Gradiance

	Homework
1.	Chapter 01: Introduction --- Discrete Math
2.	Chapter 02: Deterministic Finite Automata
3.	Chapter 02: Epsilon-NFA's
4.	Chapter 02: Nondeterministic Finite Automata
5.	Chapter 03, 04: Regular Expressions --- Basics
6.	Chapter 03: Regular Expressions --- Algebra and FA Equivalence
7.	Chapter 04: Minimization of DFA's
8.	Chapter 04: Regular Languages
9.	Chapter 05: CFG's
10.	Chapter 05: CFG's --- Additional Questions
11.	Chapter 05: CFG's --- Proofs
12.	Chapter 05: Parse Trees
13.	Chapter 05: Parse Trees --- Additional Questions
14.	Chapter 06: Pushdown Automata
15.	Chapter 07: CFG's --- Normal Forms
16.	Chapter 07: Properties of CFL's
17.	Chapter 07: Properties of CFL's --- Additional Questions
18.	Chapter 08: Turing Machines
19.	Chapter 09: Undecidabilty
20.	Chapter 10, 11: Intractability
21.	Chapter 10: Intractability --- SAT and Related Problems
22.	Chapter 10: Intractability --- Some NP Complete Problems

Gradiance Homework References

HW#	Homework (HMU, 3e)	ID	Gradiance Questions ID	Linz, 5e	Class #
#1	Chapter 01: Introduction --- Discrete Math		Based on Ch. 1 of HMU.		Class 1
		108	Remember: the contrapositive (see Section 1.3.2, p. 14) of an if-then statement S is another statement whose hypothesis is the negation of the conclusion of S and whose conclusion is the negation of the hypothesis of S.		
		109	To prove such a statement by contradiction (see Section 1.3.3, p. 16), we must prove the negation implies false.		
		110	The outline of a simple induction on integers is in Section 1.4.1 (p. 19).		
		111	See Section 1.5.2 (p. 29).		
		112	Concatenation is defined in Section 1.5, on p. 30.		
		113	Problems are discussed in Section 1.5.4 (p. 31).		
#2	Chapter 02: Deterministic Finite Automata		Based on Section 2.2 of HMU.		Class 2
		74	The informal description of how a DFA makes transitions is in Section 2.2.2 (p. 46), and the more formal notion is in Section 2.2.4 (p. 49).		
		75	The informal description of how a DFA makes transitions is in Section 2.2.2 (p. 46). Transition diagrams are explained in Section 2.2.3 (p. 47).		
		76	Example 1.23 (p. 26) is a useful model of the proof. Also helpful may be the definition of transition tables (within Section 2.2.3, on p. 48) and the formal delta notation in Section 2.2.4 (also beginning on p. 49).		
		77	This question is really about state-elimination, and reading Section 3.2.2 (p. 98) may offer a useful hint.		
		122	The information you need to understand the behavior of this DFA is in Section 2.2 (p. 45).		
#3	Chapter 02: Epsilon-NFA's		Based on Section 2.5 of HMU		Class 3
		80	Q: Suppose we use the extended subset construction from Section 2.5.5 (p. 77) You should check the details of the extended subset construction in Section 2.5.5 (p. 77). In particular, you will need to understand how the epsilon-closure of a set of states is taken as in Section 2.5.3 (p. 74).		
		81	Q: Suppose we construct an equivalent DFA by the construction of Section 2.5.5 (p. 77). You should check the details of the extended subset construction in Section 2.5.5 (p. 77). In particular, you will need to understand how the epsilon-closure of a set of states is taken as in Section 2.5.3 (p. 74).		
		82	It may be useful to review the informal meaning of transitions in an NFA (Section 2.3.1, p. 55) and an epsilon-NFA (Section 2.5.1, p. 72). Also, the formal notions of transitions of an NFA (Section 2.3.3, p. 58) and the language of an NFA (Section 2.3.4, p. 59) are useful, as		

			are the same topics for an epsilon-NFA (Section 2.5.4, p. 75).		
#4	Chapter 02: Nondeterministic Finite Automata		Based on Section 2.3 of HMU		Class 3
		78	This question asks you to apply the subset construction from Section 2.3.5 (p. 60). However, you need to do so in the "lazy" way described in Example 2.10 (p. 61).		
		79	The informal way that NFA's process their inputs can be seen in Section 2.3.1 (p. 55). The formal treatment of input processing by an NFA is in Section 2.3.3 (p. 58).		
		135	Q: Convert this NFA to a DFA, using the "lazy" version of the subset construction described in Section 2.3.5 (p. 60), so only the accessible states are constructed. See Section 2.3.5 (p. 60), especially Example 2.10 (p. 61)		
		136	The informal simulation of an NFA in Section 2.3.1 (p. 55) could be useful.		
#5	Chapter 03, 04: Regular Expressions --- Basics		Based on Sections 3.1 and 4.1 of HMU.		Class 4
		83	This exercise is intended to explore the techniques for converting from automata to regular expressions that are contained in Section 3.2.1 (p. 93).		
		85	You may wish to review the basic operators of regular expressions and how they fit together, in Section 3.1 (p. 85) as well as the extended "UNIX" operators from Section 3.3.1 (p. 109).		
		89	See Section 3.1 (p. 85) for the basic iterpretation of the regular-expression operators.		
		90	(Theorem 4.5, p. 135) Apply the pumping lemma (Section 4.1, p. 128)		Class 6
		114	Check Section 3.4.1 (p. 115). Check Section 3.4.2 (p. 116). Check Section 3.4.4 (p. 117). See Example 3.10 (p. 116).		
		119	The Kleene closure operator is defined in Section 3.1.1 (p. 86).		
		120	Section 3.1.2 (p. 87) shows how to determine the language of a regular expression.		
#6	Chapter 03: Regular Expressions --- Algebra and FA Equivalence		Based on Sections 3.2 and 3.4 of HMU.		Class 4
		84	This exercise is intended to explore the techniques for converting from automata to regular expressions that are contained in Section 3.2.1 (p. 93).		
		86	Q: Apply the construction in Figure 3.16 (p. 104) and Figure 3.17 (p. 105) The entire process is outlined in Section 3.2.3 (p. 102).		
		87	The general subject of algebraic laws for regular expressions is in Section 3.4 (p. 115). You may want to look especially at Section 3.4.7 (p. 120), where it is shown how to test a		

