Arturo Villalobos UIN: 827008236

Professor Teresa leyk

CSCE 221 sec: 503

24 April 2020

PA-5 Report

SOURCES
https://www.youtube.com/watch?v=eL-KzMXSXXI&t=505s
https://www.youtube.com/watch?v=ddTC4Zovtbc
https://www.youtube.com/watch?v=joqmqvHC_Bo

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

“On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.”

Signature: Arturo Villalobos Date: April 24, 2020

Files Included: main.cpp | graph.cpp | graph.h |

(* all functions are included are in the graph.cpp file topological_sort.cpp is not included/ used).

Description:

For this implementation of graphs i made several assumptions on the format the data was going to be given to me, structuring my program knowing that every number at the beginning of a new line would indicate the label for a given vertex, numbers following this first number represented adjacent nodes, and finally that -1 would signify the end of the adjacency list. To

actually implement this data structure, I used an adjacency list vector that held the lists of adjacent nodes for every node and a vector that only contained nodes, here calling them Vertices. The functions used are buildGraph(), displayGraph(), compute_indegree(), topological_sort(), and print_top_sort(). By reading in the data files with a specific format the Big-O is O(V+E) because I have a while loop set to run until reading it fails. For displaying the graph the big-O is also O(V+E) because while I have two for-loops I am still only accessing any element once not do the same operation over the entire graph. To compute the in-degree again the big-O is O(V+E). I am traversing each adjacency list for each node , in essence every edge connected to every node. For finding the topological sort of each graph i had to use a queue, first traversing the whole node list to find vertices that had an indegree of zero and pushing them into a stl list. While this list is not empty I traverse the adjacent list for every node in this list looking for vertices that have an indegree of 1 and pushing those vertices into the same list. This is again an O(V+E) operation. Printing the topological sort is a O(V) operation as the only thing being worked over is the vector of vertices, not lists. The reason why this algorithm detects cycles in graphs is that since there are only a finite number of vertices that have an indegree of 1 the count can never be anything else than the number of actual vertices, that is unless there is a cycle where the counter would continually increase because the cycle would keep pushing and popping in the same connected vertices , incrementing the counter. Using this implementation of topological sorting and graph representation it is not possible to use a stack however if i was using a DFS recursive function i could use a stack.

Case 1:

Vertex	1	2	3	4	5	6	7
List	2,4,5	3,4,7	4	6,7	n/a	5	6

Topological Order: 1, 2, 3, 4, 7, 6, 5.

Case 2:

Vertex	1	2	3	4	5	6	7	8
List	3	3,8	4,5,6,8	7	7	n/a	n/a	n/a

Topological Order: 1, 2, 3, 4, 5, 6, 8, 7

Case 3:

Vertex	1	2	3	4	5	6	7	8	9
List	2,4	5,7,8	6,8	5	6,9	n/a	n/a	n/a	n/a

Topological Order: 1, 3, 2, 4, 7, 8, 5, 6, 9

Case 4:

Vertex	1	2	3	4	5	6	7
List	2,4,5	3,4,7	4	6,7	4	5	6

As you will find out in the code, there is an exception to be caught if the algorithm finds a cycle within the graph. Given the values from the ‘input-cycle.data’ file, a topological sort is not possible because there is no way to meet all prerequisites for the next vertex.

Screenshots: