CPSC 535 Advanced Algorithms
Project 1: Graduation Time

Prof. Doina Bein, CSU Fullerton

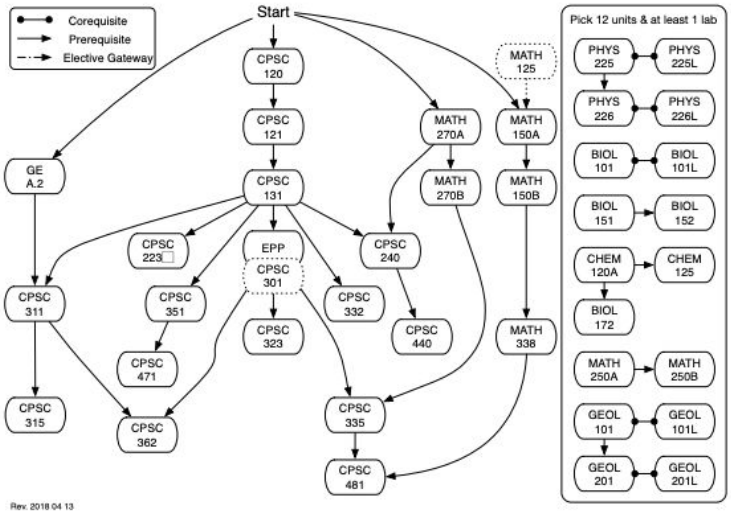
dbein@fullerton.edu

Introduction

In this project you will design and implement one algorithm related to directed acyclic graphs (DAG). You will design two algorithms, describe the algorithms using clear pseudocode, combine them and implement them using C/C++/C#/Java/Python, compile, test it, and submit BOTH the report (as a PDF file) and the files. Let G=(V,E) be a directed acyclic graph with N=|V| nodes and M=|E| arcs.

Estimating the graduation time

When pursuing a degree, a student needs to complete a number of credits in order to graduate. Credits are accumulated as the student takes various courses, some core courses, some elective. Except for a few courses, each other course has a number of course prerequisites that need to be taken before the respective course can be taken. For example, at California State University in Computer Science, in order for a student to graduate with a BS in CS, the students need to accumulate at least 120 credits, out of which 18 credits (i.e. 6 courses) are elective and the rest of 102 belong to core courses. In Figure 1, the core courses are drawn as a directed graph. For more details, see see http://www.fullerton.edu/ecs/cs/_resources/pdf/CPSC_undergraduate_handbook-2018-19.pdf.



We ignore the graphs on the right side and we focus only on the graphs on the left and center; that graph is a DAG. The problem requires computing how many semesters a student needs in order to graduate. This translates into computing the longest path in the DAG, since the number of hops on the longest path corresponds to the number of semesters needed to graduate. We ignore that there is a limit on the number of courses a student needs to take in any given semester. Before we compute the longest path, the problem requires re-labeling the nodes using topological sorting and using values in the range 0..N-1.

Topological sorting

The directed graph G=(V,E) is given in a text file called **input.txt** in which the first row contains the number of nodes N, the second row contains the # sign, and the next N rows contain the IDs of the nodes in some order, followed by a row with the # sign, followed by all the arcs. Each arc is a pair of two IDs, separated by space. Each ID is a string of maximum 8 characters.

An example of such a file **input.txt** is below, and the corresponding DAG is drawn in Figure 2.

```
6
#
A1
B1
C1
A2
B2
C2
#
A1 B1
B1 A2
A1 C1
C1 B2
C2 B2
```

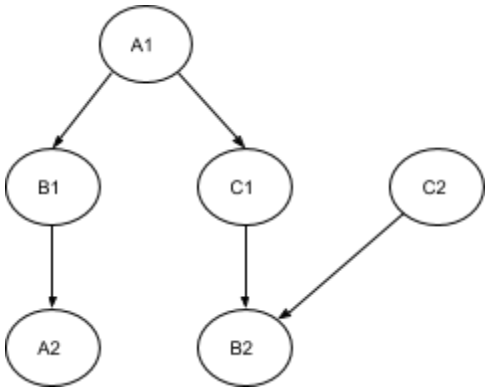


Figure 2.