			possible algebraic law by converting the variables in the equation into concrete symbols and treating the "law" as an equality of two particular languages that must be tested for its truth.		
		114	Check Section 3.4.1 (p. 115). Check Section 3.4.2 (p. 116). Check Section 3.4.4 (p. 117). See Example 3.10 (p. 116).		
		123	The techniques useful for this problem are found in Section 3.2.1 (p. 93).		
#7	Chapter 04: Minimization of DFA's		Based on Section 4.4 of HMU	Section 2.4	Class 4.2
		126	Section 1.1.1 (p. 2) gives several examples of design of automata in which states are used to remember the important aspects of the input history. The formal definition of a DFA is in Section 2.2 (p. 45), and minimizing the number of states is covered in Section 4.4.3 (p. 160).		
		127	Section 1.1.1 (p. 2) gives several examples of design of automata in which states are used to remember the important aspects of the input history. The formal definition of a DFA is in Section 2.2 (p. 45), and minimizing the number of states is covered in Section 4.4.3 (p. 160).		
		128	See Section 4.4.3 (p. 160) for the state-minimization algorithm.		
		129	See Section 4.4.3 (p. 160) for the state-minimization algorithm.		
#8	Chapter 04: Regular Languages		Based on Section 4.2 of HMU.		
		64	Homomorphisms are defined in Section 4.2.3 (p. 140).		
		65	Homomorphisms are defined in Section 4.2.3 (p. 140) and inverse homomorphisms in Section 4.2.4 (p. 142).		
		67	definitions of homomorphisms (Section 4.2.3, p. 140) and their inverses (Section 4.2.4, p. 142). It may also be useful to try to develop an induction to define *N*. The ideas behind inductive proofs are described in Section 1.4 (p. 19).		
		88	You need to interpret each of the four regular expressions, using the definitions from Section 3.1.2 (p. 87). The pumping lemma for regular languages in Section 4.1 (p. 128) may be useful in showing certain languages not to be regular.		
		91	Can you use the pumping lemma (Section 4.1, p. 128) to prove it is not regular? Can you use the pumping lemma for CFL's (Section 7.2, p. 279) to prove it is not context-free? Can you devise a pushdown automaton (Section 6.1, p. 225) to recognize this language? Can you devise a finite automaton (Section 2.1, p. 38) to do so? Can you devise a finite automaton (Section 2.1, p. 38) to recognize this language? Can you devise a pushdown automaton (Section 6.1, p. 225) to recognize this language? A finite automaton (Section 2.1, p. 38)?	Depends on the choice:	

			Can you use the pumping lemma (Section 7.2, p. 279) to show it is not a CFL? Alternatively, can you devise a finite automaton (Section 2.1, p. 38) to recognize this language? Can you use one of the pumping lemmas to show it is not regular (Section 4.1, p. 128), or not a CFL (Section 7.2, p. 279)? Can you use the pumping lemma (Section 4.1, p. 128) to prove it is not regular? Can you use the pumping lemma for CFL's (Section 7.2, p. 279) to prove it is not context-free? Can you devise a pushdown automaton (Section 6.1, p. 225) to recognize this language? Can you devise a finite automaton (Section 2.1, p. 38) to do so?		
		121	Section 3.1.2 (p. 87) is the relevant reading.		
		124	The algorithm is described in Section 4.2.2 (p. 139).		
		125	See Section 4.2.3 (p. 140).		
#9	Chapter 05: CFG's		Based on Sections 5.1 and 5.4 of HMU. Note: there are many other questions on these topics; this homework is a recommended set.	Chapter 5	Class 8
		11	See Section 5.1.3 (p. 175) for a discussion of how strings are generated by a grammar and Section 5.1.5 (p. 179) for the definition of the language defined by a grammar. Also, since there are useless productions in this grammar, Section 7.1.1 (p. 262) on eliminating useless symbols, may be relevant. 5.1.3 Derivations Using Grammar 5.1.5 The Language of a Grammar		
		16	See Sections 5.1.3 (p. 175) and 5.1.5 (p. 179) for definitions of derivations and languages. 5.1.3 Derivations Using Grammar		
		70	A possible aid is to examine the rules for derivations in Section 5.1.3 (p. 175). 5.1.3 Derivations Using Grammar		
		115	Ambiguous grammars are discussed in Section 5.4 (p. 205). In particular, the relationship between ambiguous grammars and leftmost derivations is covered in Section 5.4.3 (p. 211). 5.4 Ambiguity in Grammars and languages 5.4.3 Leftmost Derivations as a Way to Express Ambiguity		
#10	Chapter 05: CFG's --- Additional Questions		These questions, based on Section 5.1 of HMU, are not in either of the other two homeworks on CFG's, but are available for use.	Chapter 5	Class 8
		3	Derivations and the terminal strings they derive are introduced in Section 5.1.3 (p. 175).		

