Knight Foundation School of Computing and Information Sciences
COT 3100
Discrete Structures

Outline

Topic	Number of Lecture Hours	Outcome
1. Sets, Relations, and Functions 1.1. Operations on sets 1.2. Equivalence relations 1.3. Cardinality	10	1,7
2. Logic and Mathematical Reasoning 2.1. Propositional logic 2.2. Mathematical induction and recursion	10	2, 3, 7
3. Combinatorics 3.1. Combinatorial identities 3.2. Binomial theorem	5	4, 7
4. Directed and Undirected Graphs 4.1. Isomorphism of graphs 4.2. Paths 4.3. Adjacency matrices 4.4. Euler paths 4.5. Four-color problem 4.6. Planar graphs 4.7. Trees and tree traversal	10	5, 7
5. Boolean Algebras 5.1. Disjunctive normal form 5.2. Minimization of Boolean functions (Karnaugh maps)	5	6, 7

Learning Outcomes: (Familiarity → Usage → Assessment)

Sets, Relations, and Functions:

1. Explain with examples the basic terminology of functions, relations, and sets. [Familiarity]
2. Perform the operations associated with sets, functions, and relations. [Usage]
3. Relate practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context. [Assessment]
4. Describe how constructs of a chosen language (e.g., Java) are used to implement sets, functions, and relations. [Assessment]

Basic Logic:

1. Convert logical statements from informal language to propositional and predicate logic expressions. [Usage]
2. Apply formal methods of symbolic propositional and predicate logic, such as calculating validity of formulae and computing normal forms. [Usage]
3. Use the rules of inference to construct proofs in propositional and predicate logic. [Usage]
4. Describe how symbolic logic can be used to model real-life situations or applications, including those arising in computing contexts such as software analysis (e.g., program correctness), database queries, and algorithms. [Usage]
5. Apply formal logic proofs and/or informal, but rigorous, logical reasoning to real problems, such as predicting the behavior of software or solving problems such as puzzles. [Usage]
6. Describe the strengths and limitations of propositional and predicate logic. [Familiarity]
7. Convert logical statements from informal language to propositional and predicate logic expressions and then to selection statement syntax into a programming language of choice. [Implementation]

Mathematical Reasoning:

1. Identify the proof technique used in a given proof. [Familiarity]
2. Outline the basic structure of each proof technique (direct proof, proof by contradiction, and induction) described in this unit. [Usage]
3. Apply each of the proof techniques (direct proof, proof by contradiction, and induction) correctly in the construction of a sound argument. [Usage]
4. Determine which type of proof is best for a given problem. [Assessment]
5. Explain the parallels between ideas of mathematical and/or structural induction and recursively defined structures. [Assessment]
6. Explain the relationship between weak and strong induction and give examples of the appropriate use of each. [Assessment]
7. Identify and trace recursion within a programming language of choice (e.g., Java). [Usage]
8. Determine if iteration or recursion are best for a given problem.

Combinatorics:

1. Apply counting arguments, including sum and product rules, inclusion-exclusion principle and arithmetic/geometric progressions. [Usage]

2. Apply the pigeonhole principle in the context of a formal proof. [Usage]
3. Compute permutations and combinations of a set, and interpret the meaning in the context of the particular application. [Usage]
4. Map real-world applications to appropriate counting formalisms, such as determining the number of ways to arrange people around a table, subject to constraints on the seating arrangement, or the number of ways to determine certain hands in cards (e.g., a full house). [Usage]
5. Solve a variety of basic recurrence relations. [Usage]
6. Analyze a problem to determine underlying recurrence relations. [Usage]
7. Perform computations involving modular arithmetic. [Usage]
8. Describe how a programming language of choice (e.g., Java) could be used to create all ordered pairs from two sets. [Assessment]

Graphs and Trees:

1. Illustrate by example the basic terminology of graph theory, as well as some of the properties and special cases of each type of graph/tree. [Familiarity]
2. Describe how to traverse trees and graphs. [Usage]
3. Model a variety of real-world problems in computer science using appropriate forms of graphs and trees, such as representing a network topology or the organization of a hierarchical file system. [Usage]
4. Show how concepts from graphs and trees appear in data structures, algorithms, proof techniques (structural induction), and counting. [Usage]
5. Explain how to construct a spanning tree of a graph. [Usage]
6. Determine if two graphs are isomorphic. [Usage]

Course Outcomes Emphasized in Laboratory Projects / Assignments

Outcome	Number of Weeks
Sets and functions computational implementations: outcomes 1,7	3
Complex selection computational implementations: outcomes 2,7	2
Recursive computational implementations: outcomes 3,7	2
Permutation computational implementations: outcomes 4,7	2
Graph traversal program: outcomes 5,7	3
Boolean statements computational implementation: outcomes 6,7	1

Oral and Written Communication

No significant coverage

Written Reports		Oral Presentations	
Number Required	Approx. Number of pages	Number Required	Approx. Time for each
0	0	0	0

Social and Ethical Implications of Computing Topics

No significant coverage

Topic	Class time	student performance measures

Approximate number of credit hours devoted to fundamental CS topics

Fundamental CS Area	Core Hours	Advanced Hours
Algorithms:	0.5	
Software Design:		
Computer Organization and Architecture:		
Data Structures:	0.5	
Concepts of Programming Languages		

Theoretical Contents

Topic	Class time
Discrete structures	40 hours

Problem Analysis Experiences

Solution Design Experiences

The Coverage of Knowledge Units within Computer Science Body of Knowledge[1]

Knowledge Unit	Topic	Type	Lecture Hours
DS1. Functions, relations, and	1	Tier 1	10
DS2. Basic logic	2.1, 5	Tier 1	10
DS3. Proof techniques	2.2	Tier 1	5
DS4. Basics of counting	3	Tier 1	5
DS5. Graphs and trees	4	Tier 1	10

[1]See Appendix A in Computer Science Curricula 2013. Final Report of the IEEE and ACM Joint Task Force on Computing Curricula, available at:
https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf