In my gradebook, I've written

52 + C

68 + B

84 + A

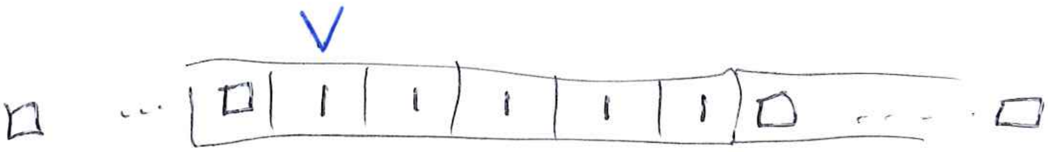
for this exam.

~~Doin~~ Making Turing machines do stuff:

We can interpret a TM as giving output - whatever is left on the tape at the end - instead of just saying "Yes" or "No".

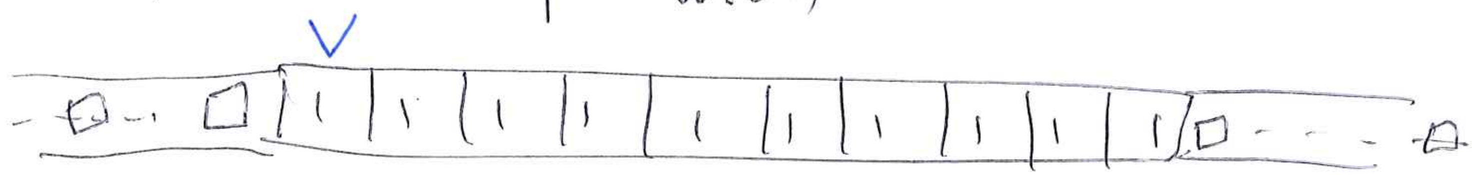
For programming TMs, it's easiest to deal with numbers in "unary" - the # is the number of 1's.

Write a TM that, given a number n as input (in unary), outputs ~~2xn.~~ $2n$. (i.e. if we start w/



$(\lambda, q_0, \underbrace{11111}_{5})$

we end up with



$(\lambda, \text{some state}, \underbrace{1111111111}_{10})$

Start state q_0:

$$(q_0, 1) \longrightarrow (q_1, x, R)$$

$$(q_1, 1) \longrightarrow (q_1, 1, R)$$

$$(q_1, \square) \longrightarrow (q_2, \cancel{x}\,y, L)$$

$$(q_1, \cancel{x}\,y) \longrightarrow (q_1, \cancel{x}\,y, R)$$

$$(q_2, \cancel{x}\,y) \longrightarrow (q_2, \cancel{x}\,y, L)$$

$$(q_2, 1) \longrightarrow (q_2, 1, L)$$

q_1 - marked a 1 - need to copy it

q_2 - copying done - move left to find next 1 to copy

Problem:

$\square \; 1 \; 1 \; 1 \; 1 \; 1 \; \square$

$\square \; X \; 1 \; 1 \; 1 \; 1 \; x$

$\square \; x \; x \; 1 \; 1 \; 1 \; x \; x \; \square$

when moving left we need to treat these x's differently

add y's instead.

$$(q_2, x) \longrightarrow (q_0, x, R)$$

How do we tell we are "done", and how to finish from there?

$\square \cdots \square \; 1 \; 1 \; 1 \; 1 \; 1 \; \square \cdots \square$

$\square \; x \; 1 \; 1 \; 1 \; 1 \; y \; \square \cdots \square$

$\square \; x \; x \; 1 \; 1 \; 1 \; y \; y \; \square \cdots \square$

$\vdots$

q_0

$\square \; x \; x \; x \; x \; x \; y \; y \; y \; y \; y \; \square$

clean-up when see a ~~q_0 on~~ y in state q_0

q_3 - move to end in clean-up phase.

$$(q_0, y) \longrightarrow (q_3, y, R)$$
$$(q_3, y) \longrightarrow (q_3, y, R)$$
$$(q_3, \square) \longrightarrow (q_4, \square, L)$$

$(q_4, y) \longrightarrow (q_4, 1, L)$

$(q_4, x) \longrightarrow (q_4, 1, L)$

$(q_4, \square) \longrightarrow (q_f, \square, R)$

q_4 - move left, converting everything to 1's, in cleanup.

What about multiplying 2 numbers?

Input: $\underbrace{\overset{a}{111111}}_{6}\,0\,\underbrace{\overset{b}{111}}_{3}$

Output should be

$$\underbrace{11 \ldots\ldots 11}_{18}$$

Strategy:

Mark ~~each~~ each a 1 in a,
make a copy of b each time
(copying involves marking and unmarking)
When all the 1's in a are marked, we are done, erase a (replace w/ blanks) and unmark all co

copies of b.

Given an input n,

if n is even, output $\frac{n}{2}$

if n is odd, output $n+1$.

There are lots of ways to check even- ~~a~~ a ~~reg.~~ finite automaton can check this by just switching btw 2 states.

(Note- a DFA is equivalent to a TM where you cannot move L)

B/c we'll need to divide by 2, and squishing a string ~~is~~ (~~i.e.~~ after some middle bits are removed) is annoying, we'll match beginning w/ end.

① Mark ~~for~~ beginning w/ x, move to end (a □ or a y) and mark last 1 w/ y, and return to beginning (the letter after the x)

② If we end up at

$$\overset{\vee}{xxx}yy$$

and try to mark an x as a y, ~~this is~~ n is odd,
Change all the x's & y's back to 1's and add a 1 at beginning.

③ If we end up at

$$xxx\overset{\vee}{y}yy$$

and try to mark a y as an x then n is even,
Change all the y's to 1's and erase all the x's.