			5.1.3 Derivations Using Grammar		
		4	The basic definitions you need for this problem are in Section 5.1.5 (p. 179), but you also need to think a bit about what these (relatively simple) grammars generate. 5.1.5 The Language of a Grammar		
		5	This is the set of words in L that begin with aa. Derivations and the terminal strings they derive are introduced in Section 5.1.3 (p. 175). 5.1.3 Derivations Using Grammar		
		6	Derivations in a grammar are introduced in Section 5.1.3 (p. 175). See the discussion of inductive proofs in Section 1.4 (p. 19). See the discussion of ambiguous grammars in Section 5.4 (p. 207). 5.1.3 Derivations Using Grammar 1.4 Inductive Proofs		
		7	A useful example is the "if" part of the proof of Theorem 5.7 in Section 5.1.5 (p. 179).		
		8	See Section 5.1.3 (p. 175) for a discussion of how strings are generated by a grammar.		
		10	See Section 5.1.3 (p. 175) for a discussion of how strings are generated by a grammar and Section 5.1.5 (p. 179) for the definition of the language defined by a grammar.		
		12	See Section 5.1.3 (p. 175) for a discussion of how strings are generated by a grammar and Section 5.1.5 (p. 179) for the definition of the language defined by a grammar.		
		14	See Section 5.1.3 (p. 175) for a discussion of how strings are generated by a grammar and Section 5.1.5 (p. 179) for the definition of the language defined by a grammar.		
		18	See Sections 5.1.3 (p. 175) and 5.1.5 (p. 179) for definitions of derivations and languages.		
		19	See Sections 5.1.3 (p. 175) and 5.1.5 (p. 179) for definitions of derivations and languages.		
		20	The "Only If" part of Theorem 5.7 (p. 180) is a useful example, as is the proof of Theorem 5.18 (p. 193). Also, see Section 1.4.2 (p. 22) on the general form of inductions on integers (which includes an induction on the lengths of derivations).		
		21	The "Only If" part of Theorem 5.7 (p. 180) is a useful example, as is the proof of Theorem 5.18 (p. 193). Also, see Section 1.4.2 (p. 22) on the general form of inductions on integers (which includes an induction on the lengths of derivations).		
		71	A possible aid is to examine the rules for derivations in Section 5.1.3 (p. 175).		
#11	Chapter 05: CFG's --- Proofs		These are some questions in which students are asked to understand proofs about context-free grammars. The material is based on Chapter 5 of HMU.	Chapter 5	Class 8
		2	A good model of this proof is Section 5.2.6 (p. 191), where there is an induction on the length of a derivation. The proof in Section 5.1.5 (p. 179) may also be instructive. 5.2.6 From Derivations to Recursive Inferences 5.1.5 The Language of a Grammar		
		9	See Section 5.1.3 (p. 175) for a discussion of how strings are generated by a grammar.		

			5.1.3 Derivations Using a Grammars		
		17	The "If" part of Theorem 5.7 (p. 179) is a useful example. Also, see Section 1.4.2 (p. 22) on the general form of inductions on integers (which includes an induction on the lengths of strings). 1.4.2 More General Forms of Integral Inductions		
		22	The "Only If" part of Theorem 5.7 (p. 180) is a useful example, as is the proof of Theorem 5.18 (p. 193). Also, see Section 1.4.2 (p. 22) on the general form of inductions on integers (which includes an induction on the lengths of derivations). 1.4.2 More General Forms of Integral Inductions		
		23	The "If" part of Theorem 5.7 (p. 179) is a useful example. Also, see Section 1.4.2 (p. 22) on the general form of inductions on integers (which includes an induction on the lengths of strings). 1.4.2 More General Forms of Integral Inductions		
#12	Chapter 05: Parse Trees		Selected questions on parse trees. Based on Section 5.2 of HMU.	Chapter 5	Class 8
		48	See Section 5.2.1 (p. 183) for a discussion of the restrictions on what parse trees for a given grammar may look like. 5.2.1 Constructing Parse Trees		
		49	Section 5.2.5 (p. 188) talks about constructing leftmost derivations from parse trees. You should think about how to change that method to produce rightmost derivations instead. Also, see Section 5.1.4 (p. 177) for the definition of rightmost derivations. 5.2.5 From Trees to Derivations 5.1.4 Leftmost and Rightmost Derivations		
		51	See Section 5.2.1 (p. 183) for a discussion of the restrictions on what parse trees for a given grammar may look like. 5.2.1 Constructing Parse Trees		
		52	See Section 5.2.1 (p. 183) for a discussion of the restrictions on what parse trees for a given grammar may look like. 5.2.1 Constructing Parse Trees		
		56	See Section 5.2.2 (p. 185) on yields of parse trees. 5.2.2 The Yield of a Parse Tree		
#13	Chapter 05: Parse Trees --- Additional Questions		These are questions based on Section 5.2 of HMU that were not selected for the main homework on the topic.	Chapter 5	Class 8
		50	Section 5.2.5 (p. 188) talks about constructing leftmost derivations from parse trees. Also,		

			see Section 5.1.4 (p. 177) for the definition of leftmost derivations.		
		53	See Section 5.2.1 (p. 183) for a discussion of the restrictions on what parse trees for a given grammar may look like.		
		54	See Section 5.2.2 (p. 185) on yields of parse trees.		
		55	See Section 5.2.2 (p. 185) on yields of parse trees.		
#14	Chapter 06: Pushdown Automata		Based on Chapter 6 of HMU.		Class 10
		59	Pushdown automata are the subject of Section 6.1 (p. 225). See especially the informal description of how these automata move in Section 6.1.1 (p. 225) and the formal definition of their behavior in terms of instantaneous descriptions in Section 6.1.4 (p. 230).		
		69	Pushdown automata are the subject of Section 6.1 (p. 225). See especially the informal description of how these automata move in Section 6.1.1 (p. 225) and the formal definition of their behavior in terms of instantaneous descriptions in Section 6.1.4 (p. 230).		
		61	Pushdown automata are the subject of Section 6.1 (p. 225). See especially the informal description of how these automata move in Section 6.1.1 (p. 225) and the formal definition of their behavior in terms of instantaneous descriptions in Section 6.1.4 (p. 230).		
		62	Pushdown automata are the subject of Section 6.1 (p. 225). Also, acceptance by final state is in Section 6.2.1 (p. 235).		
		63	The construction of pushdown automata from grammars is in Section 6.3.1 (p. 243).		
		66	See the complete algorithm for construction of the grammar in Section 6.3.2 (p. 247).		
#15	Chapter 07: CFG's --- Normal Forms		Based on Section 7.1 of HMU.		Class 9
		24	An algorithm for finding generating symbols is in Section 7.1.2 (p. 264).		
		25	The algorithm for finding nullable symbols is in Section 7.1.3 (p. 265).		
		26	Find all the nullable symbols, and then use the construction from Section 7.1.3 (p. 265) to modify the grammar's productions so there are no ε-productions. The algorithm for modifying the grammar to eliminate ε-productions is within Section 7.1.3 starting on p. 266.		
		27	The algorithm for finding unit pairs is in Section 7.1.4 (p. 268).		
		28	Section 7.1.4 (p. 268). The algorithm for eliminating unit productions is in Section 7.1.4 (p. 268).		
		29	Q: Section 7.1.5 (p. 272) The Chomsky-normal-form algorithm is in Section 7.1.5 (p. 272).		
		58	The basic components of a grammar are defined in Section 5.1.2 (p. 173). Eliminating useless symbols and productions is the subject of Section 7.1.1 (p. 262).		
#16	Chapter 07: Properties of		Based on Sections 7.2, 7.3, and 7.4 of HMU.		Class 12

	CFL's				Class 13 Class 14
		1	You should examine the statement of the Pumping Lemma in Section 7.2.2 (p. 280) and the examples in Section 7.2.3.		
		15	The complete CYK algorithm is described in Section 7.4.4 (p. 303).		
		34	Closure of context-free languages under union is a special case of the substitution operation described in Section 7.3.1 (p. 287). Theorem 7.24(1) describes the particular substitution involved, from which we can discover possible grammar modifications that perform the union.		
		35	Closure of context-free languages under concatenation is a special case of the substitution operation described in Section 7.3.1 (p. 287). Theorem 7.24(2) describes the particular substitution involved, from which we can discover possible grammar modifications that perform the concatenation.		
		38	You can use the pumping lemma (Section 7.2, p. 279) to prove this fact. Section 7.3.4 (p. 291) shows that the intersection of a regular language and a CFL is a CFL.		
		65	Homomorphisms are defined in Section 4.2.3 (p. 140) and inverse homomorphisms in Section 4.2.4 (p. 142).		
#17	Chapter 07: Properties of CFL's --- Additional Questions		These are questions based on Section 7.3 of HMU that were not selected for the main homework on the topic.		Class 12 Class 13 Class 14
		36	There is a variant of the pumping lemma (Section 7.2, p.279) that applies only to linear languages. Can you prove this lemma, making use of the fact that for a linear grammar, the path leading to *w* in Fig. 7.6 (p. 282) must have all the variables in the entire parse tree?		
		37	Section 4.2.1 (p. 133) discusses closure of regular languages under concatenation. Closure of the context-free languages under concatenation is covered in Section 7.3.2 (p. 289).		
		64	Homomorphisms are defined in Section 4.2.3 (p. 140).		
		67	You should check the definitions of homomorphisms (Section 4.2.3, p. 140) and their inverses (Section 4.2.4, p. 142). The ideas behind inductive proofs are described in Section 1.4 (p. 19).		
#18	Chapter 08: Turing Machines		based on Chapter 8 of HMU.		Class 15 Class 16

		68	Section 8.2.2 (p. 326), and the formal notion of moves between instantaneous descriptions is in Section 8.2.3 (p. 327).		
		104	Section 8.2.2 (p. 326). The formal notion of moves of a TM as a sequence of instantaneous descriptions is in Section 8.2.3 (p. 327).		
		105	The formal notion of moves of a TM as a sequence of instantaneous descriptions is in Section 8.2.3 (p. 327).		
		106	Section 8.2.2 (p. 326). The formal notion of moves of a TM as a sequence of instantaneous descriptions is in Section 8.2.3 (p. 327).		
		107	Nondeterministic Turing machines are introduced in Section 8.4.4 (p. 347).		
#19	Chapter 09: Undecidabilty		Based on Chapter 9 of HMU. Chapter 9 - Undecidability 9.1 A Language That Is Not Recursively Enumerable 9.2 An Undecidable Problem That Is RE 9.3 Undecidable Problems About Turing Machines 9.4 Post's Correspondence Problem 9.5 Other Undecidable Problems 9.6 Summary of Chapter 9 9.7 Gradiance Problems for Chapter 9 9.8 References for Chapter 9		Class 19 Class 20
		72	Section 9.5.3 (p. 415), especially Theorem 9.22 (p. 416), gives a number of undecidable problems about grammars. Start with the pumping-lemma constant *n* (Section 7.2, p. 279) for G. Testing emptiness of a context-free language is discussed in Section 7.4.3 (p. 302). Ref: 7.2 The Pumping Lemma for Context-Free Languages (p. 279) 9.5.3 The Complement of a List Language (p. 415) Theorem 9.22 Let G_1 and G_2 be context-free grammars, and let R be regular expression. Then the following are undecidable: a) Is $L(G_1) \cap L(G_2) = \emptyset$? b) Is $L(G_1) = L(G_2)$? c) Is $L(G_1) = L(R)$? d) Is $L(G_1) = T^*$ for some alphabet T? e) Is $L(G_1) \subseteq L(G_2)$? f) Is $L(R) \subseteq L(G_1)$?	12.2 Undecidable Problems for Recursively Enumerable Languages (p. 308) Theorem 12.3 (p. 309) 7.2 Pumping Lemma for Context-Free Language	

<table>
<tr>
<td></td>
<td></td>
<td>73</td>
<td>Recall the definition of a recursive Turing machine in Section 9.2.1 (p. 383). Also, you may wish to examine Section 11.2.1 (p. 487), which talks about polynomial-space-bounded Turing machines.

Choice 7: Can you compare this sort of TM to a finite automaton, in particular to a deterministic finite automata with epsilon-transitions (Section 2.5, p. 72)?

Ref:
9.2.1 Recursive Language (p.383)
11.2.1 Polynomial-Space Turing Machines (p. 487)
2.5 Finite Automata With Epsilon-Transitions (p.72)

<u>Question Explanation:</u> There is an explanation for each of the problems:
For Example: Is Comp(L(G)) equal to (0+1)*? This problem is the same as asking if L(G) is empty.</td>
<td>11.4 The Chomsky Hierarchy
Recursive Languages

Chapter 14: An Overview of Computational Complexity</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>116</td>
<td>See the discussion of Rice's Theorem in Section 9.3.3 (p. 397).

Ref:
9.3.3 Rice’s Theorem and Properties of the RE Languages

<u>Question Explanation:</u> The key observation is that given any TM M, we can design a very restricted form of TM that simulates M by writing successive ID's on a tape. The simulating TM can run back and fourth on the tape, computing the symbols of the next ID, one at a time. Remember that, unless the symbol being copied is adjacent to the head, the symbol cannot change.
On the other hand, if we limit the amount of tape that the TM may use to any computable function of the input length, then we can accept only recursive languages. The reason is that after a while, the TM must repeat an ID, and if it hasn't accepted by then, we can conclude it never will. Likewise, if we limit the number of moves to any computable function, we can accept only recursive languages.

Finally, if we limit the tape heads to move in only one direction, then nothing it writes can ever affect what it does. Thus, the TM can be simulated by a finite automaton, no matter how many tapes the TM has.</td>
<td>Rice’s Theorem (p. 311)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>117</td>
<td>Post's Correspondence Problem is treated in Section 9.4 (p. 401), and in particular you should read Section 9.4.2 (p. 404) on the "modified" PCP.

Ref:
9.4 Post’s Correspondence Problem (p.401)
9.4.2 The “Modified” PCP (p. 402)
Question Explanation: The PCP instance is (fill in the table):
<table>
<tr><td></td><td>List C</td><td>List D</td></tr>
<tr><td>0</td><td>*0*1*</td><td></td></tr>
<tr><td>1</td><td></td><td></td></tr>
</table>
</td>
<td>Ref:
12.3 The Post Correspondence Problem (p. 311)
The “Modified” PCP Problem (p. 313)</td>
<td></td>
</tr>
</table>

<table>
<tr><td></td><td></td><td></td><td><table><tr><td>2</td><td></td><td></td></tr><tr><td>3</td><td></td><td></td></tr><tr><td>4</td><td>$</td><td></td></tr></table></td><td></td><td></td></tr>
<tr><td></td><td></td><td>118</td><td>Section 9.4.3 (p. 407). We assume the TM M satisfies Theorem 8.12 (p. 353)

Ref:
9.4.3 Completion of the proof of PCP Undecidability (p. 407)
Theorem 8.12 (p. 353) - This topic under: 8.5.1 Turing Machines with Semi-infinite Tapes

Question Explanation: We know we need the following pairs:
1.
2.
3.
4.
5.</td><td>12.1 Some Problems That Cannot Be Solved by Turing Machines

Turing Machines with Semi-infinite Tapes (p. 255)</td><td></td></tr>
<tr><td></td><td></td><td>137</td><td>Section 9.3.1 (p. 392) for a discussion of what is implied by the existance of a reduction from one problem to another.

Section 9.3.2. on p. 394.

Ref:
9.3.1 Reductions (p. 392)
9.3.2 Turing Machines That Accepts the Empty Languages (p. 394)

Question Explanation:
If there is a reduction from P1 to P2, then P2 must be at least as hard as P1 (and may be harder). Moreover a solution to P2 combined with the reduction from P1 to P2, implies a solution to P1.</td><td>(p. 292)
Theorem 12.3 (p. 309)</td><td></td></tr>
<tr><td></td><td></td><td>138</td><td>Section 9.3.2 on p. 394.

Rice's Theorem is in Section 9.3.3 (p. 397).

Section 9.2.1 (p. 383).

Ref:
9.3.2 Turing Machines That Accepts the Empty Languages (p. 394)
9.3.3 Rice's Theorem and Properties of the RE Languages

Question Explanation:
L_1 is not RE. To prove this statement we can reduce the non-RE language L_e (Section 9.3.2., p. 394) to L_1. Let M be the Turing machine for the empty language L(M). Then,</td><td>Rice's Theorem (p. 311)
Theorem 12.3 (p. 309)
Rice's Theorem (p. 311)</td><td></td></tr>
</table>

			given an instance M_1 of L_e, we can reduce it to the instance (M_1,M) of L_1. M_1 is in L_e if and only if $L(M_1)$ is a subset of $L(M)$ L2 .. L3 .. L4		
#20	Chapter 10, 11: Intractability		Questions about languages classes NP and above, based on Sections 10.1, 11.1, 11.2, and 11.3 of HMU Chapter 10 - Intractable Problems 10.1 The Classes P and NP 10.2 An NP-Complete Problem 10.3 A Restricted Satisfiability Problem 10.4 Additional NP-Complete Problems 10.5 Summary of Chapter 10 10.6 Gradiance Problems for Chapter 10 10.7 References for Chapter 10 Chapter 11 - Additional Classes of Problems 11.1 Complements of Languages in NP 11.2 Problems Solvable in Polynomial Space 11.3 A Problem That Is Complete for PS 11.4 Language Classes Based on Randomization 11.5 The Complexity of Primality Testing 11.6 Summary of Chapter 11 11.7 Gradiance Problems for Chapter 11 11.8 References for Chapter 11	14 An Overview of Computational Complexity 14.1 Efficiency of Computation 14.2 Turing Machine Models and Complexity 14.3 Language Families and Complexity Classes 14.4 Some NP Problems 14.5 Polynomial-Time Reduction 14.6 NP-Completeness and an Open Question	
		92	Read Section 10.1.5 (p. 433) on polynomial-time reductions and Section 10.1.6 (p. 434) on what it means if a problem is NP-complete Ref: 10.1.5 Polynomial-Time Reduction (p. 433) 10.1.6 NP-Complete Problems (p. 434) Question Explanation: Use Choices feedback.	Ref: 14.6 Polynomial-Time Reduction (p 360) 14.7 NP-Completeness and an Open Question (p. 362)	
		93	The definition of the class P is in Section 10.1.1 (p. 426). The definition of the class NP is in Section 10.1.3 (p. 431). Ref: 10.1.1 Problems Solvable in Polynomial Time (p. 426) 10.1.3 Nondeterministic Polynomial Time (p. 431)	Ref: 14.6 Polynomial-Time Reduction (p 360)	

