1. (a)

$L = \{\langle x \rangle : \sqrt[3]{x} \in \mathbb{Z} \text{ and } x \in \mathbb{Z}\}$

(b)

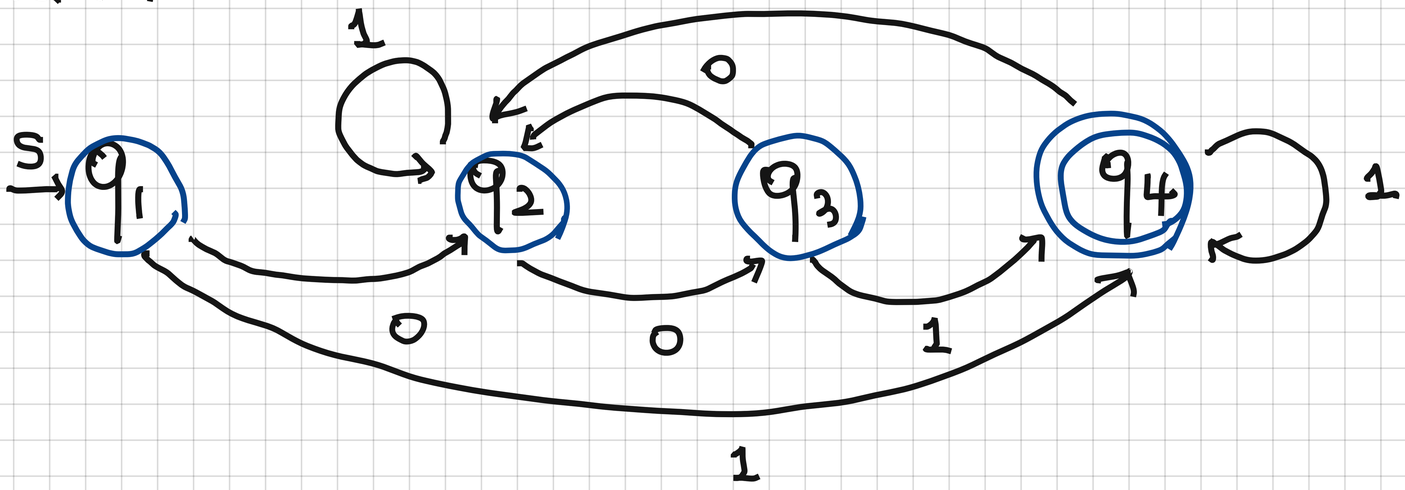
$L = \{\langle m \rangle : m \bmod 2 \equiv 1 \text{ and } m \in \mathbb{Z}\}$

(c)

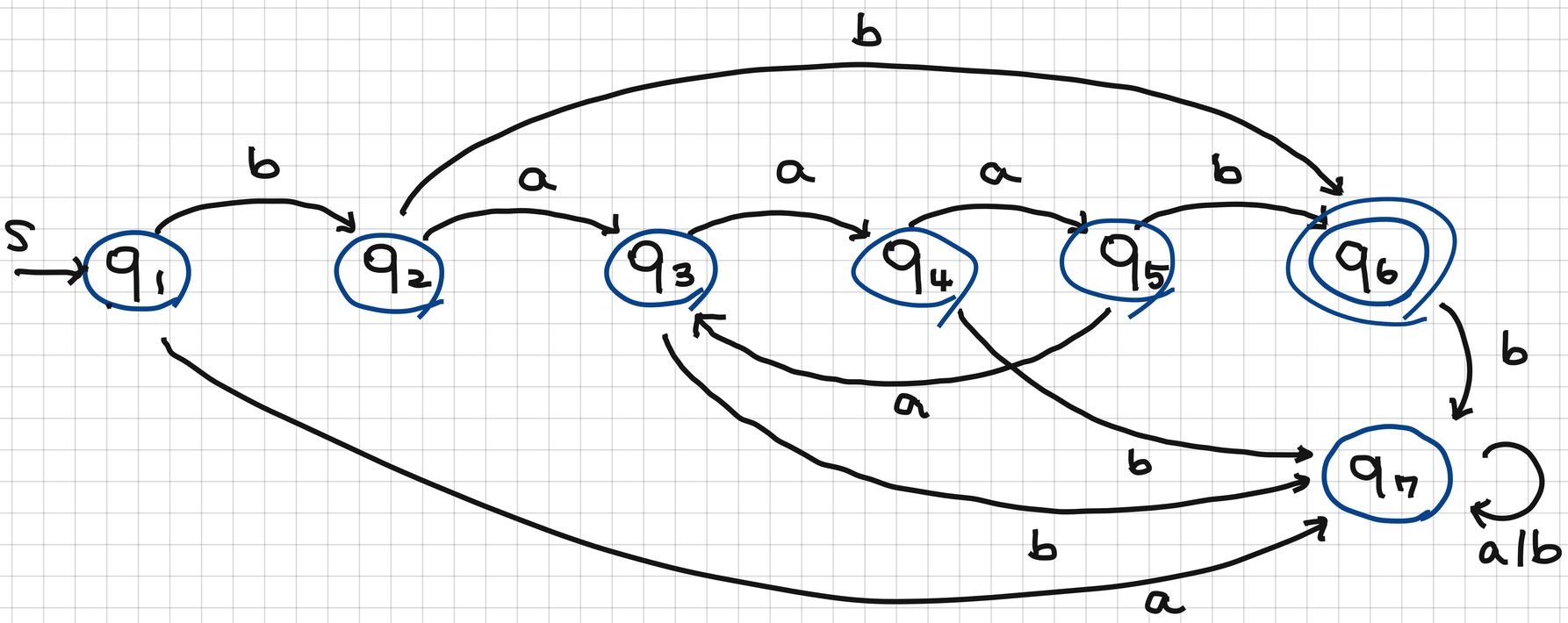
$L = \{\langle p, q \rangle : \text{Euclid}(p, q) \neq 1\}$

If GCD is not a prime number, GCD will always have factor 2 or 3.
Since 2 and 3 are prime numbers, which means the p.q share
common prime factors

2. (a)



(b) $b(aaa)^*b$



3. Some valid : aabb , aaaabb , aabbbb
The shortest language to reach q4 is "aabb"

it should not start with b since q2 can't reach q4
it can only have even numbers of (aa)s between the first aa and the last bb.

