University of Central Lancashire
UCLan

CO2412
Computational Thinking

Module structure
What is computational thinking?

Where opportunity creates success

Resources

Recommended literature:

- K. Erciyes, *Discrete Mathematics and Graph Theory*, Cham: Springer (ISBN 978-3-03061114-9), **2021**.
- P. Sanders, K. Mehlhorn, M. Dietzfelbinger, R. Dementiev, *Sequential and Parallel Algorithms and Data Structures*, Cham: Springer (ISBN 978-3-03025208-3), **2021**.

Resources

Recommended literature:

- K. Erciyes, *Discrete Mathematics and Graph Theory*, Cham: Springer (ISBN 978-3-03061114-9), **2021**.
- P. Sanders, K. Mehlhorn, M. Dietzfelbinger, R. Dementiev, *Sequential and Parallel Algorithms and Data Structures*, Cham: Springer (ISBN 978-3-03025208-3), **2021**.

Additional references for special topics, *e.g.*, today:

- K. Brennan, M. Resnick, "New frameworks for studying and assessing the development of computational thinking," in *Proceedings of AERA 2012*, Cambridge, MA: Academic Press, **2012**.

Resources

Recommended literature:

- K. Erciyes, *Discrete Mathematics and Graph Theory*, Cham: Springer (ISBN 978-3-03061114-9), **2021**.
- P. Sanders, K. Mehlhorn, M. Dietzfelbinger, R. Dementiev, *Sequential and Parallel Algorithms and Data Structures*, Cham: Springer (ISBN 978-3-03025208-3), **2021**.

Additional references for special topics, *e.g.*, today:

- K. Brennan, M. Resnick, "New frameworks for studying and assessing the development of computational thinking," in *Proceedings of AERA 2012*, Cambridge, MA: Academic Press, **2012**.

Course website:

- https://home.bawue.de/~horsch/teaching/co2412/

All essential information will be made accessible through the course website.

What is computational thinking?

CO2412

5th October 2021

What is computational thinking?

Brennan & Resnick (2012) "have developed a definition of computational thinking that involves three dimensions:

- **computational concepts** (the concepts *designers* employ [...]),
- **computational practices** (the practices *designers* develop [...]), and
- **computational perspectives** (the perspectives *designers* form about the world around them and about themselves)."

Computational thinking is more about *design* than about implementation.

What is computational thinking?

Brennan & Resnick (2012) "have developed a definition of computational thinking that involves three dimensions:

- **computational concepts** (the concepts *designers* employ […]),
- **computational practices** (the practices *designers* develop […]), and
- **computational perspectives** (the perspectives *designers* form about the world around them and about themselves)."

Computational thinking is more about *design* than about implementation.

These concepts, practices, and perspectives are easy to develop even in children; e.g., for Scratch as evaluated by Brennan & Resnick (2012) at the time, "hundreds of thousands of young creators (mostly between the ages of 8 and 16)".

What is computational thinking?

Brennan & Resnick (2012) "have developed a definition of computational thinking that involves three dimensions:

- **computational concepts** (the concepts *designers* employ [...]),
- **computational practices** (the practices *designers* develop [...]), and
- **computational perspectives** (the perspectives *designers* form about the world around them and about themselves)."

Computational thinking is more about *design* than about implementation.

These concepts, practices, and perspectives are easy to develop even in children; e.g., for Scratch as evaluated by Brennan & Resnick (2012) at the time, "hundreds of thousands of young creators (mostly between the ages of 8 and 16)". *Our module will address the theory behind computational thinking*.

Programming practice

The official programming language of the module is Python.

Programming practice

The official programming language of the module is Python.

However, Computational Thinking is a theoretical course.

It does not matter what programming languages or environments you use to implement the considered data structures and algorithms.

Any code that is well written and documented will be accepted as part of the solution to a problem from our practical/tutorial sessions.

Programming practice

The official programming language of the module is Python.

However, Computational Thinking is a theoretical course.

It does not matter what programming languages or environments you use to implement the considered data structures and algorithms.

Any code that is well written and documented will be accepted as part of the solution to a problem from our practical/tutorial sessions. If it does not run on my system (a rather average Linux installation) or if there are any other issues due to the employed environment, I may ask for clarification, e.g., by demonstrating your solution on the system where it was implemented.

