# University of Westminster
School of Computer Science and Engineering

| 5SENG003W | Algorithms – Coursework (2022/23) |
|---|---|
| Module leader | Klaus Draeger |
| Unit | Coursework |
| Weighting: | 50% |
| Qualifying mark | 30% |
| Description | Checking acyclicity of directed graphs |
| Learning Outcomes Covered in this Assignment: | This assignment contributes towards the following Learning Outcomes (LOs):<br>- LO2: Be able to apply the theory for the effective design and implementation of appropriate data structures and algorithms in order to resolve the problem at hand;<br>- LO3: Be able to analyse, predict, compare and contrast the performance of designed and implemented algorithms, particularly in the context of processing data;<br>- LO4: Be able to use a range of typical data structures and collections as part of Application Programming Interfaces (APIs) offered by programming languages;<br>- LO5: Be able to apply the theory for the definition and implementation of novel algorithms. |
| Handed Out: | October 2022 |
| Due Date | 13:00, Monday, 9th January 2022 |
| Expected deliverables | **A zip file containing the source code in Java or C++,<br>a short report (no more than 3 pages pdf).** |
| Method of Submission: | Electronic submission on Blackboard via a provided link close to the submission time. |
| Type of Feedback and Due Date:<br><br>BCS CRITERIA MEETING IN THIS ASSIGNMENT | Written feedback within 15 working days.<br><br>**2.1.1 Knowledge and understanding of facts, concepts, principles & theories**<br>**2.1.3 Problem solving strategies**<br>**2.1.5 Deploy theory in design, implementation and evaluation of systems**<br>**2.2.2 Evaluate systems in terms of quality and trade-offs**<br>**2.3.2 Development of general transferable skills**<br>**3.2.2 Defining problems, managing design process and evaluating outcomes**<br>**4.1.1 Knowledge and understanding of scientific principles**<br>**4.1.2 Knowledge and understanding of mathematical and statistical principles**<br>**4.2.1 Use theoretical and practical methods in analysis and problem solving** |

**Assessment regulations**

Refer to section 4 of the “How you study” guide for undergraduate students for a clarification of how you are assessed, penalties and late submissions, what constitutes plagiarism etc.

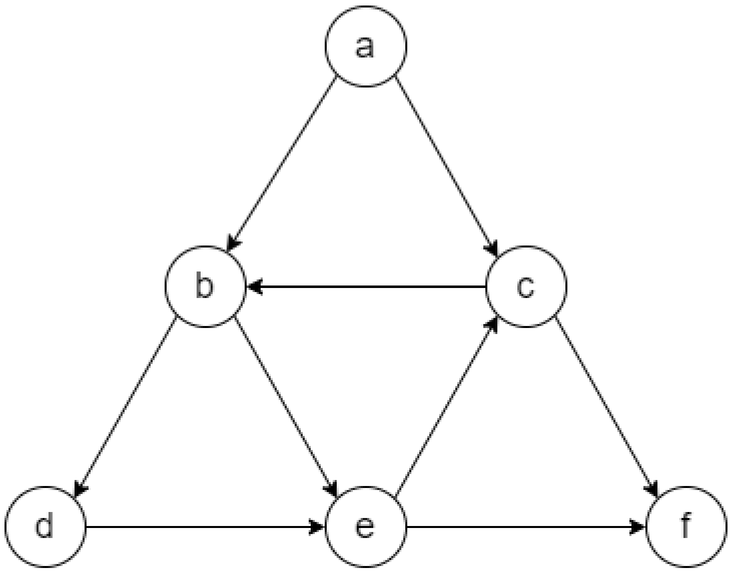
**Penalty for Late Submission**

If you submit your coursework late but within 24 hours or one working day of the specified deadline, 10 marks will be deducted from the final mark, as a penalty for late submission, except for work which obtains a mark in the range 40 – 49%, in which case the mark will be capped at the pass mark (40%). If you submit your coursework more than 24 hours or more than one working day after the specified deadline you will be given a mark of zero for the work in question unless a claim of Mitigating Circumstances has been submitted and accepted as valid.

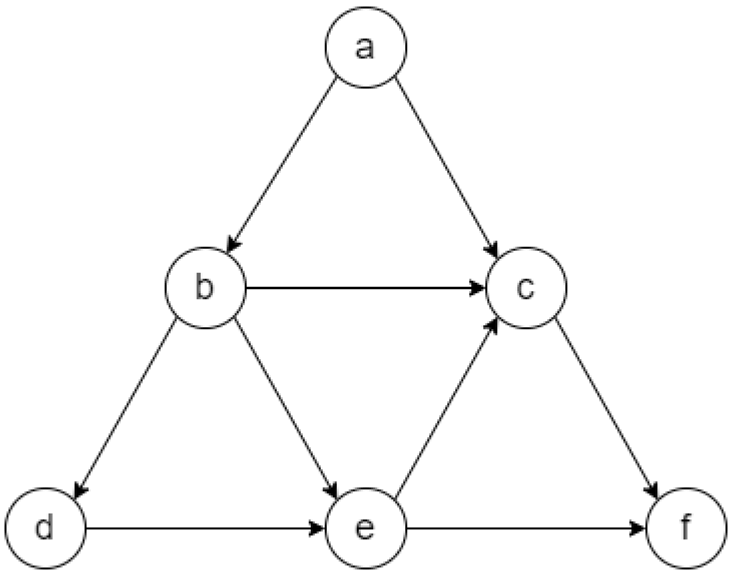
It is recognised that on occasion, illness or a personal crisis can mean that you fail to submit a piece of work on time. In such cases you must inform the Campus Office in writing on a mitigating circumstances form, giving the reason for your late or non-submission. You must provide relevant documentary evidence with the form. This information will be reported to the relevant Assessment Board that will decide whether the mark of zero shall stand. For more detailed information regarding University Assessment Regulations, please refer to the following website:**http://www.westminster.ac.uk/study/current-students/resources/academic-regulations**

# Coursework Description: Sliding puzzles

In this coursework, you are supposed to check whether a given directed graph is **acyclic**. For example, for this graph:



The answer would be "no" since there is a cycle **b->e->c->b**. On the other hand, if the edge between **b** and **c** was reversed:



The answer would now be "yes" since there is no cycle. In order to answer the question in general, there are different algorithms. We will now have a look at one of them.

