Send Orders for Reprints to reprints@benthamscience.ae

The Open Cybernetics & Systemics Journal, 2015, *9*, 359-362 359

Open Access

Topological Sort Algorithm according to the Principle of a DAG's Subgraph is still a DAG after Outputting a Start Node

Yong Wei*

Software School of Shenzhen Institute of Information Technology, Shenzhen, Guangdong, 518172, P.R. China

Abstract: The paper proves that a DAG's subgraph is still a DAG or empty after deleting a start node. Based on this conclusion, the program loops the DAG and its subgraph as parameters, calls the same method, outputs a start node. Resulting nodes must be a topological sequence when the parameter is empty, so as to implement a topological sort algorithm. This paper also proposes using key-value storage structure to represent a DAG, where each node as the key, the node's subsequent nodes as the values. Because subsequent nodes must not be start nodes, the remains must be a start nodes set after deleting each node's subsequent nodes from the nodes set.

Keywords: Adjacency matrix, adjacent table, directed acyclic graph, key-value storage, topological sort.

1. INTRODUCTION

In 1960, Jarnagin first touched the problem in PERT project management [1]. Two years later, Kahn pointed out that a DAG (directed acyclic graph) has at least one topological sequence, and he found such a topological sequence method [2, 3]. The algorithm established a start node set S (could also be a queue or stack). The program saved the start nodes in S in case any start node was found either beginning or processing. The topology sequence outputting could be summarized as the following steps:

(1) Arbitrarily output a node from the start node set.

(2) Delete the node and edges which connected the node.

(3) If more start nodes emerge in the remaining graph, save them in the S.

(4) If remaining graph is not null, continue to (1), otherwise end the program.

In accordance with the depth-first search ideas, Cormen *et al.* proposed another method to implement the topological sort algorithm [4, 5]. The characteristic of this algorithm was that the program did not output anther start node in S set immediately after processing a start node. But the program processed new emerging start node in the remaining graph firstly. The algorithm was described as follows:

(1) Output a start node.

(2) Delete the node and edges that connected with the node.

(3) If new start nodes emerge in the deleted node's subsequences, output one of them. Otherwise output anther node from S. If there are no start nodes, end the program.

(4) Go to (2).

Vernet and Markenzon further proofed that if a Hamiltonian path exists, the topological sort order is unique [6].

From the beginning of 80's last century, in order to improve the efficiency, the parallel algorithms for topological sorting emerged [7].

The past topological sorting algorithm represented DAG based on adjacency matrix or adjacency table. If the nodes which have no incoming edges were called "start nodes", almost all topological algorithms were determined from the start nodes.

This paper breaked the limitation of the adjacency matrix or adjacency table, used key-value storage structure to represent DAG, and implemented the algorithm of finding the start nodes. Moreover the paper put forward a method of finding the topological sequence based on the principle that the remaining subgraph is still a DAG after output a topological order node.

2. GRAPH'S DESCRIPTION OF THE KEY-VALUE STRUCTURE

In the key-value storage structure, key is a certain data identification. The program can find the data in a dataset through the identification, and carry out operations, such as reading, writing etc. Value refers to the data in the real world such as height, weight, date of birth, place of birth and so on.

For example, in Java Map interface is key-value storage structure, or key-value pair. The key is unique, meanwhile, the value is various. Hashtable and HashMap are implementations of Map, they provide put() method to put the key-value value into the table, then get this value through calling get () method where the key is a parameter.

The topological sort is a process to map the topology sequence. Fig. (**1**) was a typical DAG, said the priority relation between curriculum.

1874-110X/15 2015 Bentham Open

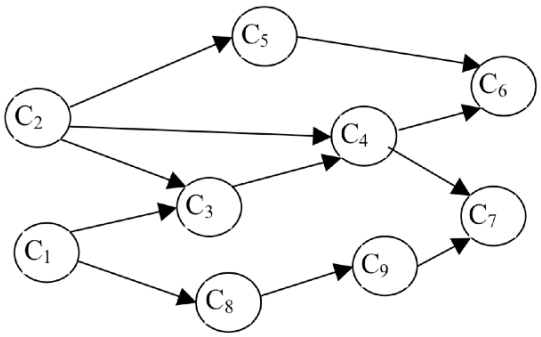

Fig. (1). The priority relation between curriculum.

In Fig. (**1**), a topological sequence was: c1, c2, c3, c4, c5, c8, c9, c7, c6. Another sequence was: c2, c5, c1, c8, c9, c3, c4, c7, c6.

The following discussed how to use key-value storage structure represent a DAG in Fig. (**1**).

