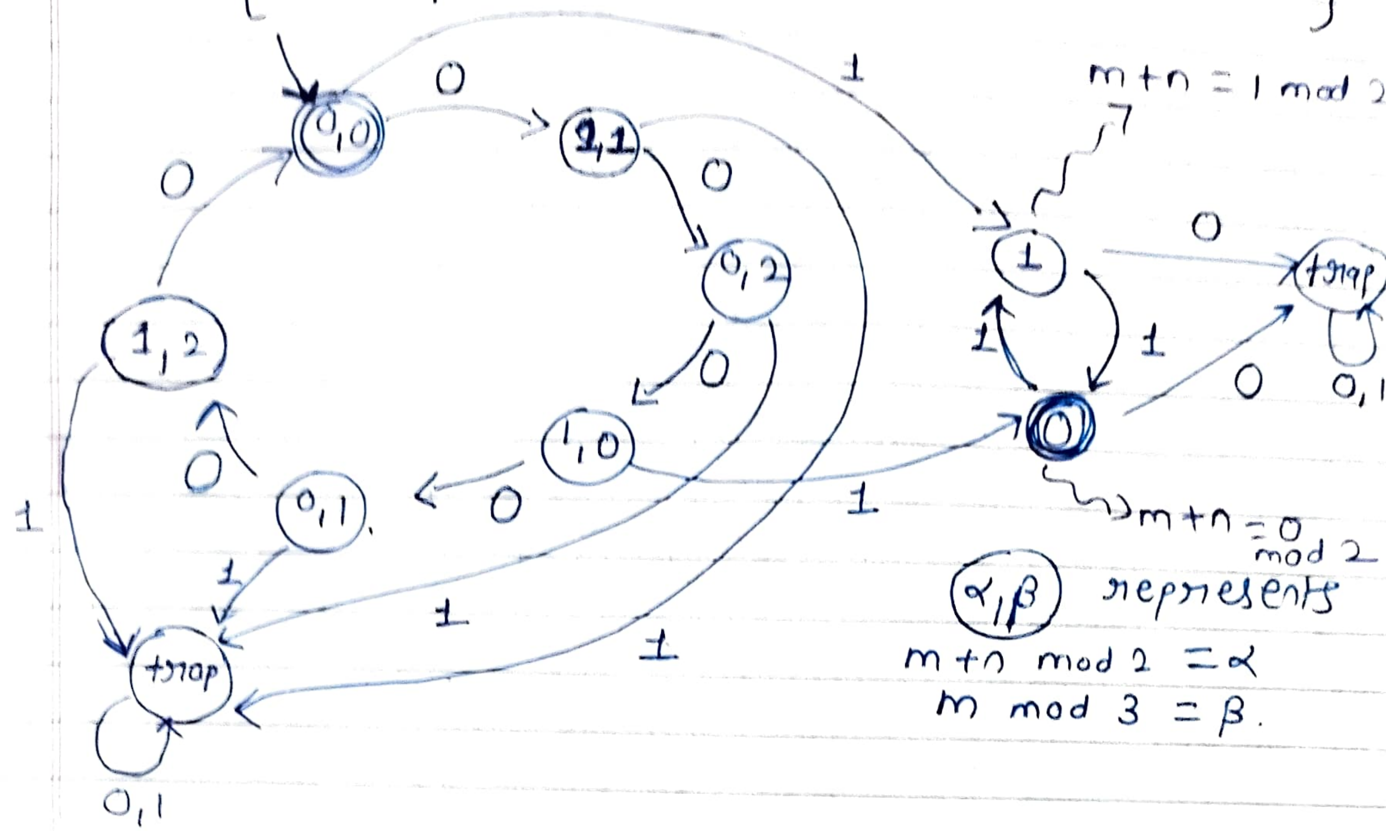
1 (i) $\{0^m 1^n \mid m+n \equiv 0 \bmod 2 \quad m \equiv 0 \bmod 3\}$



