Jupiter Publications Consortium

Vol. 2 Iss.2, pp. 14-17, 24th Sep, 2016
ISSN Print : 2454-1435 © 2016 IJRMMAE
ISSN Online : 2454-1443 © 2016 IJRMMAE
http://www.ijrmmae.in

International Journal of Research in Mechanical, Mechatronics and Automobile Engineering (IJRMMAE)

CASE STUDY OF TOPOLOGICAL SORTING IN CASE OF DIRECTED ACYCLIC GRAPH

Dr. N Guruprasad, Dr. Vishwanath Y,
Associate Professor, Information Science and Engineering Department,
New Horizon College of Engineering, Bangalore, India,
guruprasadn@newhorizonindia.edu
Gangadhar Immadi, Asha Rani Borah, Asha Rani Borah

Assistant Professor, Information Science and Engineering Department,
New Horizon College of Engineering, Bangalore, India,
gangadhari@newhorizonindia.edu
Received 09, September 2016 | Accepted 24, September 2016

ABSTRACT

Often a complex project may be decomposed into a collection of simpler tasks with the property that the completion of these tasks implies that the project has been completed. A precedence relationship exists between certain pair of tasks. The set of tasks together with the precedence's may be represented as a digraph. Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.In this paper we make an attempt to study the topological sorting on a specific digraph by making use of queue as a data structure.

Keywords: Directed Acyclic Graph, Linear ordering, Precedence relationship, queue, topological sorting

I. INTRODUCTION

In the field of computer science, a topological sort (sometimes abbreviated toposort) or topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering [1]. For instance, the vertices of the graph may represent tasks to be performed, and the edges may represent constraints that one task must be performed before another; in this application, a topological ordering is just a valid sequence for the tasks[3]. A topological ordering is possible if and only if the graph has no directed cycles, that is, if it is a directed acyclic graph (DAG)[4]. Any DAG has at least one topological ordering, and algorithms are known for constructing a topological ordering of any DAG in linear time[5].

II. APPLICATION

The canonical application of topological sorting (topological order) is in scheduling a sequence of jobs or tasks based on their dependencies; topological sorting algorithms were first studied in the early 1960s in the context of the PERT technique for scheduling in project management) [2]. The jobs are represented by vertices, and there is an edge from xto y if job x must be completed before job y can be started (for example, when washing clothes, the washing machine must finish before we put the clothes to dry). Then, a topological sort gives an order in which to perform the jobs.

In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when re-computing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in makefiles, data serialization, and

Jupiter Publications Consortium

Vol. 2 Iss.2, pp. 14-17, 24th Sep, 2016
ISSN Print : 2454-1435 © 2016 IJRMMAE
ISSN Online : 2454-1443 © 2016 IJRMMAE
http://www.ijrmmae.in

International Journal of Research in Mechanical, Mechatronics and Automobile Engineering (IJRMMAE)

resolving symbol dependencies in linkers. It is also used to decide in which order to load tables with foreign keys in databases.

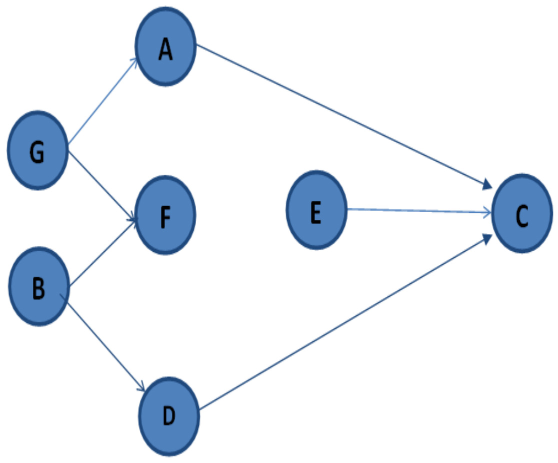
III. COMPLEXITY

The computational complexity of the problem of computing a topological ordering of a directed acyclic graph is NC^2; that is, it can be computed in $O(\log^2 n)$ time on a parallel computer using a polynomial number $O(n^k)$ of processors, for some constant k. One method for doing this is to repeatedly square the adjacency matrix of the given graph, logarithmically many times, using min-plus matrix multiplication with maximization in place of minimization. The resulting matrix describes the longest path distances in the graph. Sorting the vertices by the lengths of their longest incoming paths produces a topological ordering.

