1 Greedy Algorithms

1. **Copying Homework Problems**
You are copying a list of n problems from your textbook to the paper you do your homework on. You need to copy all the problems over, and you want to have them in order. Formally, the ith problem is l_i lines long, and you're copying them onto an ordered set of sheets of paper which are each m lines long.

   - The problems must be copied in the assigned order.
   - All problems must be copied.
   - Problems must be kept on one sheet of paper and can't be split across them.

   (a) Describe a greedy algorithm to minimize the number of pages you need.

   > **Solution:** The greedy solution is to put as many problems as possible (g) onto the first page until the next problem ($g+1$) would exceed the m line limit, then put the next problem onto the next page, until all n problems are copied.

   (b) Prove that the proposed greedy algorithm is correct. In other words, prove that the proposed algorithm satisfies each of the three constraints given in the problem.

   > **Solution:** We'll show by induction with the use of an exchange argument that all pages of the greedy solution match some optimal solution. Let G denote the solution given by our greedy algorithm. Our inductive hypothesis is that there exists some optimal solution `OPT` whose first k pages are the same as those in G.
   >
   > For the base case when $k = 0$, there is nothing to prove. The greedy and optimal pages trivially match.
   >
   > For the inductive step, assume that the first k pages of G and `OPT` are the same. Our goal is to find an optimal solution `OPT`$'$ whose first $k+1$ pages are the same as those of G.
   >
   > We know by the inductive hypothesis that the first k pages are identical, so now we consider the $(k+1)$st pages. If the $(k+1)$st are equal, then we are done. If the pages do not match, then we know that there must be more problems copied into the greedy solution than the optimal solution by the design of the greedy algorithm. The problems are copied in order, so the problems that are not on the $(k+1)$st page of `OPT` but are in the $(k+1)$st page of G must start on the $(k+2)$nd page of `OPT`. Construct `OPT`$'$ from `OPT` by copying problems in order starting from the $(k+2)$nd page onto the $(k+1)$st page until it matches the $(k+1)$th page of G. We are only moving problems forward, so we cannot increase the number of pages used to write the hw. So, the solution `OPT`' remains optimal. This concludes our inductive proof.

2. **BBBBBB**
Due to high demand, the EECS Department has finally decided to install a vending machine, called the Bob and Betty Beyster Building Beverage Bestower. Like any decent vending machine, the B^6 must provide change to customers after purchases. Because this is a building teeming with computer scientists, the machine should return the smallest number of coins

possible for any given amount of change. Due to design constraints, the machine only has space for three types of coins: 1¢, 2¢, and 5¢. The student designing the machine (who did not take EECS 376) decides that it should use a greedy algorithm to dispense change. Specifically, it first returns as many 5¢ coins as it can (without overpaying), then as many 2¢ coins as it can, then the appropriate number of 1¢ coins. The student hopes that this strategy will always return an optimal (smallest) number of coins; in this question, you (an EECS 376 student) will prove that this is indeed the case.

(a) Let c denote the amount of change and let n_1, n_2 and n_5 respectively denote the number of 1¢, 2¢, and 5¢ coins returned by the greedy algorithm. Likewise, let n_1^*, n_2^* and n_5^* denote the number of 1¢, 2¢, and 5¢ coins in an optimal solution. (That is, the sum $n_1^* + n_2^* + n_5^*$ is minimum.)

i. Show that $n_1^* \leq 1$.

> **Solution:** We want to show that in an optimal solution, the number of 1¢ coins (n_1^*) is at most one. Suppose, for establishing a contradiction, that $n_1^* \geq 2$. Then we may simply replace two of the 1¢ coins by one 2¢ coin, thereby reducing the total number of coins returned by one, which contradicts the assumed minimality of the original solution.

ii. Show that $2n_2^* + n_1^* \leq 4$. In other words, the *total amount* given in 1¢ and 2¢ coins at most 4.

> **Solution:** We shall continue with a another proof by contradiction, to show that $2n_2^* + n_1^* \leq 4$. For contradiction suppose that the total value change associated to n_1^* and n_2^* is more than 4 cents, i.e., $2n_2^* + n_1^* > 4$. If $n_1^* = 0$, then $2n_2^* > 4$, i.e., $n_2^* \geq 3$. Then we can replace three of the two-cent coins with one 5-cent and one one-cent coin, thereby contradicting optimality. If $n_1^* = 1$, then $2n_2^* > 3$, i.e., $n_2^* \geq 2$. Then we can replace two two-cent coins and one one-cent coin with a single five-cent coin, again contradicting optimality. These are the only cases because $n_1^* \leq 1$ by the previous part.

iii. Show that $n_5^* \geq n_5$ and conclude that $n_5^* = n_5$.

> **Solution:** We want to show that an optimal solution chooses at least as many 5¢ coins as the greedy solution does. Assume for contradiction that it doesn't for some total amount of change
>
> $$c = n_1^* + 2n_2^* + 5n_5^* = n_1 + 2n_2 + 5n_5.$$
>
> By assumption, $n_5^* < n_5$, so subtracting $5n_5^*$ from both sides gives us
>
> $$n_1^* + 2n_2^* = n_1 + 2n_2 + 5(n_5 - n_5^*) \geq n_1 + 2n_2 + 5.$$
>
> Because n_1, n_2 are non-negative, this means that $n_1^* + 2n_2^* \geq 5$, but this contradicts the previous part.

iv. Similarly, conclude that $n_2^* = n_2$. **Hint:** Use the previous parts.

> **Solution:** By the above, $n_5^* = n_5$, so we have $n_1^* + 2n_2^* = n_1 + 2n_2$. To show that $n_2^* = n_2$, we argue similarly to the previous part. If $n_2^* < n_2$, then by subtracting $2n_2^*$ from both sides we have $n_1^* \geq n_1 + 2 \geq 2$. This contradicts the first part.

v. Conclude that $n_1^* = n_1$. That is, the greedy algorithm outputs an optimal solution

> **Solution:** From the previous parts we have $n_5^* = n_5$ and $n_2^* = n_2$. It follows immediately that we must have $n_1^* = n_1$ as well, so the greedy solution is in fact optimal.

(b) In a parallel universe, the B^6 only has space for 1¢, k¢ and 5¢ coins, where k is an integer strictly between 1 and 5. Find a k such that the greedy approach does **not** return the optimal number of coins for every amount of change. Note that you just proved in (a) that with $k = 2$ the greedy algorithm gives the optimal answer.

> **Solution:** It is important to note that one counter-example will prove that not all amounts of change are optimal. Consider returning 8¢ in change with the denominations 1¢, 4¢ and 5¢ (here, $k = 4$). The greedy algorithm will return change in the following order: 5¢, 1¢, 1¢, 1¢ (a total of 4 coins). However, upon observation, we see that an optimal solution is 4¢, 4¢ (a total of 2 coins).

2 Decision Problems and Languages

1. Determine if the following problems are decision problems. If the problem is a decision problem, formulate a language containing exactly the "yes" instances of the decision problem (make sure to specify how you would encode the elements of the language). If the problem is not a decision problem, explain how you could change it into a decision problem.

(a) Given a set of cities, find a set of three cities whose name share at least 3 letters?

> **Solution:** This not a decision problem, as the problem is not asking for a yes/no answer. We could change it to a decision problem by rewording it to: *Given a set of cities, is there a set of three cities whose name share at least 3 letters?*

(b) Given a list of classes (specified by (s_c, e_c) where s_c is the start time of class c and e is the end time of class c), is it possible to enroll in at least three classes without having any overlaps in lecture times?

> **Solution:** This is a decision problem. Elements of the language could be encoded using the natural encoding for a pair of integers (which is basically just one integer, then the next followed by another with a spacer in between). The set would be $\{L : \exists c_1, c_2, c_2 \in L\ c_1$, c_2, and c_3 do not overlap$\}$, where c_1, c_2 and c_3 are distinct class intervals.

2. For each of the following decision problems, (i) define a reasonable (finite) alphabet Σ, (ii) give the encoding (over the alphabet) of a representative input object, and (iii) define a language L for the corresponding decision problem. Use the notation $\langle X \rangle$ for the encoding of object X as a string over the alphabet.

   (a) Given a binary string b, is the number of 0s even?

   (b) Does a given non-negative base-10 integer x has 3 as the last digit?

   (c) Is a given array A of n integers sorted?

Solution:

(a) i. $\Sigma = \{0, 1\}$

ii. For a binary string b, write it as a string of bits. For example, if b is the binary string "010", then $\langle b \rangle = \texttt{010}$.

iii. $L = \{\langle b \rangle : b \text{ is a binary string with even number of 0s.}\}$

(b) i $\Sigma = \{\texttt{0}, \ldots, \texttt{9}\}$.

ii For an integer k, write it in base 10 in the usual way, as a string of digits. For example, if k is the integer forty-seven, then $\langle k \rangle = \texttt{47}$

iii A corresponding language would be $L_{EndsWith3} = \{\langle k \rangle : k \bmod 10 = 3\}$. It would also be acceptable to copy the phrasing of the decision problem, i.e., $L_{EndsWith3} = \{\langle k \rangle : k \text{ written in base 10 has 3 as the last digit}\}$

(c) i. $\Sigma = \{\texttt{0}, \ldots, \texttt{9}, \texttt{[}, \texttt{,}, \texttt{]}\}$. (Notice that a comma is one of these characters.)
Note: we can't just use the decimal digits alone if we want to use some special symbols to indicate the start/end of the array and to separate the elements.

ii. For an array A, encode its elements as in the previous part, and list those encodings with appropriate separators. For example, the array A with entries one, two, three, and four would have $\langle A \rangle = \texttt{[1,3,2,4]}$.

iii. The corresponding language is

$$L_{sorted} = \{\langle A \rangle : A \text{ is a sorted array of non-negative integers.}\}.$$

3. The language which represents all positive integers divisible by 4 can be written as:

   (a) $L = \{n \in \mathbb{Z}^+ : n \mod 4 = 0\}$

   (b) $L = \{n, k \in \mathbb{Z} : n/4 = k\}$

   (c) $L = \{n \in \mathbb{Z}^+ : \gcd(n, 4) = 1\}$

   (d) $L = \{n, k \in \mathbb{Z}^+ : n \mod 4 = 2\}$

   (e) None of the above

Solution: (A)

3 DFAs

1. True or False: A DFA can decide the following language:

$$\{x : x \text{ is a binary string of length 4 that starts with two 1's}\}$$

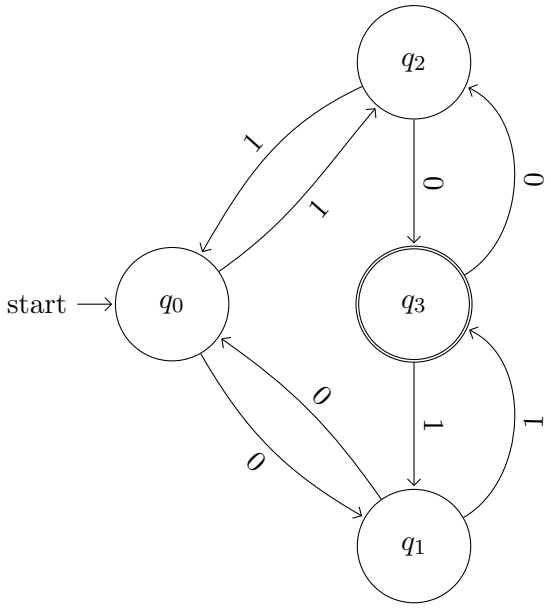
Solution: True. The language can be written as a regular expression: $\{11(0|1)(0|1)\}$

2. True or False: A DFA can decide the following language: $\{x : x \text{ is a palindrome}\}$

Solution: False. DFA's do not have infinite memory (they have a finite number of states) so while they could detect palindromes in a fixed length string, detecting a palindrome in a string with arbitrary size is not possible.

In general, it's a bad sign if the DFA needs to have an entirely different (and growing) subset of states for each increasing input size. This means that the DFA would need to have an infinite number of states (infinite memory) in order to solve a problem at arbitrary length.

3. Consider the following DFA:



(a) Recall that a DFA is formally defined as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$. Define what $Q, \Sigma, \delta, q_0, F$ represent and identify them for the above DFA.

Solution:

Q**:** This is the full set of states, $\{q_0, q_1, q_2, q_3\}$

Σ**:** This the the set of characters the DFA reads, $\{0, 1\}$

δ**:** The is the transition function for the DFA,

state	input	next state
q_0	0	q_1
q_0	1	q_2
q_1	0	q_0
q_1	1	q_3
q_2	0	q_3
q_2	1	q_0
q_3	0	q_2
q_3	1	q_1

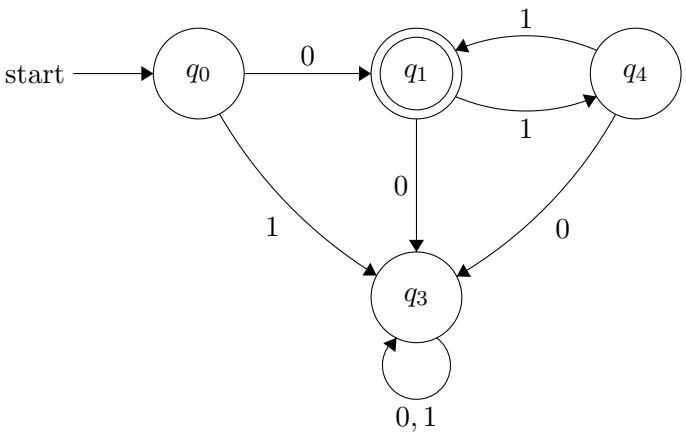
q_0**:** This is the start state; this is sometimes denoted by an arrow labelled start

F**:** This is the *set* of accept states, $\{q_3\}$

(b) What language does the above DFA decide?

Solution: The DFA checks if there are an odd number of 0's and 1's in the bit string. Some inputs that the program accept: $\{010101, 000001, 111110, 11000001\}$, and inputs that the program doesn't accept: $\{0, 1, 100, 1100\}$.

4. Consider the following DFA, where the alphabet is $\Sigma = \{0, 1\}$.



(a) For each of the following strings, determine whether the string is accepted by the DFA:

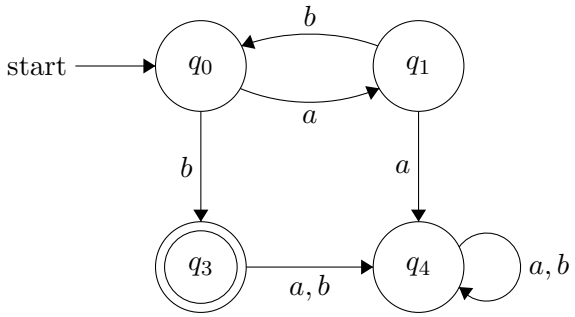
- ε
- 0
- 1
- 010
- 011
- 111

Solution: $0, 011$ are accepted. $\varepsilon, 1, 010, 111$ are rejected.

(b) What is the language that the DFA decides? Provide your answer as a regular expression.

Solution: Regular expression: $0(11)^*$.

5. Consider the following DFA, where the alphabet is $\Sigma = \{a, b\}$.



(a) For each of the following strings, determine whether the string is accepted by the DFA:

- ε
- a
- b
- bb
- aab
- $abbabb$
- $ababb$

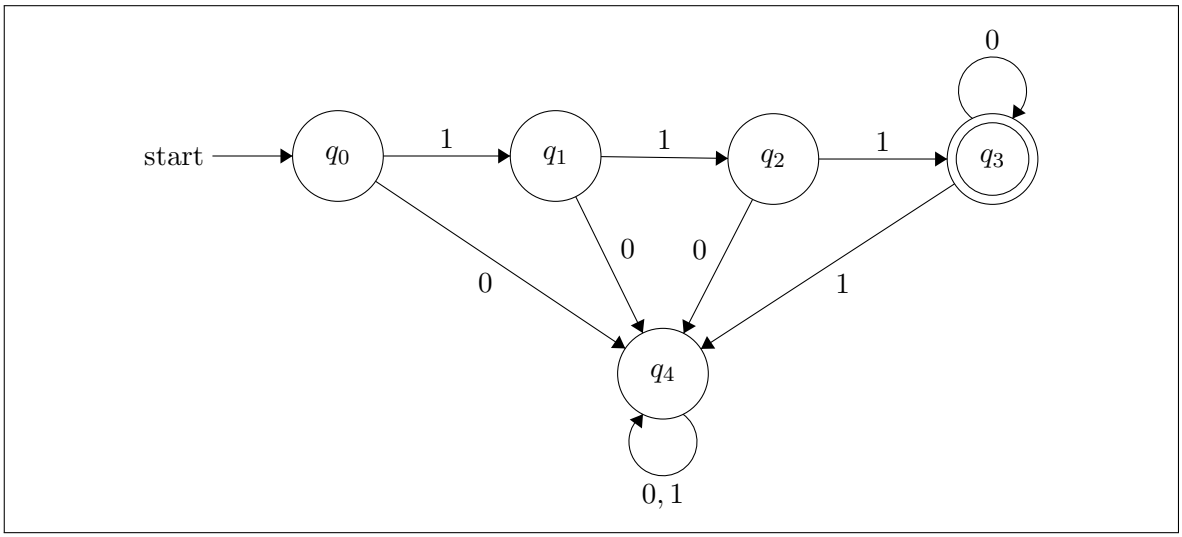
Solution: $b, ababb$ are accepted.

(b) What is the language that the DFA decides? Provide your answer as a regular expression.

Solution: Regular expression: $(ab)^*b$.

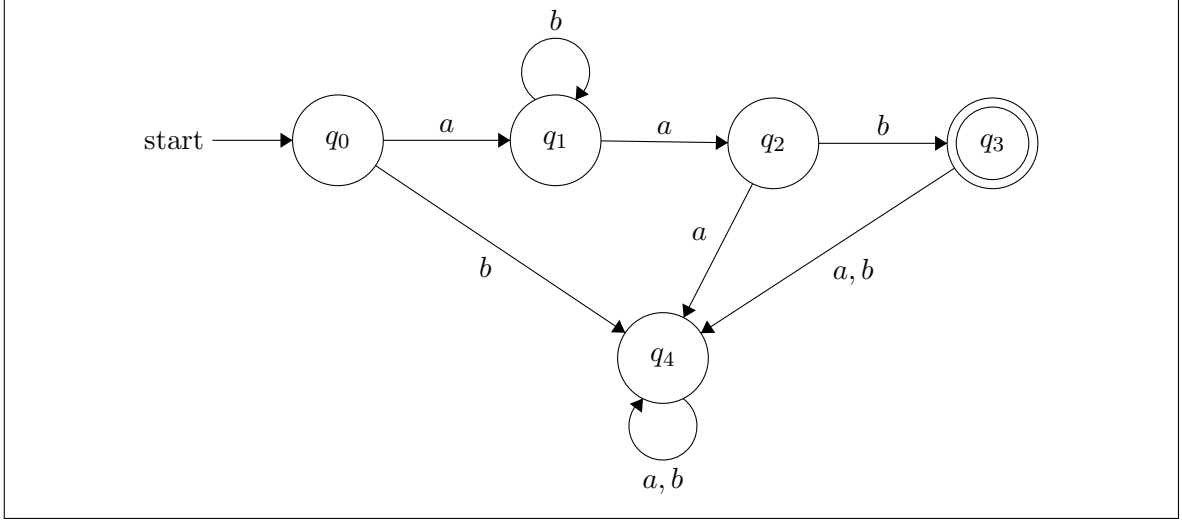
6. Draw a DFA that decides the language 1110^*.

Solution:



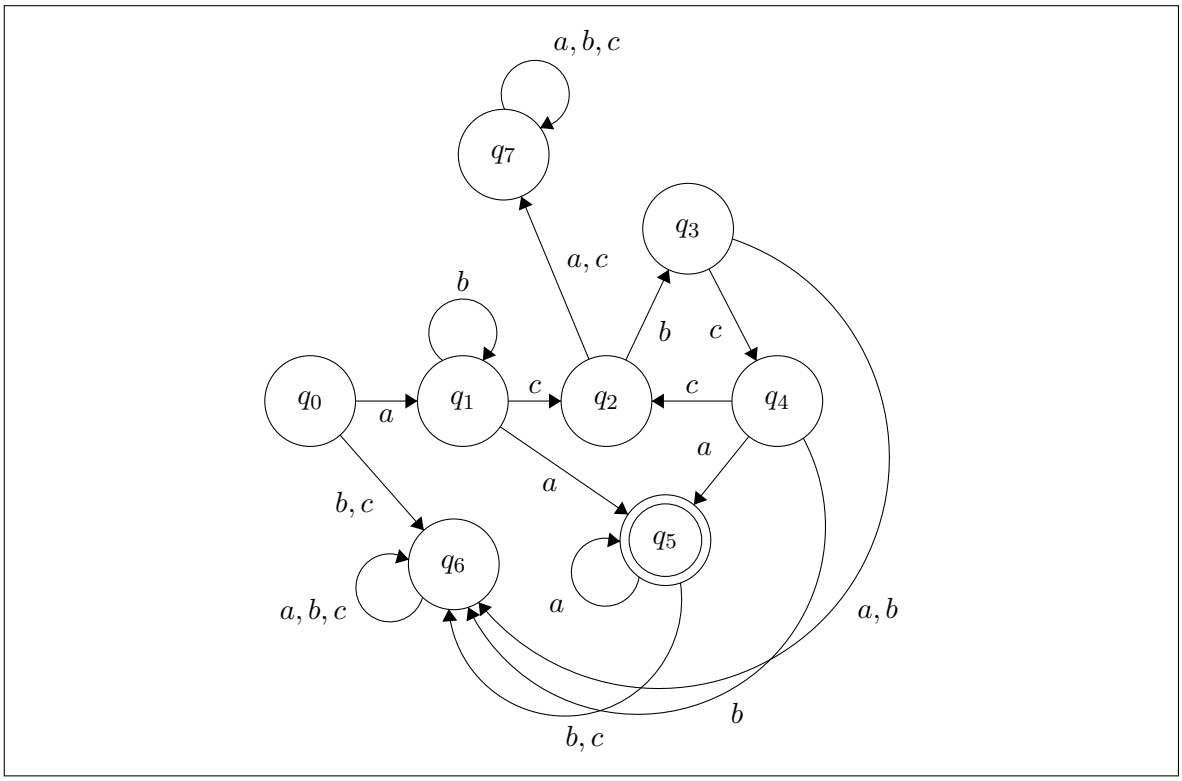
7. Draw a DFA that decides the language ab^*ab.

Solution:



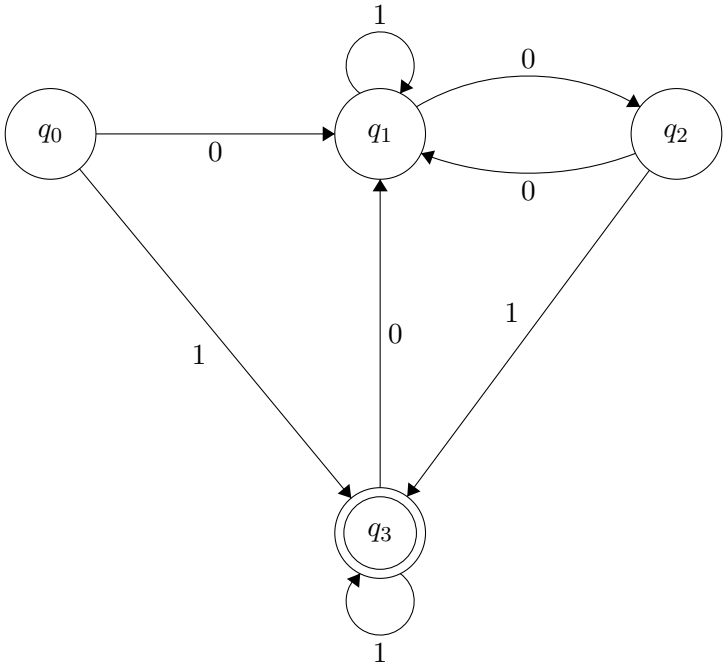
8. Draw a DFA that decides $ab^*(cbc)^*aa^*$

Solution: Note that q_7 is not actually necessary, it was just added so the diagram is neater.



9. $\{x \in \{0,1\}^* \mid x \text{ has an even number of 0s and the last character of } x \text{ is 1}\}$.

Solution: Every time an odd number of 0s has been read, we are in q_1; we can read one more zero to move to q_2, which represents having read an even number of 0s. We then end up in q_3 (the accept state) whenever the latest input was a 1.



10. Draw a DFA that accepts only the set of binary strings of length $3n$, for some integer $n \geq 0$, that contain exactly two 0's for every "block" of 3 characters.

    For example, 100010 is in the language because it has two blocks, 100 and 010, which both have *exactly* two 0's. The empty string, ε, is also (vacuously) in the language. On the other hand, 00100 is not in the language because its length is not a multiple of 3 so it cannot be split into blocks, and 110001 is not in the language because there is only one 0 in its first block, 110.

Solution: