|  |  |
| --- | --- |
| **School of Computing and Information Sciences**  **Course Title:** Discrete Structures **Date:** 9/14/16 | |
| **Course Number:** COT-3100  **Number of Credits:** 3 | |
| **Subject Area:** Foundations | **Subject Area Coordinator:**  Geoffrey Smith  **email:** [sm](mailto:smithg@cis.fiu.edu)[ithg@cis.fiu.edu](mailto:ithg@cis.fiu.edu) |
| **Catalog Description:** Align mathematical and computational concepts by applying computing to propositional logic, sets, functions, relations, induction, recursion, combinatorics, Boolean algebra, graph and trees. | |
| **Textbook:**  Discrete Structures in Python  <http://www.cs.utah.edu/~ganesh/misc/2100NEWbook.pdf> | |
| **References:**  MIT OpenCourseWare Mathematics for Computer Science | |
| **Prerequisites Courses:** MAC1105 and (COP 2210 or COP2250) | |
| **Corequisite Courses:** | |

Type: Required for CS and IT Majors.

This course is acceptable as an alternative to MAD-2104 for CS majors

and MAD-1100 for IT majors, and satisfies the discrete requirement.

Prerequisites Topics:

1. Solve algebraic equations
2. Selection statements
3. Iteration
4. Encapsulation using functions
5. Writing programs that use selection, iteration and encapsulation

Course Outcomes:

1. Master definitions and theorems involving sets, relations and functions.
2. Write programs that use sets and functions.
3. Be familiar with propositional logic.
4. Write programs that use complex selection logic.
5. Be familiar with mathematical reasoning, including mathematical induction and recursion.
6. Write recursive programs that demonstrate induction.
7. Be exposed to combinatorics.
8. Write programs that can generate permutations.
9. Be familiar with graph theory.
10. Write programs that traverse graphs and trees.
11. Be exposed to Boolean Algebras.
12. Write programs that parse Boolean statements

**Relationship between Course Outcomes and Program Outcomes**

|  |  |
| --- | --- |
| **BS in CS: Program Outcomes** | **Course Outcomes** |
| a) Demonstrate proficiency in the foundation areas of Computer Science including mathematics, discrete structures, logic and the theory of algorithms | 1, 2, 3, 4, 5, 6 |
| b) Demonstrate proficiency in various areas of Computer Science including data structures and algorithms, concepts of programming languages and computer systems. |  |
| c) Demonstrate proficiency in problem solving and application of software engineering techniques |  |
| d) Demonstrate mastery of at least one modern programming language and proficiency in at least one other. |  |
| e) Demonstrate understanding of the social and ethical concerns of the practicing computer scientist. |  |
| f) Demonstrate the ability to work cooperatively in teams. |  |
| g) Demonstrate effective communication skills. |  |

**Assessment Plan for the Course & how Data in the Course are used to assess Program Outcomes**

Student and Instructor Course Outcome Surveys are administered at the conclusion of each offering, and are evaluated as described in the School’s Assessment Plan: <http://www.cis.fiu.edu/programs/undergrad/cs/assessment/>

**Outline**

|  |  |  |
| --- | --- | --- |
| **Topic** | **Number of Lecture Hours** | **Outcome** |
| 1. Sets, Relations, and Functions    1. Operations on sets    2. Equivalence relations    3. Cardinality | 10 | 1 |
| 1. Logic and Mathematical Reasoning    1. Propositional logic    2. Mathematical induction and recursion | 10 | 2, 3 |
| 1. Combinatorics    1. Combinatorial identities    2. Binomial theorem | 5 | 4 |
| 1. Directed and Undirected Graphs    1. Isomorphism of graphs    2. Paths    3. Adjacency matrices    4. Euler paths    5. Four-color problem    6. Planar graphs    7. Trees and tree traversal | 10 | 5 |
| 1. Boolean Algebras    1. Disjunctive normal form    2. Minimization of Boolean functions (Karnaugh maps) | 5 | 6 |

**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. Write a program that demonstrates union, intersection, and complement of lists. [Usage]
5. Write a program that computes the Cartesian product of two lists. [Usage]
6. Write a program that determines if a set of ordered pairs is a function. [Usage]

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.
8. Write a program that tests if two logical statements are equivalent.
9. Apply formal logic proofs and/or informal, but rigorous, logical reasoning to predict the behavior of software. [Usage]

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. Write a program that tests input for a condition, such as positive, negative, a set, all primes, all powers of two.
8. Write programs that implement recursive functions.
9. 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. Write a program that counts the number of unique elements in three sets.
9. Write a program that creates all ordered pairs from two sets.
10. Explain why only n-1 iterations are needed to sort n elements.
11. Write a program that uses 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).

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. 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]
5. Explain how to construct a spanning tree of a graph. [Usage]
6. Determine if two graphs are isomorphic. [Usage]
7. Write programs that traverse trees using including pre-, post-, and in-order traversal of trees. [Usage]
8. Implement Dijkstra’s algorithm for finding the shortest path in a graph.
9. Write programs that use graphs and trees in applications from data structures, algorithms, proof techniques (structural induction), and counting. [Usage]

# Course Outcomes Emphasized in Laboratory Projects / Assignments

|  |  |
| --- | --- |
| **Outcome** | **Number of Weeks** |
| Sets and functions program:  outcomes 1,2 | 3 |
| Complex selection program:  outcomes 3,4 | 2 |
| Recursive program:  outcomes 5,6 | 2 |
| Permutation program:  outcomes 7,8 | 2 |
| Graph traversal program:  outcomes 9,10 | 3 |
| Parse Boolean statements program:  outcomes 11,12 | 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**](#bookmark0)

|  |  |  |  |
| --- | --- | --- | --- |
| **Knowledge Unit** | **Topic** | **Type** | **Lecture Hours** |
| DS1. Functions, relations, and sets | 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 |

1See Appendix A in Computer Science Curricula 2013. Final Report of the IEEE and ACM Joint Task Force on Computing Curricula, available at: <http://www.acm.org/education/CS2013-final-report.pdf>