

Foundations Course Articulation Renewal Proposal
ICS-241: *Discrete Mathematics for Computer Science II*
As a Symbolic Reasoning (FS) Course

Leeward Community College
Spring 2008

Page	Contents
2	Course Description & Changes
3	Hallmark 1 Discussion
5	Hallmark 2 Discussion
7	Hallmark 3 Discussion
9	Hallmark 4 Discussion
11	Hallmark 5 Discussion
13	Hallmark 6 Discussion
14	Sample Syllabus

I. Course Description, Student Learning Outcomes.

▪ Course Number, Title & Catalog Description

ICS-241: Discrete Mathematics for Computer Science II (3 credits): Includes program correctness, recurrence relations and their solutions, divide and conquer relations, graph theory, trees and their applications, Boolean algebra, introduction to formal languages and automata theory. * (45 lecture hours)

▪ Student Learning Outcomes

Upon completion of ICS-241, the student should be able to

Analyze issues and apply more complex mathematical problem solving skills to plan courses of actions in high-level decision-making situations, using:

1. Graphs and trees.
2. Boolean algebra.
3. Finite-state machines.
4. Formal languages
5. Program correctness.
6. Solving recurrence relations

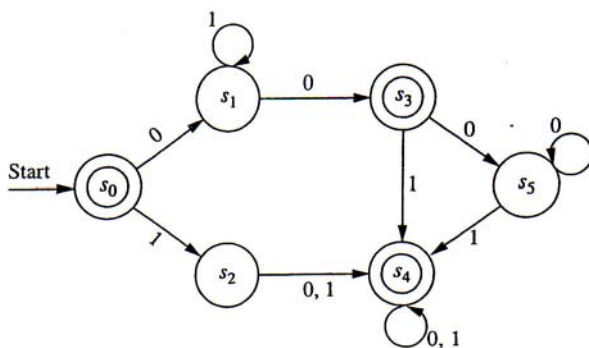
II. Changes.

No significant changes have been made in ICS-241 since the original request for foundations designation was approved. During a system-wide meeting held during the Fall 2005 semester resulting in an articulation agreement between all campuses offering this course, the Course Description and Student Learning Outcomes were reworded for clarity and consistency throughout the system. The content of the course was not altered by the rewording to either the course description or the student learning outcomes.

III. Assessing the Course: Below are samples of course materials that illustrate how the course meets the Symbolic Reasoning (FS) Foundations Hallmark.

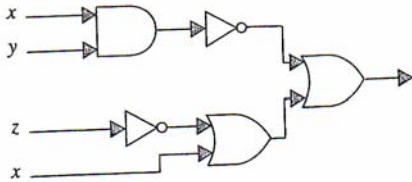
1. *Students will be exposed to the beauty, power, clarity and precision of formal systems. How will the course meet this hallmark?*

The breadth of material covered in ICS-241 includes some of the major topics of mathematics. They include recursive algorithms, program correctness, structured programs, graph theory, trees and their applications, relations and their properties, and Boolean algebra. The material covers modeling computation by covering formal language recognition and the relationship of Turing Machines to computer programming. Faculty in the ICS discipline will introduce applications of these topics to common problems encountered by computer science students. For example, the material on finite state automata is directly related to building of compilers for programming languages. The compilers must be taught to accept certain language constructs and reject others. In the section on Boolean Algebra, students are taught how to construct digital logic circuits which they will encounter in hardware design courses. The students study algorithms in the section on binary trees where they encounter searching strategies for ordered data. In the section on graph theory, students learn several algorithms used to find the shortest path between two points which they will encounter in network routing design. Methods of proving the correctness of algorithms and structured programming techniques will be useful to the student throughout most of their future coursework and careers. The above are just a few examples of how material taught in this course is applied by computer science students throughout their schoolwork and careers. These application examples are carefully selected to expose the beauty, power, clarity and precision of formal computational systems. In addition, the homework exercises are specifically selected to show students the power and precision of these formal systems.