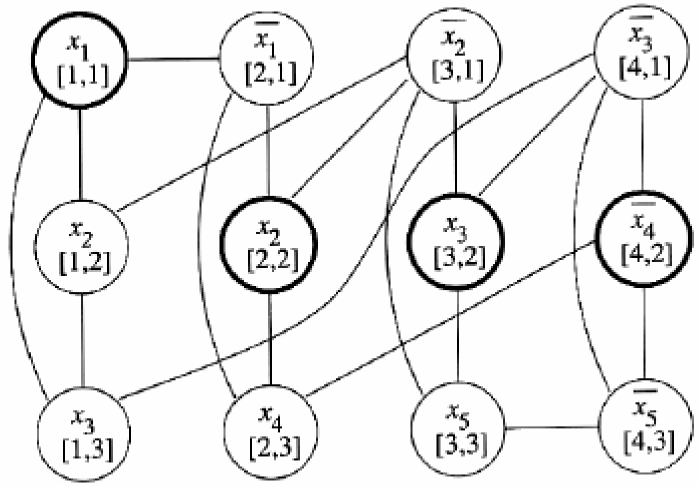
			Question Explanation: Use Choices feedback.		
		96	The conditions for a reduction to be polynomial-time are in Section 10.1.5 (p. 433). Ref: 10.1.5 Polynomial-Time Reduction (p. 433) Question Explanation: Use Choices feedback.	Ref: 14.6 Polynomial-Time Reduction (p 360)	
		102	See Section 9.2.1 (p. 383). See Section 11.1 (p. 484). See Section 11.2.2 (p. 488). See Section 11.2.3 (p. 490). Ref: 9.2.1 Recursive Language (p. 383) 11.1 Complements of Languages in NP (p. 484) 11.2.2 Relationships of PS and NPS to previously Defined Classes (p. 488) 11.2.3 Deterministic and Nondeterministic Polynomial Space (p. 490) Question Explanation: Since we do not know whether P=NP, or whether NP=co-NP (i.e., whether NP is closed under complementation), we do not know whether any of A, B, or C is __?___. If we know P=NP, then surely A and B are __?__. But if P=NP, then co-NP=NP=P, since P is closed under complementation. Thus, C would be ___?___as well.	Ref: 11.4 The Chomsky Hierarchy Recursive Languages Chapter 14: An Overview of Computational Complexity	
		103	The Hamilton-path problem is discussed in Exercise 10.4.5 (p. 477). Also look at the material on (the related) weighted Hamilton circuits problem in Section 10.4.5 (p. 471). Hint: Consider the running time of Dijkstra's algorithm (not covered in the book). The complexity of SAT is in Section 10.2.3 (p. 440). Complexity of the QBF problem is discussed in Section 11.3.4 (p. 496). Ref: 10.4.5 Undirected Hamilton-Circuit Problem (p. 471) 10.2.3 NP-Completeness of the SAT Problem (p. 440) 11.3.4 PS-Completeness of the QBF Problem (p. 496) Question Explanation: Use Choices feedback.	**Ref:** Example 14.7 The Hamiltonian Path Problem (p. 357) SAT: Example 14.2 (p. 348) Example 14.6 (p. 357) Example 14.9 (p. 360) Theorem 14.5 The Satisfiability Problem (SAT) is NP-complete. (p. 363) QBF: quantified Boolean formula problem (QBF) External Reading	
		139	Relevant reading includes most of Section 10.1 (p. 426). The question is based on definitions of NP, NP-hardness, NP-completeness, and Theorems 10.4 and 10.5 in Section 10.1.6, p. 434--435.	14 An Overview of Computational Complexity 14.1 Efficiency of Computation 14.2 Turing Machine Models and Complexity 14.3 Language Families and Complexity Classes 14.4 Some NP Problems 14.5 Polynomial-Time Reduction	

			Ref: 10.1 The Classes P and NP 10.4 Additional NP-Complete Problems 10.5 Summary of Chapter 10 10.1.6 NP-Complete Problems Question Explanation: The question is based on definitions of NP, NP-hardness, NP-completeness, and Theorems 10.4 and 10.5 in Section 10.1.6, p. 434—435. Use Choices feedback.	14.6 NP-Completeness and an Open Question	
#21	Chapter 10: Intractability --- SAT and Related Problems		Based on Sections 10.2 and 10.3 of HMU. Ref: 10.2 An NP-Complete Problem 10.3 A Restricted Satisfiability Problem	14 An Overview of Computational Complexity 14.1 Efficiency of Computation 14.2 Turing Machine Models and Complexity 14.3 Language Families and Complexity Classes 14.4 Some NP Problems 14.5 Polynomial-Time Reduction 14.6 NP-Completeness and an Open Question	
		69	The satisfiability problem is defined in Section 10.2.1 (p. 438). Ref: 10.2.1 The Satisfiability Problem (p. 438) Question Explanation: All choices fall into one of four categories (possibly with clauses reordered)	Ref: Theorem 14.5 The Satisfiability Problem (SAT) is NP-complete. (p.363) (p. 348 – p. 363)	
		94	The definition of 3-CNF is in Section 10.3.1 (p. 448). Note that the reduction of SAT to CSAT in Theorem 10.13 (p. 452) and of CSAT to 3-SAT in Theorem 10.15 (p. 457) Ref: Theorem 10.13: CSAT is NP-complete. (p. 452) Theorem 10.15: 3SAT is NP-complete. (p. 457) Question Explanation: The simplest way to proceed is to use the distributing law of OR over AND, three times, to distribute u+v over wxyz. The result is ….	Ref: CSAT is the problem: given a Boolean expression in CNF, is it satisfiable? Example 14.2 (p. 348) 3SAT: Example 14.9 (p. 360) Example 14.10 (p360)	
		95	The polynomial-time reduction from SAT to CSAT, as described in Section 10.3.3 (p. 452). Ref: 10.3.3 NP-Completeness of CSAT Question Explanation: The first subexpression to which we apply the transformation is vw. The AND rule is simple: take the AND of the clauses for each side. That gives us (v)(w) as the CNF expression.	Ref: Example 14.2 (p. 348)	

			Next, we work on u+(vw). The rule for OR requires us to introduce variable y1. It is added positively to all the clauses on the left side and negatively to all clauses on the right side. That gives us ____________. Finally, we apply the same transformation to (u+(vw))+x, introducing y2. The final answer is ________________.		
		97	Theorem 10.15 (p. 457) The reduction of CSAT to 3SAT is in Section 10.3.4 (p. 456). Question Explanation: (a+b) becomes ________ (c+d+e+f) becomes ____________ (g+h+i+j+k+l+m) becomes _________	Ref: 14.7 NP-Completeness and an Open Question Theorem 1 4.5 The Satisfiability Problem (SAT) is NP-complete. Example 14.1 1 (p. 363)	
		144	Ref: N/A Question Explanation: If each clause has at least two literals with different truth values, they end up in different partitions of the cut. The maximum contribution from a single clause to the total cut-size is hence ____. For 4 clauses, the total is ____. Complemented and uncomplemented literals for the same variable will necessarily be in different partitions of the cut and hence each such pair contributes 1 to the cut-size. There are _____ such pairs in E with a total contribution of 7. Hence maximum cut-size = ___ + ____ = ______.		
#22	Chapter 10: Intractability --- Some NP Complete Problems		Questions about reductions and the meaning of certain NP-complete problems based on Section 10.4 of HMU. Ref: 10.4 Additional NP-Complete Problems (458)	Ref: 14.7 NP-Completeness and an Open Question	Class 22 Class 23
		98	Theorem 10.18 (p. 460), which reduces 3SAT to Independent Sets. The reduction of 3SAT to Independent-Set is explained in Section 10.4.2 (p. 459). Ref: Theorem 10.18: The independent-set problem is NP-complete. (p 460) 10.4.2 The Problem of Independent Sets. (p. 459) Question Explanation: There are always edges between nodes [*i*,1], [*i*,2], and [*i*,3] for any *i*. In addition, there are edges between nodes corresponding to a literal and its complement. These edges are:	Ref: 14.5 Polynomial-Time Reduction 14.6 NP-Completeness and an Open Question	

