The Failure Function Algorithm

```
function FAILURE FUNCTION(p = p_1 ... p_l)
L1 { f(1) <- 0
   { i <- 0
L2 for j ∈ {2, ..., l} do
L3     i <- f(j - 1)
L4     while p_j ≠ p_{i+1} and i > 0 do
           i <- f(i)
       end while
L5     { if p_j ≠ p_{i+1} and i = 0 then
       {     f(j) <- 0
       { else
L6     {     f(j) <- i + 1
       end if
   end for
end function
```

Theorem: the failure function algorithm computes f in $O(l)$ steps, where l = length (p)

Proof:

- L3 and L5 have constant cost | constant number of time units
  - L3: just assigning $f(j-1)$ to i → constant cost (independent of l)
  - L5: boolean conditions and assignment → constant cost
- L4: cost of the "while" statement is proportional to the number of times i is decreased by the statement $i = f(i)$ on L4 following the "do"
  - by definition of f, $f(i) < i$
  - The only way i is increased/incremented is by assigning $f(j) = i+1$ (L6), then incrementing j by 1 (L2), and setting $i = f(j-1)$ (L3)

Proof continued:

- since $i = 0$ initially (L1) and L6 is executed at most $l-1$ times, we conclude that the while statement (L4) cannot be executed more than l times
  → therefore, L4 is time $O(l)$
- This takes care of any potential nested for-loop issues → ∴ alg is $O(l)$

Some concluding thoughts

- we can show that M_p will be in state i after reading $t_1 t_2 \dots t_k$ iff $p_1 p_2 \dots p_i$ is the longest prefix of p that is a suffix of $t_1 t_2 \dots t_k$
  Thus, M_p correctly finds the leftmost (first) exact occurence of p in the text $t = t_1 t_2 \dots t_m$
- the failure function is a linear time algorithm $(O(|p|))$, and using it for pattern matching makes it so M_p will execute at most $2|t|$ state transitions on the input text t.
- Thus, we can determine whether p is a substring of t by tracing out the state transitions of M_p on input t.
  Thus, $O(|p| + |t|)$ is the time complexity of the KMP alg, independent on the size of the alphabet Σ

We want to construct a DFA for the language A^*p (anything followed by pattern p)

- The DFA makes exactly 1 state transition per input letter

<u>algorithm</u> (constructing a DFA for A^*p)

<u>input</u>: pattern p over alphabet A

<u>output</u>: A DFA M such that $L(M) = A^* p$

<u>steps</u>:

1. use the failure function to construct f(p)
2. let $M = (S, A, \delta, S_0, \{l\})$ →
3. construct δ as follows:

$$\boxed{\begin{array}{l} \underline{\text{for}}\ j = 1, \dots, l\ \text{do}: \\ \quad \delta(j-1, p_j) = j \\ \underline{\text{for}}\ \text{each}\ a \in A, \\ \quad \text{if}\ a \neq p_1: \delta(0, a) = 0 \\ \underline{\text{for}}\ j = 1, \dots, l\ \text{do}: \\ \quad \underline{\text{for}}\ \text{each}\ a \in A\ \text{and}\ a \neq p_{j+1}\ \text{do}: \\ \quad\quad \delta(j, a) = \delta(f(j), a) \end{array}}$$

this essentially constructs the skeleton

S: set of states

A: alphabet

δ: transition function ~ takes in a state and a symbol and returns the next state (i.e. $\delta(0,a) = 1$)

S_0: start state

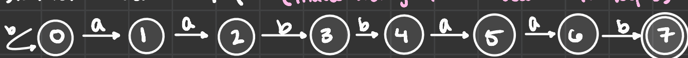
$\{l\}$: set of final/accepting states

Now, let's walk through the alg w/ an example:

<u>input</u>: $p = aabbaab$

$t = a\ b\ a\ a\ b\ a\ a\ b\ b\ a\ a\ b$

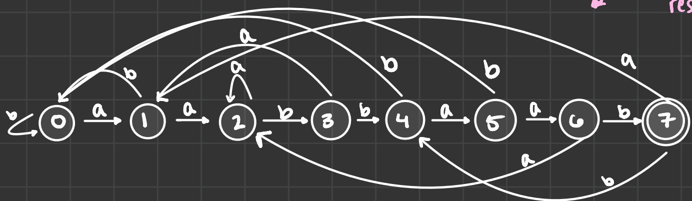
skeleton machine M_p: (made using first + second for-loops)



failure function:

p_i	a	a	b	b	a	a	b
i	1	2	3	4	5	6	7
$f(i)$	0	1	0	0	1	2	3

Third for-loop used to construct the rest of the DFA:



$\delta(1,b) = \delta(f(1),b) = \delta(0,b) = 0$

$\delta(2,a) = \delta(f(2),a) = \delta(1,a) = 2$

$\delta(3,a) = \delta(f(3),a) = \delta(0,a) = 1$

$\delta(4,b) = \delta(f(4),b) = \delta(0,b) = 0$

$\delta(5,b) = \delta(f(5),b) = \delta(1,b) = 0$

$\delta(6,a) = \delta(f(6),a) = \delta(2,a) = 2$

$\delta(7,a) = \delta(f(7),a) = \delta(3,a) = 1$

$\delta(7,b) = \delta(f(7),b) = \delta(3,b) = 4$

input:	a	b	a	a	b	a	a	b	b	a	a	b
State: 0	1	0	1	2	3	1	2	3	4	5	6	7