Turing Machine

Turing machine was invented in 1936 by Alan Turing. A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells. It consists of a read/write head which reads the input tape. After reading an input symbol, it may be replaced with another symbol, or it moves one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

A TM can be described as a 7-tuple $(Q, \sum, \Gamma, \delta, q_0, Z_0, F)$
Q : A finite set of states.
Γ : A finite set of tape symbols.
B: A blank symbol.
Σ A finite set of input symbols or alphabets.
δ : A mapping from (transition function) $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$, Where L and R represents left move and right move, respectively.
q_0: The start state or initial state.
F: The set of final states.

Input representation: Turing machine follows the unary representation. I.e., to represent the integer '5', we use '1 1 1 1 1'. Unary representation for zero is not defined.

Turing machine's transition functions: The $\delta(q_0, 1) = (q_1, A, R)$ represents that on reading the input '1' at the state q_0, the control moves to the state q_1 by writing the symbol 'A' in place of '1', and the read/write head moves exactly one cell right. Similarly, $\delta(q_0, 1) = (q_0, 1, L)$ describes that on reading '1' at q_0, the read/write head alone is moves one cell left without changing the state and the tape symbol. The move $\delta(q_0, \$) = (q_f, \$, H)$ says on reading \$ at q_0, halt the Turing machine.

In this scribe, we shall discuss TM from the perspective of 'machine as an acceptor' and 'machine as a computing device'. Note: Finite Automata and Pushdown Automata are acceptors.

1 TM as a recognition device

1. We have seen that $\{a^n b^n c^n\}$ is not a context-free language. Therefore we cannot design PDA recognizing $\{a^n b^n c^n\}$. We shall look into a TM that recognizes L such that $L = \{a^n b^n c^n\}, n \geq 0$

   Approach / Logic:
   Step 1: On reading $'a'$, replace it with X then move right.
   Step 2: On reading $'b'$, replace it with Y then move right.
   Step 3: On reading $'c'$, replace it with Z then change the direction and move left.
   Step 4: Move left until we get first X.
   Step 5: Repeat above steps till all $'a'$, $'b'$ and $'c'$ are marked as X, Y and Z, respectively.
   Step 6: If everything is marked (I.e there are no a, b, c in the input tape) that means string is accepted, otherwise rejected.

   Explanation of TAPE movement:
   Input is "*aaabbbccc*" (scan string from left to right)

   - Change first $'a'$ to $'X'$ and move right until reach $'b'$ (skip all other $a's$).
   - Change first $'b'$ to $'Y'$ and move right until reach $'c'$ (skip all other $b's$).
   - Change first $'c'$ to $'Z'$ and move left until reach $'X'$ (skip all $Z's$, $b's$, $Y's$, $a's$). Now, change the direction and move one cell right.
   - Repeat the above process.

- When TAPE read/write head reaches $'X'$ and next symbol is $'Y'$ that means $a's$ are finished.
- Check for $b's$ and $c's$ by going till \$ (in right) and skip $'Y'$ and $'Z'$.
- If $'b'$ and $'c'$ are not found that means string is accepted.
- If $'b'$ or $'c'$ are present in the input tape then reject, or after changing $'a'$ to $'X'$, if no $'b'$ or $'c'$ are in the tape, then reject.

TAPE movement for string "*aaabbbccc*"

