Topological Sorting

De Melo, Bianca
CS 560
Computer Science Department
Illinois Institute of Technology
Chicago, Illinois 60616
bdemalo@hawk.iit.edu

Abstract - **This paper defines what topological sorting is, shows some of the most known algorithms to this kind of sorting and discusses its use in many other fields, such as Computer Science and Project Management. It also lists examples of ordered graphs that need to have its vertices ordered using topological sorting and discusses the non-uniqueness in the results of most of the graphs that are sorted. Topological sorting is frequently used to define, in a chained list of events, which one comes first and what comes next, being really useful in scheduling and planning, modular programming, such as in other applications in many fields.**

Index Terms - Algorithms, Applications, Computer Science, Topological sorting.

DEFINITION

Topological sorting is one of many possible ways to solve problems involving dependency resolutions. This type of sorting consists in ordering vertices of a directed graph, such that for every edge (u,v), u comes before v in the order.

Topological sorting has lots of applications, such as solving dependency issues in modular programming, or ordering tasks in many types of projects so it becomes more clear which tasks must be completed first in order to keep the project going.

To make the topological order possible, the graph that represents the tasks or the objects of interest needs to be directed and acyclic. When a directed graph has cycles, it's not possible to establish which vertices come first.

ALGORITHMS

There are lots of algorithms to topological sorting, most of them run in linear time, $O(|V|+|E|)$.

Kahn's algorithm works separating the vertices that have predecessors of the ones that don't and inserting them in a list in the following order: first, one the vertices with no predecessors (also called 'start nodes') is picked, in any order, and then, for each one of the start nodes, all the edges that come out of them are removed. If another vertex becomes without its predecessors after the edge removal, it goes to the set of start nodes, and can be picked from that moment on.

The depth-first search based algorithm starts examining the vertices that have edges pointing to them, and not the opposite, as Kahn's algorithm.

Both algorithms return error when the graphs given initially have directed cycles. When a graph has a directed cycle, it is not possible to return and ordered path, since there is a mutual dependency between two nodes.

The existence of cycles in the graphs that are to be ordered is a decisive factor concerning whether the graphs can or cannot be ordered. The existence of mutual dependencies between nodes means that both nodes are interdependent and it's not possible to define which one comes before the other.

KAHN'S ALGORITHM

This section shows Kahn's algorithm.
Algorithm: TopologicalKahn
Input: A directed acyclic graph G(V,E)
Q ← Queue({u ϵ V: deg_{in}(u) = 0})
i ← 1
while Q ≠ 0
u ← pop(Q)
ord(u) ← i
i ← i + 1
for each (u, v) ϵ E
E ← E \ {(u,v)}
if deg_{in} (v) = 0
push(Q, v)
return ord
Output: all the vertices in V sorted in topological order, represented by 'ord'.
Kahn's algorithm has a linear complexity of $O(|V|+|E|)$.

DEPTH-FIRST SEARCH BASED ALGORITHM

This section describes the depth-first search based algorithm to topological sorting.
Algorithm: TopologicalDFS
Input: A directed acyclic graph G(V,E)
S ← Set({u ϵ V: deg_{out}(u) = 0})
L ← empty
for each u ϵ S
visit (u)
Output: all the vertices in V sorted in topological order in the list L. The algorithm visit (node n) returns the ordered vertices.

Algorithm: visit (node n)
Input: node n

if n has not been visited, **then** mark n as visited
for each v, (v, n) ϵ E
visit (v)
L.add (n)
return L
Output: L, the list with the ordered vertices.

EXAMPLES

In the following section, a few examples of the uses of topological sorting will be listed.

The example below shows applications of topological sorting in modular programming. In order to develop the modules as quickly as possible, it's important to know what modules can be fully developed first, so when the team moves to the next module there are no pending tasks in the finished module.

Of course this example does not transfer completely to the real world, since errors often appear on software and the developers need to go back and fix what's wrong in a module, but even so it's important to know what to do first, to avoid as much as possible jumping from one module to another from time to time - so mutual dependency is not recommended. For instance, if a graph representing the architecture of a software has cycles in their modules, it often means there is some kind of problem, and cycles can and should be removed when we're talking about software maintenance.

