Questions 3, 4, and 5 consider a very simple computer called the Plum. Plum has a finite set $Q$ of states, two registers $R$ and $S$, and a read-only input tape containing symbols in $\Sigma$. Each register can hold a nonnegative integer, which can be arbitrarily large. Plum operates with a specific positive integer $m$ (which is fixed once and for all when the machine is built), and operates as follows. Let \( \text{rem}(A, m) \) be the remainder when $A$ is divided by $m$, and \( \text{quo}(A, m) \) be the quotient when $A$ is divided by $m$.

Each transition reads the next character as long as there is unread input. Once all input is read, a special symbol, $\text{EOF}$, is the result of the read. Each transition examines (1) the current state, (2) \( \text{rem}(R, m) \), and (3) the symbol from $\Sigma \cup \{\text{EOF}\}$ from the read. Based on this, it determines a new state. It also determines an adder $\alpha$ with $0 \leq \alpha < m$, and does one of the following:

- **R**: replaces $R$ by \( \text{quo}(R, m) \) and $S$ by $m \times S + \alpha$, or
- **S**: replaces $R$ by $m \times (\text{quo}(R, m) + \alpha) + \text{rem}(S, m)$ and $S$ by \( \text{quo}(S, m) \).
- **T**: replaces $R$ by $m \times \text{quo}(R, m) + \alpha$ and leaves $S$ unchanged.

Thus the transition function is a mapping

$$Q \times \{0, \ldots, m-1\} \times (\Sigma \cup \{\text{EOF}\}) \mapsto Q \times \{0, \ldots, m-1\} \times \{R, S, T\}$$

Initially $R = S = 0$. Plum accepts when it enters a designated accept state. You may need to make further assumptions to answer the questions, so state them clearly if you do.

The main thing we notice is that the Plum has no tape and the TM has no registers. Otherwise they are basically the same. How can we represent a configuration of a TM on the Plum when we have no tape? A TM configuration looks like $w_1 w_2 \cdots w_{i-1} w_i \cdots w_\ell$. We are going to remember the state $q$ in a state of the Plum. We still need to know the tape contents and where the tape head is. So we are going to encode $w_1 w_2 \cdots w_{i-1} w_i$ in one register, and $w_{i+1} \cdots w_n$ in the other. These are strings over an alphabet on $\gamma = |\Gamma|$ letters. But we can view them as integers written in base $\gamma$. Indeed we use any one-to-one mapping $\psi$ from $\Gamma$ to $\{0, \ldots, \gamma-1\}$ for which $\psi(B) = 0$ (B is the Turing machine blank). It will be convenient to encode $w_{i+1} \cdots w_n$ as $\sum_{j=0}^{\ell-i-1} \psi(w_{i+1+j}) \gamma^j$.

Given this integer, you can get $w_{i+1}, \ldots, w_\ell$ by repeatedly dividing by $\gamma$ and taking the remainder! This will be how I view the register $S$.

I want the tape characters nearest the tape head to the the ones with the lowest powers of $\gamma$ in the representation, so I encode $w_1 w_2 \cdots w_{i-1} w_i$ as $\sum_{j=0}^{i-1} \psi(w_{i-j}) \gamma^j$ in register $R$. Notice that when I do this, if I take $m = \gamma$, \( \text{rem}(R, m) \) is $\psi(x)$ where $x$ is the symbol that the tape head of the TM is pointing at.

**The Questions:**

1. Draw a state diagram for a Turing machine that decides the language \( \{1^n \# b : b \text{ is the binary representation of } n\} \). (For example, 11111#101 is in the language, but 111#110 is not.)

   Give a brief explanation of your design. The idea is to perform unary to binary conversion.
Show that for every Turing-recognizable language $L$

Draw a state diagram for a Turing machine that decides the language of palindromes in $L$. Transitions in $\delta_{\text{plum}}$ are:

- $\delta_{\text{plum}}(s, 0, \sigma) = (s, \psi(\sigma), S)$ for every $\sigma \in \Sigma$;
- $\delta_{\text{plum}}(s, 0, \text{EOF}) = (r, 0, R)$;
- $\delta_{\text{plum}}(r, x, \text{EOF}) = (r, x, R)$ for every $0 < x < m$;
- $\delta_{\text{plum}}(r, 0, \text{EOF}) = (q_0, 0, S)$;
- if $\delta_m(q, a) = (q', b, R)$ then $\delta_{\text{plum}}(q, \psi(a), \text{EOF}) = (q', \psi(b), S)$; and
- if $\delta_m(q, a) = (q', b, L)$ then $\delta_{\text{plum}}(q, \psi(a), \text{EOF}) = (q', \psi(b), R)$.

For all other cases $\delta_{\text{plum}}$ goes to $q_{\text{reject}}$. 

(2) Draw a state diagram for a Turing machine that decides the language of palindromes in $\{a, b\}^*$ that have the same number of $a$s and $b$s. Give a brief explanation of your design. The idea is to perform the computation in three phases in order to check both requirements first that the string is a palindrome, then that the number of $a$s and $b$s is the same as well as handling some problems that arise with a 1-way infinite tape.

