# HOMEWORK 5

Points: 100 Topics: Graphs, topological sort.

Plagiarism/collusion: You should not read any code (solution) that directly solves this problem (e.g. implements DFS, topological sorting or other component needed for the homework). The graph representation provided on the Code page and the pseudocode and algorithm discussed in class provide all the information needed. If anything is unclear in the provided materials check with us. You can read materials on how to read from a file, or read a Unix file or how to tokenize a line of code, but not in a sample code that deals with graphs or this specific problem. E.g. you can read tutorials about these topics, but not a solution to this problem (or a problem very similar to it). You should not share your code with any classmate or read another classmate's code. You are allowed to use in your solution any of the code posted on our class Code webpage.

## Main program requirements

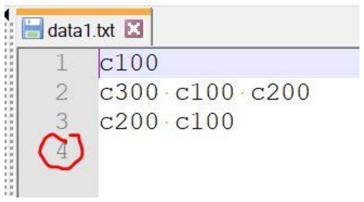
**In Canvas, under this assignment, there are 2 videos that explain the homework itself and how to do some of the work. Make sure you watch them**.

Given a list of courses and their prerequisites, compute the order in which courses must be taken so that when taking a course, all its prerequisites have already been taken.

See sample runs in run.pdf .

## Data, File Format and Size Constraints, Other Files

- The data (input) files are in Unix format (with Unix End-Of-Line). See the image at the end of this document.
- Size limits:
  - The file name will be at most 30 characters. E.g. *Long_FileName_With_30Chars.txt*
  - A course name will be at most 30 characters. E.g. *Long_CourseName_With_30Chars_1*
  - A line in the file will be at most 1000 characters.
  - There will be at most 50 lines with data (that is, at most 50 unique courses).
- The file ends with an empty new line. See:



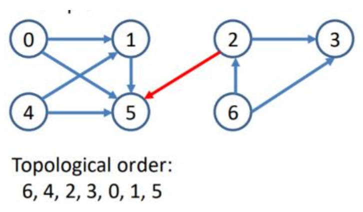
```
c100
c300 c100 c200
c200 c100
```

- Each line (except for the empty line) has one or more course names.
- Each course name is a single word (without any spaces). E.g. CSE1310 (with no space between CSE and 1310).
- There is no empty space at the end of the line.
- There is exactly one empty space between any two consecutive courses on the same line. (You do not need to worry about having tabs or more than one empty space between 2 courses.)
- The first course name on each line is the course being described and the following courses are the prerequisites for it. E.g.

  ```
  CSE2315 CSE1310 MATH1426
  ENGL1301
  ```

  The first line describes course `CSE2315` and it indicates that `CSE2315` has 2 prerequisite courses, namely: `CSE1310` and `MATH1426`. The second line describes course `ENG1301` and it indicates that `ENG1301` has no prerequisites.

- You can assume that there is exactly one line for every course, even for those that do not have prerequisites (see `ENGL1301` above). Count the number of lines in the file to get the total number of courses.
- The courses are not given in any specific order in the file.
- files :
  - run.pdf - sample runs
  - slides.txt – input file that will generate the graph shown in the image below. (The last digit in the course number is the same as the vertex corresponding to it in the drawn graph. You can also see this in the vertex-to-course name correspondence in the sample run for this file.)



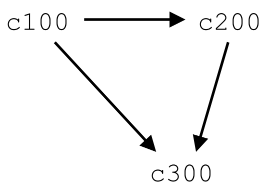
  - data0_easy.txt - If you cannot handle the above file format, this is an easier file format that you can use, but there will be 15 points lost in this case. More details about this situation are given in Part 3.

## Specifications:

1. Allowed:
    1. structs, macros, typedef.
    2. static memory for arrays (e.g. int arr[100]) and/or dynamic memory (with malloc() or calloc() )
2. Not allowed:
    1. Global variables or static variables (The exception is using macros to define constants for the size limits. E.g. `#define MAX_COURSE_LENGTH 30`
3. All the code must be in C (not C++, or any other language)
4. Code that does not compile receives 0 credit.
5. The program must read from the user a filename. The filename (as given by the user) will include the extension, but NOT the path. E.g.: *data0.txt*
6. You can open and close the file however many times you want.
7. You must create a directed graph corresponding to the data in the file.
    1. The graph will have as many vertices as different courses listed in the file.
    2. You can represent the vertices and edges however you want.
    3. You do NOT have to use a graph struct. If you can do all the work with just the 2D table (the adjacency matrix) that is fine.
    4. Must **implement the topological sorting covered in class**.
    5. Use the adjacency matrix to represent the graph edges.
    6. Adjacency list representation is not allowed.
    7. For each course that has prerequisites, there is an edge, from each prerequisite to that course. The direction of the edge indicates the dependency. The actual edge will be between the vertices in the graph corresponding to these courses.
    8. E.g. file data0.txt has:

        ```
        c100
        c300 c200 c100
        c200 c100
        ```

        Meaning:

        (The above drawing is provided here to give a picture of how the data in the file should be interpreted and the graph that represents this data. Your program should not print this drawing. See the sample run for expected program output.)

        From this data you should create the correspondence:

        ```
        vertex 0 - c100
        vertex 1 - c300
        vertex 2 - c200
        ```

Represent the graph using adjacency matrix (the row and column indexes are provided for convenience):

```
    |   0   1   2
-----------------
   0|   0   1   1
   1|   0   0   0
   2|   0   1   0
```

e.g. `E[0][1]` is `1` because vertex `0` corresponds to `c100` and vertex `1` corresponds to `c300` and `c300` has `c100` as a prerequisite. Notice that `E[1][0]` is not `1`.

9. **You must use the correspondence given here: vertex 0 for the course on the first line, vertex 1 for the course on the second line, etc.** See data1.txt and its sample run. For the data below

```
cs40 cs30 cs10 MATH1421
cs20
cs800
cs50 cs40
MATH101
MATH1421 MATH101
cs10
cs30 cs10
```

**You should make the vertex number to course name correspondence based on lines in file:**

**0 – cs40**

**1 – cs20 (NOT cs30)**

**2 – cs800 (NOT cs10)**

**3 – cs50 (NOT MATH1421)**

**4 – MATH101 (NOT cs20)**

**5 – MATH1421**

**6 – cs10**

**7 – cs30**

10. Print the courses in topological sorted order as shown in class. This should be done using the topological sorting covered in class, including the DFS covered in class. There is no topological order if there is a cycle in the graph; in this case print an error message. If in DFV-visit when looking at the (u,v) edge, the color of v is GRAY then there is a cycle in the graph (and therefore topological sorting is not possible). See the Lecture on topological sorting.

## Suggestions for improvements (not for grade)

1. CSE Advisors point out to students the "longest path through the degree". That is the longest chain of course prerequisites (e.g. CSE1310 ---> CSE1320 --> CSE3318 -->...). It gives a lower bound on the number

of semesters needed until graduation. Can you find this path? Can you calculate for each course the LONGEST chain ending with it? E.g. in the above example, there are 2 chains ending with c300 (size 2: just c100-->c300, size 3: c100-->c200-->c300) and you want to show longest path 3 for c300. Can you calculate this number for each course?

2. Allow the user to the enter a list of courses taken so far (from the user or from file) and print a list of the courses they can take (they have all the prerequisites for).
3. Ask the user to enter a desired number of courses per semester and suggest a schedule (by semester).

---

## Implementation suggestions

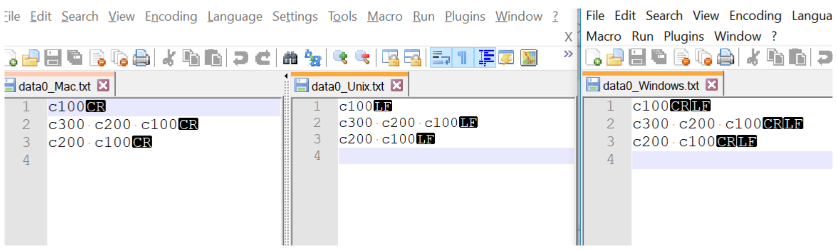
1. Creating vertex number-course name correspondence. Open the file twice.
    1. First, from each line in the file read only the first course name (ignore the rest of the line) and add it in an array of course names. Say we name this array `names`. This will ensure you have the correct vertex numbers-course name correspondence (as in the sample run).
    2. Open the file again (or use rewind() ) and read again from the first line on. Process each line to extract the edges and add them in your graph matrix.
2. To add an edge in the graph you will need to find the vertex number corresponding to a given course name. E.g. find that `c300` corresponds to vertex `1` and `c200` corresponds to vertex `2`. Now you can set `E[2][1]` to be `1`. To help with this, write a function that takes as arguments `names`, the array of [unique] course names, and one course name and returns the index of that course in the `names` array. Use that index as the vertex number. (This is like the `indexOf()` method in Java.)
3. Reading from file:
    1. If you cannot extract the different course names in a line from the file correctly, you can use a modified input file that shows, on each line, the number of courses. **Note that if you use this simplified type of input file you will lose 15 points**. If your program works with the "easy files", please name your C program *courses_graph_easy.c*, so that the TAs will know to run it with the easy input files.

       Instead of
       ```
       c100
       c300 c200 c100
       c200
       ```

       the file would have:
       ```
       1 c100
       3 c300 c200 c100
       1 c200
       ```

       that way the first data on each line is a number that tells how many courses (strings) are after it on that line. Everything is separated by exactly one space. All the other specifications are the same as for the original file (empty line at the end, no space at the end of any line, length of words, etc). See file data0_easy.txt

4. One tricky bug comes from having the wrong EOL in your input file. Check the EOL symbol in your input files. To see all the non-printable characters that may be in a file, find an editor that shows them. E.g. in Notepad++ : open the file, go to View -> Show symbol -> Show all characters. YOU SHOULD TRY THIS! In general, not necessarily for this homework, if you make the text editor show the white spaces, you will know if what you see as 4 empty spaces comes from 4 spaces or from one tab or show other hidden characters. This can help when you tokenize. E.g. here I am using Notepad++ to see the EOL for files saved with Unix/Mac/Windows EOL (see the CR/LF/CRLF at the end of each line


File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
File Edit Search View Encoding Langua
Macro Run Plugins Window ?
data0_Mac.txt
data0_Unix.txt
data0_Windows.txt

```
c100CR
c300 c200 c100CR
c200 c100CR
```

```
c100LF
c300 c200 c100LF
c200 c100LF
```

```
c100CRLF
c300 c200 c100CRLF
c200 c100CRLF
```

## What to submit

Submit only **courses_graph.c (or courses_graph_easy.c) .** Your program should be named courses_graph.c if it reads from the normal/original files. If instead it reads from the 'easy' files, name it courses_graph_easy.c .

As stated on the course syllabus, programs must be in C, and must run on omega.uta.edu, the VM, Ubuntu or Unix/Linux system.

**IMPORTANT:** Pay close attention to all specifications on this page, including file names and submission format. Even in cases where your answers are correct, points will be taken off liberally for non-compliance with the instructions given on this page (such as wrong file names, wrong compression format for the submitted code, and so on). The reason is that non-compliance with the instructions makes the grading process significantly (and unnecessarily) more time consuming. Contact the instructor or TA if you have any questions.

## Grading Criteria

a) Code that does not compile: 0 points. (-10 pts for each small syntax error, e.g. missing ; )
b) Code compiles with warnings: 20%-50% penalty
c) 6 pts - coding style: student name at the top, consistent indentation, variable names, comments.
d) 14 pts - No Valgring errors. If any memory-related error is reported (including invalid read or invalid write) all points are lost. If "conditional jump depends on uninitialized variable" errors are present but no other memory related errors, 10 out of the 14 points will be lost.

e) 15 pts - Code correctly reads from the posted files.
   - All 15 points will be lost if the 'easy' files are used (the ones that have the number of words at the beginning of each line - see details about this in the homework text)

f) 10 pts - Correct mapping vertex number to course name is printed.
   - **If this part is wrong, everything that shows vertex numbers or course names will not get any credit as it would depend on a wrong code (parts g) and i) below).**
   - This part is correct only if EVERY course name is mapped to the same index as in the sample output for EVERY test case.
   - If only one course number in one test case is mapped incorrect, it is all considered wrong.
   - Double-check that for cycle0.txt, data1.txt and slides.txt you get the same mapping as in the sample runs.

g) 25 pts - Correct edge information is created and printed (as either an adjacency matrix or adjacency list). The direction of each edge must be correct. Note that you do not need to draw an edge. The direction is inferred from your printing of the adjacency matrix or adjacency list. If e) is wrong, this part gets no credit as well.

h) 10 pts - program correctly identifies if there is a cycle and prints a message about that and it does NOT attempt to print the courses in topological order.

i) 20 pts - Courses are printed in correct topological order. It must show both the COURSE (e.g. c100 below) and the VERTEX number (e.g. vertex 0 below). If e) is wrong, this part gets no credit as well.

```
1. - c100 (corresponds to graph vertex 0)
2. - c200 (corresponds to graph vertex 2)
3. - c300 (corresponds to graph vertex 1)
```