~~There are some simplifying assumptions. The maximum number of nodes will be 100.~~

The set of edges is read from the file as an edge list, but there are other ways to represent a graph in C++. One easy approach is to use a 0-1 adjacency matrix. Recall that a 0-1 adjacency matrix is a square matrix of size NxN where N is the number of nodes, and each element is either 0 or 1:

A[i][j] = 1 if and only if there is an arc from node i to node j

We would like the nodes to be renumbered from 0..N-1 such that if (i,j) is an arc from i to j, then i < j. The topological sort will give an order in which courses need to be taken. Please refer to the class notes and also CLRS, pages 612-615 on how to perform topological sort on the given DAG.

Computing Longest Path in a DAG

The longest path problem is the problem of finding a simple path of maximal length in a DAG, i.e. among all possible simple paths in the DAG, the problem is to find the longest one. Since we assume that the DAG is unweighted, it suffices to find the longest path in terms of the number of edges.

For a general graph, computing the longest path is NP-hard, i.e. exponential solution. Well, why not simply enumerate all the paths, and find the longest one? Because, there can be exponentially many such paths! For example in Figure 3, there are 2^n different paths from vertex 1 to n, so computing the longest one takes exponential time.

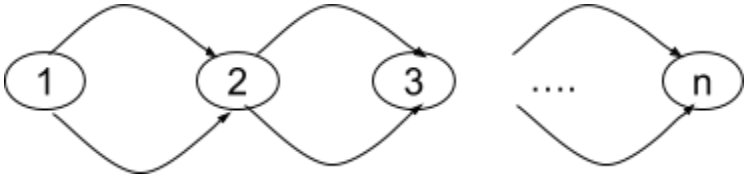


Figure 3

Fortunately, the longest path in a DAG does have optimal substructure, which allows us to solve it using dynamic programming. Please refer to the class notes and also
http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/Docs/longest-path-in-dag.pdf

What Output is expected

The output should display the list of nodes in topological sorting, separated by whitespace (space, tab, new line, etc.) followed by a new line, then followed by the length of the longest path and a longest path.

For example, for the graph in Figure 2, a sample output will be:

```
Topological sorting:
C2  A1  C1  B2  B1  A2
Length of the longest path:
2
A longest path:
A1  C1  B2
```

Grading rubric

The total grade is 35 points. Your grade will be comprised of three parts, Form, Function, and Report:

- Function refers to whether your code works properly (18 points).
- Form refers to the design, organization, and presentation of your code. The instructor will read your code and evaluate these aspects of your submission (6 points):
    - README.md completed clearly = 2 points
    - Style (whitespace, variable names, comments, helper functions, etc.) = 2 points
    - C++ Craftsmanship (appropriate handling of encapsulation, memory management, avoids gross inefficiency and taboo coding practices, etc.) = 2 points
- Report (11 points) divided as follows:
    - Summary of report document (2 points)
    - Pseudocode of overall algorithm, topological sort, and longest path (2 points each, total 6 points)
    - Four screenshots: one for the group members and three for the three sample input files (1 point each, total 3 points)

Obtaining and Submitting Code

This document explains how to obtain and submit your work:

GitHub Education / Tuffix Instructions

Here is the invitation link for this project:
https://classroom.github.com/g/NBdn4Ul5

Implementation

You are provided with the following files.

1. README.md contains a brief description of the project, and a place to write the names and CSUF email addresses of the group members. You need to modify this file to identify your group members.
2. LICENSE contains a description of the MIT license.
3. biggraph.txt contains a graph with 50 nodes, organized as input.txt file in the project description
4. graph01.txt contains a graph with 6 nodes, organized as input.txt file in the project description
5. graph02.txt contains the example from Figure 2, organized as input.txt file in the project description

What to Do

First, add your group members' names to README.md. Then write clear pseudocode for both topological sorting as well as the longest path in a DAG algorithms, describe what parameters are needed to execute the program and submit it as a PDF report.Your report should include the following:

1. Your names, CSUF-supplied email address(es), and an indication that the submission is for project 1.
2. A full-screen screenshot with your group member names shown clearly. One way to make your names appear in Atom is to simply open your README.md.
3. The pseudocode for the two algorithms
4. A brief description on how to run the code.
5. Three snapshots of code executing for the three given input files.

Then implement your algorithm in C/C++/C#/Java/Python. Submit your PDF by committing it to your GitHub repository along with your code.