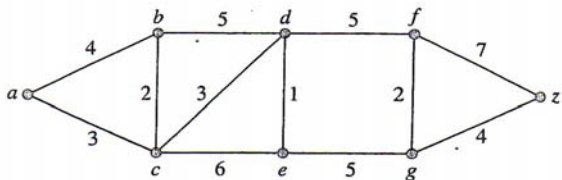


The above diagram represents an accepting state machine which is covered in the material on finite-state automata. The student is asked to be able to draw a finite state machine from a state table and create a state table from a drawing. The student is asked to describe the bit strings or language accepted by the machine. For the machine to accept a bit string, when tracing the bit string from the initial state, it must end at an accepting state (accepting states are represented by a double circle). It's easy to see that the above machine will accept any bit string of length two or greater that begin with a 1. It will also accept any bit string of length three or more which begins with 001. Of course, there are other descriptions necessary to fully describe the language accepted by

the above machine. The student is expected to be able to apply the logical inference rules learned in ICS-141, to convert a machine to a simpler machine that accepts the same language if a simpler machine can be found. It is easy for the student to understand that the study of automata is necessary to learn how to design vending machines. This study has applications to computer programming and the building of compilers, which must be taught to accept the language written by a computer programmer into bits which can be processed by a computer system.



The above diagram represents a simple circuit design that is covered in ICS-241. In the section on digital circuit design, the students are asked to be able to draw a circuit using various types of logic gates that produce certain output based on the inputs provided to the circuit. The students also have to trace through circuit designs to determine the outputs produced by the circuit based on the inputs provided. The students must apply the logical inference rules learned in ICS-141 to determine if a simpler circuit using fewer logic gates can be designed to produce the same output as a circuit with more logic gates. The students will also have to design logic circuits. The material covered in this section of the course has a direct impact on future required coursework in processor design.



The above diagram is a representation of a weighted graph. When studying graphs, students learn various terminology such as vertex and edge which are obvious. They also learn to calculate the degree of a vertex, which is the number of edges connected to it. In the above diagram, vertices c and d are both degree four while a and z are degree two. The student learns and applies other terms such as walk, path and circuit. There are special types of paths called Euler paths and Hamilton paths and the student is asked to identify if the graph exhibits those types of paths. With weighted graphs, the student identifies the shortest paths that exist in the graph. There are obvious applications to design of both transportation systems, networks and routing of data through communication systems associated with weighted graphs.

Students are expected to solve problems of the following type:

1. Draw an accepting state machine from the state table provided.
2. Identify the language accepted from the drawing of a machine provided.
3. Create a circuit using fewer logic gates which produces the same output as the circuit provided.
4. Identify the vertices, edges and degree of each vertex in the graph provided.
5. Find a Euler path in the graph provided, or explain why one does not exist.
6. Find a circuit in the graph provided from vertex c to vertex k, or explain why one does not exist.

2. Instructors will help students understand the concept of proof as a chain of inferences. How will instructors help students understand this concept?

There are two sections of the course where we explicitly focus on the notion of a proof. While covering the concept of program correctness we use the notion of rules of inference, conditional statements, and loop invariants to establish the correctness of simple programs. While covering formal languages and automata theory we explicitly prove many of the properties of these systems from first principles.

In the section on program correctness, students are asked to prove that program segments are valid. An example of this type of problem and its solution follow:

Verify that the following program segment is correct with respect to the initial assertion T and the final assertion $y \geq x$.

if ($x > y$) then

$y := x$;

Solution: When the initial assertion is true and $x > y$ is true, the assignment $y := x$ is carried out. Hence, the final assertion, which asserts $y \geq x$, is true in this case. Moreover, when the initial assertion is true and $x > y$ is false, so that $y \geq x$, the final assertion is again true. Hence, using the rule of inference for program segments of this type, this program is correct with respect to the given initial and final assertions.

In the section on program correctness, students are also asked to use loop invariants to prove that a program segment is correct. In the following example and its solution, students verify that the program segment terminates with $\text{factorial} = n!$ when n is a positive integer.

$i := 1$;

$\text{factorial} := 1$;

while ($i < n$)

begin

$i := i + 1$;

$\text{factorial} := \text{factorial} * i$;

end

Solution: Let p be the assertion “factorial = $i!$ and $i \leq n$.” We first prove that p is a loop invariant. Suppose that, at the beginning of one execution of the while loop, p is true and the condition of the while loop holds; in other words, assume that factorial = $i!$ and that $i < n$. The new values i_{new} and factorial_{new} of i and factorial are $i_{\text{new}} = i + 1$ and factorial_{new} = factorial * $(i+1) = (i+1)! = i_{\text{new}}!$. Because $i < n$, we also have $i_{\text{new}} = i + 1 \leq n$. Thus, p is true at the end of the execution of the loop. This shows that p is a loop invariant.

Next, consider the program segment. Just before entering the loop, $i = 1 \leq n$ and factorial = $1 = 1! = i!$ both hold, so p is true. Because p is a loop invariant, the rule of inference just introduced implied that if the while loop terminates, it terminates with p true and with $i < n$ false. In this case, at the end, factorial = $i!$ and $i \leq n$ are true, but $i < n$ is false; in other words, $i = n$ and factorial = $i! = n!$, as desired.

Finally, we need to check that the while loop actually terminates. At the beginning of the program i is assigned the value, 1, so after $n-1$ traversals of the loop, the new value of i will be n , and the loop terminates at that point.

Students are expected to solve problems of the following type:

1. Verify that the following program segment is correct with respect to the initial assertion that $y = 3$ and the final assertion that $z = 6$.

```
x := 2;
z := x + y;
if (y > 0) then
  z := z + 1;
else
  z := 0;
```

2. Use a loop invariant to prove that the following program segment for computing the n th power, where n is a positive integer, of a real number x , is correct.

```
power := 1;
i := 1;
while (i ≤ n)
begin
  power := power * x;
  i := i + 1;
end
```

*3. Instructors will teach students how to apply formal rules or algorithms.
How will instructors meet this hallmark?*

The notion of formal rules and algorithms is fundamental to all topics in this course. Certainly we emphasize the notion of algorithms while discussing recursive algorithms. Obviously we use formal rules to discuss the notion of formal languages as first presented by the linguist Noam Chomsky. We cover other concepts of algorithms when discussing relations, graph theory, trees and Boolean algebra.

When recursive algorithms are introduced, students are shown the difference between a recursive algorithm and an iterative algorithm. A recursive algorithm is defined by one or more base cases which stop the recursion and a recursive case which defines the next element of a series in terms of one or more previous elements in the series. An iterative solution uses one or more loops. One of the classical examples is $n!$, where n is a positive integer.

The iterative solution is

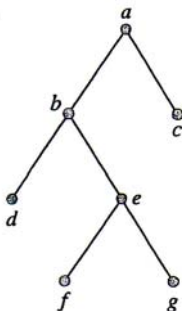
```
function factorial (int n)
val = 1;
while (n > 1)
begin
    val = val * n;
    n = n - 1;
end
```

The recursive solution

```
function int factorial (int n)
if (n == 1)
    return 1
else
    return (n * factorial(n-1))
```

In divide and conquer relations, which break a problem down in half, then half those halves, etc. until a solution can be found, students discover that many divide and conquer problems can be implemented as recursive algorithms which run more efficiently than their iterative counterparts. For example, we examine both the linear search (iterative) and binary search (recursive) algorithms and determine the binary search algorithm is more efficient. The students examine the insertion sort (iterative) and the mergesort and quicksort (both recursive) and find that the recursive algorithm is more efficient.

Other algorithms used in computer science are more easily understood if we express them recursively. For example, when studying the material on trees, we might encounter a problem depicted as the one below:



- a. List the order in which nodes are visited when doing a preorder traversal. A preorder traversal (visit root, traverse left subtree, traverse right subtree).
a, b, d, e, f, g, c.
- b. List the order in which nodes are visited when doing an inorder traversal. A preorder traversal (traverse left subtree, visit root, traverse right subtree).
d, b, f, e, g, a, c
- c. List the order in which nodes are visited when doing an postorder traversal. A postorder traversal (traverse left subtree, traverse right subtree, visit root).
d, f, g, e, b, c, a

The recursively defined algorithms for binary tree traversals are traced by the students and implemented on the computer to verify that they are correct. An inorder traversal of an ordered binary tree will display information which is sorted in ascending order.

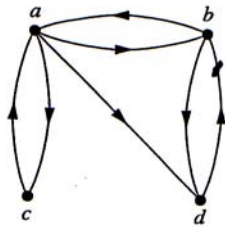
Students are expected to solve problems of the following type.

1. Create an iterative algorithm to calculate the sum of the first n positive integers, when n is a positive integer.
2. Create a recursive algorithm to calculate the sum of the first n positive integers, when n is a positive integer.
3. Trace through the binary search algorithm for data contained in a sorted array.
4. Display the order in which nodes of a tree are visited in a preorder, inorder, and postorder traversal of a given pictorial representation of a tree.

4. Students will be required to use appropriate symbolic techniques in the context of problem solving, and in the presentation and critical evaluation of evidence. What symbolic techniques will be required and in what contexts? How will presentations and evaluations of evidence be incorporated into the course?

Symbolic techniques are fundamental to all topics in this course. Students are required to use symbolic representations for the concepts of programs, graphs, trees, languages, and automata. They will use these representations in homework and exam exercises. In graph theory, symbolic techniques are used to represent graphs and the results of basic graph oriented operations on them. In Boolean logic students are asked to design logic circuits that perform specific logical and computational functions such as addition. In the Modeling of Computation students are introduced to the concept of finite state automata and are asked to design automata that give change in a vending machine as well as to recognize formal languages.

For example, a student might see the representation of a directed graph, such as the following:



When covering the material on graphs, the student is asked to:

- a. Identify the vertices of the graph a, b, c, d
- b. Identify the edges of the graph (a,b), (b,a), (a, c), (c,a), (a,d), (b,d), (d,b)
- c. Express the graph as an adjacency list

initial vertex	terminal vertex
a	b,c,d
b	a,d
c	a
d	b

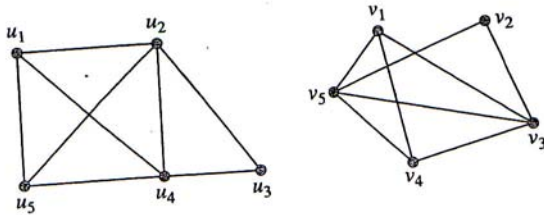
- d. Express the graph as an adjacency matrix

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

When covering the material on relations, the student is expected to use a model of a relation depicted as a directed graph and be able to answer questions such as the following (based on the same graph used for the previous questions):

- Is the relation reflexive? A relation that is reflexive has an ordered pair (a,a) for every vertex of the directed graph. In other words, a loop is depicted at each vertex if it represents a reflexive relation. No, this is not reflexive.
- What is the reflexive closure of the relation? The reflexive closure of the relation adds the ordered pairs to the relation which are necessary to make the relation reflexive. In this case, a loop is needed at each vertex, so the reflexive closure adds the ordered pairs: (a,a) , (b,b) (c,c) (d,d)
- Is the relation symmetric? A relation that is symmetric has an ordered pair (b,a) for every ordered pair (a,b) which is in the relation. This relation is not symmetric because it contains an ordered pair (a,d) but does not contain the required ordered pair (d,a) .
- What is the symmetric closure of the relation? The symmetric closure adds the ordered pairs required to make the relation symmetric. In this case, the ordered pair (d,a) needs to be added to the relation.

Similarly, the student is asked if the relation depicted by the directed graph is transitive and to express the transitive closure of the relation. A relation which is transitive has an ordered pair (a,c) whenever the relation also has ordered pairs (a,b) and (b,c) . The student is asked to determine if the relation is an equivalence relation. An equivalence relation is a relation that exhibits the properties of being reflexive, symmetric and transitive.



In the study of graphs, the students are asked to look at a drawing of a graph such as the one on the right above and determine if it is a bipartite graph or to look at two graph drawings such as those above and determine if the graphs are isomorphisms of each other.

Many other examples of use of symbolic techniques are depicted in answers to hallmark questions one and three earlier in this document. These techniques are used extensively in this course throughout the presentation of the following major topics, relations, graphs, trees, automata, and digital logic circuits.

5. *The course will not focus solely on computational skills. What reasoning skills will be taught in the course?*

In the area of program correctness we focus on the notion of logically proving that a program meets its stated objectives. Students are exposed to the notions of indirect proof, proof by contradiction, proof by cases, counterexample, mathematical induction, iteration, etc. The use of word problems also helps the student develop reasoning skills.

In the section on recurrence relations, the students encounter several word problems that develop reasoning skills. An example of such a problem with its solution, follows:

Suppose that a person deposits \$10,000 in a savings account at a bank yielding 4% per year with interest compounded annually. a.) create a recurrence relation which can be used to find the values of the sequence. b.) Use the recurrence relation to determine the first three values of the sequence. c.) Create an explicit formula to find any one value of the sequence. d.) Use your explicit formula to find the value in the account after 30 years.

Solution:

a.) $P_0 = 10,000; P_n = (1.04)P_{n-1}$

b.) 10,000

$$(1.04)(10,000) = 10,400$$

$$(1.04)(10,400) = 10,816$$

c.) $P_n = (1.04)^n P_0$

d.) $P_{30} = (1.04)^{30}(10,000) = 32,433.98$

When studying advanced counting problems, such as those that apply the inclusion-exclusion principle, students develop reasoning skills. The inclusion-exclusion principle counts objects based on their membership in overlapping sets. For example, to calculate the number of objects in the union of two overlapping sets A and B, you apply the principle as follows:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

The number of elements in A union B is equal to the number of elements in A plus the number of elements in B minus the number of elements in A intersect B. You do the final subtraction because you have counted these elements twice, once when counting the number of elements in A and another time when counting the number of elements in B.

For three overlapping sets, the calculation is expressed as:

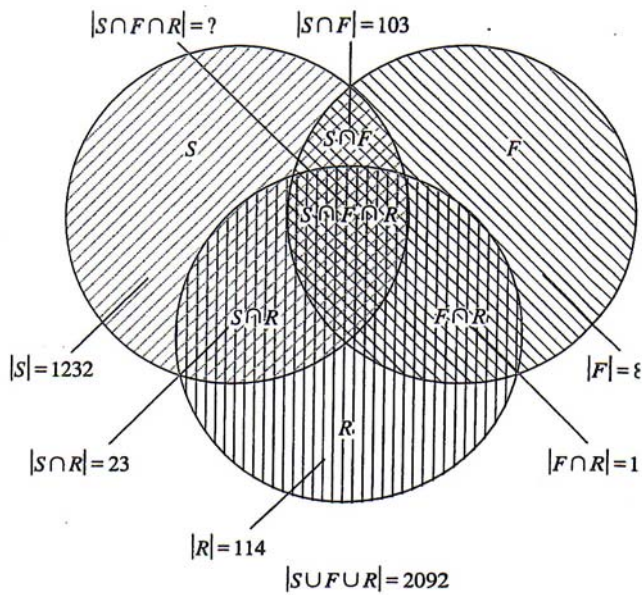
$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

The students solve problems such as the following when studying the inclusion exclusion principle:

A total of 1232 students have taken a course in Spanish, 879 have taken a course in French, and 114 have taken a course in Russian. Further, 103 have taken courses in both Spanish and French, 23 have taken courses in both Spanish and Russian, and 14 have taken courses in both French and Russian. If 2092 students have taken at least one of Spanish, French and Russian, how many students have taken a course in all three languages?

Solution. $2092 = 1232 + 879 + 114 - 103 - 23 - 14 + \text{unknown}$. Using simple arithmetic and algebraic manipulation, you can solve for the unknown value, 7.

Students are also asked to prepare a Venn diagram to display the solution symbolically. The use of Venn diagrams helps the student to understand the inclusion – exclusion principle.



Examples of problems that students are expected to solve are:

1. Assume the population of the world was 6.2 billion people in 2002 and that the population is growing at a compounded rate of 1.3% per year. a.) Find a recurrence relation for the human population of the world n years after 2002. b.) Find an explicit formula for the human population of the world n years after 2002.
2. In a survey of 270 college students, it is found that 64 like brussels sprouts, 94 like broccoli, 58 like cauliflower, 26 like both brussels sprouts and broccoli, 28 like both brussels sprouts and cauliflower, 22 like both broccoli and cauliflower, and 14 like all three vegetables. How many of the 270 students surveyed do not like any of these vegetables?

6. Instructors will build a bridge from theory to practice and show students how to traverse this bridge. How will instructors help students make connections between theory and practice?

There are a number of illustrations in the answers to the first five questions that can be applied to answer this hallmark question. There are several applications that can be made to computer science practices in this course. Students will see how algorithms can be implemented in programming languages and will compare the efficiency of those algorithms. Students will see the direct application to digital logic circuit design and the design of processors that they will encounter in future coursework. Machine and compiler design are applications made when studying the material on finite-state automata and formal language acceptance sections of the course. Algorithms from the sections on graphs and trees are implemented and run on a computer. Practical applications of the use of binary search trees quickly determine if an element exists in an ordered collection.

Additionally, there are direct applications to database theory and implementation taken from the study of relations and their operations. A join operation on a relation is actually what occurs whenever a relational database management system, such as MS-Access relates the data that exists in two separate tables of the database. In this section of the course, students apply the concepts and operations of relations using SQL statements in a relational database such as those created by MS-Access.

LEEWARD COMMUNITY COLLEGE
Mathematics and Natural Sciences Division
Course Syllabus
ICS 241 -- DISCRETE MATH FOR COMP SCI II (3.0 credits)

Instructor:

Office Hours:

Office Location:

Contact Information:

Catalog Course Description:

Includes program correctness, recurrence relations and their solutions, divide and conquer relations, graph theory, trees and their applications, Boolean algebra, introduction to formal languages and automata theory. * (45 lecture hours)

Co-requisites:

None

Prerequisites:

ICS-111 and ICS-141, or consent of instructor.

Recommended Preparations:

None

Required Textbook:

"Discrete Mathematics and Its Applications" 6th Edition by Kenneth H. Rosen.

Required Supply:

1 Scientific Calculator

Student Learning Outcomes:

Upon completion of ICS-241, the student should be able to

Analyze issues and apply mathematical problem solving skills to plan courses of actions in decision-making situations, using:

- 1. Graphs and trees.
- 2. Boolean algebra.
- 3. Finite-state machines.
- 4. Formal languages
- 5. Program correctness.
- 6. Solving recurrence relations

Grading Policy:

- ★ Quizzes: A total of five quizzes worth 25 points each will be given in the course. Your best four quiz scores contribute 100 points toward your final grade.
- ★ Exams: There are two exams worth 50 points each. The first exam covers the first half of the course and the second exam covers the second half of the course. Exams are worth 100 points.

Final Grades will be based upon:

A	100-90%	200 - 180 points
B	89.9-80%	179 - 160 points
C	79.9-70%	159 - 140 points
D	69.9-60%	139 - 120 points
F	59.9 - 0%	119 - 0 points

Grading Policies:

- 1. No make-up exams will be given unless your absence is due to a medical emergency or is work related. Written verification from your doctor or employer is required before permission for a make-up exam is granted. No exceptions!!!
- 2. Quizzes must be completed by the date due.

Student Contributions:

Students are expected to spend at least one to two hours per class period in addition to class time on this course. Regular attendance and active participation in class activities and discussions are vital for success in this course.

Student with Disabilities Statement:

Students with documented disabilities who believe that they may need accommodations in this class are encouraged to contact the Coordinator of the KAKO'O 'IKE (KI) program as soon as possible to ensure that such accommodations are implemented in a timely fashion. The KI office is located in L-208, across from the elevator in the library building or call for information at 455-0421.

COURSE SYLLABUS

(subject to revision)

Topic/Homework

Sec. 4.5: Program Correctness
Sec. 7.1: Recurrence Relations
Sec. 7.2: Solving Linear Recurrence Relations
Sec. 7.3: Divide & Conquer Algorithms & Recurrence Relations
Sec. 7.5: Inclusion-Exclusion Principle
Sec. 7.6: Applications of the Inclusion-Exclusion Principle
Quiz One

Sec. 8.1 : Relations and Their Properties
Sec. 8.2: n-ary Relations and Their Applications
Sec. 8.3: Representing Relations
Sec. 8.4: Closures of Relations
Sec. 8.5: Equivalence Relations
Quiz Two

Sec. 9.1: Graphs and Graph Models
Sec. 9.2: Graph Terminology and Special Types of Graphs
Sec. 9.3: Representing Graphs and Graph Isomorphisms
Sec. 9.4: Graph Connectivity
Sec. 9.5: Euler and Hamilton Paths
Sec. 9.6: Shortest-Path Problems
Quiz Three

Exam One

Sec. 10.1: Introduction to Trees
Sec. 10.2: Applications of Trees
Sec. 10.3: Tree Traversals
Sec. 10.4 Spanning Trees
Sec. 10.5: Minimum Spanning Trees
Quiz Four

Sec. 11.1: Boolean Functions
Sec. 11.3: Logic-Gates

Sec. 11.4: Minimization of Circuits
Sec. 12.1: Languages and Grammars
Sec. 12.2: Finite-State Machines with Output
Sec. 12.3: Finite-State Machines with No Output
Sec. 12.4: Language Recognition
Sec. 12.5: Turing Machines
Quiz Five

Exam Two