In extreme, unexpected cases you may be required to rewrite code in Python.

Programming practice

Algorithms are best discussed using **pseudocode**:

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Since the analysis of an algorithm should occur at a level of abstraction higher than that of its practical implementation (as code), simple generally comprehensible notations are often more suitable than syntactically correct code in any given programming language.

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.

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.

program analysis

algorithm design

graphs and trees

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.

program analysis

algorithm design

graphs and trees

logic

randomness and probability

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.

program analysis

algorithm design

graphs and trees

logic

randomness and probability

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.

program analysis

algorithm design

graphs and trees

logic

randomness and probability

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.

program analysis **algorithm design** **graphs and trees** **logic** **formal languages** **complexity** **randomness and probability**

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.

program analysis

algorithm design

graphs and trees

logic

formal languages

complexity

randomness and probability

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.**

program analysis **algorithm design** **graphs and trees** **logic** **formal languages** **complexity** **randomness and probability**

Module structure

On the topic of **program analysis**, we will:

- Consider the space (memory) and time efficiency of algorithms;
- Describe asymptotic scaling behaviour using Landau $O(n)$ notation;
- Analyse algorithms formally via pre-/postconditions of statements;
- Review concurrency and scalability for massively-parallel computing.

program analysis **algorithm design** **graphs and trees** **logic** **formal languages** **complexity** **randomness and probability**

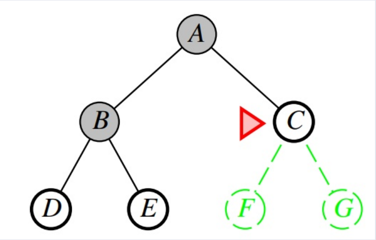
Module structure

On the topic of **algorithm design**, we will:

- Compare and apply algorithm design strategies such as recursion, divide-and-conquer, greedy algorithms, dynamic programming;
- Look at common data structures and their specification and implementation;
- Apply algorithm design to *sorting* as a highly relevant use case.

program analysis | **algorithm design** | **graphs and trees** | **logic** | **formal languages** | **complexity** | **randomness and probability**

Module structure



On the topic of **graphs and trees**, we will:

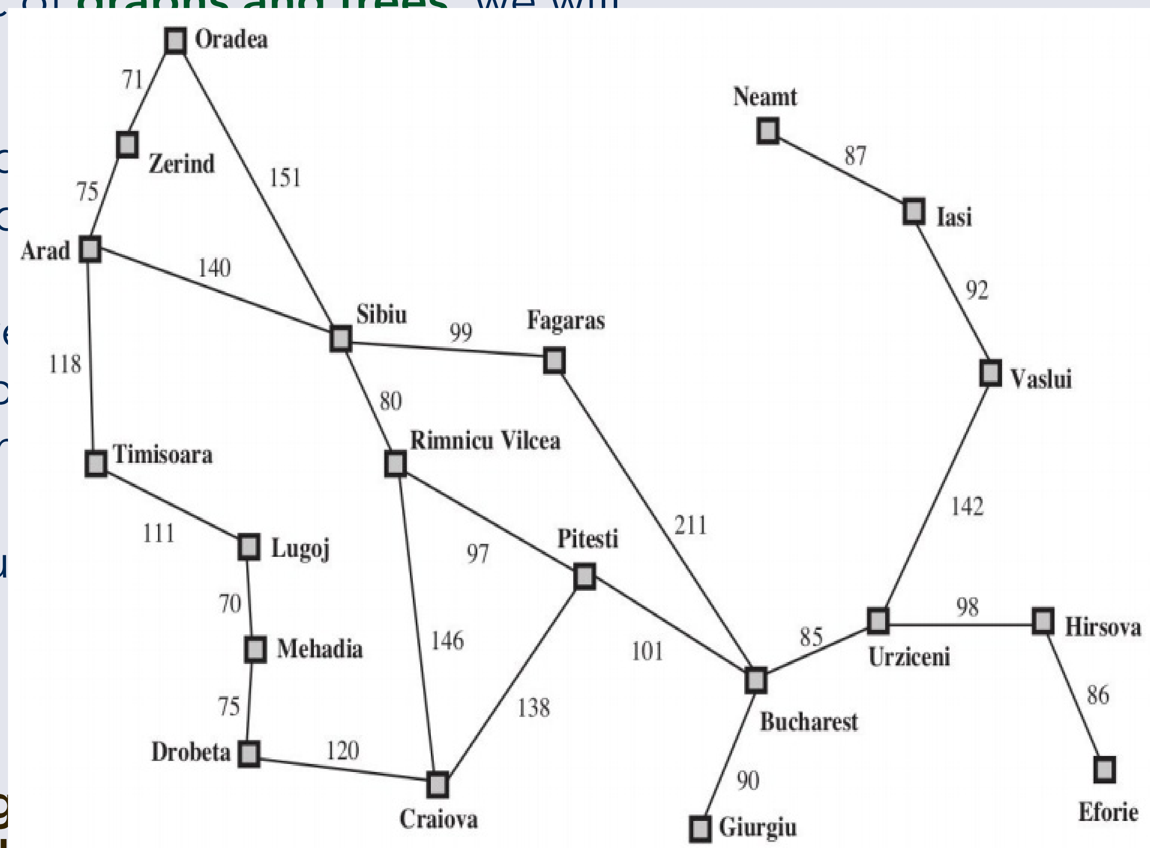
- Introduce graph theory and its basic definitions and concepts, including trees as a special case;
- Address basic tasks/problems when dealing with graphs, e.g., computing the shortest paths or connected components, strategies for graph traversal, and the application of trees to sorting and searching;
- Discuss numerical and mathematical representations of graphs.

