Overview

Given a directed graph G = (V,E) in txt file, the program can successfully read in data and implement topological sort algorithm using DFS. Number of charges (C command operations) on vertices and edges are calculated. Topological order of the input can be obtained with a complexity of O(|V|) time with a constant factor.

Structure, functions, and variables

Input vertices are read from file and stored in the graph that we declared , with total number of input vertices, indegree, outdegree and color of vertices obtained for further algorithm implementation. Number of operations on vertices and edges are tracked with separate counters which count the time of visits and transverses charged on vertices and edges. Functions and important variables are declared as below.

```
void PrintHelp();
void PrintWarnning();
bool DataImport(char* inFileName);
bool DataExport(char* outFileName, bool hasCyc);
bool ValidateFileName(char* filename, char* extension);
void ResetData();
bool DFSTplgOrdering();
bool DFSVisiting(int ver);

/*!variables declaration starts here*/
int*  graph         = NULL;
int   verNum        = 0;      //number of input vertices
int*  ingoing       = NULL;   //indegree of vertices
int*  outgoing      = NULL;   //outdegree of vertices
int*  verVisitCount = NULL;   //time of visits charged on vertices
int*  edeVisitCount = NULL;   //time of traverses charged on edges
int*  color         = NULL;   //vertex color
int*  result        = NULL;   //output result pointer array
int   resType  = 0;
int   index    = 0;
```

Figure 1 Functions and variables

Before reading data direct in, we validate input files with file extension by validateFileName() function, then by manipulating file pointer, we can obtain total number of input vertices. Applying dynamic memory allocating, data from the input are properly read and stored regardless of total vertices numbers, which guarantees this graph structure great flexibility. Absolute path is also permitted for data import to ensure import success.

```
graph         = (int*)malloc((verNum+1)*(verNum+1)*sizeof(int));
ingoing       = (int*)malloc((verNum+1)*sizeof(int));
outgoing      = (int*)malloc((verNum+1)*sizeof(int));
color         = (int*)malloc((verNum+1)*sizeof(int));
result        = (int*)malloc( verNum   *sizeof(int));
verVisitCount = (int*)malloc((verNum+1)*sizeof(int));
edeVisitCount = (int*)malloc((verNum+1)*(verNum+1)*sizeof(int));
```

Figure 2 Dynamic memory allocation

Outdegrees and indegrees values are initialized as attributes to the start and end vertices by traversing each edge during data import. In the DFS algorithm, each time a vertex is visited and operated on (e.g. color changes), we update its counter for charge.

Runtime Analysis

We traverse each vertex to see if there is any vertex has its in-degree equal to 0 and its color being white, then put it in our result array. Then we traverse each edge with this 0 in-degree vertex as start vertex, if the color of the end vertex is gray then we add the vertex in to our result array. Here index is used to keep record of vertex priority. Therefore, each edge is visited once (or to say operated once) and each vertex is visited(operated) no more than twice. Hence, total number of operations charged on edges is |E|, and total number of operations charged on vertices is O(|V|), so our total number of operations (C commands) charged is thus O(|V|+|E|) with a constant factor equals to 2.
(**Note**: Since we handle the indegree and outdegree calculation under our DataImport() function, and there is no further modification or update on the values of degrees afterwards, we consider the calculation as part of the initialization and therefore do not charge the corresponding operations on vertices nor edges in our algorithm and do not include them in counting.)

Table 1 Result of number of operations

input	Total charges on vertices	Total charges on edges	Total operations
In1	25	24	49
In2	33	34	67
In3	14	14	28 (Cycle detected!)
In4	71	163	234
In5	111	318	429
In6	152	556	708
In7	194	990	1184

Program actual runtime analysis

When analyzing actual runtime of the whole program, if taken degree initialization and updates for vertices into consideration, number of degree updates of a vertex should be no larger than (|V|-1) in the worst case. In this case the total operations charged on vertices is (|V|*(|V|-1)) and total number of operations charged on edges should be (2*|E|) since we need to traverse each edge to update degrees of the start and end vertices. Therefore, the total number of operations should be $O(|V|^2 + 2*|E|)$ in the worst case if taking degree updates into account and consider it in the actual runtime of the whole program.

Functionality evaluation of the program

Some extra codes are introduced in the program to make sure that the program functions well. For example, ValidateFile() function is introduced to ensure that the input files are validated. We also allow absolute path input for the program. Meanwhile, ResetData() function is also introduced to deal with corner cases. Applying dynamic memory allocating during data import enables the program to deal with inputs with various size.