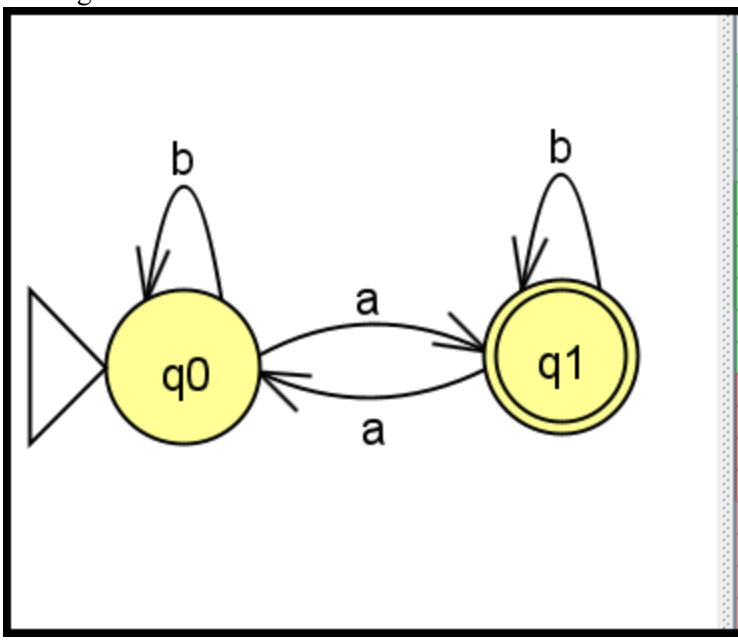
Final Exam Questions

Question 1: (20 points)

a) Let Σ = {a, b}. Construct a DFA for the language {w | w has an odd number of a's}. You can either give the formal definition of the DFA or draw a state diagram of the DFA.



Input	Result
a	Accept
ab	Accept
ba	Accept
aaa	Accept
aaab	Accept
baaa	Accept
ababa	Accept
bababa	Accept
abbbbbb	Accept
ababbbba	Accept
	Reject
aa	Reject
aba	Reject
baa	Reject
aab	Reject
abab	Reject
baaaa	Reject
aabbbbbb	Reject

b) Let Σ = {0, 1}. Give regular expressions generating the following language, {w| every odd position of w is a 1}

- $(1(0 \cup 1))^*(1 \cup \varepsilon)$

c) CFG G is given below:

$$S \to RT$$

$$R \to TR|a$$

$$T \to TR|b$$

- Does G accept string w=baba? If so, show the derivation tree.

S	starting variable
RT	$S \to RT$
RTR	$T \to TR$
$TRTR$	$R \to TR$
$bRTR$	$T \to b$
$baTR$	$R \to a$
$babR$	$T \to b$
$baba$	$R \to a$

  - As shown by the derivation tree, yes G accepts string w=baba

d) Are 7289 and 8029 relatively prime? Show the calculations that led to your conclusions.

(1) No, this pair of values are not relatively prime, since two numbers are relatively prime if 1 is the largest integer that evenly divides them both and these values share 37 as a common

divisor. We can calculate this using the Euclidean algorithm to find $\gcd(7289, 8029) = 37$. Our Process is as follows:

1) Using Division Algorithm, find q and r to write a=bq+r
2) Do step 1 again, but now use b as your new a, and use r as your new b
3) Stop when your $r = 0$. Your b from the last step is your final answer.

(2) Proof:

Step 1: Using Division Algorithm, find q and r to write $a = bq + r$
- $a = bq + r$
- $7289 = 8029 + r$
- $7289 = 8029(0) + 7289$

Step 2: Do step 1 again, but now use b as your new a, and use r as your new b
- $a = bq + r$
- $8029 = 7289q + r$
- $8029 = 7289(1) + 740$

Step 3: Do step 1 again, but now use b as your new a, and use r as your new b
- $a = bq + r$
- $7289 = 740q + r$
- $7289 = 740(9) + 629$

Step 4: Do step 1 again, but now use b as your new a, and use r as your new b
- $a = bq + r$
- $740 = 629q + r$
- $740 = 629(1) + 111$

Step 5: Do step 1 again, but now use b as your new a, and use r as your new b
- $a = bq + r$
- $629 = 111q + r$
- $629 = 111(5) + 74$

Step 6: Do step 1 again, but now use b as your new a, and use r as your new b
- $a = bq + r$
- $111 = 74q + r$
- $111 = 74(1) + 37$