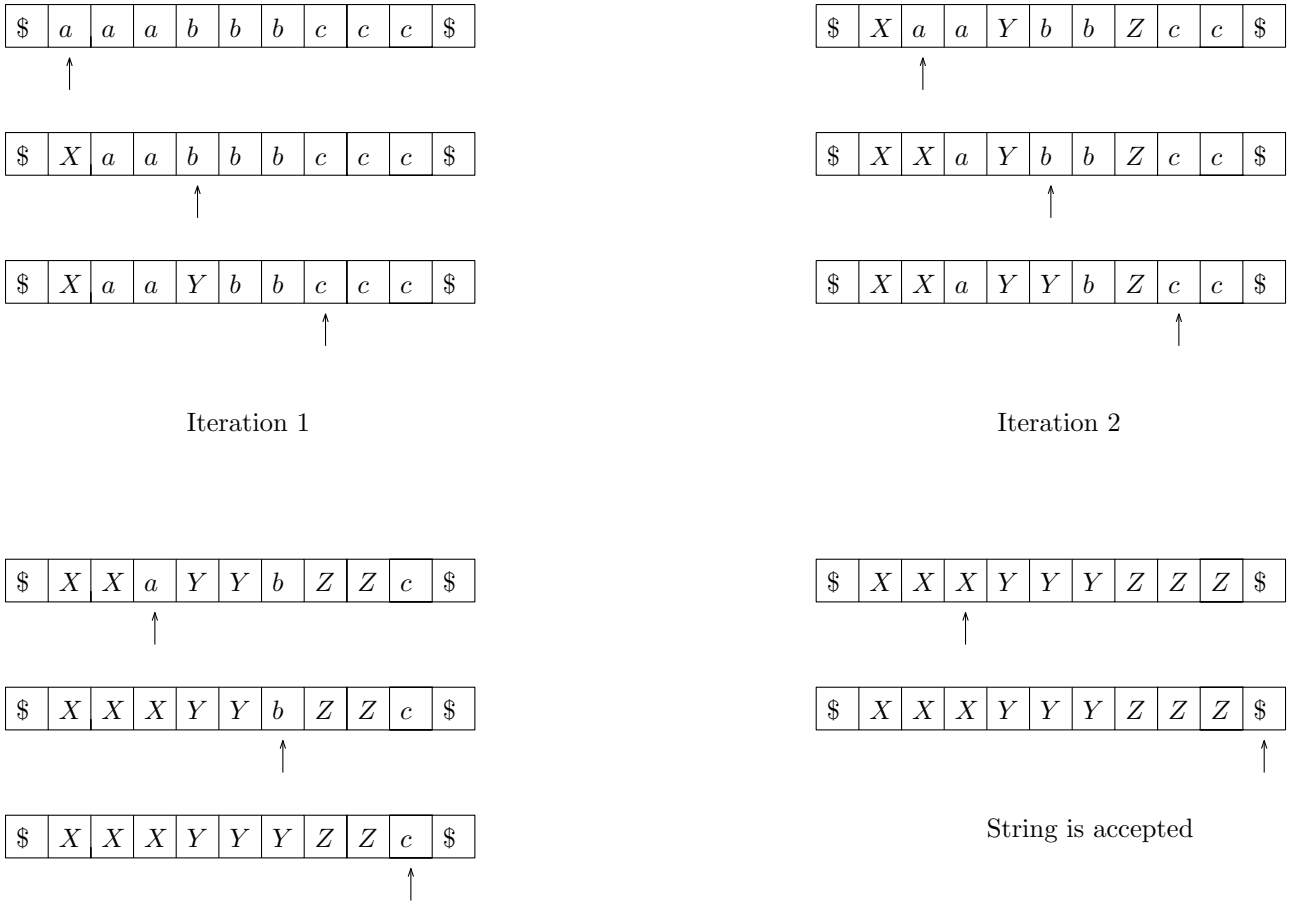

Figure 1: *TAPE movement for string "aaabbbccc"*

State Transition Functions:

Transition for the first iteration:
$\delta(q_0, a) = (q_1, X, R)$
$\delta(q_1, a) = (q_1, a, R)$
$\delta(q_1, b) = (q_2, Y, R)$
$\delta(q_2, b) = (q_2, b, R)$
$\delta(q_2, c) = (q_3, Z, L)$
$\delta(q_3, b) = (q_3, b, L)$
$\delta(q_3, Y) = (q_3, Y, L)$
$\delta(q_3, a) = (q_3, a, L)$

$\delta(q_3, A) = (q_0, A, R)$

Leftover transitions:
$\delta(q_2, Z) = (q_2, Y, R)$
$\delta(q_1, Y) = (q_1, Y, R)$
$\delta(q_3, Z) = (q_3, Z, L)$

Other valid strings:
$\delta(q_0, Y) = (q_4, Y, R)$
$\delta(q_4, Y) = (q_4, Y, R)$
$\delta(q_4, Z) = (q_4, Z, R)$
$\delta(q_0, \$) = (q_f, \$, HALT)$

Other invalid strings:
$\delta(q_4, b) = (q_r, b, HALT)$
$\delta(q_4, c) = (q_r, c, HALT)$
$\delta(q_1, Z) = (q_r, Z, HALT)$
$\delta(q_2, \$) = (q_r, \$, HALT)$
Cases like *aacc*, *bbcc*, *aabb*, *a*, *b*, *c* are handled below
$\delta(q_0, c) = (q_r, c, HALT)$
$\delta(q_0, b) = (q_r, b, HALT)$
$\delta(q_1, c) = (q_r, c, HALT)$

Remark: In the above example, the TM behaves like an acceptor. Accept all strings $x \in L$ and stop the machine at q_f, and reject all strings $x \notin L$ and stop the machine at q_r. We next discuss TM as a computing device. Note: So far we have designed a TM that performs basic arithmetic operations like addition, subtraction, multiplication, division and comparison. An interesting question is that can TM solves any computational problem. In this section, we show that any computational problem can be solved by TM. We shall now design a TM that solves other computational problems using above designed TM as a black box.

2. We shall look into four different ways on how a Turing machine accepts $L = \{ww \mid w \in \{a, b\}^*\}$.

   Logic 1: For each symbol that is read, replace tape symbol that appears after two \$ by 1. Perform division by 2 operation on those 1's. The number of 1's left represents the middle point of two strings. Replace the \$ that appears after input as 0. Swap 0 and an input symbol on reading each 1 that appears after \$. After the swaps, the tape consists of $w0w$. Compare the each alphabet of first string and alphabet of second string.

   $$abab\$\$$$
   $$Abab\$\$1$$
   $$ABab\$\$11$$
   $$ABAb\$\$111$$
   $$ABAB\$\$1111$$

   Perform division by 2:

   $$ABAB\$\$11\$\$$$
   $$ABAB0\$11\$\$$$
   $$ABAB0\$01\$\$$$
   $$ABA0B\$01\$\$$$
   $$ABA0B\$00\$\$$$
   $$AB0AB\$00\$\$$$

Logic 2: Place a 0 at right after the input symbols. On reading each input symbols from left swap 0 and one symbol on the right. When there is no more swaps, the position of 0 represents the middle position of two strings.

Logic 3: Place a \$ before and after input alphabets. At each iteration, swap an alphabet and \$ from left, and swap an alphabet and \$ from right. When two \$'s are adjacent, halt the machine.

Logic 4: Observe that if $x = ww$ then $x' = w^R w$. For each consecutive substring of a given string we find its reverse. If all the substrings are not of the form $w^R w$, then the give string is not of the form ww. Suppose at least one substring is of the form form $w^R w$, then the give string is of the form ww. We invoke (i) TM for reversing w (ii) TM for finding substring of a string. We know that there exists a PDA that checks the strings of the form $ww^R(w^R w)$.

3. We shall look into three different ways on how a Turing machine accepts $L = \{www \mid w \in \{a, b\}^*\}$.

   Logic 1: Similar to logic 1 of ww, perform division by 3, thereby we can identify w.

