CSI2101 Discrete Structures: Introduction

Lucia Moura

Winter 2010

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking
  4. quantitative thinking (counting)

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking
  4. quantitative thinking (counting)
  5. analytical thinking

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking
  4. quantitative thinking (counting)
  5. analytical thinking
  6. applied thinking

Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking
  4. quantitative thinking (counting)
  5. analytical thinking
  6. applied thinking
- Question: How these 5 aspects appear in the the activities listed above?

Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))))

Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))))
- propositional logic
  propositional calculus allows us to make logical deductions formally

Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))))
- propositional logic
  propositional calculus allows us to make logical deductions formally
- predicate logic
  make a proposition to depend on a variable and we get a predicate; here the logical deductions include quantifiers (for all, there exists) in front of the predicates)

Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))))
- propositional logic
  propositional calculus allows us to make logical deductions formally
- predicate logic
  make a proposition to depend on a variable and we get a predicate; here the logical deductions include quantifiers (for all, there exists) in front of the predicates)
- methods of proof: direct, by contraposition, by contradiction
  use what you learned in formal/symbolic logic, to guide your reasoning on mathematical proofs (written in paragraph form)

Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))))
- propositional logic
  propositional calculus allows us to make logical deductions formally
- predicate logic
  make a proposition to depend on a variable and we get a predicate; here the logical deductions include quantifiers (for all, there exists) in front of the predicates)
- methods of proof: direct, by contraposition, by contradiction
  use what you learned in formal/symbolic logic, to guide your reasoning on mathematical proofs (written in paragraph form)
- logic in programming
  imperative programming: conditional statements (if-then-else, do-while)
  logic programming languages (e.g. prolog): uses the rules of predicate logic

Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))))
- propositional logic
  propositional calculus allows us to make logical deductions formally
- predicate logic
  make a proposition to depend on a variable and we get a predicate; here the logical deductions include quantifiers (for all, there exists) in front of the predicates)
- methods of proof: direct, by contraposition, by contradiction
  use what you learned in formal/symbolic logic, to guide your reasoning on mathematical proofs (written in paragraph form)
- logic in programming
  imperative programming: conditional statements (if-then-else, do-while)
  logic programming languages (e.g. prolog): uses the rules of predicate logic
- logic in circuits

Relational Thinking

- It deals with the following type of structures:

Relational Thinking

- It deals with the following type of structures:
  - sets

Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions

Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**

Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**

Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**

Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?

Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?
- Question 2: give examples of situations where they can be applied in computer science.

Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?
- Question 2: give examples of situations where they can be applied in computer science.
  - Databases: table=relation; record=n-ary tuple

Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?
- Question 2: give examples of situations where they can be applied in computer science.
  - Databases: table=relation; record=n-ary tuple
  - Dependency of task executions (partial ordering); topological sorting: order tasks respecting dependencies.

Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?
- Question 2: give examples of situations where they can be applied in computer science.
  - Databases: table=relation; record=n-ary tuple
  - Dependency of task executions (partial ordering); topological sorting: order tasks respecting dependencies.
  - Graphs: networks (communication, roads, social), conflicts (timetabling, coloring maps), hierarquies (rooted trees), diagrams (binary relations).

Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.

Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.
- Recursive definitions
  e.g. binary trees, recursive geometry/fractals

Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.
- Recursive definitions
  e.g. binary trees, recursive geometry/fractals
- Proofs by induction
  Prove that $P(n)$ is true for all $n \geq 0$:
  basis: $P(0)$ is true + induction step $P(n) \Rightarrow P(n+1)$

Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.
- Recursive definitions
  e.g. binary trees, recursive geometry/fractals
- Proofs by induction
  Prove that $P(n)$ is true for all $n \geq 0$:
  basis: $P(0)$ is true + induction step $P(n) \Rightarrow P(n+1)$
- Recursive data structures
  e.g. binary search trees

Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.
- Recursive definitions
  e.g. binary trees, recursive geometry/fractals
- Proofs by induction
  Prove that $P(n)$ is true for all $n \geq 0$:
  basis: $P(0)$ is true + induction step $P(n) \Rightarrow P(n+1)$
- Recursive data structures
  e.g. binary search trees
- Recursive algorithms
  e.g. binary search, mergesort, solving towers of Hanoi.

