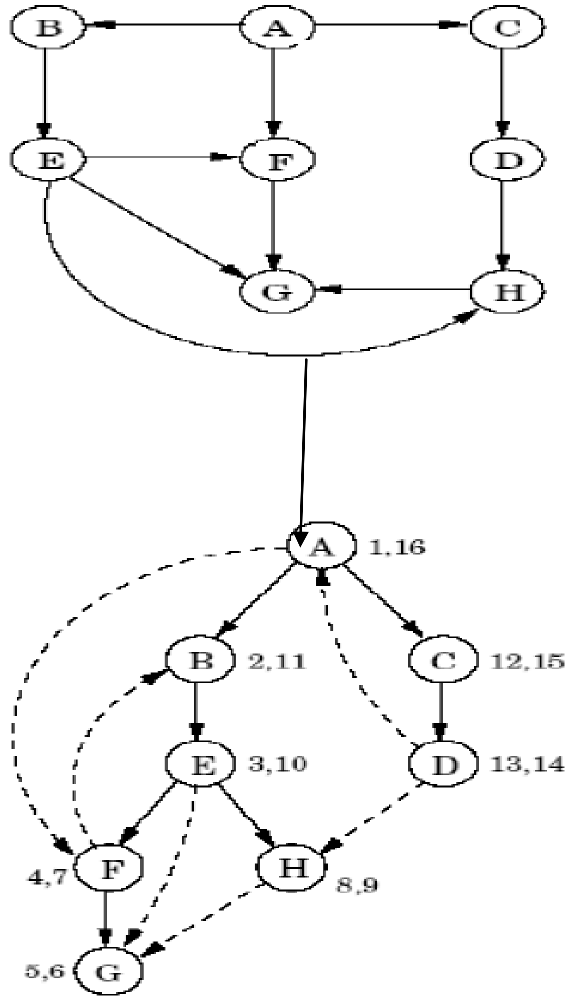
Important quality about topological sort is that it is never unique. There can exist a wide range of possibilities depending on the way/method used to solve the problem.
There are a number of algorithms that can be used to solve the scenario i.e find the linear ordering of the tasks. One such algorithm is depth-first search.

DEPTH-FIRST SEARCH

Depth first search is a type of search in which the search takes place by visiting the first child node of the graph that appears and then keeps going deeper and deeper until the last node is found. This algorithm was described by Cormen et al. After completing this process it loops back, meaning that it returns to the most recent node that it hadn't completely finished analyzing. In this type of implementation all the visited nodes are added to the stack for analysis. DFS looks for nodes with edges pointing to a specific node rather pointing from it. The algorithm loops through each node in the graph in a pattern that eventually starts a mechanism that eliminates the node that has already been visited since the beginning of the topological sort.

```
Topological-Sort(G)[1]
{
1. Call dfsAllVertices on G to compute f[v] for each vertex v
2. If G contains a back edge (v, w) (i.e., if f[w] > f[v]) , report error ;
3. else, as each vertex is finished prepend it to a list;     // or push in stack
4. Return the list;    // list is a valid topological sort
}[1]
```

Running time is O(V+E), which is the running time for DFS.



Topological order: A C D B E H F G

In this algorithm each node and edge are visited only once, therefore the algorithm exists in linear time.

Advantages:

- Memory requirement is linear with respect to search graph.
- Time complexity for this algorithm is O(b^d). DFS is time limited.
- If DFS finds a solution without exploring much during its path then it would consume very less time and space.

Disadvantages:

- There exists a possibility that it may go down the left-most path forever.
- There is probability that a finite graph can generate an infinite tree.[2]
- This algorithm does not guarantee to find a solution.
- There is a limited possibility to find a minimal solution if there are multiple solutions.

SOURCE REMOVAL ALGORITHM

The main technique used in this algorithm is to repeatedly identify and remove a source and all the corresponding edges associated to it. This process has to be continued until there is no vertex left or there is no source left in the remaining vertices.

The algorithm follows three simple steps:

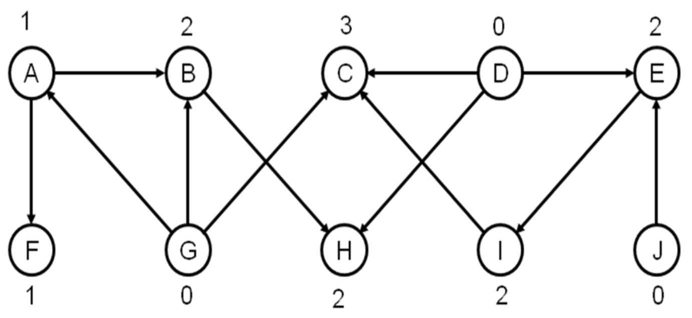
- Pick a source (a vertex) and output it
- Remove the source and all the edges associated with it
- Keep repeating till the graph is empty

The algorithm is implemented in such a manner that it visits all the vertices in a topological sort manner. This algorithm usually uses an array to record the in-degree vertices. As a result there is no explicit need to delete vertices and edges. The algorithm uses a priority queue to record vertices with-in degree zero that hasn't been visited yet.