   Logic 2: Place $\alpha,\ \beta,\ \gamma$ after the input alphabets. Perform shift operation such that α is shifted three places to its left, β is shifted by two places to its left and γ is shifted by one place to its left.

   $$abaabaaba$$

   Add $\alpha,\ \beta,\ \gamma$:

   $$abaabaaba\alpha\beta\gamma$$
   $$abaaba\alpha aba\beta\gamma$$
   $$abaaba\alpha a\beta ba\gamma$$
   $$abaaba\alpha a\beta b\gamma a$$
   $$aba\alpha abaa\beta b\gamma a$$
   $$aba\alpha ab\beta aab\gamma a$$
   $$aba\alpha ab\beta aa\gamma ba$$
   $$\alpha abaab\beta aa\gamma ba$$
   $$\alpha aba\beta aba\gamma aba$$

   Once α hits the left end of the tape, β is between the first w and second w, and γ is between second w and third w, if the string is valid. If there any mismatch in strings during comparison, then reject the string and halt the machine. Else accept the string and halt the machine.

   Logic 3: For every third symbol that is being read, replace blank after \$ as 1. The number of 1's after \$ represents length of w.

Can we generalize such that Turing machine accepts $L = \{ww \dots w \mid w \in \{a, b\}^*\}$, $|ww \dots w| = k$.

2 Turing machine as a computing device

Turing's original view of his machine was as a computer of integer-valued functions. In his scheme, integers were represented in unary, as blocks of a single character, and the machine computed by changing the lengths of the blocks or by constructing new blocks elsewhere on the tape. We shall show some examples how a Turing machine can compute arithmetic operations.

2.1 Turing machine (TM) performs addition:

The Turing machine M starts with a tape consisting of 1^m01^n surrounded by blanks. We shall assume that in all our cases the left end is fixed. M halts with 1^{m+n} on its tape, surrounded by blanks or tape symbols.

1. We shall see two different ways of performing addition of two numbers, $M(a+b)$:

   Logic 1: M replaces 0 by 1 and moves right. Once it hits \$, moves left and replaces 1 with \$.
   The transition function for this approach is:

   $$\delta(q_0, 1) = (q_0, 1, R)$$

   $$\delta(q_0, 0) = (q_1, 1, R)$$

   $$\delta(q_1, 1) = (q_1, 1, R)$$

   $$\delta(q_1, \$) = (q_2, \$), L)$$

   $$\delta(q_2, 1) = (q_f, \$, \text{Halt})$$

   Logic 2: M repeatedly finds leftmost remaining 1, replaces it by a blank and replaces blank appearing after 0 by 1. It continues until no 1's are left before 0.

   Suppose that if we want to perform addition of n numbers.

2. **Instance:** $n, a_1, \ldots, a_n$, n is fixed.
   Question: Σa_i
   Logic: M replaces 0 by 1 and moves right after changing the state from q_i to q_{i+1}. When it reaches \$, the state of M is q_{n-1}. Once it reaches \$, it moves left and changes 1 to 0 until it reaches the state $q_{n-1+(n-1)} = q_{2n-2}$. After reaching q_{2n-2} state, halt the machine.
   Note that in this case before designing the value of n is known.

3. **Instance:** $a_1, \ldots, a_n$
   Question: Σa_i
   Logic: After reading each 0 replace it with 1, the corresponding 1 is placed after \$. Then the number of 1's are flipped as 0's before \$ are exactly the number of 1's after \$. Halt the machine once all 1's after \$ are exhausted.

   1101011101\$

   1111011101\$1

   1111011101\$1

   1111111101\$11

   1111111111\$111

   1111111110\$011

   1111111100\$001

   1111111000\$000

   Note that in this case the value of n is dynamic and not known during the design.