Quantitative Thinking

- counting,

Quantitative Thinking

- counting,
- combinations, permutations, arrangements,

Quantitative Thinking

- counting,
- combinations, permutations, arrangements,
- the pigeonhole principle,

Quantitative Thinking

- counting,
- combinations, permutations, arrangements,
- the pigeonhole principle,
- **discrete probability,**

Quantitative Thinking

- counting,
- combinations, permutations, arrangements,
- the pigeonhole principle,
- **discrete probability,**
- **counting operations in algorithms,**

Quantitative Thinking

- counting,
- combinations, permutations, arrangements,
- the pigeonhole principle,
- **discrete probability,**
- **counting operations in algorithms,**
- **estimating growth of functions, big-Oh notation.**

Analytical Thinking

- Apply previous tools to analyze problems of interest such as:

Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.

Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - loop invariants,

Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - loop invariants,
    - program correctness and verification.

Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - loop invariants,
    - program correctness and verification.
  - Writing algorithms that are efficient.

Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - loop invariants,
    - program correctness and verification.
  - Writing algorithms that are efficient.
    - algorithm complexity,

Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - loop invariants,
    - program correctness and verification.
  - Writing algorithms that are efficient.
    - algorithm complexity,
    - analysis of algorithms.

Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - loop invariants,
    - program correctness and verification.
  - Writing algorithms that are efficient.
    - algorithm complexity,
    - analysis of algorithms.
- Question: How previous tools can be applied in each of the above areas?

Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - loop invariants,
    - program correctness and verification.
  - Writing algorithms that are efficient.
    - algorithm complexity,
    - analysis of algorithms.
- Question: How previous tools can be applied in each of the above areas?
  - This question will be answered more fully by the studies in this course.

Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.

Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills

Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills
  - modelling

Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills
  - modelling
- Before using tools we need to learn the language and methods.

Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills
  - modelling
- Before using tools we need to learn the language and methods.
- A lot of the course will focus on acquiring the mathematical skills. But we don't want to lose sight of their use in applications.

Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills
  - modelling
- Before using tools we need to learn the language and methods.
- A lot of the course will focus on acquiring the mathematical skills. But we don't want to lose sight of their use in applications.
- Here we discuss some application problems illustrated in the following slides by Prof. Zaguia (2008): IntroZaguia2008.pdf

Calendar description:

CSI2101 Discrete Structures (3,1.5,0) 3 cr. Discrete structures as they apply to computer science, algorithm analysis and design. Predicate logic. Review of proof techniques; application of induction to computing problems. Graph theory applications in information technology. Program correctness, preconditions, postconditions and invariants. Analysis of recursive programs using recurrence relations. Properties of integers and basic cryptographical applications. Prerequisite: MAT1348.

Objectives:

- Discrete mathematics form the foundation for computer science; it is essential in every branch of computing.
- In MAT1348 (discrete mathematics for computing) you have been introduced to fundamental problems and objects in discrete mathematics.
- In CSI2101 (discrete structures) you will learn:
  - more advanced concepts in discrete mathematics
  - more problem solving, modelling, logical reasoning and writing precise proofs
  - how to apply concepts to various types of problems in computing: analyse an algorithm, prove the correctness of a program, model a network problem with graphs, use number theory in cryptography, etc.

Textbook

References:

- Kenneth H. Rosen, Discrete Mathematics and Its Applications, Sixth Edition, McGraw Hill, 2007.
  (same textbook as normally used for MAT1348; we will use different sections!)

Topic by topic outline: (approximate number of lectures, order may vary)

1. Introduction (1)
2. Propositionl logic (2)
3. Predicate logic (2)
4. Rules of inference/proof methods (2)
5. Basic number theory and applications (4)
6. Induction and applications. (4)
   Program correctness and verification (1)
7. Solving recurrence relations. Complexity of divide-and-conquer algorithms. (3)
8. Graphs (3)