STRING MATCHING

DATA STRUCTURES AND ALGORITHMS (INFO-F413)

CHARLOTTE NACHTEGAEL

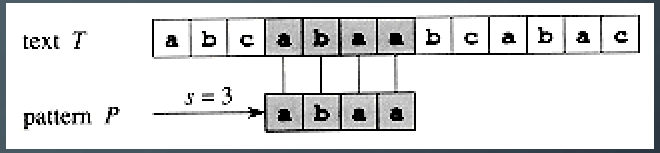
YOUR MISSION, IF YOU ACCEPT IT

Your favorite agent 007 called you with an important task : find the indexes of the secret word in the text to use the numbers to defuse an destructive bomb menacing the world !

The fate of the world rests on your shoulders….

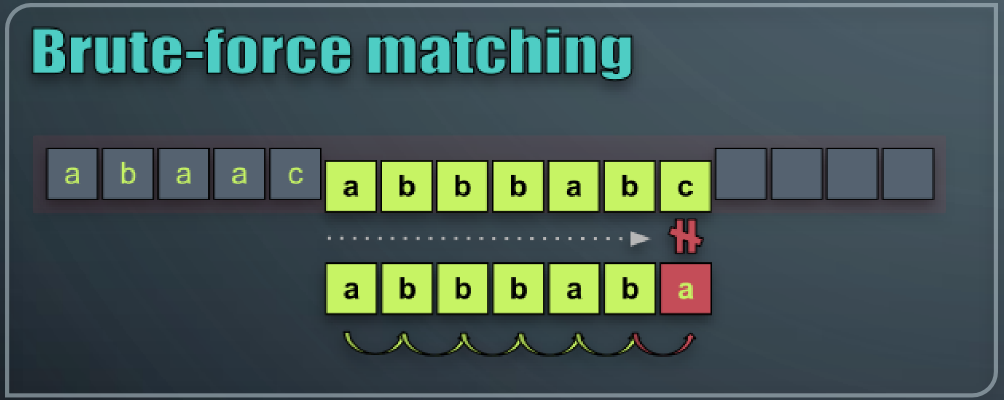
FORMALLY SPEAKING

- Text is an array T[1…n]
- Pattern/Secret word is an array P[1…m]



- P occurs with shift s in text T (or, equivalently, that pattern P occurs beginning at position s + 1 in text T) if 0 ≤ s ≤ n - m and T[s + 1 . . s + m] = P[1 . . m] (that is, if T[s + j] = P[j], for 1 ≤ j ≤ m).
- The string-matching problem is the problem of finding all valid shifts with which a given pattern P occurs in a given text T.

FIRST IDEA : LET'S DO THIS ONE-BY-ONE

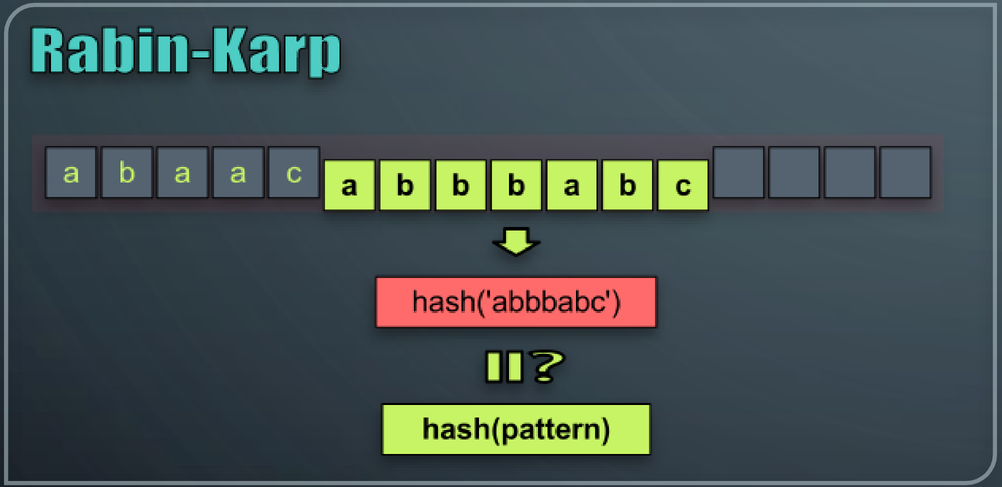


Brute-force matching

- Checks the condition P[1 . . m] = T[s + 1 . . s + m] for each of the n - m + 1 possible values of s
- Θ((n-m+1)m) !

➔ Too long, the world would explose before you finish !

SECOND IDEA : LET'S DO HASHING !



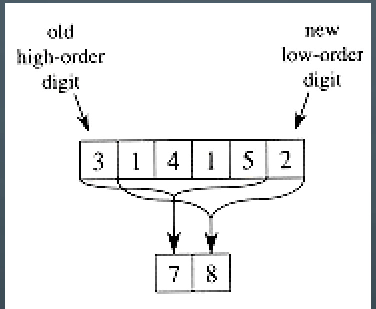
- Introduction of a powerful technique : the *hash function*
- Calculate a number H for the pattern and each substring of T of length m

➔ Better than one-by-one as you compare two integers and not two strings character by character !

ROLLING HASH

With d, the size of alphabet and q, a large prime number

$$H(T) = (T[0]d^{m-1} + T[1]d^{m-2} + \cdots T[m-1]d^{0})\ \%\ q$$

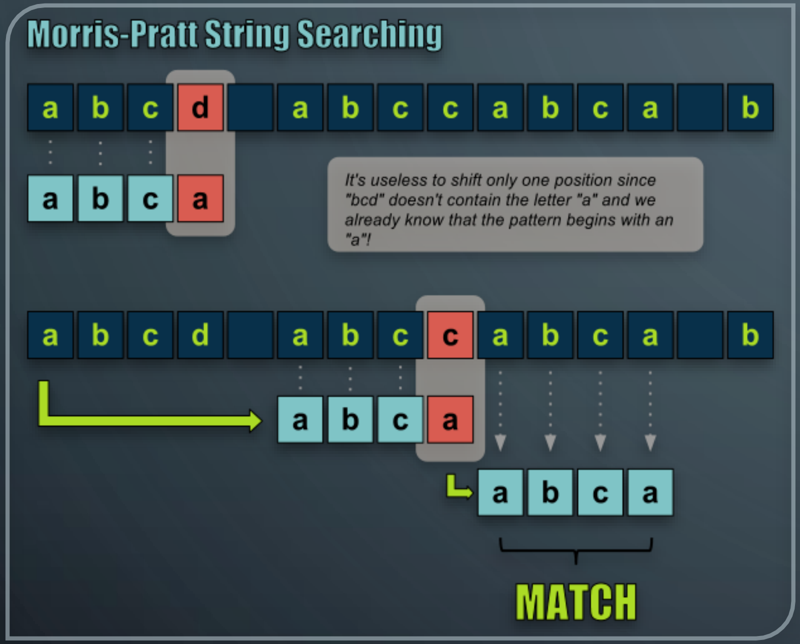


$$H(T_{s+1}) = \ (d(T_s - T[s+1]d^{m-1}\%q) + T[s+m+1])\ \%\ q$$

➔ Preprocessing O(m) + Looking at the text O(n) * rolling hash to look O(1)

➔ O(m + n)

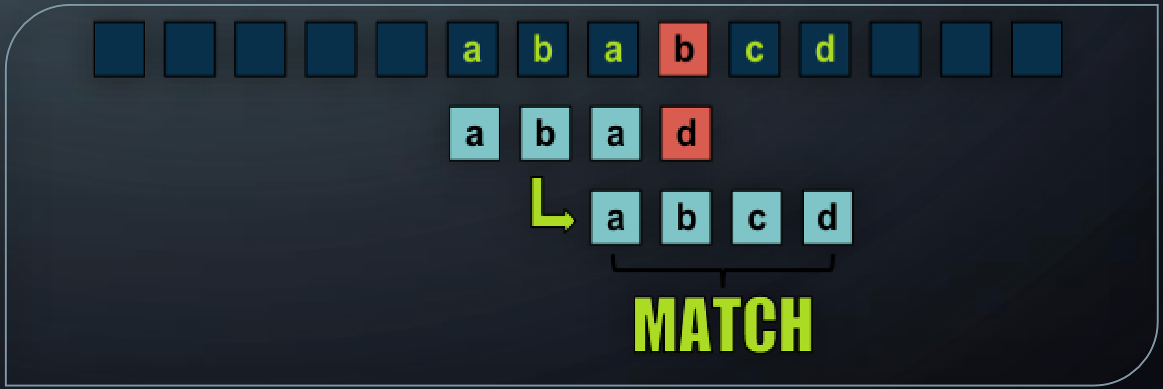
THIRD IDEA : LET'S JUMP FROM ONE INTERESTING PLACE TO ANOTHER



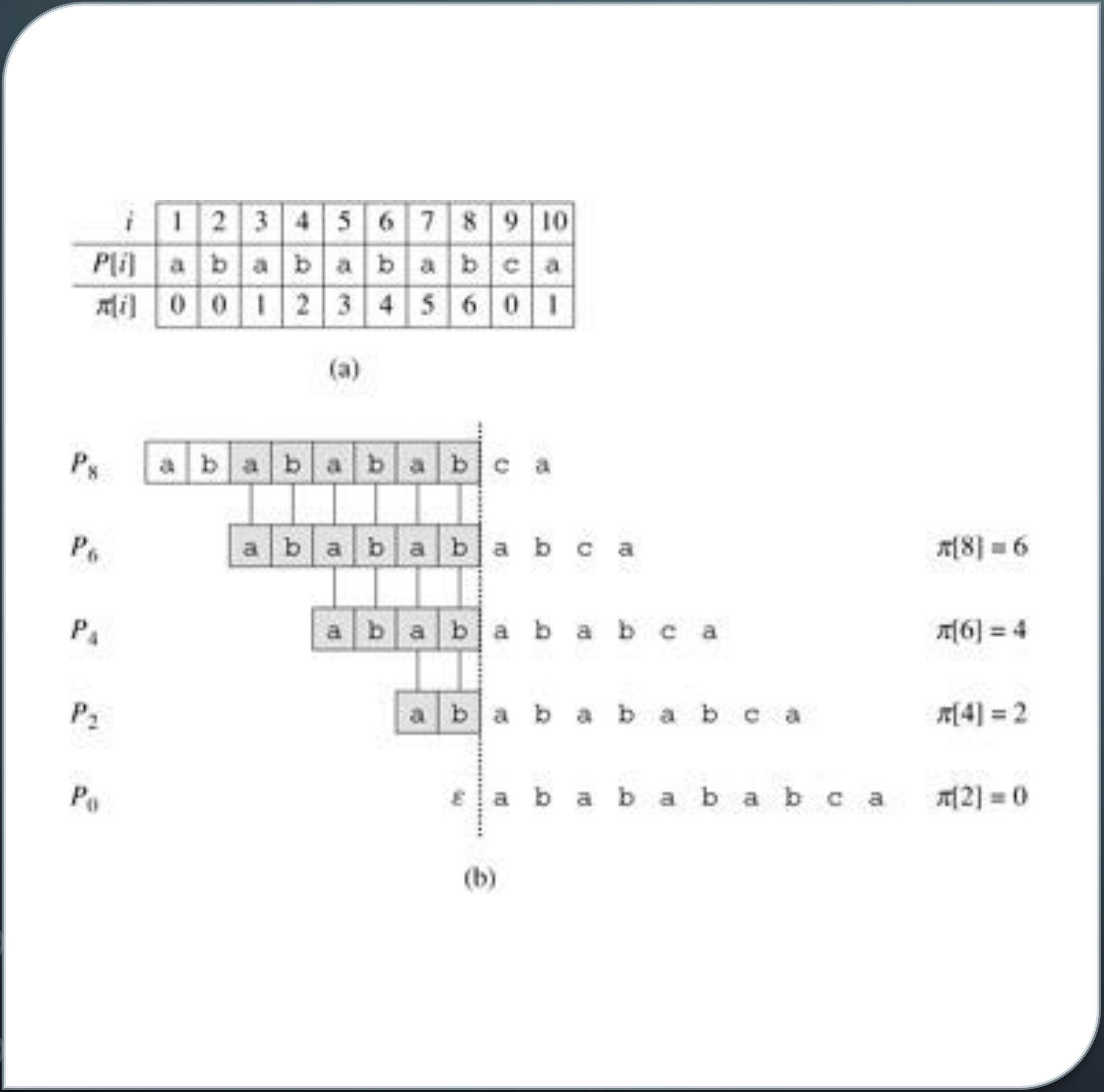
Morris-Pratt String Searching

It's useless to shift only one position since "bcd" doesn't contain the letter "a" and we already know that the pattern begins with an "a"!