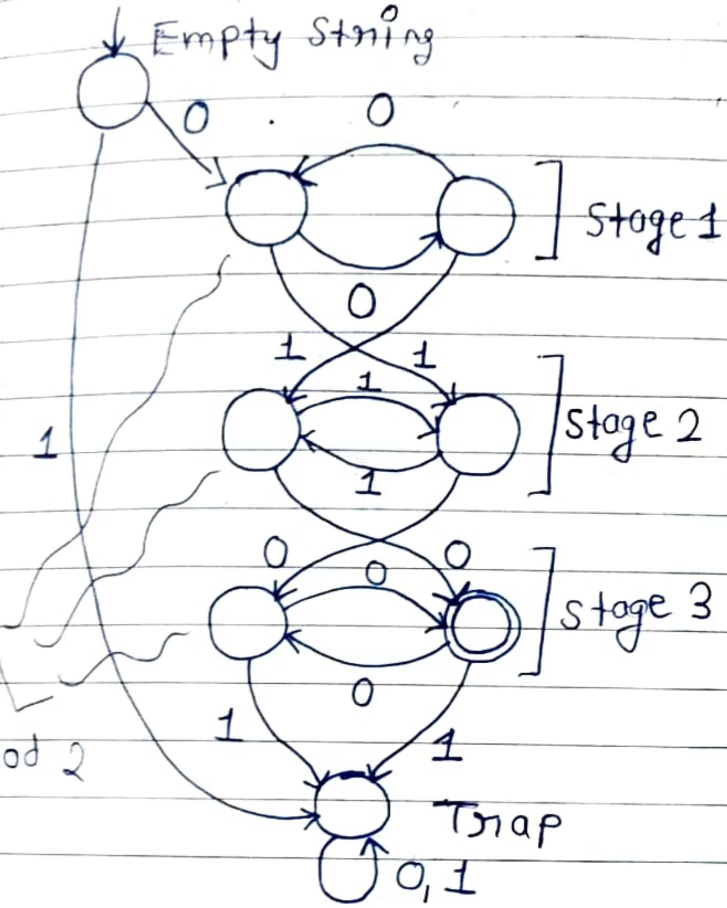
idea

(ii) when first '1' comes, check that remainder of m w.r.t is 3 is 0. If after '1' we get '0', go to trap.

problem set - 2

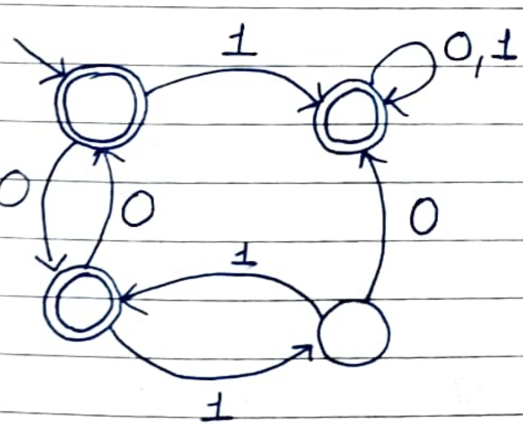
(a)

(ii)



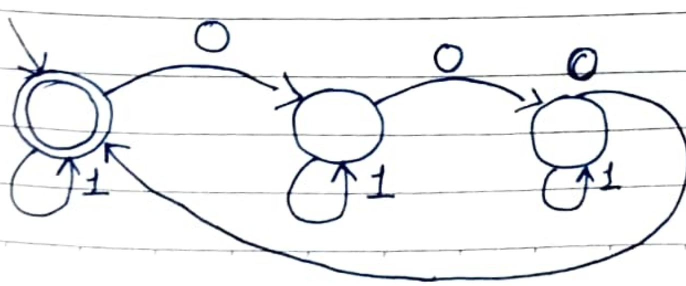
Intuition:-
We keep record of the stage in which the string is. Stages are :① only zeros as of now ② zeros followed by ones ③ zeros followed by ones, followed by zeros. At the same time we also store the modulo w.r.t 2.