program analysis **algorithm design** **graphs and trees** **logic** **formal languages** **complexity** **randomness and probability**

Module structure

On the topic of graphs and trees, we will:

- Introc ... pts, incluc
- Addre ... e.g., comp ... strategies for graph ... searching;
- Discu ... aphs.



program analysis

alg design

and trees

logic

languages

randomness nd probability

Module structure

On the topic of **logic**, we will:

- Introduce propositional logic and the semantics of logical formulas;
- Consider deductive logical reasoning by inference and address basic problems for logical expressions such as their satisfiability;
- Introduce first-order logic as a powerful formalism by which most application scenarios, including from program analysis, can be covered.

program analysis **algorithm design** **graphs and trees** **logic** **formal languages** **complexity** **randomness and probability**

Module structure

On the topic of **formal languages**, we will:

- Formalize the definition of a computational problem in terms of a language and the word problem for a given language;
- Introduce ways for specifying formal languages, e.g., by regular expressions and more generally by generative grammars;
- Introduce finite automata and the Turing machine, a simple theoretical computer model for purposes of decidability and complexity analysis.

program analysis **algorithm design** **graphs and trees** **logic** **formal languages** **complexity** **randomness and probability**

Module structure

On the topic of **complexity**, we will:

- Remove ourselves from considering specific algorithms by introducing complexity, i.e., the efficiency of the best possible algorithm for a given problem, as an additional layer of abstraction;
- Introduce the hierarchy of complexity classes, NP-complete problems, and the P = NP problem;
- Characterize the complexity of common problems from graph theory and from logic.

program analysis **algorithm design** **graphs and trees** **logic** **formal languages** **complexity** **randomness and probability**

Module structure

On the topic of **randomness and proabability**, we will:

- Introduce and/or review basic concepts from probability theory;
- Apply statistics and discrete mathematics to probability;
- Discuss randomness and random number generators;
- Consider randomized algorithms that can help at addressing computationally challenging problems.

program analysis **algorithm design** **graphs and trees** **logic** **formal languages** **complexity** **randomness and probability**

Grading

Number of Assessments	Form of Assessment	% weighting	Size of Assessment/ Duration/ Wordcount	Category of assessment	Learning outcomes being assessed
1	Examination	40%	1.5 hours	Written exam	1,2,3,4
1 (split into parts)	Practical work involving the selection, implementation and evaluation of algorithms and data structures	60%	2,000 words equivalent	Coursework	1,2,3,4,5

To pass this module, you must achieve a grade of 40% or above aggregated across all the assessments.

University of Central Lancashire
UCLan

CO2412 Computational Thinking

Module structure
What is computational thinking?

Where opportunity creates success