## The sink elimination algorithm

A **sink** in a directed graph is a vertex with no outgoing edges, like vertex **f** in the examples above. An acyclic graph will always have a sink (can you figure out why this is true?). Of course the converse is not true: a graph with a sink is not necessarily acyclic (the first graph above is a counterexample).

But we can use sinks as the basis of an algorithm using this idea:

If a graph is acyclic, it has a sink. Also, removing that sink gives us again an acyclic graph (since any cycles would already have existed in the original graph), so there is again a sink, and we can repeat this until the graph is empty. That is, we use the following steps:

1. If the graph is empty, return "yes" (represented by a suitable value like true or 1).
2. If the graph has no sink, return "no" (represented by a suitable value like false or 0).
3. Otherwise, remove a sink from the graph and repeat.

For example, starting from the first graph above, we would first remove **f** and then return "no" since there is now no sink. For the second graph, we would remove **f, c, e, d, b, a** in this order and then return "yes" because there is nothing left. Try to go through this example with pen and paper to make sure you understand how it works.

## Tasks to be performed:

**Task 1 (10 marks).** Set up a project (Java or C++) as you did for the tutorial exercises.
**Task 2 (20 marks).** Choose and implement a data structure which can represent directed graphs. It must provide the necessary infrastructure for the algorithm, such as finding a sink and removing a vertex. If you use one of the graph representations from the lecture and tutorials, consider how efficiently the operations can be performed on each.
**Task 3 (10 marks).** Add a simple parser which can read the description of a graph from an input file. The input files will contain pairs of numbers, one per line, like:

```
1  2
3  1
2  5
```

Meaning that there are edged from vertex 1 to vertex 2 etc.
**Task 4 (20 marks).** Implement an algorithm to determine if the given graph is acyclic. You can use the one presented above, but if you are curious and want to do some more research, you could also use a different one. The implementation should produce additional output to show how it obtained the answer. For example, for the sink elimination algorithm, it should print the sinks it has found and eliminated.
**Task 5 (10 marks).** Make your program find and output a cycle in the given graph in case it is not acyclic.
**Task 6 (30 marks).** Write a brief report (no more than 3 A4 pages) containing the following:

a) A short explanation of your choice of data structure and algorithm.
b) A run of your algorithm on two small benchmark examples, one of which is acyclic and the other one isn't. This should include the supporting information as described in Task 4.
c) A performance analysis of your algorithmic design and implementation. This can be based either on an empirical study, e.g., doubling hypothesis, or on purely theoretical considerations, as discussed in the lectures and tutorials. It should include a suggested order-of-growth classification (Big-O notation).

## To be submitted:

- Your zipped source code (for Tasks 1 to 5) in Java or C++. Your source code shall include header comments with your student ID and name.
- The report (PDF format) about the algorithmic performance analysis (Task 6). Your name and student ID number must be on the first page of the report.

## Coursework marking scheme:

| Criterion and range | Indicative mark | Comments |
|---|---|---|
| **Task 1 (0-10 marks)** | 8-10 | A compilable and executable project has been created and follows guidelines for writing clear code such as https://introcs.cs.princeton.edu/java/11style |
| | 4-7 | A compilable and executable project has been created but does not follow programming guidelines such as of the *Java* related example above. |
| | 0-3 | No project has been created, or it is prone to compilation or runtime errors. |
| **Task 2 (0-20 marks)** | 16-20 | A data structure has been implemented, which satisfies the following principles of abstraction:<br>- Builds on top of already existing programming language specific data structures, e.g., array.<br>- Allows directed graphs as given by the input to be represented<br>- Fits the purpose of the intended algorithm and nature of the problem. |
| | 5-15 | A working data structure has been implemented but is limited in functionality. |
| | 0-4 | A data structure has not been implemented or does not work properly. |
| **Task 3 (0-10 marks)** | 8-10 | A parser has been implemented and is able to initialise the data structure from a given input file. It can handle any input file which has the format given in the problem description. |
| | 4-7 | A parser has been implemented and is able to initialise the data structure from some input files, but not others. |
| | 0-3 | A parser has not been implemented or does not work properly. |
| **Task 4 (0-20 marks)** | 16-20 | The correct answer ("yes" for acyclic graphs, "no" otherwise) can be found for any given graph. The implementation outputs the steps of the solution in enough detail that it can be checked independently. |
| | 5-15 | The correct answer can be obtained, but the implementation does not work for all possible inputs, or does not provide enough information to justify its result. |
| | 0-4 | Not done, or does not work properly |
| **Task 5 (0-10 marks)** | 8-10 | Produces a correct cycle in case the given graph contains one. |
| | 4-7 | Produces a correct cycle for some graph containing one. |

| | | |
|---|---|---|
| | 0-3 | Produces incorrect cycles. |
| **Task 6 (0-30 marks)** | 25-30 | The student has submitted a full report explaining their solution and its algorithmic performance based on sufficient empirical data or a formal analysis |
| | 6-24 | The student has submitted a report, but some of the contents (explanation of the data structure, explanation of the algorithm, discussion of algorithmic performance) are lacking. |
| | 0-5 | Not done, or no relevant contents. |