ENEE 641 Project Report:
Topological Sort

YUE WANG ,114934802
School of Electrical and Computer Engineering
University of Maryland

1. Project Overview

The purpose of this report is to apply topological algorithm to a directed graph to find out one topological order, or report that it contains a cycle , how to implement the algorithm, and analyzes its time complexity. the process is to read adjacency list of graph from text files, conduct topological sort(using DFS), then output the sorted graph in output files.

2. Code Review

In this project, I apply the topological sort algorithm on given directed graphs. The code is written in C . there are mainly three parts in this project:

```
int main(int argc, char*argv[])
{
   start = clock();
    Readgraph(fp,argv);
    HasCycle = TopologicalSort();
    end = clock();
    PrintGraph(fp,argv);

    return 0;
}
```

- *void Readgraph(FILE* fp,char*argv[])*

This sub function mainly read adjacency list from input file. firstly. an array of size 101 has been assigned to store the vertices.then.fget() has been used to get the adjacency vertex information and allocate space for adjlist[][], then save all the vertex into this array.

- *int TopologicalSort()*

This algorithm is designed based on the DFS(int Dfs_Visit(int Adjdex)).it will help to find one topological order and store the data in array Sorted[],or report that if it contains a cycle using flag ‘HasCircle’. Meanwhile, to get the number of operations charged to this algorithm, some extra codes are necessary.

```
 int i=1;
    Sorted = (int*)malloc(NVertices* sizeof(int));//get an empty linked list
    while (i<=NVertices)
    {
        Vertices[i].color=WHITE;//initialization color
        Vertices[i].operation=1;//initialization the operation value of each vertex
        ++i;
    }

    i=1;
    while((i<=NVertices))
    {
        ++Vertices[i].operation;//get vertex operation
        if(Vertices[i].color==WHITE)
        {
            if(Dfs_Visit(i))//get the cycle
            {
                return 1;
            }
        }
        ++i;
    }

    return 0;
```

- *void PrintGraph(FILE* fp,char*argv[])*

This function will help to get the sorted order vertices in output files and the operations charged to vertices and edges. it also can print out is there any cycle included in any of these graphs. if the flag ”HasCycle” is true, then it reports the cycle. For input graph file1, operations charged to each vertices and each edge from vertex x t vertex y and all vertices and edges needed to be counted and output. For other input graph files, output the sorted vertices. Total number of operations charged to vertices and edges need to be counted as well. At last, it needs to report the complexity respect with V and E.

3. Algorithm

To find out one topological order of given directed graph G=(V,E), the main algorithm used in this project is topological sort and DFS.

3.1 DFS

in this algorithm, each vertices has three different color status(white, gray, black),which can help to indicate their state during the depth first search operation. initially, every vertex is white.if it is discovered it will be colored gray. when it is finished. it will be colored black. the whole adjacency list then has been examined completely.

3.2 Topological Sort

A topological sort of a graph is an ordering of its vertices along a horizontal line so that all directed edges go from left to right.

```
L ={φ}←sorted vertices will be contained in this empty list
while there are vertices with white color do
   select an white vertex n
   visit(n)
function visit(vertex n)
   if the color of n is gray then show there is a cycle
   if the color of n is white
      then
      mark n with gray color
      for each vertex m with an edge from n to m do
         visit(m)
      mark n with black color
      unmark n temporarily
      add n to head of L
```

3.3 Cycle Detection with the DFS

we can check for cycle in individual trees by checking back edges.To detect a back edge, keep track of vertices currently in recursion stack of function for DFS traversal. If we reach a vertex that is already marked gray, then there is a cycle in the tree.

3.4 Asymptotic Analysis

Performing a topological sort costs O(V+E) time. because it is based on depth-first search, which mainly takes the same amount of time O(V+E). and inserting each searched vertex onto the the front of linked list costs O(1) time. if we assume that |V|=O(E), then we have time complexity which is equal to O(E).

Vertices[Adjdex].operation has been used to count the the total number of operation charged to vertices and adjList[k][0].operation has used to represent the total number of operation charged to edges.

In the readgraph function. getting every adjacency vertex from file gives E operations. coloring every vertices into white and conduct DSF give V operations. then the vertex which has been discovered will be

colored gray gives V operation. after the search finished, each vertex will be colored black, which still gives V operations. the vertices will be explored one more time to figure out if the list contains cycle, which gives total V operation on all vertices. then the total operations for vertices is 4V+E. the total operation for edge would be E because each edge only be explored once. thus, there are total (4V+2E) operations charged to edges and vertices.