- You look for the prefix of your pattern within your pattern
- Shift directly to the next beginning of pattern
- If no next prefix, shift further



BUILDING THE PREFIX TABLE



i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

(a)

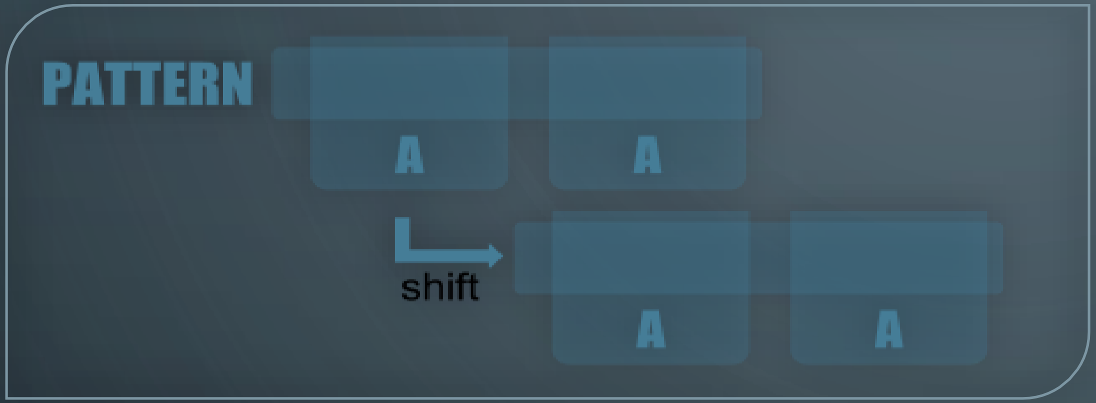
P_8 a b a b a b a b c a
P_6 a b a b a b a b c a $\pi[8] = 6$
P_4 a b a b a b a b c a $\pi[6] = 4$
P_2 a b a b a b a b c a $\pi[4] = 2$
P_0 ε a b a b a b a b c a $\pi[2] = 0$

(b)

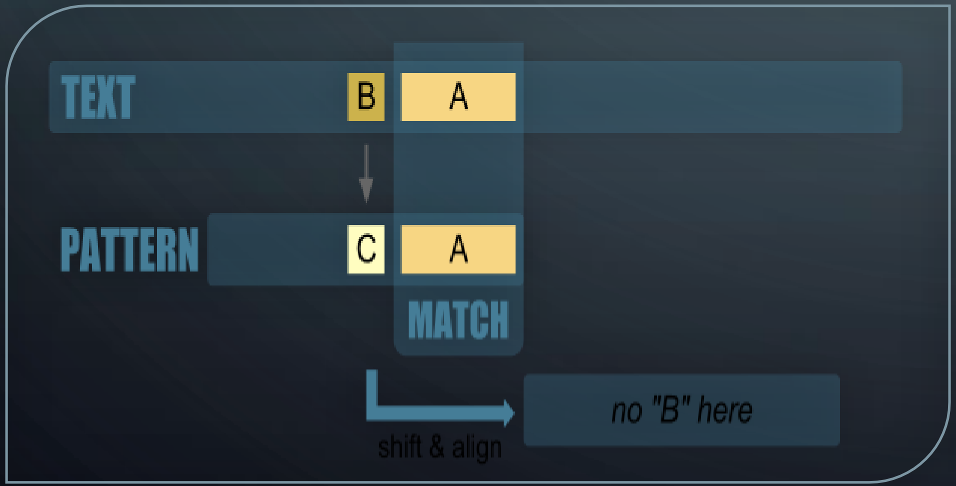
```
COMPUTE-PREFIX-FUNCTION(P)
 m ← length[P]
 π[1] ← 0
 k ← 0
 for q ← 2 to m
      do while k > 0 and P[k + 1] ≠ P[q]
              do k ← π[k]
       if P[k + 1] = P[q]
             then k ← k + 1
         π[q] ← k
return π
```

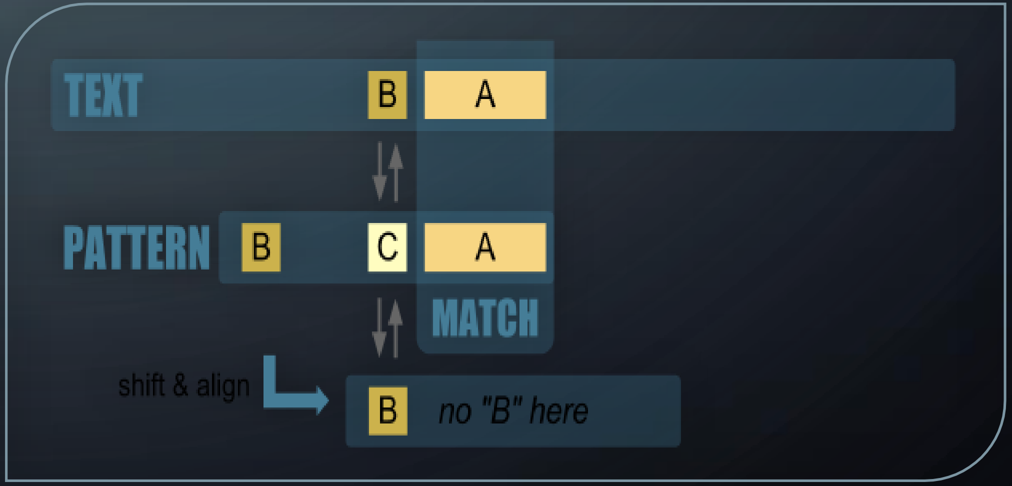
➔ Preprocessing O(m) + Looking in the text O(n) ➔ O(n + m)