Step 7: Do step 1 again, but now use b as your new a, and use r as your new b
- $a = bq + r$
- $74 = 37q + r$
- $74 = 37(2) + 0$

Step 8: Stop when your $r = 0$. Your b from the last step is your final answer.
- $\gcd(7289, 8029) = 37$

❖ By definition, two numbers are relatively prime if 1 is the largest integer that evenly divides them both. Thus, in order to prove that 7289 and 8029 are relatively prime, I calculated their greatest common divisor (gcd) using the Euclidean algorithm, and

obtained $\gcd(7289, 8029) = 37$. As such, I have successfully proven that 1274 and 10505 are not relatively prime as they share 37 as a common divisor.

Question 2: (10 points): Prove the language $\{0^m 1^n \mid m \neq n\}$ is not regular

By applying the pumping lemma for regular languages, I successfully proved that the language is not a regular language. I began the proof under the assumption that A was a regular language and then identified a contradiction in the necessary conditions. As this assumption led to a contradiction, A does not satisfy the conditions required for a language to be considered regular.

Proof

For this proof, lets assume the language name is A. If we also assume that A is regular, the Pumping Lemma definition tells us that any string s in A can be 'pumped' at least a 'pumping length' of p , then divided into three pieces $s = xyz$. For it to be regular, it must satisfy the following conditions:

(1) $\forall i \geq 0, xy^i z \in A$
(2) $|y| \geq 0$
(3) $|xy| \leq p$

We can test these conditions by doing the following:

- Assume A is regular
- Let p be the pumping length
- Choose a string s in language A to test: $s = 0^P 1^{P+P!}$
  - As we assume A is regular, s is a member of A
  - Because $|s| = p + (p + p!)$, and $p + (p + p!) \geq p$, then $|s| \geq p$
  - Since both statements above are true, the Pumping Lemma definition tells us that s can be split into three pieces $s = xyz$
- For conditions (2) $|y| \geq 0$ and (3) $|xy| \leq p$ to be met, piece y must contain only 0's
  - For example, consider $p = 1$

$$0^P 1^{P+P!} = 0^1 1^{1+(1)}$$
$$0^1 1^2 = 011$$

  - To split $s = xyz$, with $|xy| \leq 3$, y may only contain 0's

$$011$$
$$y\ z$$

- According to condition (1) $\forall i \geq 0, xy^i z \in A$
  - assume $i = 2$

$$xy^i z \in A$$
$$xy^2 z \in A$$

  - Visual Representation

$$0011$$
$$yy\ z$$

- $xy^2z \notin A$
    - However, there is no possible division of 0011 that will result in the required $\{0^m1^n \mid m \neq n\}$ format. This is evident in the visual representation above as it has the format 0^21^2 and $2 = 2$, yet our formula states that $m \neq n$. Since I was able to prove that this string does not meet the condition (1) for Pumping Lemma.s, I have successfully proven that it is not regular.

Question 3: (15 points)

a) What is the difference between Turing Recognizable and Turning Decidable? Could you give a language which is Turing recognizable and not Turing decidable?

Language L is considered Turing Recognizable if there exists a Turing Machine M that determines if a given input string w is a member of the language or not by accepting and halting, rejecting and halting, or looping. Of these three possible outcomes, if $w \in L$, then M halts in q_{accept} and if $w \notin L$, M either halts in q_{reject} or enters a loop and does not halt. The language recognized by the Turing Machine is denoted as $L(M)$.

The terms Recursive Enumerability and Turing recognizability are often used interchangeably due to the Turing Machine (TM) adaptation, the enumerator. An enumerator is a TM with an attached printer where it produces, or enumerates, strings to an output tape. Language L is considered Turing Recognizable "if and only if some enumerator enumerates it." In other words, L is Turing Recognizable if and only if there exists A TM M that accepts and halts on all strings in L and loops or rejects all strings not in L.

However, this concept has its limitations. In some cases, it can be difficult to determine if a machine is looping or is still computing. This problem arises because there is no way to predict how long it will take for a machine to solve a problem. For example, if an enumerator's output tape does not contain string w, there is no possible way to distinguish if the machine is still processing or if w is not part of the language recognized by the machine.

Turing Decidability arises precisely to address this challenge. Language L is considered Turing Decidable if there exists A TM M that determines if a given input string w is a member of the language or not, halting in a state of accept or reject. Of these two possible outcomes, if $w \in L$, then M halts in q_{accept} and if $w \notin L$, M halts in q_{reject} .

The main difference between Turing Decidable and Turing Recognizable is that Turing Decidable machines will always halt and 'decide' a definitive answer on all inputs, rather than continuing to loop or process indefinitely. As such, Turing Decidability can determine whether an input belongs to a language or not, whereas Turing Recognizability may not necessarily provide a definite answer for all cases.

It is important to note that the concept of Turing Decidability is a subset of Turing Recognizability, which means that every Turing decidable problem is also Turing recognizable. However, it is essential to note that not all problems are Turing decidable. Turing Recognizability is a broader concept that encompasses all problems that can be recognized by a TM, regardless of whether they can be decided by such machines or not. For example, the language $a_{TM} = \{(M, w) | \ M \ is \ a \ TM \ and \ M \ accepts \ w\}$ is considered Turing Recognizable but not Turing Decidable. This is because a universal TM U is able to

recognize whether the TM M accepts a given string w as input. Essentially, it does this by simulating M on w and accepting if M ever enters its accept state, and rejecting if M ever enters its reject state - if M accepts w, then U accepts, and if M rejects w, then U rejects. However, if M loops on w, then U will also loop and never halt. As U could potentially end up looping on w, it does not meet the definition of Decidability which states a Decidable TM must halt on all inputs. However, it is considered Recognizable because it will either accept, reject, or loop, on all inputs. This example shows the importance of understanding the subtle differences between Turing Decidability and Turing Recognizability, along with their implications and limitations in the field of computation theory. Such limitations are important to understand because there are some problems that cannot be solved algorithmically, even by the most powerful computing model we know of, the Turing machine.

b) What is the relation between P, NP and NP-Complete? Could you find a language which belongs to the class of NP but not belong to class of P?

P, NP and NP-Complete pertain to different classes of decision problems in computational complexity theory, such that each one represents a different type of problem with different properties and characteristics.

A P-Problem is a problem that is efficiently decidable in polynomial, such as n^k, time on a deterministic single-tape Turing machine. These problems are considered "tractable" and the class of problems that are realistically solvable can computed using the function:

$$P = \bigcup_k TIME(n^k)$$

"union of all polynomial time functions"

An NP-Problem stands for "nondeterministic polynomial time" and it is a decision problem that can be solved by a non-deterministic Turing machine in polynomial time. In other words, given a potential solution to an NP problem, it can be verified to be correct or incorrect in polynomial time. However, finding a solution may require a non-polynomial amount of time.

NP-Complete refers to a class of problems whose "individual complexity is related to that of the entire class." In other words, if an efficient, polynomial time algorithm exists for solving a problem within the class, then an efficient algorithm exists for solving all problems in NP. An example of an NP-Complete problem includes the Boolean satisfiability problem.

There is much conjecture regarding the relationships between P, NP, and NP-Complete.While every problem in P is also in NP, it is not known whether NP problems are in P. Similar to how P is a subset of NP, NP-Complete is a subset of NP.

An example of a language which belongs to the class of NP but not belong to class of P is the SUBSET-SUM problemconcerning integer arithmetic. This problem asks whether, given a collection of numbers and a target integer t, there is a non-empty subcollection of the set that adds up to t. While the problem can be solved in nondeterministic polynomial time, there is no known polynomial time algorithm for solving the problem deterministically. This means that the SUBSET-SUM problem is considered to be in the class of NP but not in the class of P.

c) Are all languages Turing Recognizable? Briefly explain your answer.

No, not all languages are Turing Recognizable. According to the book, if both a language and its complement are Turing-recognizable, the language is decidable. However, in Question 3a, we showed

that $a_{TM} = \{(M, w) | \ M \ is \ a \ TM \ and \ M \ accepts \ w\}$ was Recognizable but not decidable. As such, for this to be true, the co-Turing Recognizable language $\overline{a_{TM}}$ can not be Turing Recognizable.

Question 4: (15 points) Categorize the following languages:

a) $B = \{a^n b^n c^n \ | n \geq 0\}$
b) SAT (satisfiability) problem.
c) Hamilton path problem.
d) HALT problem.
e) Co-HALT problem.

Turning-recognizable language(s):	a b c
Turning-unrecognizable language(s):	
Turning-recognizable but undecidable language(s):	d e
Class P problem(s):	
NP-completeness problem(s):	b c

Question 5: (20 points) A triangle in an undirected graph is a 3-clique. Show that TRIANGLEϵP, where TRIANGLE={ <G> | G contains a triangle}.

