Mandatory Project 4: Critical paths in PERT charts

Project Report

Niveditha Varadha Chandrasekaran
12/6/2016

CONTENTS

1. ABSTRACT

The aim of this project is to implement the algorithm for finding critical paths (DAGs representing projects).

2. PROBLEM STATEMENT

This project aims at implementing the following:

a) Finding topological order in DAGs using inDegree of the vertices
b) Finding reverse topological ordering using Depth First Search finish times of the vertices
c) Computing the Early completion (EC), Latest Completion (LC) and Slack of the projects represented as vertices
d) Finding Critical nodes and tight edges of the Graph representing projects using EC and LC.
e) Finding the length of the Critical path and enumerating all critical paths.

3. METHODOLOGY

Let s and t be dummy nodes where s is the start node and has no incoming edges and t is the end node and has no outgoing edges. Let u.d contain the duration of the projects.

a) Finding topological order in DAGs using inDegree of the vertices:

Add all the vertices whose in degree is 0 to a queue. While the queue is not empty, remove a vertex from the queue and add it to a list which contains the topological order. Also for each edge in the adjacency list of this vertex u get the other end vertex v and decrement it's in degree. If v.indegree is 0 then add it to the queue. Do this until the queue is empty.

b) Finding reverse topological ordering using Depth First Search finish times of the vertices:

Perform Depth First Search on the graph and as each vertex finished add them to a list which contains the reverse of the topological order.

c) Computing the Early completion (EC), Latest Completion (LC) and Slack of the projects represented as vertices:

Assign s.ec as 0. For every vertex u assign u.ec as u.d. Now for every vertex u in topological order and for every edge e of the vertex u get the other end vertex v, if v.ec is lesser than u.ec + v.d then update v.ec as u.ec + v.d.

Assign t.lc as t.ec. For every vertex u assign u.lc as t.lc. Now for every vertex u in reverse topological order calculate u.slack = u.lc - u.ec and for every edge e of the vertex u get the other end vertex v, if v.lc greater than u.lc-u.d then assign v.lc as u.lc-u.d.

d) Finding Critical nodes and tight edges of the Graph representing projects using EC and LC:

For each vertex in the graph if u.ec is equal to u.lc then it is a critical node. For every edge (u,v) in the graph, if u and v are critical nodes and if u.lc + v.d = v.lc then that edge is marked as tight edge.

4. DEVELOPMENT PLATFORM

IDE: Eclipse. Version: Mars.1 Release (4.5.1)
Java Version: 1.8
Operating System: Windows 10

5. SAMPLE INPUT/OUTPUT

Sample input:

```
11 12
1 3 1
1 4 1
2 4 1
2 5 1
3 6 1
4 6 1
4 7 1
5 7 1
5 8 1
6 9 1
7 9 1
8 9 1
3 2 3 2 1
3 2 4 1 0 0
```

Input Explanation:

If the input graph has N vertices, the last two (N-1 and N) are dummy nodes and are not incident to any edges in the input. These two nodes are to be used for s and t. The last two lines are the duration (d) of the projects.

Sample output:

10
1 3 6 9

Task	EC	LC	Slack
1	3	3	0
2	2	4	2
3	6	6	0
4	5	6	1
5	3	5	2
6	9	9	0
7	7	9	2
8	7	9	2
9	10	10	0

4
1
1 3 6 9

Time: 8 msec.
Memory: 1 MB / 123 MB.

Output Explanation:

Line 1: Length of a critical path
Line 2: A critical path
Line 3: blank
Line 4: header of table (Task EC LC Slack)
Next n lines: Task number, its earliest completion time, latest completion time, and slack.
Line n+1: Number of critical nodes
Line n+2: Number of critical paths (k)
Next k lines: one critical path per line.

6. ANALYSIS OF RESULT

Running Time analysis for finding critical paths on the inputs:

Input file	Running time (ms)	Memory Used(MB)	Memory Total(MB)
pert.10.15.txt	28	1	123
pert.100.150.txt	24	1	123
pert.100.500.txt	67	3	123
pert.1000.5000.txt	125	14	123
Input with many critical paths	42	1	123

7. CONCLUSION

The algorithm to find critical paths in PERT charts was implemented my making use of topological ordering of DAGs and computing EC, LC and slack and all the critical paths were enumerated. Appropriate use of data structures, efficient usage of iterators and indexes plays a key role in getting efficient running times as the input size increases.

8. REFERENCES

- https://en.wikipedia.org/wiki/Critical_path_method