(iii)



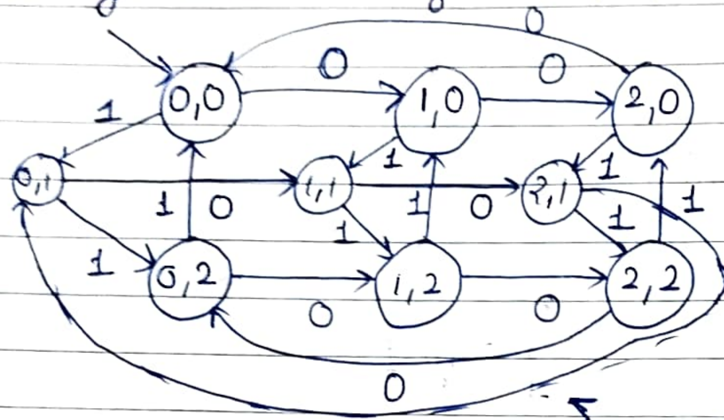
Idea:-
If the number of 1's is 0 mod 2 in the whole string, it's accepted. for odd i.e 1's are 1 mod 2, we have to ensure a prefix with even 0's and odd 1's.

(iv)



Idea : Just keep record of $n_0(w) \bmod 3$

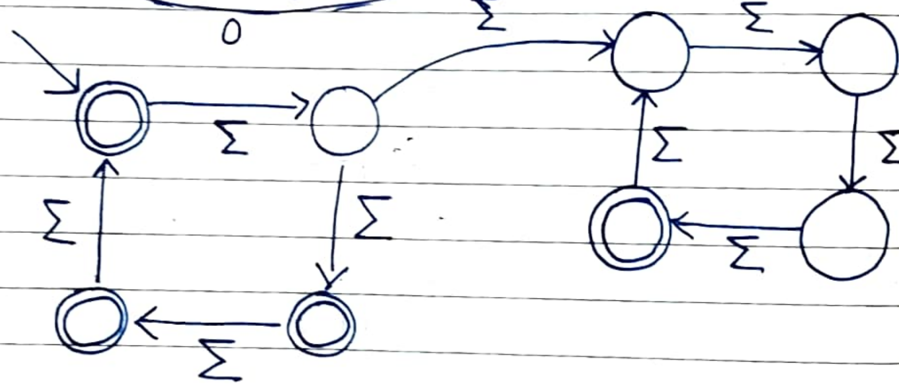
for general $\alpha n_0(w) + \beta n_1(w) = k \mod 3$



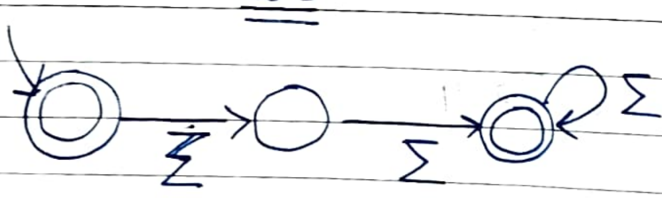
choose those accepting states for which the above is true.

(b)

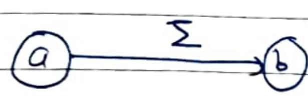
(i)



<u>or</u>



Here



means

for all letters there is outgoing edge from a to b.

Idea:-

only string "a" where a is any letter from Σ are not acceptable. or simply strings of length 1.