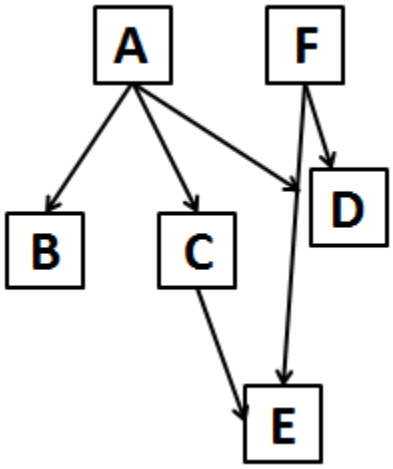

FIGURE 1
GRAPH REPRESENTING MODULES IN A SOFTWARE PRODUCT.

In figure 1, we have a graph that represents modules in a software product that we would like to order topologically to decide which parts should be developed first so every model completed can run fully, even without all the parts working.

In figure 2, it's presented one of the many solutions. It can also be considered that some of those modules could be developed in parallel so more time can be saved in the development activity.

But despite of this observation, one possible order is A -> B -> F -> D -> C -> E.

Another possible order is A -> C -> F -> D -> E -> B.

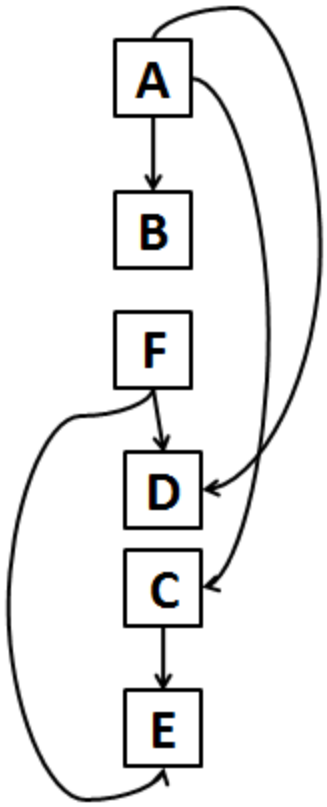

FIGURE 2
GRAPH SHOWN IN FIGURE 1 ORDERED USING TOPOLOGICAL SORTING.

Another common problem in topological sorting is the one with the order to wear each piece of clothing.

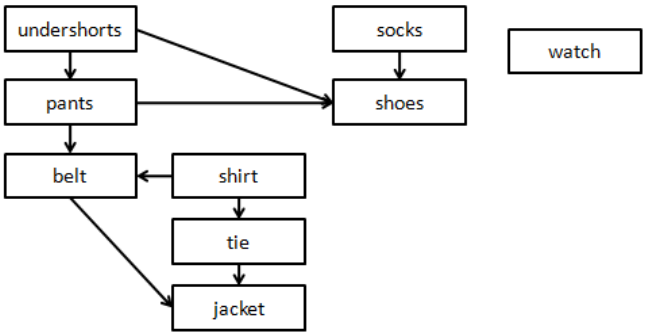

FIGURE 3
GRAPH REPRESENTING THE CORRECT ORDER TO WEAR CLOTHES.

Figure 3 shows a graph that represents the correct way to wear pieces of clothing so no one wears a tie under the shirt or wears their shoes before having the socks on. It's also important to wear underwear before the pants and to only wear the shoes after wearing the pants.

There are lots of restrictions in this graph, but there are also things that can be worn in any step of the process, like the watch.