∴ aa (aa)*(bb)* bb

4.

The key reason for this is that the DFA has a finite number of states,
and it processes the input string symbol by symbol, transitioning from
one state to another based solely on the current symbol and its current state.
It does not have the capability to remember past states or positions.

Now, let's assume that there exists a DFA that can solve this problem.
This DFA would have a finite set of states, and the input alphabet would consist of the instructions N, W, S, and E.

After executing an instruction string, the robot can end up at the origin in various ways.
For example, if the instruction string is "NWSE," the robot would end up back at the origin.
However, there are multiple paths to achieving this. It could go NW, then SE, or it could go N, then W, then S, then E.
Each of these paths would require the DFA to be in a different state to recognize the final position correctly.
Since there are infinitely many possible instruction strings, the DFA would need infinitely many states to account
for all possible paths, which is not possible for a DFA (by definition, a DFA has a finite number of states).

5. (a)

The greedy algorithm for the fractional knapsack problem works as follows:

1. Calculate the value-to-weight ratio (v_i / w_i) for each item i in the set of available items.
2. Sort the items in decreasing order of their value-to-weight ratio.
3. Initialize the total value of the knapsack (let's call it totalValue)
   to 0 and initialize the remaining capacity of the knapsack (let's call it remainingCapacity)
   to the maximum capacity of the knapsack.
4. Iterate through the sorted list of items, starting with the item with the highest value-to-weight ratio.
5. For each item i, if its weight (w_i) is less than or equal to the remaining capacity, add the entire item
   to the knapsack by adding its value (v_i) to totalValue and subtracting its weight from remainingCapacity.
6. If the weight of item i is greater than the remaining capacity, add a fraction of it to the knapsack.
   Specifically, add remainingCapacity / w_i times the value and weight of item i to totalValue and remainingCapacity,
   respectively.
7. Continue this process until you either add all of an item or fill the knapsack to its capacity.
8. Finally, return totalValue as the maximum achievable value for the given knapsack capacity.

The greedy algorithm for the fractional knapsack problem provides the optimal value because it prioritizes
adding items with the highest value-to-weight ratio first.
This approach ensures that the knapsack is filled with items that contribute the most value per unit of weight.

(b)

Suppose there exists a solution to the 0-1 knapsack problem (the binary knapsack problem)
that yields a higher total value than the solution to the fractional knapsack problem for the same weights,
values, and knapsack capacity.

Let OPT_binary be the optimal solution to the 0-1 knapsack problem,
and let OPT_fractional be the optimal solution to the fractional knapsack problem.

Assumption: OPT_binary > OPT_fractional

Now, let's consider OPT_binary.
It represents the highest total value achievable in the 0-1 knapsack problem,
which means it includes a set of items selected from the given set of items that maximizes
the total value without exceeding the knapsack's capacity.

We can convert this solution OPT_binary into a solution for the fractional knapsack problem as follows:

For each item in OPT_binary, include the entire item in the fractional knapsack (i.e., set t = 1).
This is because in the 0-1 knapsack problem, you either include an item entirely or exclude it.
Calculate the total value of the items added to the fractional knapsack using the same weights and values as in OPT_binary.
This converted solution for the fractional knapsack has the same total value as OPT_binary, which means:

Total value of converted solution = OPT_binary

However, we assumed that OPT_binary > OPT_fractional. But now, we have shown that the converted solution for
the fractional knapsack (with the same total value) is greater than OPT_fractional. This contradicts our assumption.

Therefore, our assumption that OPT_binary > OPT_fractional must be incorrect.
In other words, the optimal value for the fractional knapsack is at least as large as that for the original 0-1 knapsack problem
for the same weights, values, and knapsack capacity.

(c)

Consider the following set of items with their values (v_i), weights (w_i), and the knapsack capacity (W):

Item 1: v_1 = 10, w_1 = 5
Item 2: v_2 = 6, w_2 = 4
Item 3: v_3 = 12, w_3 = 8
Knapsack capacity: W = 10

Now, let's apply the greedy approach described in part (a):

Calculate the value-to-weight ratios for each item:

Item 1: v_1 / w_1 = 10 / 5 = 2
Item 2: v_2 / w_2 = 6 / 4 = 1.5
Item 3: v_3 / w_3 = 12 / 8 = 1.5
Sort the items in decreasing order of their value-to-weight ratios:

Sorted order: Item 1, Item 2, Item 3
Initialize the total value of the knapsack (totalValue) to 0 and the remaining capacity (remainingCapacity)
to the maximum capacity of the knapsack (W = 10).

Start adding items based on their value-to-weight ratios:

Add Item 1: totalValue = 0 + 10 = 10, remainingCapacity = 10 - 5 = 5
Add Item 2: totalValue = 10 + 6 = 16, remainingCapacity = 5 - 4 = 1
At this point, we can see that there is not enough capacity to add Item 3 (w_3 = 8), so the algorithm stops.

The greedy approach selects Item 1 and Item 2, resulting in a total value of 16. However,
the optimal solution to the original 0-1 knapsack problem is to select Item 1 and Item 3, which yields a total value of 22.