University of Central Lancashire UCLan School of Psychology and Computer Science	**UCLan Coursework Assessment Brief**	2021-2022
	Module Title: Computational Thinking Module Code: CO2412	
	Paths in Labelled Graphs	This assessment is worth 60% of the overall grade

Learning outcomes: Upon successful completion of this module, a student will be able to:

1. Use methods including logic and probability to reason about algorithms and data structures;
2. Compare, select, and justify algorithms and data structures for a given problem;
3. Analyse the computational complexity of problems and the efficiency of algorithms;
4. Use a range of notations to represent and analyse problems;
5. Implement and test algorithms and data structures.

THE AIM

The present coursework assessment serves the purpose of documenting the attainment of learning outcomes for the CO2412 module, specifically:

2. Designing algorithms and selecting and justifying data structures suitable for problems that involve analysing paths in a labelled graph;
3. Analysing the efficiency of your algorithm and measuring the performance of your implementation;
4. Using graphs as a notation and formal representation;
5. Implementing your algorithm and the required data structures and testing your implementation.

Accordingly, it consists of multiple tasks that contribute to the grade as follows:

A) 15%: **Exploration and discussion** of ideas for potentially suitable algorithms and data structures.
B) 10%: Justified **selection and design** of an algorithm and the required data structures.
C) 10%: Analysis of the algorithm as regards its **asymptotic efficiency** (in time and space/memory).
D) 10%: Analysis of the algorithm as regards its **correctness**.
E) 20%: **Implementation** of your selected design by a code that solves the problem specified below.
F) 10%: **Validation** of your code, confirming its correctness for representative instances of the problem.
G) 10%: **Performance measurement** based on representative instances of the problem.
H) 15%: **Documentation** and user guidelines.

THE PROBLEM

This problem deals with **sparse labelled graphs** where *both the nodes and the edges are labelled*.

The user or some other external software component provides the following **input**:

- A labelled graph g
- Two sequences p, q of edge labels

The graph g contains labelled nodes and labelled edges. Each of the sequences p and q contains one or multiple labels. It is up to you to decide on data structures for the graph and the sequences and on an appropriate way of passing them to your algorithm.

Your algorithm and implementation should determine whether there are any **nodes** in the graph that are simultaneously connected **both** by a path with edge labels corresponding to the sequence p **and also** by a path with edge labels corresponding to the sequence q. See the section right below for an illustration.

The **output** should communicate the following information to the user or an external software component:

- Whether or not *at least one* pair of nodes *m*, *n* exists in the graph *g* such that *m* is connected to *n* via a path consisting of edges labelled *p* and also by a path consisting of edges labelled *q*.

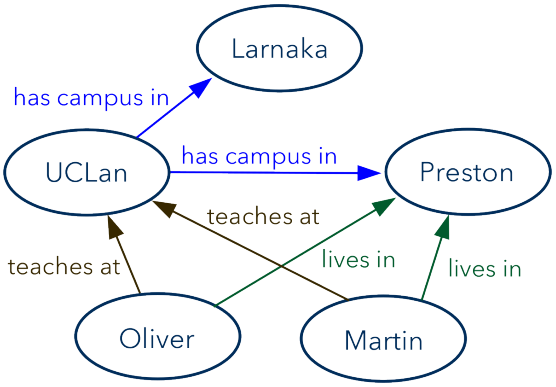
  To satisfy this criterion, the first edge of such a path from *m* to *n* should be labelled *p*[0], the second edge should be labelled *p*[1] (if *p* has more than one element), and so on, and similarly for a second path with edge labels from *q*, connecting the same nodes *m* and *n*.

- If that is the case, a pair of node labels *label_m*, *label_n* should be returned (or communicated to the user or external software component in a suitable way) where *label_m* is the label of *m* and *label_n* is the label of *n* for two nodes *m* and *n* matching the description above.

You can expect the typical input for *g* to be a sparse graph, but your algorithm and code should also function correctly in the case of a dense graph.

EXAMPLE

In the example below, Python list notation is used for the edge label sequences *p* and *q*. Python tuple (pair) notation is used for pairs of node labels. You are not tied to using Python or these specific data structures.



If *g* is the graph given above and the two sequences of edge labels are *p* = ["teaches at", "has campus in"] and *q* = ["lives in"], there are two pairs of nodes connected by paths with the appropriate edge labels. The corresponding pairs of node labels would be ("Oliver", "Preston") and ("Martin", "Preston"). Your algorithm would not need to find both of these pairs, it is sufficient to find one of them.

GUIDANCE ON TASKS

A) Exploration and discussion: Discuss what algorithm design strategies are applicable to the problem, and how a rough structure or concept for multiple potentially suitable algorithms might look like. Evaluate what needs for data structures arise in addressing the problem using these algorithms, *e.g.*, for storing the graph data, for paths in the graph, for sequences of nodes, or for storing and recalling partial solutions (these are examples, not a list of items that you should address – this is up to your judgment). Discuss multiple potential data structures that might be used to address these needs. Refer to the academic literature where appropriate. Limit yourself to a maximum of four pages.

B) Selection and design: Based on advantages and disadvantages of multiple potential approaches vis-a-vis the requirements from addressing the problem, make a justified decision in favour of one specific algorithm and associated data structures. (The advantages and disadvantages should be discussed in task A, not here; in part B it is enough to reference them and explain in what way they justify your design choices.) Describe your final algorithm and your final data structures following disciplinary good practice. Refer to the academic literature where appropriate. Make it absolutely clear beyond doubt for what purposes you will reuse libraries and/or external code to which you have the license, and what part of the implementation you will do by yourself, as only your own work can be considered for grading. Limit yourself to a maximum of four pages.

C) Asymptotic efficiency: Determine the worst-case time efficiency and the worst-case space efficiency of your algorithm, using Landau notation ("big-O notation"), *i.e.*, with respect to asymptotic efficiency. Clearly define your measure(s) for the problem size, such as the number of nodes, the number of edges, the maximum degree of a node, the number of elements of the sequences p and q, and/or anything else that you deem appropriate. Also discuss how you would expect your algorithm and its implementation to perform on average for typical sparse labelled graphs (it is up to you to explain what you would regard as typical/average). Limit yourself to a maximum of four pages.

D) Correctness: Justify beyond doubt that your algorithm indeed solves the problem correctly. This need not be a mathematical/logical proof, and certainly need not be based on line-by-line formal verification. Target an average reader with a background in computing. Limit yourself to a maximum of four pages.

E) Implementation: The code solving the problem. You may work in Python and using Jupyter Notebook or alternatively use any other widespread programming language, compiler/interpreter, and environment. (In that case, if your submission does not run on the system used by UCLan faculty during grading, you may be asked for a live demonstration.) Where using libraries or pre-existing code, make sure that you have the right to do so, and that this is pointed out clearly both in the code itself and in your text addressing task B. Only your own work can be taken into account for grading.

F) Validation: Develop code that generates representative instances of the problem (that is, typical graphs and typical sequences of edge labels), and use them to validate your implementation, corroborating that it is a correct implementation of your algorithm and indeed solves the problem correctly. Limit yourself to a maximum of two pages for a brief statement summarizing the validation procedure and its outcome.

G) Performance measurement: Conduct a performance measurement as a function of the problem size, similar to how it was done in the lectures and lab sessions of our module. Limit yourself to a maximum of two pages for a brief statement summarizing the performance measurement outcome, the conditions under which it was conducted (CPU architecture, *etc.*), and to what extent it agrees with the analysis from task C.

H) Documentation: A concise (readme-style) documentation should be provided that explains how exactly your code can be used **i)** directly by a user, and **ii)** by another programmer who would like to include your implementation into a larger body of work. This should make it clear in which way the input is communicated to your program, and in which way your program is communicating its output. (Typically, this would be through arguments and the return value of a function; however, file I/O or any other viable interface in line with the specification is equally acceptable.) Keep it brief and limit yourself to a maximum of two pages. Additionally, the source code itself needs to be sufficiently documented to be intelligible to an average programmer; it should be recognizable how the code development follows the algorithm design.

GRADING CRITERIA

1st criteria (70% to 100%)

- The work provides evidence of thorough and substantial independent algorithm design and code development. The submitted material makes it easy for a developer or user with a disciplinary background in computing to test, employ, and further develop the code. The implementation makes it easy for users to provide their own graph data, and it is well documented how a user would need to proceed for that purpose.
- The arguments in favour of the selected algorithms and data structures are correct and convincing, and it is documented in detail how the design choices relate to the requirements imposed by the problem. The employed and potential alternative algorithms and data structures are summarized concisely, with a recognizable focus on aspects that are relevant to the present work, referencing the original sources and/or more recent relevant academic literature.
- Validation and performance measurements are conducted in line with and potentially exceeding the specification. A sound reasoning is provided for what could be typical/representative instances of the problem in practice, grounding the average-case efficiency analysis, and such representative instances are systematically generated and used for the validation and the performance

measurements. Multiple well-characterized measures of the problem size are employed correctly for the asymptotic efficiency analysis. The validation is successful, and a clear connection is made between the outcome of the performance measurements and the average-case asymptotic time efficiency, plausibly discussing explanations for apparent deviations if any such deviations occur. Limitations as to what can be achieved with a reasonable amount of computational resources are deduced correctly from the efficiency analysis and the performance measurements.