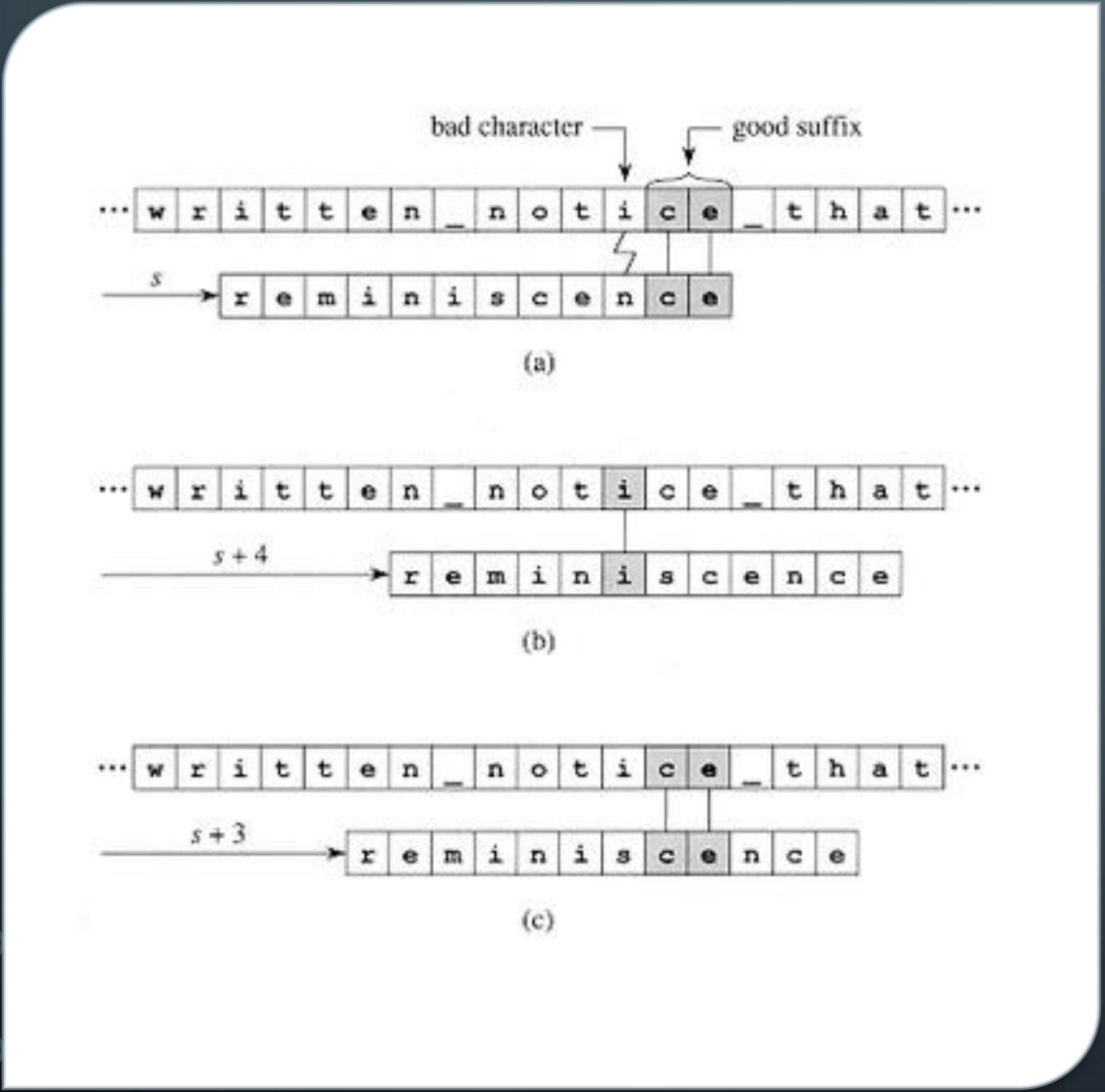
FOURTH IDEA : LET'S BEGIN WITH THE END



- The letters of the pattern are compared from right to left !
- Two kinds of shifts :
  - Good suffix shift
  - Bad character shift






bad character
good suffix
… w r i t t e n _ n o t i c e _ t h a t …
s
r e m i n i s c e n c e
(a)
… w r i t t e n _ n o t i c e _ t h a t …
s + 4
r e m i n i s c e n c e
(b)
… w r i t t e n _ n o t i c e _ t h a t …
s + 3
r e m i n i s c e n c e
(c)

THE BAD CHARACTER & THE GOOD SUFFIX

- Bad character : shift to the last occurrence of the bad character in the pattern
- Good suffix : shift to the next pattern found with the suffix

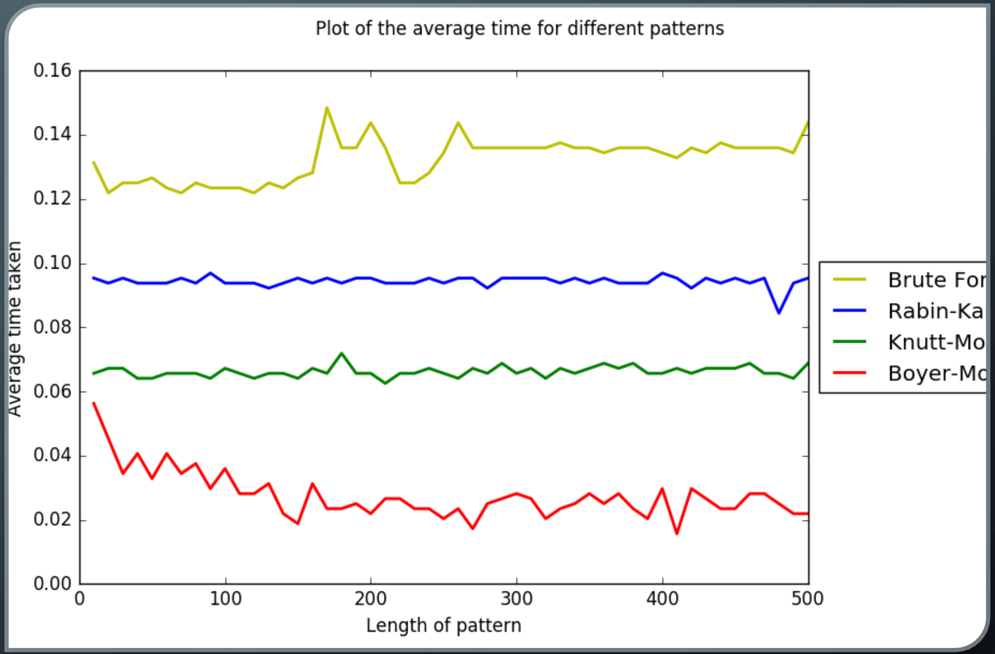
➔ Choose the biggest shift !