4. Experimental Results

After applying this algorithm,I get the outputs and summerize as following:
Table 1

Inputfile	NVertices	NEdges	Vertices operations	Edge operations	Total Operations	Cycle(N/Y)	Runtime(ms)
in1	15	24	84	24	108	N	271
in2	20	34	114	34	148	N	79
in3	30	83	46	7	53	Y	236
in4	40	163	323	163	486	N	94
in5	60	318	558	318	876	N	109
in6	80	556	876	556	1432	N	132
in7	100	990	1390	990	2380	N	325

From the table, it shows that there is a cycle in in3.txt. all the number of operations that vertices and edges charged can be found in this table.the total operation can help to validate the time complexity O(V+E). The sorted vertices of each input graph can be found below:
Table 2

input graph	Sorted Vertices
in1	15 9 7 5 3 8 1 10 11 6 4 13 12 14 2
in2	20 17 15 14 13 9 8 4 16 5 19 18 3 2 1 7 10 11 12 6
in3	There is a cycle in this directed graph!
in4	40 38 32 27 23 7 3 18 2 11 12 19 21 22 30 31 13 5 24 28 25 9 15 14 6 17 33 16 4 37 20 8 26 29 1 34 10 35 36 39
in5	51 60 53 27 26 16 52 15 7 35 54 5 19 39 21 28 25 32 12 4 48 23 55 36 10 3 2 29 22 38 40 50 14 31 30 13 1 44 41 24 33 37 43 56 6 8 46 45 47 9 34 57 49 58 17 18 20 59 11 42
in6	76 70 61 68 48 37 35 46 42 47 55 45 38 71 39 65 33 60 31 66 27 26 74 19 18 17 10 77 9 28 78 8 6 1 67 63 52 75 30 40 49 50 11 20 64 22 56 53 62 29 79 32 54 57 41 15 80 51 43 34 58 59 36 21 72 73 24 23 69 2 3 4 13 44 5 25 7 12 14 16
in7	36 100 24 26 14 59 61 3 16 88 19 34 89 43 54 47 74 10 87 37 40 90 52 38 35 93 44 53 27 71 62 83 95 86 94 98 57 2 17 68 39 11 63 85 28 99 29 60 45 48 65 41 30 96 64 76 42 15 55 4 77 46 97 18 20 58 66 67 56 75 21 1 69 6 70 84 91 72 5 7 73 9 12 31 78 49 13 79 8 22 32 92 80 81 82 23 50 25 33 51

The result of Graph 1(in1.txt)can be found below:

```
Sorted vertices can be found below:
15 9 7 5 3 0 1 10 11 6 4 13 12 14 2

Vertex 1: 5
Vertex 2: 8
Vertex 3: 4
Vertex 4: 7
Vertex 5: 4
Vertex 6: 6
Vertex 7: 4
Vertex 8: 6
Vertex 9: 4
Vertex 10: 5
Vertex 11: 7
Vertex 12: 7
Vertex 13: 6
Vertex 14: 7
Vertex 15: 4

Edge from Vertex 1 to Vertex 4: 1
Edge from Vertex 1 to Vertex 6: 1
Edge from Vertex 1 to Vertex 10: 1
Edge from Vertex 3 to Vertex 6: 1
Edge from Vertex 3 to Vertex 8: 1
Edge from Vertex 3 to Vertex 4: 1
Edge from Vertex 4 to Vertex 12: 1
Edge from Vertex 4 to Vertex 13: 1
Edge from Vertex 4 to Vertex 14: 1
Edge from Vertex 6 to Vertex 2: 1
Edge from Vertex 7 to Vertex 8: 1
Edge from Vertex 7 to Vertex 1: 1
Edge from Vertex 9 to Vertex 14: 1
Edge from Vertex 9 to Vertex 11: 1
Edge from Vertex 9 to Vertex 12: 1
Edge from Vertex 10 to Vertex 11: 1
Edge from Vertex 11 to Vertex 12: 1
Edge from Vertex 11 to Vertex 4: 1
Edge from Vertex 11 to Vertex 2: 1
Edge from Vertex 12 to Vertex 14: 1
Edge from Vertex 12 to Vertex 2: 1
Edge from Vertex 14 to Vertex 2: 1
Edge from Vertex 15 to Vertex 11: 1
Edge from Vertex 15 to Vertex 13: 1

total operations detail:
Total number of operations charged to all vertices is: 84
Total number of operations charged to edges is: 24
Total number of operations is: 108
```

Figure 1

5 conclusion

In this project. DFS based topological algorithm in C has been used to apply to different adjacency lists to find out their topological order. after analyzing the code, the time complexity for this algorithm is O(V +E). the total operations charged to edges and vertices are 4V + 2E.