# Testing/Identifying a Topological Sorting

You're given a set of $n$ Directed Acyclic Graphs $G_1, G_2, \ldots, G_n$ over the same set of $m$ vertices $V$. You're also given a permutation of the set of vertices $(v_1, v_2, \ldots, v_m)$. What is the best algorithm that could identify the graphs among $G_1, G_2, \ldots, G_n$ that have $(v_1, v_2, \ldots, v_m)$ as a topological sort? Could someone test whether $(v_1, v_2, \ldots, v_m)$ is a topological sort of a DAG $G$ over $V$ in sub-linear time?

ds.algorithms graph-algorithms directed-acyclic-graph

edited Jan 20 '11 at 4:57

Hsien-Chih Chang 張顯
5,715 2 31 69

asked Jan 20 '11 at 4:09
Steve

6 Are you able to build a data structure based on the set of graphs before being presented with the ordering of the vertices? You need to look at all $n$ graphs and all $m-1$ edges in the ordering, so unless you're allowed to preprocess the graphs somehow, it doesn't seem like you could beat linear time. – mjqxxxx Jan 20 '11 at 5:29

Hsien-Chih Chang, what would be a good pre-processing technique that can allow a better solution? Some type of hashing? I guess you can beat linear time if can approximate the solution (probabilistic algorithm). – user2471 Jan 21 '11 at 0:37

@user2471: As I said in your previous answer, this post is written by @Steve, not me ;) – Hsien-Chih Chang 張顯之 Jan 21 '11 at 1:00

Sorry Hsien-Chih Chang, my question was meant for everyone :) – user2471 Jan 21 '11 at 2:04

@user2471, no need to apologize! Hope someone who is familiar to this question will post a nice answer :D – Hsien-Chih Chang 張顯之 Jan 21 '11 at 5:28

## 2 Answers

This can be done in nearly linear time.

Let the permutation be $\pi = (v_1, \ldots, v_m)$, and let $k = k(\pi)$ be the number of steps needed to check a single edge $(u, v)$ against $\pi$. It is then enough to check that each of the $M_i$ edges of $G_i$ is compatible with $\pi$, which can be done in $O(kM_i)$ time, or $O(k \sum M_i)$ overall.

By preprocessing $\pi$ one can reduce $k$ down to two lookups in an array containing $m$ entries each of $\log\ m$ size, and a comparison between two $(\log\ m)$-bit entries in the array; the array element $a[w]$ contains the index of $w$ in $\pi$ considered as an ordered list. This means that $k = O(\log\ m)$ yielding $O((\log\ m) \sum M_i)$ time overall for the upper bound.

As @mjqxxxx points out, every edge of every graph may be relevant. This creates a lower bound of $\Omega(K \sum M_i)$ steps, where $K$ is the least amount of work that needs to be done for every graph edge; it is possible that some approaches can amortize the cost so that $K = o(\log\ m)$. This is still going to be $\Omega(\sum M_i)$ at best, so there is not much of a gap left.

edited Apr 23 '11 at 15:52

answered Apr 20 '11 at 23:21

András Salamon
11.8k 3 40 117

What algorithm can be used to make k = log m. Is it suffix trees? – vincent mathew Mar 30 at 11:03

Trivial Method:

```
GS = {G1,G2...G3}
for v in (v1,v2,...vm)
      remove all graphs from GS where indegree(v) != 0
      remove v and attached edges from remaining GS
```

It is not as fast as you want. But it solves one problem that "there can be multiple valid topological orderings of DAG". And finding them all is not a good idea.

answered Jan 20 '11 at 5:46

Pratik Deoghare
1,636 11 26