Algorithm goes as follows:

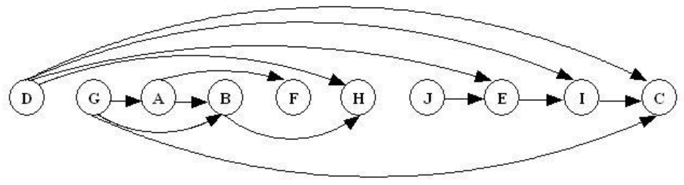
[4]

```
int topologicalOrderTraversal( ){
   int numVisitedVertices = 0;
   while(there are more vertices to be visited){
      if(there is no vertex with in-degree 0)
          break;
      else{
 select a vertex v that has in-degree 0;
 visit v;
 numVisitedVertices++;
 delete v and all its emanating edges;
      }
   }
  return numVisitedVertices;
}
```



D	G	A	B	F	H	J	E	I	C

[2]



[2]

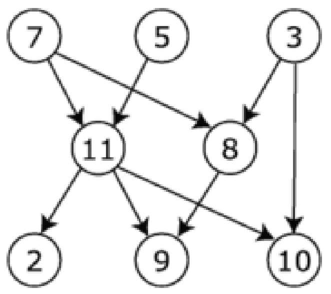
Efficiency is same as that of DFS.

(O(|V|+|E|)) ALGORITHM

This algorithm works by choosing vertices in the same order as if they were eventually sorted topologically. The algorithm starts by listing start nodes; these are nodes that have no incoming edges. Then insert them into a set S. Be sure that there exists a node in an acyclic graph.

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edges
while S is non-empty do
   remove a node n from S
   insert n into L
   for each node m with an edge e from n to m do
      remove edge e from the graph
      if m has no other incoming edges then
         insert m into S
if graph has edges then
   return error (graph has at least one cycle)
else
   return L (a topologically sorted order) [3]
```

For a graph usually a DAG, a solution is created in L. This solution may not necessarily be unique. If the graph has a cycle then topological sorting would become impossible.
A very important property of topological sorting is that the result is never fixed. It depends upon the order in which nodes n are removed from the set S.



[3]

- 7, 5, 3, 11, 8, 2, 9, 10 (visual left-to-right, top-to-bottom)
- 3, 5, 7, 8, 11, 2, 9, 10 (smallest-numbered available vertex first)
- 3, 7, 8, 5, 11, 10, 2, 9
- 5, 7, 3, 8, 11, 10, 9, 2 (fewest edges first)
- 7, 5, 11, 3, 10, 8, 9, 2 (largest-numbered available vertex first)
- 7, 5, 11, 2, 3, 8, 9, 10[3]

ANALYZING TOPOLOGICAL SORT

A topological sort has the property that if all the sorted consecutive vertices are connected by edges, then these edges tend to form a directed Hamiltonian path. The existence of such a path shows that topological sort is unique. In case a topological sort does not form a Hamiltonian path, then DAG should have multiple topological orderings.

APPLICATIONS

1. Topological sorting is used for planning and scheduling.
2. It is also used to identify defects with a bit of modification.
3. It is an effective method to detect cycles in a graph.
4. It is commonly used in identifying errors in DNA fragment assembly.

CONCLUSION

This paper explains topological sorting, its definition. Multiple implementations and also advantages over each other. It explains in detail all the algorithms used to achieve topological sorting and illustrates the same using concrete examples. It explains the differences in cyclic and acyclic graphs and how their presence changes the entire analysis of each algorithm. It also highlights the time and space complexity involved during topological sort. Towards the end it compares the consecutive connected vertices to Hamiltonian path. The paper ends by mentioning the multi-domain implementation/application of topological sorting.

ACKNOWLEDGMENT

This research paper was made possible through the help and support from everyone. I would specially like to thank Professor Charles Bauer for his support and encouragement and also for his invaluable detailed advices on organization and presentation of the paper.

REFERENCES

1. http://www.cs.sunysb.edu/~algorith/files/topological-sorting.shtml
2. http://intelligence.worldofcomputing.net/ai-search/depth-first-search.html
3. http://en.wikipedia.org/wiki/Topological_sorting
4. http://www.cs.nott.ac.uk/~nza/G5BADS06/lecture18.pdf
5. http://www.cs.duke.edu/~reif/courses/alglectures/skiena.lectures/lecture15.pdf
6. http://www.cse.cuhk.edu.hk/~taoyf/course/2100sum11/lec14.pdf