(Hint: you need to design an algorithm to tell if an undirected graph contains a triangle in polynomial time.)

To show that $TRIANGLE \in P$, we need to show that there exists an algorithm that decides whether a given input graph G contains a triangle in polynomial, such as n^k, time on a deterministic single-tape Turing machine.
This can be done by checking all the possible sets of vertices in G, and check whether they form a 3-clique. This technique considers all possible sets of 3 vertices in the input graph with n total vertices, which can be done in $O(n^3)$ time complexity.

This technique can be shown in the following algorithm:

Input: A graph G with n vertices and m edges.
Output: "Yes" if G contains a triangle, and "No" otherwise.

Consider for triangle G, V is the set of nodes and E is the set of edges, such that $G = (V, E)$

M = On input $< G >$, where G is an undirected graph:

(1) For each set of 3 vertices $\{u, v, w\}$ in G do the following:
    a. Check if G contains edges $\{(u, v), (v, w), (u, w)\}$. If each edge exists, then we have found a triangle in G and can stop the algorithm.
(2) If no such set of vertices exists to meet these conditions, then G does not contain a triangle.

As step 1 runs in polynomial time ($O(n^3)$ time), and there are a polynomial number of steps ($O(n^3)$), this algorithm runs in polynomial time. Therefore, $TRIANGLE \in P$.

Question 6: (20 points) Given a $G = (V, E)$, a vertex cover in G is a subset $V' \subseteq V$ such that for all $(u, w) \in E, u \in V' \text{ or } w \in V'$

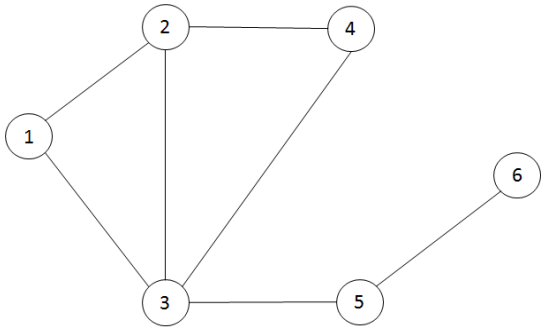


Figure 1

Figure 1 shows an undirected graph including 6 nodes. Answer the questions below:

1. Does Figure 1 have a vertex cover including four nodes? If so, given an example of 4-node vertex cover?
    a. Yes, Figure 1 has a vertex cover including four nodes. An example of such cover is $\{2,3,4,5\}$.

2. Does Figure 1 have a vertex cover including three nodes? If so, given an example of 3-node vertex cover?
    b. Yes, Figure 1 has a vertex cover including three nodes. An example of such cover is $\{2,3,5\}$.

3. What is the minimum vertex cover (including least nodes) in Figure 1?
    c. The minimum vertex cover for G is 3, as any less does not ensure all edges in the graph are covered

4. Let $VC = \{< G, k > \mid G \text{ has a } VC \text{ of size } \leq k\}$. VC Is this problem a NP problem? Prove your conclusion.

Yes, the problem of finding a vertex cover of size k or less, as described by the language $VC = \{< G, k > \mid G \text{ has a } VC \text{ of size } \leq k\}$., is an NP problem. To prove this, we must show that given a potential solution, or certificate, to the problem (i.e., a vertex cover of size k), we can verify whether it is a correct solution or not in polynomial time.

Given a graph G and a set of vertices S, we can verify whether S is a vertex cover of G in polynomial time as follows:

1. Check if S is a subset of V
    a. This can be done in $O(S)$ time.
2. Check if every edge $\{u, v\}$ in E, at least one of u and v belongs to S.
    a. This can be done in $O(E)$ time.
3. If both conditions above are met, then S is a valid vertex cover of G

Since each of these steps can be completed in polynomial time, VC is an NP problem.

4. Is VC a NP-complete problem? Why?

Since we know that VC is in NP, we need to also prove that VC is NP-hard to show that it is a NP-complete problem. We can prove that VC is NP-hard by showing the NP-complete problem, CLIQUE, can be reduced to the VC problem. For example, CLIQUE determines if graph G with n-cliques contains a clique size k. The complement of graph G, G', can be constructed to show that the existence of a clique of size k in G is equivalent to $|V| - k$ in G'. As such, any instance of the CLIQUE problem can be reduced to to the VC problem, this proving that VC is NP-hard and consequently, NP-complete.