2.2 TM performs subtraction:

$M(a-b),\ a \geq b$; M starts with a tape consisting of 1^m01^n surrounded by tape symbols. When M halts, 1^{m-n} is on the tape surrounded by tape symbols.
Initially,M is in state q_0 moves right on reading 1's, and on reading a 0, it changes its state to q_1 and moves right. At state q_1, if it reads 1, it replaces 1 with B and changes its state to q_2. The state q_2 moves left, while moving left if B is read, it moves left. After reading a 0 at state q_2, it moves left. Once it hits a 1 after reading a 0, it changes state to q_3, replaces 1 with B and moves right. At state q_3, if reads B or 0, it moves right by staying in the same state. At q_3 on reading 1, M changes its state from q_3 to q_2 and replaces 1 to B. When q_3 on reading \$, it goes to q_f and goes to halt.

$$\delta(q_0, 1) = (q_0, 1, R)$$

$$\delta(q_0, 0) = (q_1, 0, R)$$
$$\delta(q_1, 1) = (q_2, B, L)$$
$$\delta(q_2, B) = (q_2, B, L)$$
$$\delta(q_2, 1) = (q_3, B, L)$$
$$\delta(q_2, 0) = (q_2, 0, L)$$
$$\delta(q_3, B) = (q_3, B, R)$$
$$\delta(q_3, 0) = (q_3, 0, R)$$
$$\delta(q_3, 1) = (q_2, B, L)$$
$$\delta(q_3, \$) = (q_f, \$, \text{Halt})$$

Similarly, a Turing machine can perform subtraction when $a < b$.

$$\delta(q_0, *) = (q_0, *, R)$$

$$\delta(q_0, 1) = (q_0, 1, R)$$
$$\delta(q_0, 0) = (q_1, 0, R)$$
$$\delta(q_1, 1) = (q_2, B, L)$$
$$\delta(q_2, B) = (q_2, B, L)$$
$$\delta(q_2, 1) = (q_3, B, L)$$
$$\delta(q_2, 0) = (q_2, 0, L)$$
$$\delta(q_3, 1) = (q_2, B, L)$$
$$\delta(q_2, *) = (q_4, *, R)$$
$$\delta(q_4, B) = (q_4, B, R)$$
$$\delta(q_4, 0) = (q_4, 0, R)$$
$$\delta(q_4, 1) = (q_5, 1, L)$$
$$\delta(q_4, \$) = (q_5, \$, L)$$
$$\delta(q_5, B) = (q_f, 1, \text{Halt})$$

2.3 TM performs sorting

Initially the input tape contains numbers represented in unary, and each number is separated by 0. In order to perform sorting, each element needs to be compared. Let us assume that $a_1, \ldots, a_n$ be the numbers represented in unary. Comparison operation can be simulated using $a - b$ as a subroutine. Initially, $a_1,\ a_2$ are compared. Suppose that $a_2 > a_1$, then swapping is done. Similarly, $a_i,\ a_{i+1}$ is compared and swapped, if required. Hence, during each iteration exactly one element is in its right place. When there is no more swapping, then the Turing machine halts.

Do you remember the famous swap operation without using third variable!

$$\begin{aligned} a &= a + b \\ b &= a - b \\ a &= a - b \end{aligned} \qquad \begin{aligned} a &= a \oplus b \\ b &= a \oplus b \\ a &= a \oplus b \end{aligned}$$

2.4 TM for various well-defined problems

1. Input: Integer n, Output: Check n is prime or not.

   The approach is to check whether 2 is factor of n, 3 is a factor of n and so on up to $n - 1$. We shall now explain how this division by 2, division by 3 is performed in the Turing machine.

   Division by 2: To check whether a number n is divisible by 2, place n 1's in the input tape. Start moving the read/write head one cell right and for every second '1' in the tape, write $'A'$ in the tape in place of '1'. At the end of this procedure, if we see no 1's in the tape after the last $'A'$, then the number is divisible by 2 and 2 is factor. Otherwise, 2 is not a factor.

   Division by 3: Similar to division by 2, for three 1's, we mark 'A' in the input tape in place of the third '1', and at the end if there are no 1's, then n is divisible by 3. Otherwise, 3 is not a factor. We continue the above procedure for values 4,5, until $n - 1$. If the division procedure returns 'NO' for all division routines, then the number is a prime.

