*I'm getting lost in the research paper @ the good suffix rule. Thus I'm going to try and develop some intuition for it before re-attacking the white paper.

GeeksforGeeks Article

- requires a preprocessing table to be generated

Let t be a substring of text T which is matched w/ substring of pattern P. Shift pattern until:

↳ NOTE !substring! not just a char

Case 1 Another occurrence of the substring t in P is matched w/ t in T

Ex.

	0	1	2	3	4	5	6	7
T	A	B	A	A	B			
P	C	A	B	A	B			

↓ Note bad suffix rule (b.s.r) would usually just shift by 1

C A B ... ⟶

To do this search through P of occurrence t("AB").

=> Weak rule of original Boyer-Moore and not highly effective (Strong Good Suffix rule coming up)

Note we are looking for the match or prefix going right to left

=> following algorithm's typical structure I guess

Case 2 A prefix of P matches w/ suffix of t in T

Now we repeat case 1, but no occurrence of t in P

∴ We look for a suffix of t matching w/ a prefix of P and try to align them

i	0	1	2	3	4	5	6	7	8	9	10
T	A	A	B	A	B	A	B	A	C	B	A
P	A	B	B	A	B						

!t   t

but

note w/ b.s.r we'd only shift by 1

=> A B B A B

A B B A B

I understand what the algorithm is doing, however, I am not quite grasping how we don't skip past potential matches

Trying to break

purposefully changed pattern

i	0	1	2	3	4	5	6	7	8	9	10
T	A	A	B	A	B	A	B	A	Z	B	A
P	A	B	A	B	A						

↳ immediately failed trying to break. ugh.

=> I'm going to just accept that this works for the moment (or rather I'm going to stop trying to understand why it works, I don't doubt the algorithms efficacy. Hopefully when I return to the paper I will get a better intuition for this)

Case 3 P moves past t

this happens if both c.1 and c.2 are not satisfied

Strong Good Suffix Hueristic

Substring $t = P[i \text{ to } n]$ gets matched w/ t in T and $c = P[i-1]$ is the mismatched character

=> Deviating from case 1 we'll search for t in P which is not preceeded by the character c

The <u>closest</u> occurrence is then aligned w/ t in T by shifting the pattern P

Ex:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T	A	A	B	A	B	A	B	A	C	B	A	C	A	B	B	C	A	B
P	A ✓	A	C	C	A	C	C	A	C									
=>							A	A	C	C	A	C	C	A	C			

* the examples that have been chosen are interesting b/c they have small alphabets
=> this usually implies a use case for KMP
=> probably done this way for simplicity

Preprocessing For Good suffix heuristic