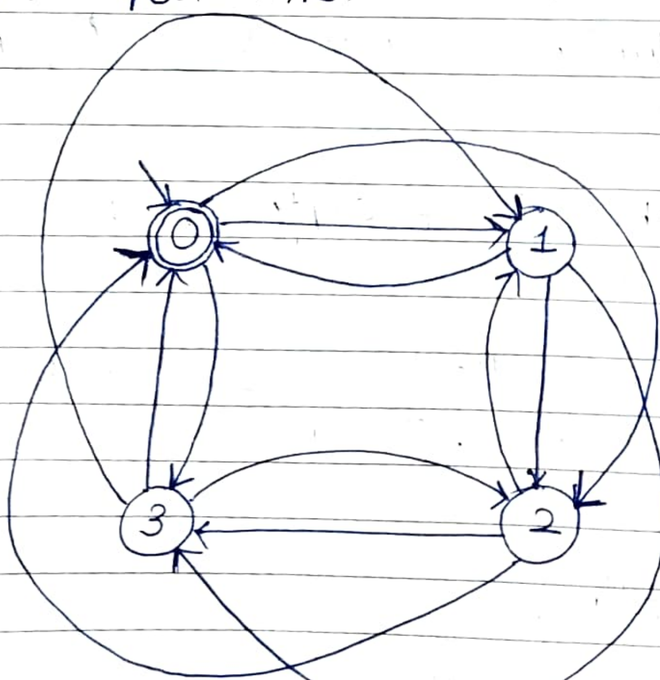
Alternative to $b(i)$

~~3.~~ we can think of this as a problem where we want to find if there is a to divide total number of characters into three parts such that

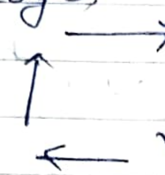
$$\underbrace{|u|}_{} + 2\underbrace{|v|}_{} + 3\underbrace{|x|}_{} = 0 \bmod 4.$$

cardinality of respective parts

~~At~~ for every character we have a choice to keep it in any of the three parts. Here is a NFA for this.

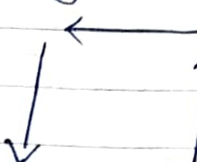


edges



represents decision of putting the next character in box 1

edges

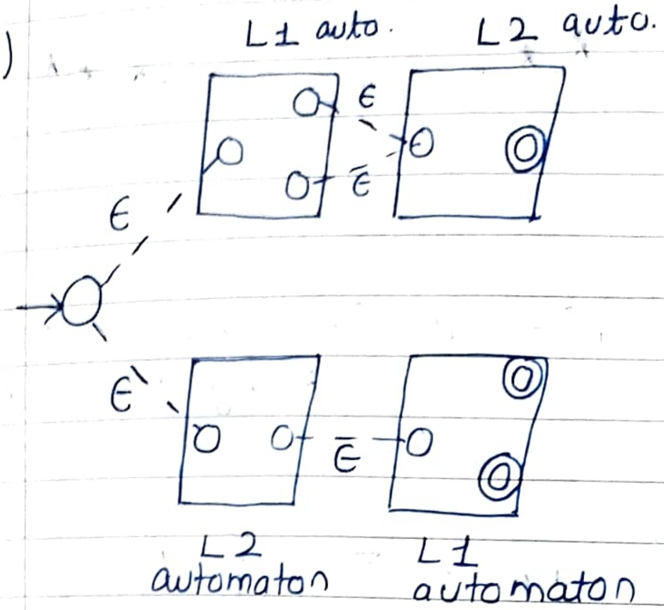


represents decision of putting in box 3.

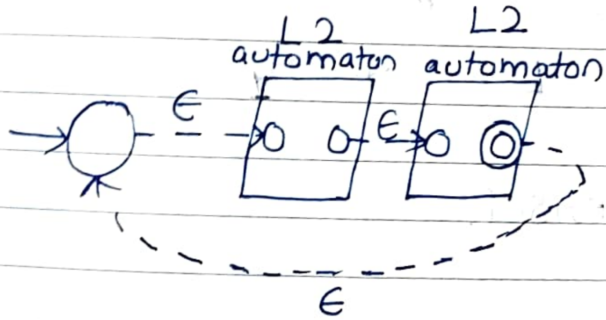
edges

represents decision of putting in box 2.

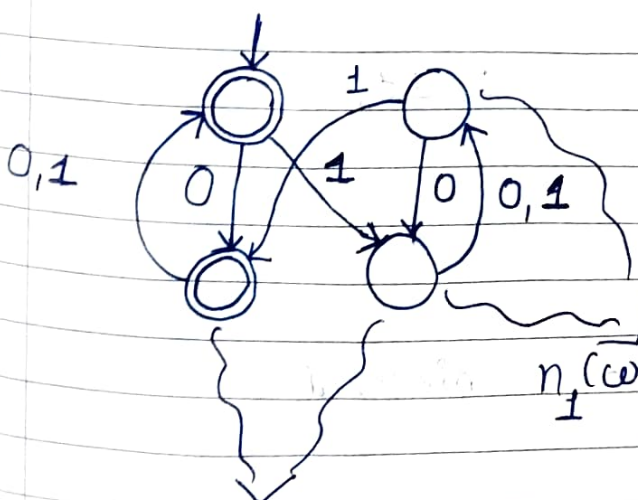
(ii)



(iii)



(iv)



idea :- we have to ignore every alternate character. We also need to keep record of $n_1(w') \bmod 2$.

$n_1(w') \bmod 2 = 1$

this state mean next character has to be avoided.

2 (a)

$r_1 = 1^*(1+0)^*0^*$ $\quad r_2 = (0^*1^*)^*$

$\underline{\underline{r_1}}$

$(1+0)^*$ is universal (i.e. any finite string belongs to its language)

why?

$1+0 \Rightarrow \{0,1\}$

So

$(1+0)^* \Rightarrow \Sigma^*$ where $\Sigma = \{0,1\}$

r_1 is $1^*(1+0)^*0^*$

consider any string (finite) α

So

$\epsilon \cdot \alpha \cdot \epsilon \in$ Lang of r_1

Hence

r_1 is universal.

$\underline{\underline{r_2}}$

$L(0^*) = \{\epsilon, 0, 00, ---\}$

$L(1^*) = \{\epsilon, 1, 11, ---\}$

$0 \in L(0^*1^*)$

$1 \in L(0^*1^*)$

So

$L((0^*1^*)^*)$ is also universal.

$L(r_1) = L(r_2)$

(b) $r_1 = ((0+1)^*0)^*0$

$r_2 = (0+1)^*0^*0$

$10 = 1\cdot\epsilon\cdot 0 \in r_2$

$r_1 \; (\underbrace{(0+1)^*0}{})^*$

explicitly added/concatenated a zero. so all strings which belongs to $L((0+1)^*0)$ has zero at the end.

$\alpha \in L(((0+1)^*0)^*)$ either is empty string or ends with 0.

So
$\alpha \in L(r_1)$ either is "0" or ends with atleast two zeros. [mean ....00]

$10 \notin L(r_1)$

$L(r_1) \neq L(r_2)$

(c) $r_1 = (0+1)^*01(0+1)^*$

$r_2 = 1^*(0+1)^*0(0+1)^*1$

Every string which belongs to $L(r_2)$ ends with '1'.

$010 = \epsilon\cdot 0\cdot 1\cdot 0 \in L(r_1)$

$0\cdot 1\cdot 0 \notin L(r_2)$

$L(r_1) \neq L(r_2)$

③

We have to know a good upper bound on the number of distinct regular languages that can be represented by automaton of size(states) $\leq n$.
$\Sigma = \{0,1\}$
(DFA)

We will use a scheme to generate such DFA's. Consider a n states which are labelled. for now assume $n \geq 3$.

$S_1, S_2, - - - -, S_n$.

Now, for every state we have to decide the arriving state for each letter.

$$\Sigma = \{0, 1\}$$

~~So~~ for each letter, n options. So $n \times n = n^2$ ways of making outgoing arrows, from state s_i. Hence we will get

$$\underbrace{n^2 \cdot n^2 \cdot n^2 \cdots}_{n} = n^{2n}$$ different DFA's which could be possibly equivalent

But DFA is complete when we mention the start and accepting states. Consider the scheme below:

for every ~~DFA~~ graph formed in this way keep S_1 as start state. Now for each such graph make $(n-1 + n-1)$ copies.

for first $n-1$ copies mark $\{S_n\}, \{S_n, S_{n-1}\}, \{S_n, S_{n-1}, S_{n-2}\}, - - - - \{S_n..$ as accepting states.

for next $n-1$ copies
mark
$\{S_1\}, \{S_1, S_n\}, \{S_1, S_n, S_{n-1}\}, \text{------} \{S_1, S_n, \cdots S_2\}$
as accepting.

thus we have

$n^{2n}(2)(n-1)$ DFAs, need not be language wise different, but each DFA represent some language, giving us a multi set of languages.

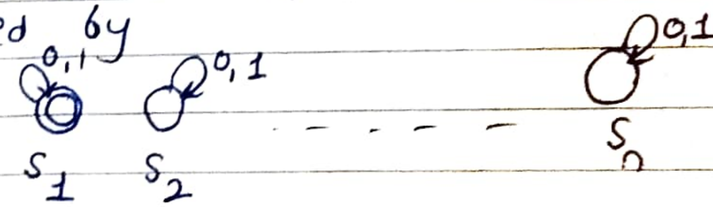
To show that $2(n-1)(n^{2n})$ is a valid upper bound, it is sufficient to show that language represented by any DFA with states $\leq n$ is included in our collection.

If $< n$ then add  to graph.

Take any DFA, states $\leq n$. If there are x ~~starting s~~ accepting states, label them $S_n, S_{n-1}, \text{---}, S_{n-(x-1)}$ in any order. Always label the starting state as S_1. Rest all states can be labelled in any way. It is not difficult to see that one such DFA can also be formed as per our scheme.

x are after excluding the accepting state which is also start state.

If $x == n-1$, then the DFA represent $(1+0)^*$ which is covered by



Thus

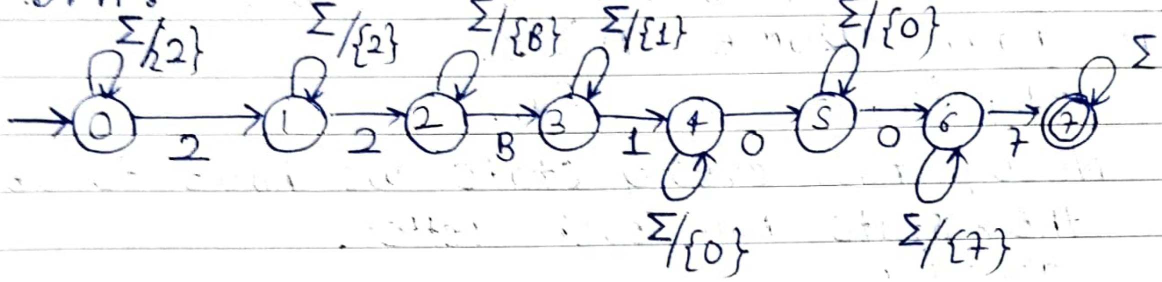
$2(n-1)n^{2n}$ is valid upper bound for $n \geq 3$. for $n=1,2$, reader can get the exact count.

4.

(a) Rollno. - 22B1007 (As an example)

(b) Basically, we have to come up with DFA which accepts any string in which "22B1007" is present as subsequence.

DFA :-



This DFA is basically the implementation of the greedy algorithm for checking if a string is present as a subsequence in another string.

Keep looking for first character, if found then start looking for next character. Repeat this till you find the last character.

Suppose it is present, then we can show that the string will reach the accepting state.

- - - - 2 - - - 2 - - - B - - - - 1 - - - 0 - - - 0 - - - 7 - - -

by this moment, the DFA must be in state ≥ 3

After this it must be in ≥ 4

Inverse,

Suppose it could reach the accepting state which means we have used the edges (that's only way to go to it). Hence 22B1007 in this sequence has appeared in the string w.