Java provides a hash data structure of Hashtable class. Its object has a key-value pair, uses put() method to put the key-value data to the table, then sees the key as a parameter, calls get() method to get the value.

A section of the following Java code represented the graph in Fig. (**1**) using a Hashtable object graph, where the key represented a node, the value said the subsequent nodes of the node. If a node had multi-subsequent nodes, separated them by spaces.

The following steps added all nodes to the object graph:

```
graph.put("c1","c3 c8");
graph.put("c2","c3 c4 c5");
graph.put("c3","c4");
graph.put("c4","c6 c7");
graph.put("c5","c6");
graph.put("c6","");
graph.put("c7","");
graph.put("c8","c9");
graph.put("c9","c7");
```

The graph was a Hashtable object, where the key represented a node, the value said the subsequent nodes.

3. FINDING THE START NODES OF DAG REPRESENTED BY KEY-VALUE STORAGE

In Fig. (**1**), c1 and c2 were start nodes, because they had not ancestral nodes. The core of the topological sorting algorithm was how to find the graph's start nodes.

Below was a description of the topological sorting algorithm's core-finding the start nodes of the DAG represented by key-value storage structure.

If allnodes was a set of the all nodes, the allnodes contained all of the start nodes after the following loop.

Algorithm 1: finding all start nodes of DAG

```
While (more nodes with key exist){
Object value = g.get(key);
allnodes = allnodes - value;
}
```

Algorithm 1 continued to scan each node, got its post nodes through the get() method, Because the post nodes must not be start nodes, deleted them until scanning all the nodes. After while-loop, allnodes contained all start nodes. For example, in Fig. (**1**) the value of allnodes was:[c9, c8, c7, c6, c5, c4, c3, c2, c1], according to algorithm 1, constantly scanned all the nodes, found its subsequent nodes, removed them from allnodes which remained two start nodes [c2, c1] at the end. The following steps demonstrated this process:

c9's subsequent node was[c7], removed it, remains: [c9, c8, c6, c5, c4, c3, c2, c1]

c8's subsequent node was[c9],removed it,remains [c8, c6, c5, c4, c3, c2, c1]

c6's subsequent node was null, remains: [c8, c6, c5, c4, c3, c2, c1]

c5's subsequent node was [c6], removed it, remains: [c8, c5, c4, c3, c2, c1]

c4's subsequent nodes were[c6, c7], removed them, remains: [c8, c5, c4, c3, c2, c1]

c3's subsequent node was[c4], removed it, remains: [c8, c5, c3, c2, c1]

c2's subsequent nodes were[c3, c4, c5], removed them, remains: [c8, c2, c1]

c1's subsequent nodes were[c3, c8], removed them, remains: [c2, c1]

Result was start nodes' set: [c2, c1]

4. THE TOPOLOGICAL SORT ALGORITHM BASED ON PRINCIPLE THAT DAG'S SUBGRAPH IS STILL A DAG OR NULL

We could prove that after removed a start node and it's connecting edges from a DAG, the remaining subgraph was still a DAG. So when output a node, we could call the same method repeatedly to output a start node until the subgraph was null.

The proofing processing: if G isGAD, G' is G's subgraph where an arbitrary start node and its connecting edges are deleted. Because G is a DAG, G exists at least a topological sequence. If G' is not DAG and had no topological sequence, G also has not the topological sequence. So G' must be DAG. Qed.

So we could establish a topological sort algorithm 2 as follows:

Algorithm 2: outputting a topological sequence output a arbitrary node from the start node set; delete it from the DAG graph; repeat the same operations until the remaining subgraph is null; get a topological sequence.

According to algorithm 2, continued the previous steps, below demonstrated the topological sort implementation of Fig. (**1**):

(1) Output an arbitrary start node c2

Removed c2, remains: [c9, c8, c7, c6, c5, c4, c3, c1]

c9 's subsequent node was[c7], removed it, remains: [c9, c8, c6, c5, c4, c3, c1]

c8's subsequent node was [c9], removed it, remains: [c8, c6, c5, c4, c3, c1]

c6's subsequent node was null, remains: [c8, c6, c5, c4, c3, c1]

c5's subsequent node was [c6], removed it, remains: [c8, c5, c4, c3, c1]

c4's subsequent nodes were [c6, c7], removed them, remains: [c8, c5, c4, c3, c1]

c3's subsequent node was[c4], removed it, remains: [c8, c5, c3, c1]

c1's subsequent nodes were [c3, c8], removed them, remains: [c5, c1]

Result was start nodes' set: [c5, c1]

(2) Output an arbitrary start node c5

Removed c5, remains: [c9, c8, c7, c6, c4, c3, c1]

Scanned each node, deleted its subsequent nodes, the rest was start nodes's set: [c1]

(3) Output the start node c1

Removed c1, remains: [c9, c8, c7, c6, c4, c3]

Scanned each node, deleted its subsequent nodes, the rest was start nodes's set: [c8, c3]

(4) Output the start node c8

Removed c8, remains: [c9, c7, c6, c4, c3]

Scanned each node, deleted its subsequent nodes, the rest was start nodes's set: [c9, c3]

(5) Output the start node c9

Removed c9, remains: [c7, c6, c4, c3]

Scanned each node, deleted its subsequent nodes, the rest was start nodes's set: [c3]

(6) Output the start node c3

Removed c3, remains: [c7, c6, c4]

Scanned each node, deleted its subsequent nodes, the rest was start nodes's set: [c4]

(7) Output the start node c4

Removed c4, remains: [c7, c6]

Scanned each node, deleted its subsequent nodes, the rest was start nodes's set: [c7, c6]

(8) Output the start node c7

Removed c7, remains: [c6]

Scanned each node, deleted its subsequent nodes, the rest was start nodes's set: [c6]

(9) Output the last start node c6, ended the program.

The above steps got a topological sequence: c2, c5, c1, c8, c9, c3, c4, c7, c6.

5. CONCLUSION

The graphs generally have two kinds of storage structure, adjacency matrix and adjacency table. The adjacency matrix is a nice data structure, but for graph with more vertices and less edges this structure has a great waste of storage space. Therefore we usually represent the graphs with adjacency table, it is a storage structure of arrays mixed with linked-lists. The adjacency table uses linkedlists to link all subsequent nodes, so it saves space relatively. But as a mixed structure, this adds complexity of algorithm.

This paper uses key-value storage structure in computer language to express a graph. In this storage method, key is a certain data identification, value refers to the data in the real world. Each node in the DAG graph as key, all the subsequent node as the value, thus we implement key-value graph storage structure.

Key-value storage structure does not appear redundant space contrast to adjacency matrix, so as to achieve the purpose of saving space. Adjacency table storage structure is mixed structure of arrays and linkedlists, this increases the complexity of the algorithm, key-value storage structure does not exist the problem. In contrast, key-value storage structure not only saves space, but also eases algorithm.

The key of the topological sort algorithm step is to find the start nodes, which have no incoming edges in the DAG. Therefore we scan all nodes one by one, then delete its subsequent nodes because all the subsequent nodes must not be the start node. The rest must be the start nodes set after the end of scanning.

According to the principle that a DAG which an arbitrary start node and its connected edges have been deleted is also a DAG or empty, the paper puts forward the topological sort algorithm:

(1) Find the start nodes in the graph.

(2) Output an arbitrary start node, delete it and its connected edges.

(3) If the subgraph is empty end the program, otherwise goto (1)

To DAG and its subgraph, the processing of finding the start node and outputting the sequences is exactly the same in the algorithm, so we can use recursion to make the code more concise and clear.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

Declared none.

REFERENCES

[1] M. P. Jarnagin, *Automatic Machine Methods of Testing PERT Networks for Consistency*, Technical Memorandum No. K-24/60, Dahlgren, Virginia: U. S. Naval Weapons Laboratory, 1960.

[2] K. B. Arthur, "Topological sorting of large networks", *Communications of the ACM 5*, vol. 11, pp. 558-562, 1962, doi:10.1145/368996.369025 .

[3] D. E. Knuth, *The Art of Computer Programming*: vol. 1, Fundamental Algorithms. (3rd ed.) Reading, MA: Addison-Wesley, 1997.

[4] T. E. Robert, "Edge-disjoint spanning trees and depth-first search", *Acta Informatica,* vol. 6, no. 2, pp. 171-185, 1976, doi:10.1007/BF00268499 .

[5] C. H. Thomas, L. E. Charles, R. L. Ronald, S. Clifford, "*Section 22.4: Topological sort*", Introduction to Algorithms (2nd ed.), MIT Press McGraw-Hill, pp. 549-552, 2001.

[6] V. Oswaldo, M. Lilian, "Hamiltonian problems for reducible flowgraphs", In: *Proc. 17th International Conference of the Chilean Computer Science Society (SCCC '97)*, pp. 264-267, 1997, doi:10.1109/SCCC.1997.637099 .

[7] D. Eliezer, N. David, S. Sartaj, "Parallel matrix and graph algorithms", *SIAM Journal on Computing*, vol. 10, no. 4, pp. 657-675, 1981, doi:10.1137/0210049, MR 635424 .

Received: September 16, 2014 Revised: December 23, 2014 Accepted: December 31, 2014