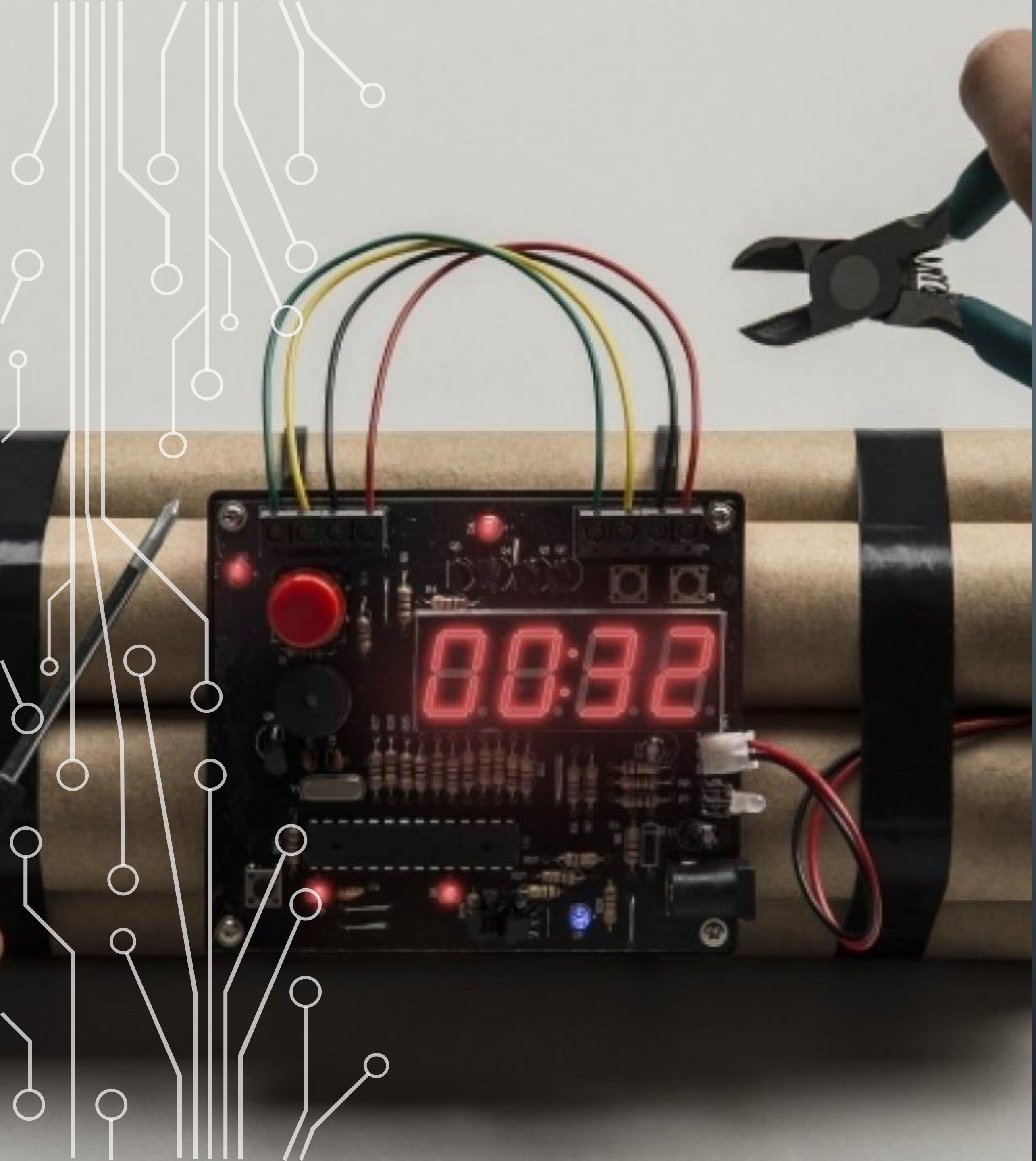
➔ Preprocessing O(m) and O(m + size alphabet) + O(m) time to validate each shift s

➔ O((n-m+1)m) but in practise : Ω(n/m) and O(nm)

WHAT'S THE BEST IDEA ?

- How many words must we find ?
- What is the size of your alphabet ?
- What is the size of your pattern ?
- What is the size of your text ?





Thank you for listening !

Now, it’s your turn to save the world !

But before that, any questions ?