- It is proven beyond doubt and intelligibly to disciplinary experts that the algorithm solves the problem correctly in all cases. The documentation of the code makes it easy to appreciate that the code is indeed an implementation of the suggested algorithm.

2.1 criteria (60% to 69%)

- The work provides evidence of substantial independent algorithmic thinking and code development. The submitted material makes it easy for a developer or user with a disciplinary background in computing to test and further develop the code. The implementation makes it viable for competent developers to provide their own graph data, and sufficient documentation for this is provided.
- Some intelligible reasoning is provided to justify the design choices; it is outlined in what way the design choices are grounded in the requirements imposed by the problem. The employed algorithms and data structures are introduced jointly with potential alternatives, properly identifying them by correct technical terms in line with disciplinary good practice and including relevant literature references.
- Validation and performance measurements are conducted in line with good practices. Representative instances of the problem are systematically generated and used for the validation and the performance measurements. A well-characterized and suitable measure for the problem size is provided and employed correctly for the asymptotic efficiency analysis. The validation is successful, and the outcome of performance measurements is compared to the efficiency analysis correctly.
- A sound and plausible argument is made in favour of the correctness of the suggested algorithm.

2.2 criteria (50% to 59%)

- The work provides evidence of the competency to implement algorithms and data structures concerning all the relevant aspects of the module. The implementation is sufficiently well-documented such that a developer with a disciplinary background in computing is enabled to test and further develop the code. It is possible for competent developers to provide their own graph data without having to rewrite any substantial part of your code.
- The design choices are discussed, including an intelligible statement on the relation between the design and the requirements imposed by the problem. The employed algorithms and data structures are introduced jointly with potential alternatives, properly identifying them by correct technical terms in line with disciplinary good practice.
- There is some evidence of a systematic validation and assessment of the resource requirements. The validation is successful in all cases, or an explanation is provided if there remain any potential issues in rare special cases. It is made intelligible why we may expect the suggested algorithm to be correct.

Pass criteria (40% to 49%)

- The work provides evidence of the competency to implement algorithms and data structures concerning some of the relevant aspects of the module, addressing the problem in a meaningful and intelligible way. The implementation is sufficiently documented to permit testing by a developer with a disciplinary background in computing.
- An intelligible statement is included concerning the relation between the design and the problem. The employed algorithms and data structures are described sufficiently to be intelligible.

PREPARATION FOR THE ASSESSMENT

Module weeks 1 to 11, and literature recommended therein. We will continue to discuss graph-related problems in the new year. Confer the course website: https://home.bawue.de/~horsch/teaching/co2412/

Note that a simple data structure implementation for sparse labelled graphs using incidence lists has been

provided and discussed in the lecture (see incidence-list-graph.ipyb Jupyter Notebook). You are allowed but not required to reuse, extend, or tailor this particular implementation for the present problem. The same applies to all the other examples and Jupyter Notebooks that have been or will be discussed in our module.

RELEASE DATE AND HAND-IN DEADLINE

Assessment release date: 16th December 2021
Assessment deadline date and time: 2nd April 2022, 24.00 BST.

SUBMISSION DETAILS

To be submitted by 2nd April 2022, end of day (24.00 hours, BST time zone), via Blackboard. The whole work (software and other documents) should be submitted as one archive file, *e.g.*, in zip, tar.gz, or tar.bz2 format. It should contain one folder **src/** with all source files (*e.g.*, py or ipynb files) and, additionally, one folder **doc/** with all relevant documentation and gradable material other than source files.

HELP AND SUPPORT

- If you have not yet made the university aware of any disability, specific learning difficulty, long-term health or mental health condition, please complete a Disclosure Form. The Inclusive Support team will then contact to discuss reasonable adjustments and support relating to any disability. For more information, visit the Inclusive Support site.
- To access mental health and wellbeing support, please complete our online referral form. Alternatively, you can email wellbeing@uclan.ac.uk, call 01772 893020 or visit our UCLan Wellbeing Service pages for more information.
- If you have any other query or require further support you can contact "The <i>," The Student Information and Support Centre. Speak with them for advice on accessing university services as well as library services. Whatever your query, their expert staff will be able to help and support you. For more information, how to contact them, and opening hours visit Student Information and Support Centre.
- If you have any valid mitigating circumstances, entailing that you cannot meet an assessment submission deadline, and you wish to request an extension, you will need to apply online prior to the deadline.

Disclaimer: The information provided in this assessment brief is correct at time of publication. In the unlikely event that any changes are deemed necessary, they will be communicated clearly via e-mail and a new version of this assessment brief will be circulated.	Version: U