Discrete Structures (DS)

Discrete structures are foundational material for computer science. By foundational we mean that relatively few computer scientists will be working primarily on discrete structures, but that many other areas of computer science require the ability to work with concepts from discrete structures. Discrete structures include important material from such areas as set theory, logic, graph theory, and probability theory.

The material in discrete structures is pervasive in the areas of data structures and algorithms but appears elsewhere in computer science as well. For example, an ability to create and understand a proof—either a formal symbolic proof or a less formal but still mathematically rigorous argument—is important in virtually every area of computer science, including (to name just a few) formal specification, verification, databases, and cryptography. Graph theory concepts are used in networks, operating systems, and compilers. Set theory concepts are used in software engineering and in databases. Probability theory is used in intelligent systems, networking, and a number of computing applications.

Given that discrete structures serves as a foundation for many other areas in computing, it is worth noting that the boundary between discrete structures and other areas, particularly Algorithms and Complexity, Software Development Fundamentals, Programming Languages, and Intelligent Systems, may not always be crisp. Indeed, different institutions may choose to organize the courses in which they cover this material in very different ways. Some institutions may cover these topics in one or two focused courses with titles like "discrete structures" or "discrete mathematics," whereas others may integrate these topics in courses on programming, algorithms, and/or artificial intelligence. Combinations of these approaches are also prevalent (e.g., covering many of these topics in a single focused introductory course and covering the remaining topics in more advanced topical courses).

DS. Discrete Structures (37 Core-Tier1 hours, 4 Core-Tier2 hours)

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
DS/Sets, Relations, and Functions	**4**		**N**
DS/Basic Logic	**9**		**N**
DS/Proof Techniques	**10**	**1**	**N**
DS/Basics of Counting	**5**		**N**
DS/Graphs and Trees	**3**	**1**	**N**
DS/Discrete Probability	**6**	**2**	**N**

DS/Sets, Relations, and Functions

[4 Core-Tier1 hours]

Topics:

- Sets
  - Venn diagrams
  - Union, intersection, complement
  - Cartesian product
  - Power sets
  - Cardinality of finite sets
- Relations
  - Reflexivity, symmetry, transitivity
  - Equivalence relations, partial orders
- Functions
  - Surjections, injections, bijections
  - Inverses
  - Composition

Learning Outcomes:

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]

DS/Basic Logic

[9 Core-Tier1 hours]

Topics:

- Propositional logic (cross-reference: Propositional logic is also reviewed in IS/Knowledge Based Reasoning)
- Logical connectives
- Truth tables
- Normal forms (conjunctive and disjunctive)
- Validity of well-formed formula
- Propositional inference rules (concepts of modus ponens and modus tollens)
- Predicate logic
    - Universal and existential quantification
- Limitations of propositional and predicate logic (e.g., expressiveness issues)

Learning Outcomes:

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]

DS/Proof Techniques

[10 Core-Tier1 hours, 1 Core-Tier2 hour]

Topics:

[Core-Tier1]

- Notions of implication, equivalence, converse, inverse, contrapositive, negation, and contradiction
- The structure of mathematical proofs
- Direct proofs
- Disproving by counterexample
- Proof by contradiction
- Induction over natural numbers
- Structural induction
- Weak and strong induction (i.e., First and Second Principle of Induction)
- Recursive mathematical definitions

[Core-Tier2]

- Well orderings

Learning Outcomes:

[Core-Tier1]

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 to recursion and recursively defined structures. [Assessment]
6. Explain the relationship between weak and strong induction and give examples of the appropriate use of each. [Assessment]

[Core-Tier2]

7. State the well-ordering principle and its relationship to mathematical induction. [Familiarity]

DS/Basics of Counting

[5 Core-Tier1 hours]

Topics:

- Counting arguments
  - Set cardinality and counting
  - Sum and product rule
  - Inclusion-exclusion principle
  - Arithmetic and geometric progressions
- The pigeonhole principle
- Permutations and combinations
  - Basic definitions
  - Pascal's identity
  - The binomial theorem
- Solving recurrence relations (cross-reference: AL/Basic Analysis)
  - An example of a simple recurrence relation, such as Fibonacci numbers
  - Other examples, showing a variety of solutions
- Basic modular arithmetic

Learning Outcomes:

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]

DS/Graphs and Trees

[3 Core-Tier1 hours, 1 Core-Tier2 hour]

Cross-reference: AL/Fundamental Data Structures and Algorithms, especially with relation to graph traversal strategies.

Topics:

[Core-Tier1]

- Trees
  - Properties
  - Traversal strategies
- Undirected graphs
- Directed graphs
- Weighted graphs

[Core-Tier2]

- Spanning trees/forests
- Graph isomorphism

Learning Outcomes:

[Core-Tier1]

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. Demonstrate different traversal methods for trees and graphs, including pre-, post-, and in-order traversal of trees. [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]

[Core-Tier2]

5. Explain how to construct a spanning tree of a graph. [Usage]
6. Determine if two graphs are isomorphic. [Usage]

DS/Discrete Probability

[6 Core-Tier1 hours, 2 Core-Tier2 hour]

Topics:

[Core-Tier1]

- Finite probability space, events
- Axioms of probability and probability measures
- Conditional probability, Bayes' theorem
- Independence
- Integer random variables (Bernoulli, binomial)
- Expectation, including Linearity of Expectation

[Core-Tier2]

- Variance
- Conditional Independence

Learning Outcomes:

[Core-Tier1]

1. Calculate probabilities of events and expectations of random variables for elementary problems such as games of chance. [Usage]
2. Differentiate between dependent and independent events. [Usage]
3. Identify a case of the binomial distribution and compute a probability using that distribution. [Usage]
4. Apply Bayes theorem to determine conditional probabilities in a problem. [Usage]
5. Apply the tools of probability to solve problems such as the average case analysis of algorithms or analyzing hashing. [Usage]

[Core-Tier2]

6. Compute the variance for a given probability distribution. [Usage]
7. Explain how events that are independent can be conditionally dependent (and vice-versa). Identify real-world examples of such cases. [Usage]