- In phase 1, we shift all characters of the input string one position to the right by keeping track of the last character read in the states, and we insert a special beginning of input symbol (\#) in the leftmost cell of the tape. If the first character is a blank, the tape is empty, and we immediately accept, since the empty string is a palindrome that has the same number of $a$s and $b$s.
- In phase two, we sweep back and forth over the input string, matching characters in the first half of the string to the second half of the string by marking matched characters from the outside edges in. (We only accept even palindromes since $|w| = \#a + \#b$s and $\#a = n = \#b$s, so $|w| = 2n$, an even number. So we do not need to worry about palindromes such as aba.) We replace as with $0$s and $b$s with $1$s to indicate marked characters. If we cannot match a symbol from the first half to its mirror in the second half, we reject. Then, we spin back to the beginning of the string, denoted by $\#$, replacing $0$s with $a$s and $1$s with $b$s on our way to regain the original string.
- In phase 3, we check to see if the number of $a$s is the same as the number of $b$s. If we read an a, we move right until we find a matching b, and vice versa. If at any time we encounter a blank before finding a match, then the number of $a$s and $b$s is not equal, so we reject. If we match all as to bs, we accept.

For this state diagram, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, 0, 1, \#, \_\}$ (where $\_\$ is blank.) The state diagram is attached as a separate sheet.

(3) Show that for every Turing-recognizable language $L$, there is a Plum computer that recognizes $L$. (Hint: Can you turn a Turing machine into an equivalent Plum?) Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be the TM that recognizes $L$. Suppose that $m = |\Gamma|$ and let $\psi$ be the mapping described above. Let’s build a Plum. The states of the Plum are $Q \cup \{s, r\}$ and the new start state is $s$. Transitions in $\delta_{\text{plum}}$ are:

- $\delta_{\text{plum}}(s, 0, \sigma) = (s, \psi(\sigma), S)$ for every $\sigma \in \Sigma$;
- $\delta_{\text{plum}}(s, 0, \text{EOF}) = (r, 0, R)$;
- $\delta_{\text{plum}}(r, x, \text{EOF}) = (r, x, R)$ for every $0 < x < m$;
- $\delta_{\text{plum}}(r, 0, \text{EOF}) = (q_0, 0, S)$;
- if $\delta_m(q, a) = (q', b, R)$ then $\delta_{\text{plum}}(q, \psi(a), \text{EOF}) = (q', \psi(b), S)$; and
- if $\delta_m(q, a) = (q', b, L)$ then $\delta_{\text{plum}}(q, \psi(a), \text{EOF}) = (q', \psi(b), R)$.

For all other cases $\delta_{\text{plum}}$ goes to $q_{\text{reject}}$. 

2
In state $s$, the Plum encodes the input into register $R$ and then goes to state $r$. In state $r$ the Plum moves everything into register $S$ (essentially moving the TM tape head back to the beginning of the tape). Then it places just the first symbol in register $R$ and enters the TM states. The TM moving left is doing an $R$ operation on the Plum; moving right it is doing an $S$. Because Plum simulates the TM step for step, always properly representing its configuration, Plum accepts exactly when the TM does.

There is one small technicality. If the TM moves left off the left hand edge of the tape, the Plum may keep computing. So we should first modify the TM so that it never hangs in this way.

(4) Show that if language $L$ is recognized by a Plum computer, it is Turing-recognizable. (Hint: Can you turn a Plum into an equivalent Turing machine?) This is quite similar to question 3. When the Plum starts, it first reads all of its input and modifies the registers. Then it does more computation after the input is read. We build a 2-tape deterministic TM, with one tape for the Plum input and one for its two registers. Tape 2 is a 2-way infinite tape (infinite in both directions). The tape alphabet on tape 2 is $\{0, \ldots, m - 1\}$, and we take 0 to be the blank. We permit the tape head on tape 2 to move left (L), right (R), or stay in the same place (S).

The TM has the same states as the Plum. Whenever $\delta_{plum}(q, x, \sigma) = (q', y, D)$ for $D \in \{R, S, T\}$, there is a Turing machine transition that applies
- when the current state is $q$;
- when the tape 1 head is pointing at character $\sigma$ and $\sigma \in \Sigma$ or is pointing at a blank and $\sigma = EOF$;
- when the tape 2 head is pointing at character $x$ and so does the following
  - changes state to $q'$;
  - moves the tape 1 head right one cell;
  - changes the $x$ under the tape 2 head to a $y$; and
  - moves the tape 2 head left if $D = R$, moves it right if $D = S$, and leaves it in the same position if $D = T$.

This TM simulates the Plum step by step. The only minor issues that remain are that we have used two tapes, one of which is 2-way infinite, and permitted one tape head to stay in the same place. But we could produce a standard TM that simulates this using our multitape results for the first two, and introducing new states that move right and then left when the tape head is to stay in the same place (much like we did with PDAs).

(5) (not to be graded) Somebody sold me a bad Plum! I can change the transition function, but it only works with $m = 1$. What languages can it recognize? When $m = 1$, the remainders computed are always 0. So the registers always contain the number 0. As a result they are of no use, and what I am left with is almost a DFA. The only difference is that it can keep computing when it hits EOF. This does not change its power, so the languages that I can recognize are exactly the regular languages.