IV. UNIQUENESS

If a topological sort has the property that all pairs of consecutive vertices in the sorted order are connected by edges, then these edges form a directed Hamiltonian path in the DAG. If a Hamiltonian path exists, the topological sort order is unique; no other order respects the edges of the path. Conversely, if a topological sort does not form a Hamiltonian path, the DAG will have two or more valid topological orderings, for in this case it is always possible to form a second valid ordering by swapping two consecutive vertices that are not connected by an edge to each other. Therefore, it is possible to test in linear time whether a unique ordering exists, and whether a Hamiltonian path exists, despite the NP-Hardness of the Hamiltonian path problem for more general directed graphs

Consider the following graph:



Algorithm:

```
1) Find the indegreeINDEG(N) of each node N of G.
2) Put in a queue all the nodes with zero indegree.
3) Repeat steps 4 and 5 until the queue is empty.
4) Remove the front node N of the queue by setting FRONT := FRONT+1.
5) Repeat the following for each neighbour M of node N :
   a) Set INDEG(M) := INDEG(M)-1
          [ This deleted edge from N to M ]
   b) If INDEG(M)=0 then : add M to rear of queue
   [ End of loop ]
   [ End of step 3 loop ]
6) Exit.
```

Adjacency List of the above graph is as follows:

A: C
B: D, F

Vol. 2 Iss.2, pp. 14-17, 24th Sep, 2016
ISSN Print : 2454-1435 © 2016 IJRMMAE
ISSN Online : 2454-1443 © 2016 IJRMMAE
http://www.ijrmmae.in

C:
D: C
E: C
F:
G: A, F

The step of the algorithm is depicted below:

1) Find the in-degree of all the nodes N of the graph.

INDEG(A)=1 INDEG(B)=0 INDEG(C)=3 INDEG(D)=1
INDEG(E)=0 INDEG(F)=2 INDEG(G)=0

2) Initially add to the queue each node with zero in-degree.

FRONT = 1 REAR = 3 QUEUE : B E G

3a) Remove front element B from queue by setting FRONT:=FRONT+1

FRONT = 2 REAR = 3 QUEUE : B E G

3b) Decrease by 1 the in-degree of each neighbour of B

INDEG(D)=1-1=0 INDEG(F)=2-1=1

The neighbour D is added to the rear of the queue since its in-degree is now zero.

FRONT = 2 REAR = 4 QUEUE : B E G D

4a) Remove front element E from queue by setting FRONT:=FRONT+1

FRONT = 3 REAR = 4 QUEUE : B E G D

4b) Decrease by 1 the in-degree of each neighbour of E

INDEG(C)=3-1=2

Since in-degree is non zero, QUEUE is not changed.

5a) Remove front element G from queue by setting FRONT:=FRONT+1

FRONT = 4 REAR = 4 QUEUE : B E G D

5b) Decrease by 1 the in-degree of each neighbour of G

INDEG(A)=1-1=0 INDEG(F)=1-1=0

Both A and F are added to the rear of the queue as follows:

FRONT = 4 REAR = 6 QUEUE : B E G D A F

6a) Remove front element D from queue by setting FRONT:=FRONT+1

FRONT = 5 REAR = 6 QUEUE : B E G D A F

6b) Decrease by 1 the in-degree of each neighbour of D

INDEG(C)=2-1=1

Jupiter Publications Consortium

Vol. 2 Iss.2, pp. 14-17, 24th Sep, 2016
ISSN Print : 2454-1435 © 2016 IJRMMAE
ISSN Online : 2454-1443 © 2016 IJRMMAE
http://www.ijrmmae.in

International Journal of Research in Mechanical, Mechatronics and Automobile Engineering (IJRMMAE)

Since in-degree is non zero, QUEUE is not changed. The graph G now looks like below where node D and its edge is deleted.

7a) Remove front element A from queue by setting FRONT:=FRONT+1

FRONT = 6   REAR = 6   QUEUE : B E G D A F

7b) Decrease by 1 the in-degree of each neighbour of A

INDEG(C)=1-1=0

Add C to the rear of the queue since its in-degree is now zero.

FRONT = 6   REAR = 7   QUEUE : B E G D A F C

8a) Remove front element F from queue by setting FRONT:=FRONT+1

FRONT = 7   REAR = 7   QUEUE : B E G D A F C

8b) Node F has no neighbours, so no change takes place.

9a) Remove front element C from queue by setting FRONT:=FRONT+1

FRONT = 8   REAR = 7   QUEUE : B E G D A F C

8b) Node C has no neighbours, so no change takes place.

The queue has no front element, hence the algorithm is completed. The elements in the array QUEUE gives the required topological sort T of graph G which is as follows:

T: B E G D A F C

V. CONCLUSION

In this paper we made use of queues as data structure for understanding topological sorting. Further it can be solved by making use of recursive call in stacks.

REFERENCES

[1] Transaction Processing Performance Council. TPC benchmark DS – Standard Specification, 2017, 2.4.0

[2] Transaction Processing Performance Council. TPC benchmark H – Standard Specification, 2014. Version 2.17.1.

[3] A. Floratou, U.F. Minhas and F. Ozcan,"SQL-on-Hadoop: Full Circle Back to Shared-Nothing Database Architecture", Proc.VLDB Endowment, vol. 7, no. 12, pp. 1295-1306, 2014.

[4] N. Poggi, J.L. Berral, T. Fenech, D. Carrera,J. Blakeley, U.M. Minhas and N. Vujic, "The State of SQL-on-Hadoop in the Cloud," Proc. IEEE Big Data Conference, pp. 529–551, 2016.

[5] Robert, G, Gallager. Principles of Digital Communication[M]. American: Cambridge University Press, 2008.

[6] Yan Jie.Multi-carrier CDMA based on the SIC multi-user detection technology [D]. Nanjing: Hohai University, 2008.