Pattern Matching

- Given string T(text) and P(pattern), the pattern matching problem consists of finding a substring of T equal to P.

Brute-Force Algorithm

- O(nm)
- Worst case:
  - T = aaa... bh
  - P = aaab
  - → may occur in images and DNA sequence
  - → unlikely in English text

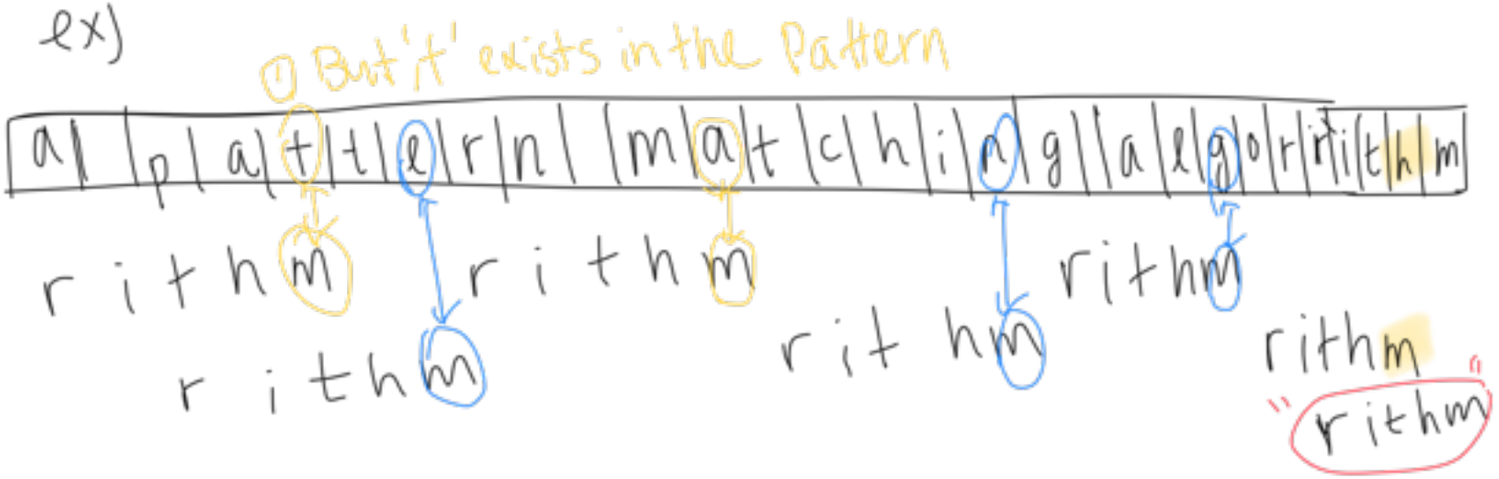
Boyer-Moore Heuristics

- Based on two heuristics
  - ① Looking-glass heuristic: Compare P with

a subsequence of T moving backwards

(v) character-jump heuristics: mismatch happen

ex)



Last-Occurance Function

- Ex:
  - $\Sigma = \{a, b, c, d\}$
  - P = abacd

c	a	b	c	d
LCW	4	5	3	-1

- O(m+s) (m: size of P, s: size of Σ)

Analysis

- Boyer-Moore: O(nm+s)
- worst case:
  - T = aaa...a
  - P = baaa

→ may in images and DNA sequences

- It's significanty faster than brute force on English text

The KMP Algorithm

- It compares to the text in left-to-right, but shifts the pattern more intelligently than brute-force algorithm.
- When a mismatch occurs, what is the most we can shift the pattern so as to avoid redundant comparisons?
- failure function: defined as the size of the largest prefix that is also a suffix of P[1...j]
  - can be represented by an array, O(m) time

ex

① failure function

Pattern: "GACAGATGA"

index	Pattern	F(j)

index		
0	G	0
1	GA	0
2	GAC	0
3	GACA	0
4	GACAG	1
5	GACAGA	2
6	GACAGAT	0
7	GACAGATG	1
8	GACAGATGA	2

step1. F(7)=2

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Text	C	G	T	A	C	C	C	G	A	C	A	G	A	T	G	A	C	A	G	A
Pattern	G	A	C	A	G	A	T	G	A											

step2.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Text	C	G	T	A	C	C	C	G	A	C	A	G	A	T	G	A	C	A	G	A
Pattern			G	A	C	A	G	A	T	G	A									

step3.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Text	C	G	T	A	C	C	C	G	A	C	A	G	A	T	G	A	C	A	G	A
Pattern								G	A	C	A	G	A	T	G	A				

* 12에서, i와 j는 항상 align.

if - 만약 failure function이 있으면 → 그 다음 index로 i and j jump 가!

else - 없으면, failure function 계산 후 → 그 만큼의 index 만큼 자리 옮기기!

Tries