		99	The definition of the problem Independent-Set is at the beginning of Section 10.4.2 (p. 459). Ref: 10.4.2 The Problem of Independent Sets. (p. 459) An independent set can have at most four nodes. Since Independent-Set is an NP-complete problem, it should not surprise us that even for as simple an instance as this, it is rather hard to reason about how many independent nodes there are. But here is a rough argument for why there can be no more than four. Question Explanation: First, we may as well pick A, because if a maximal independent set included B or G instead, we could replace it by A. If it included neither B nor G, we could add A and get a larger independent set Likewise, we may as well include L. Now, G, B, K, and F are eliminated; they cannot be chosen for a maximal independent set containing A and L. We can surely pick two of the remaining nodes --- C, D, E, H, I, and J. For example, we can add H and E to the maximal independent set, giving us 4 nodes: {A,E,H,L}. However, picking any of these six nodes eliminates at least two others. Thus, we could never pick three of these six. There is no possibility of a maximal independent set with __?__ nodes.	Ref: 14.6 NP-Completeness and an Open Question	
		100	The definition of the problem Node-Cover is in Section 10.4.3 (p. 463). Ref: 10.4.3 The Node-Cover Problem (p. 463) Question Explanation: A node cover must have at least ___?___ nodes. Since Node-Cover is an NP-complete problem, it should not surprise us that even for as simple an instance as this, it is rather hard to reason about how many nodes we need to cover all edges. But here is a rough argument for why there must be at least eight. First, we may as well eliminate A. The triangle among A, B, and G tells us we must pick at least two of these three, and A covers no edge that the other two do not. Likewise, we may as well assume L is not in the minimal node cover. So we have {B,F,G,K} as a subset of our node cover. This set covers all edges except those that have both ends in {C,D,E,H,I,J}. The triangle among C, H, and I tells us we must pick at least two of those, and the triangle among D, E, and J says we need two of those. That is sufficient; for example, picking C, D, I, and J covers all the remaining edges, giving us a node cover of {____________}.	Ref: External Reading	

		101	The problem of finding a minimum-weight Hamilton circuit is really the Traveling-Salesman problem, as described in Section 10.4.5 (p. 471). You may want to skim Section 10.4.4 (p. 465) for the proof of NP-completeness of the directed version of the Hamilton-circuit problem. Ref: 10.4.5 Undefined Hamilton Circuits and the TSP (p. 471) 10.4.4 The Directed Hamilton-Circuit Problem (p. 465) Question Explanation: Nodes B and E only have two incident edges, so each of those four edges must be in any Hamilton circuit, even though they are the edges of highest weight. Thus, there is only one Hamilton circuit --- the one that goes around the outside of the diagram. If we start at A, for example, this circuit is ___________.	Ref: 14.6 Exercises (P. 362)	
		140	The Independent-Set problem is shown to be NP-complete in the text (Theorem 10.18, Section 10.4.2, p. 460) by a reduction from 3SAT. In the text, the Node-Cover problem is proved to be NP-complete (Theorem 10.20, Section 10.4.3, p. 464) by a reduction from the Independent-Set problem (See Figure 10.8 on p. 461). Ref: 10.4.2 The Problem of Independent Sets. (p. 459) 10.4.3 The Node-Cover Problem (p. 463) Theorem 10.20; The node-cover problem is NP-coplete Question Explanation: The solution is based on establishing correspondence between satisfiable assignments for the expression E and node covers for the graph G. Each column of the graph corresponds to a clause (See Figure 10.8 on p. 461). Let *m* be the number of clauses in E. We claim that E is satisfiable if and only if G has a node cover of size *2m*.	Ref: Example 14.7 (p. 357)	

			 Figure 10.8: Construction of an independent set from a satisfiable boolean expression in 3-CNF		
		141	The coloring problem is defined in Exercise 10.4.2 (p. 474). Ref: Exercise 10.4.2: This is Exercise for section 10.4 (p. 474) Question Explanation: The solution is based on establishing a correspondence between satisfiable assignment for a 3SAT expression and a valid coloring for the corresponding graph. *! **Exercise 10.4.2:** The *coloring problem* is: given a graph G and an integer k, is G "k-colorable"; that is, can we assign one of k colors to each node of G in such a way that no edge has both of its ends colored with the same color. For	Ref: External Reading	

example, the graph of Fig. 10.1 is 3-colorable, since we can assign nodes 1 and 4 the color red, 2 green, and 3 blue. In general, if a graph has a k-clique, then it can be no less than k-colorable, although it might require many more than k colors.