2. Input: Integer n, Output: $n!$.
   Factorial of n is computed as a sequence of multiplications; $n \times (n-1)\times, \ldots, \times 1$. We know that multiplication can be performed using repeated additions, and Turing machine can perform addition. Therefore, factorial of a given number can be computed.

3. Fibonacci series / sequence. Input: Integer n, Output: $0, 1, 1, 2, 3, 5, 8, 13, \ldots, n^{th}$ term.
   We can call addition routines repeatedly to get the Fibonacci series.

4. Input: Integer n, Output: 2^n
   This involves multiplication. We multiply 2, n times. This can be done using addition routines. We can extend this as follows; Input: Integer x, n, Output: x^n. Suppose x and n are negative fractions. Negative fractions can be represented in unary. For example, -1.5 can be represented as $1B1Z1111\$$, B, Z is a delimiter.

5. Input: Integer n, Output: $\lfloor \log_2 n \rfloor$.
   We write $y = \log_2 x$
   $x = 2^y$
   $2^k \implies x$
   $2^{k-1} \implies \frac{x}{2}$
   $2^{k-2} \implies \frac{x}{4}$
   $2^{k-3} \implies \frac{x}{8}$
   $\vdots$
   $2^0 \implies 1$

The number of iterations represents the value of $\lfloor \log_2 n \rfloor$.
Logic: For every second '1', write '1' after $ symbol. Repeat the process until all 1's are exhausted. Now consider the 1's that are present after $ as an input and repeat the above step.

6. Input: Integer n, Output: $a\%b$ (a *mod* b)
   TM for division is used as a black box to compute a % b.

7. Input: Integer x, Output: $\sin x$
   We can use Taylor series to compute $\sin x$. Taylor series involves division, factorial, addition/ subtraction. All these operations are possible in TM.

8. Input: Integer x, n, Output: $\sqrt[n]{x}$
   We write $\sqrt[n]{x} = y$
   $x = y^n$. This can be done similar to problem 6.

9. How to check a given function is bijective or not.
   Consider a function (1,2)(2,3)(3,1)
   1 0 11 00 11 0 111 00 1111 0 1
   If we encounter 10 after any 00 then it is not a function (I.e we have other pairs starting with 1). Using similar approach, we can find whether a function is 1-1, onto.

10. Input: Integer n, Output: all prime factors of n
    We call division routine to find the factors of n. Let the factors be $F = \{x_1, x_2, \ldots\}$. Call the Prime_check routine for each element in F to check whether it is prime or not. This shows that there exists a Turing machine that outputs all prime factors of a given number. But there is no other computing machine that lists all prime factors of a given number.

11. Finding Integration.
    Integration is to find the area of irregular objects. We can plot this in a graph and count the number of squares to find the area

12. Input: Integer n, Output: $(-1)^{\log_2(n)}$.
    We can rewrite this as $y = (-1)^{\log_2(n)}$
    $\implies \log y = \log_2 7 \cdot \log_2 -1$,
    Note that $\log_2 -1$ is not defined. This implies that there does not exist an algorithm that solves this problem. Hence, Turing machine does not exist.
    Remark: If a problem is not well defined then there does not exist a computing device that solves it.

13. Input: Integer n, Output: (i) Determinant of A (ii) A^{-1} (iii) Check A is square or not (iv) Check A is skew.
    All these problems can be solved by invoking addition routine as a black.

14. Input: Set S, Output: Is S satisfies some property?
    For example, Input: set A, Property: is A sorted. For any two elements a_i, a_{i+1}, if $a_i \leq a_{i+1}$ then we conclude that A is sorted. Comparison can be done by invoking subtraction routine.

    Remark: Any formula based computation uses only basic arithmetic operations. We know that Turing machine can perform basic arithmetic operations. Therefore, any formula based computation can be done using Turing machine.

Note: TM can perform / solve
(i) Any computation that can be performed by a digital circuit.
(ii) Any formula based computation
(iii) Problems that are solvable / implementable using C/Python?PHP/Verilog

For any mathematical model, in particular TM, input representation is crucial. Set of all solvable problems can be represented in unary form. We shall see how the inputs are represented.