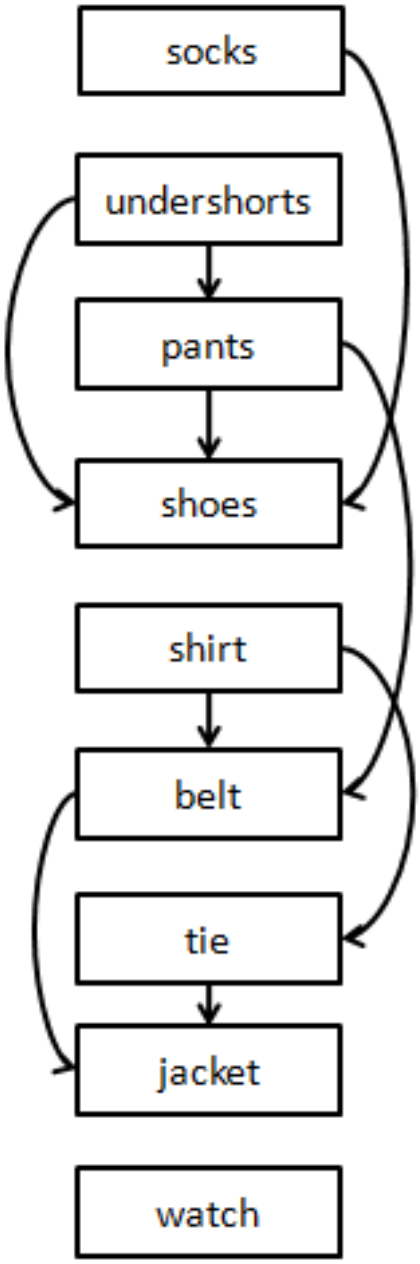

FIGURE 4
ONE OF MANY POSSIBLE SOLUTIONS TO THE PROBLEM SHOWN IN FIG. 3.

Figure 4 shows one of many solutions for the problem presented in Figure 3. Some of other solutions would be adding the watch in any other position desired by the person who's wearing the pieces of clothing.

The person could also leave the socks and shoes to be worn last, and the jacket too.

As said before, it's usually uncommon to have uniqueness in the sorting results, as most of them do have other solutions equally correct.

Uniqueness in topological sorting are proved to occur in graphs that have Hamiltonian path, where there is only one initial node that leads to only one other node and so on, until it reaches the end of the graph, so there is only one first step to be taken and only one way to go from there, but that's unusual.

There are tons of examples that could be taken to demonstrate the applications of topological sorting, but the two problems shown in the paper are two of the most common problems seen and both have similar abstractions.

OTHER APPLICATIONS OF TOPOLOGICAL SORTING

Other applications of topological sorting are in database, when trying to solve a dependency problem with foreign keys, or prerequisites of classes in an specific course, logic synthesis, and resolving different kinds of dependencies in lots of different situations.

ACKNOWLEDGMENT

The algorithms presented in this paper are proven to be efficient ones and useful in many aspects and areas of knowledge. Topological sorting is an efficient way to organize different types of tasks and it's one of the most popular organizing methods.

REFERENCES

Auno, M., Andermo, M., 'Lecture 5: Graph Algorithms 1', *DD2458 - Problem Solving and Programming under pressure*. Taken from <http://www.csc.kth.se/utbildning/kth/kurser/DD2458/popup08/anteckningar/lecnotes05/lecnotes05.pdf>

Brum, A., 'Lecture 12: Graph Algorithms I', *CMU 15-451 - Algorithms*. Taken from <http://www.cs.cmu.edu/~avrim/451f08/lectures/lect1002.pdf>

McDonald, S., 'Topological Sorting Acyclic Directed Graphs', *Stephen McDonald's Blog*. Taken from <http://blog.jupo.org/2012/04/06/topological-sorting-acyclic-directed-graphs/>

Zhang, H., '22c:21 - Computer Science II Data Structures - Topological Sort', *22c:21 - Computer Science II Data Structures*. Taken from <http://homepage.cs.uiowa.edu/~hzhang/c21/notes/18TopoSort.pdf>

Various, 'Topological Sort', *Rosetta Code*. Taken from <http://rosettacode.org/wiki/Topological_sort>

Rao, R., 'Lecture 20: Topo-Sort and Djikstra's Greedy Idea', *CSE 326 - Data Structures and Algorithms*. Taken from <https://www.cs.washington.edu/education/courses/326/03wi/lectures/RaoLect20.pdf>

Various, 'Topological Sorting', *Wikipedia, The Free Encyclopedia*. Taken from <http://en.wikipedia.org/wiki/Toopological_sorting>

Wisman, R. F., 'Topological Sorting', *C455 Analysis of Algorithms - Class Notes*. Taken from <http://homepages.ius.edu/rwisman/C455/html/notes/Chapter22/TopSort.htm>