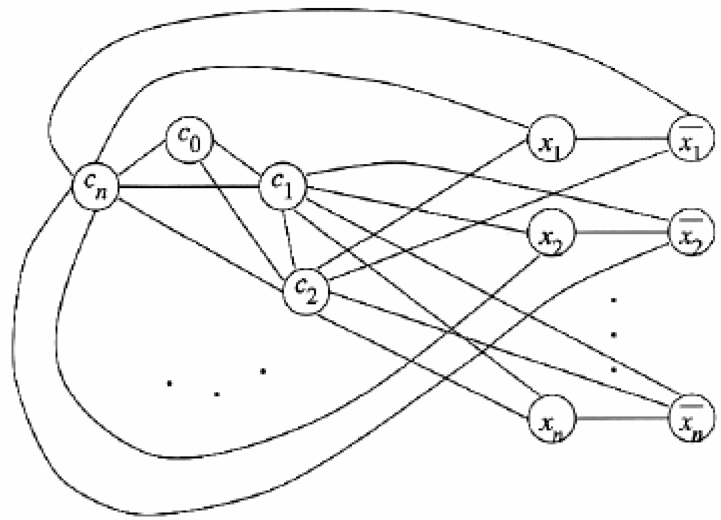


Figure 10.13: Part of the construction showing the coloring problem to be NP-complete

In this exercise, we shall give part of a construction to show that the coloring problem is NP-complete; you must fill in the rest. The reduction is from 3SAT. Suppose that we have a 3-CNF expression with n variables. The reduction converts this expression into a graph, part of which is shown in Fig. 10.13. There are, as seen on the left, $n+1$ nodes $c_0, c_1, \ldots, c_n$ that form an $(n+1)$-clique. Thus, each of these nodes must be colored with a different color. We should think of the color assigned to c_j as "the color c_j."

Also, for each variable x_i, there are two nodes, which we may think of as x_i and $\overline{x_i}$. These two are connected by an edge, so they cannot get the same color. Moreover, each of the nodes for x_i are connected to c_j for all j other than 0 and i. As a result, one of x_i and $\overline{x_i}$ must be colored c_0, and the other is colored c_i. Think of the one colored c_0 as true and the other as false. Thus, the coloring chosen corresponds to a truth assignment.

To complete the construction, you need to design a portion of the graph for each clause of the expression. It should be possible to complete the coloring of the graph using only the colors c_0 through c_n if and only if each clause is made true by the truth assignment corresponding to the choice of colors. Thus, the constructed graph is $(n+1)$-colorable if and only if the given expression is satisfiable.

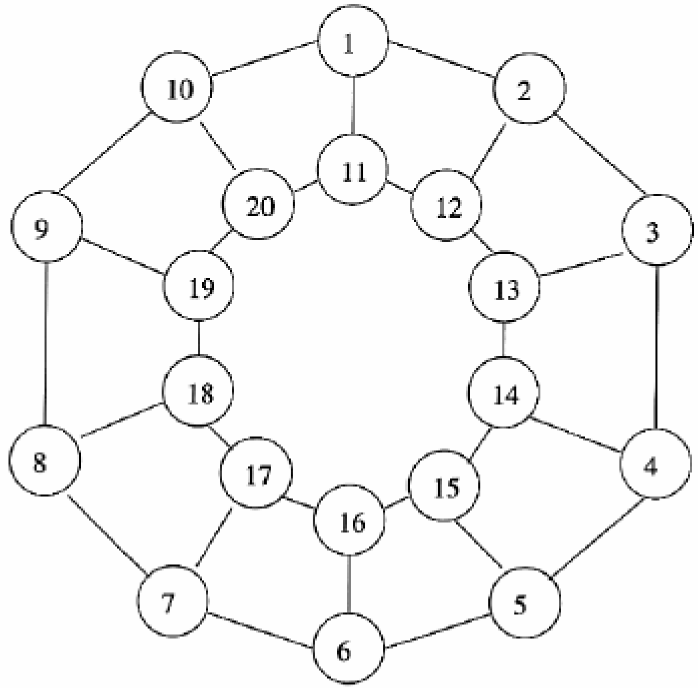
<table>
<tr><td></td><td></td><td></td><td>


Figure 10.14: A graph
</td><td></td><td></td></tr>
<tr><td></td><td></td><td>142</td><td>
Theorem 10.18 (p. 460).
The clique problem is defined in Exercise 10.4.1 (p. 473).

Ref:
Theorem 10.18: The independent-set problem is NP-complete. (p. 460)
Exercise 10.4.1: This is Exercise for section 10.4 (p. 473)

* **Exercise 10.4.1:** A *k-clique* in a graph G is a set of k nodes of G such that there is an edge between every two nodes in the clique. Thus, a 2-clique is just a pair of nodes connected by an edge, and a 3-clique is a triangle. The problem CLIQUE is: given a graph G and a constant k, does G have a k-clique?
a) What is the largest k for which the graph G of Fig. 10.1 satisfies CLIQUE?
b) How many edges does a k-clique have, as a function of k?
c) Prove that CLIQUE is NP-complete by reducing the node-cover problem to CLIQUE.

Question Explanation:
The solution is based on establishing a correspondence between satisfiable assignments of
</td><td>
Ref:
Example 14.8 (p. 358)
(p. 358 – p. 362)
</td><td></td></tr>
</table>

			the 3-CNF expression E and cliques in the complement H of G. Let *m* be the number of clauses in the E. Then we claim that E is satisfiable if and only if H has an *m*-clique.		
		143	Some of the issues regarding "size" of inputs are illustrated in Section 10.1.2 (p. 426). Ref: 10.1.2 An Example: Kruskal's Algorithm. Question Explanation: All the propositions are in fact ___?____. Whether something is part of the input or not may make a difference about whether a problem is NP-complete or not or solvable in polynomial time or not. The algorithm A_1 will not count as a polynomial time algorithm for problem P_1, where C is part of the input. But algorithm A_2 is a polynomial-time algorithm for P_2 and A_3 is a polynomial time algorithm for P_4. How does algorithm A_1 work? Let $S = \{x_1, x_2,..., x_n\}$. Let P(i,s) be TRUE if there is a subset of $\{x_1, x_2,..., x_i\}$ that sums to *s* and FALSE otherwise. Then the recurrence $P(i,s) = P(i-1,s)$ OR $P(i-1,s-x_i)$ can be used to design a simple dynamic programming algorithm that fills up an *n*-by-C sized table with P(i,s) values. Algorithm A_2 works almost in the same way as A_1, except that c_0 is known in advance and hence the size of the table and the time to fill it are both O(n). Problem P_3 can be shown to be NP-complete by a reduction from 3SAT. Algorithm A_3 for problem P_4 works by finding all possible subsets of size *m* of nodes of graph G. For each such subset V' of size *m*, check all possible pairs (u,v) of nodes in V' for existence of an edge (u,v).	Ref: External Reading	