- Bit: 0/1
- Word: eg. $5 \implies 11111$
- Characters: Char can be represented using ASCII value. ASCII values are integers. If the ASCII value is x then use x number of $1's$.
- Set/ Arrays / Vectors / Ordered set :
  $\{w_1, w_2, \ldots, w_n\} \implies w_1\ 0\ w_2\ 0\ \ldots 0\ w_n \implies 0111\ 0\ 111\ 0 \ldots 0\ 11$
- Set of sets / collection of arrays / Vectors $\implies$ Matrix
  Matrices are used to denote images, relations, functions anc cross products. $w_{11}\ 0\ w_{12}\ 0\ \ldots 0\ w_{13}\ 00\ w_{21}\ 0\ w_{22}\ 0 \ldots$.
- Sequence of matrices are videos: $M_1\ 000\ M_2\ 000\ M3\ 000 \ldots, 000\ M_n$

Let us consider a classical $P - Q$ problem (I.e) TM accepts another program Q as an input and print YES if Q runs into an infinite loop, otherwise NO.
Assume that the input program is C-program. Note that the number of C-program is countably infinite. We map Σ^* to natural numbers as follows $\Sigma^* \leftrightarrow \mathbf{N}$.

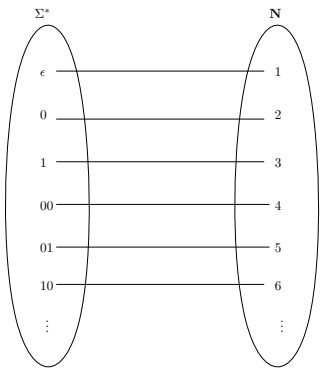

Figure 2: $\Sigma^* \leftrightarrow \mathbf{N}$.

Now we shall revisit the question.
Program P (Input Q)
{
print YES if Q runs into an infinite loop
NO otherwise
}
Note that a programs runs into an infinite loop if and only if case 1, case 2, ... , case k. Here k is countable infinite. Since the number of inputs to this problem P is countably infinite and Q runs into an infinite loop for a countably infinite number of cases, the decision to print yes in P is for a countably infinite number of cases. The decision to print YES or NO cannot be determined, hence P is unsolvable. Therefore, no P exist.

3 Can we solve 3-SAT using TM

The Satisfiability Problem (SAT) is the first problem that was shown to be NP-Complete. The 3-Satisfiablity problem (3-SAT), each clause has at most three literals. 3-SAT is also known to be NP-complete.

Input: The boolean variable $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ such that $C_i = (x_j \wedge x_k \wedge x_l)$ and $x_j,\ x_k,\ x_l$ can be either true or false literal.
Question: Does there exists a truth values $x_1, \ldots, x_n$ such that the boolean expression (circuit) is evaluated to true.
We can observe that the problem is equivalent to listing all binary strings of length n. Each binary string can be verified that whether it evaluates to true or not. If the circuit is satisfiable, then there exists at least one assignment which evaluates to true.
Listing of all binary strings of length n using Turing machine:
We know that binary strings are elements over $\{0, 1\}^*$. Let X_1 be the tape symbol that corresponds to 0 and X_2 be the tape symbol that corresponds to 1. The Turing machine M lists all possible binary strings as follows: Initially all n positions in a n-length string is X_1. Copy the string as such leaving exactly one blank symbol after the initial string. Replace the first symbol of the copied string as X_2. Continue the copy operation n times and at each time i^{th} symbol is changed as X_2. Copy the initial n-length string and replace the first two symbols as X_2, then second and third symbols as X_3, and so on. There will be 2^n strings listed.
Evaluation of the Boolean expression (circuit):
For each string listed, M checks whether the assignment evaluates the Boolean expression to true or not.

Thus, 3-SAT can be solved by using Turing machine.