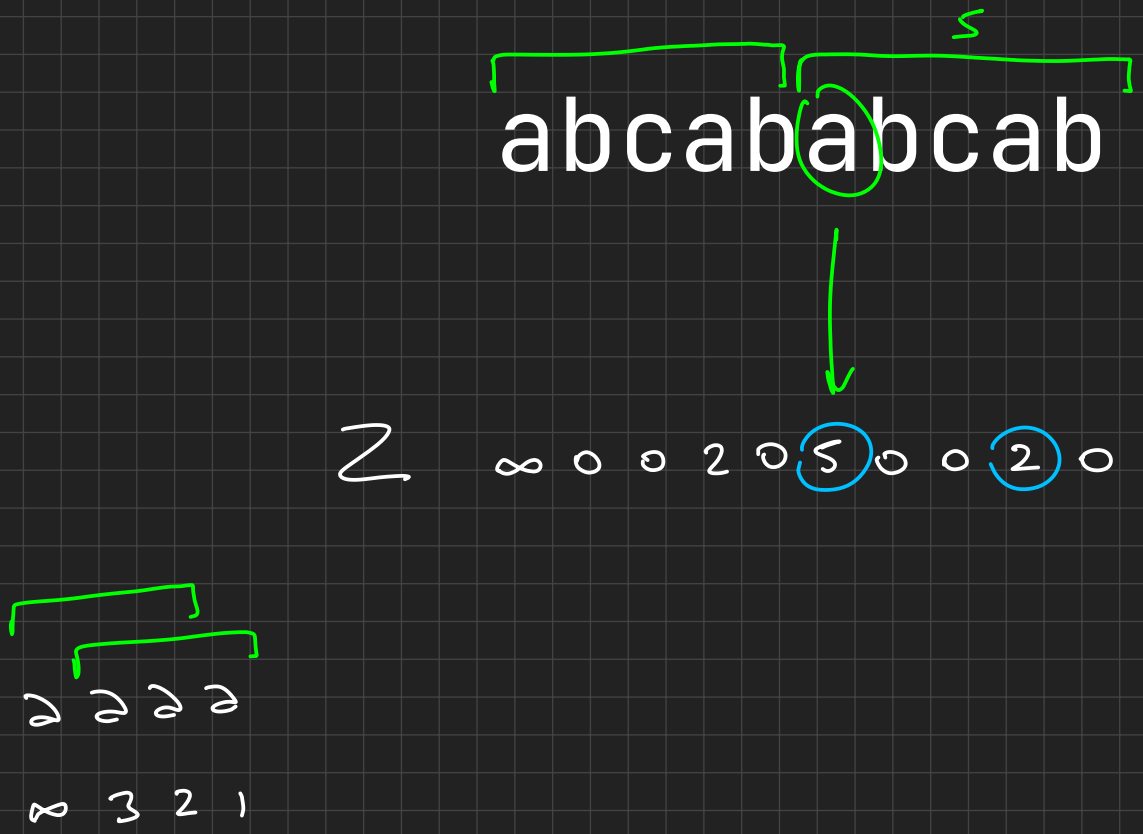
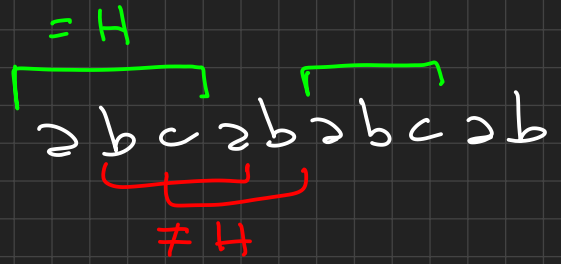
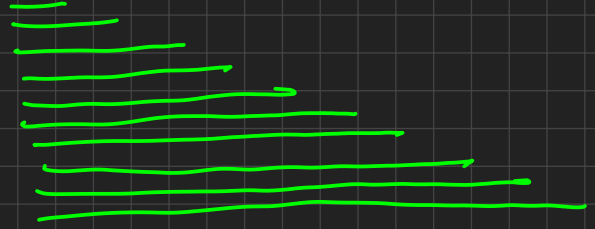

s
abcababcab
2 ∞ 0 0 2 0 5 0 0 2 0
2 2 2 2 2
∞ 3 2 1



pattern = abc

hash = H1

a b c a b a b c a b



precalcolo gli hash dei prefissi in O(n)

$\therefore =$ stessa cosa per i suffissi

- hash di una sottostringa generica (di s) in $O(1)$

a b c a b a b c a b

$\stackrel{?}{=}$

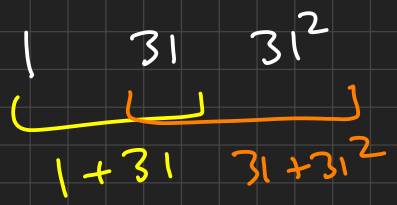
prefix sum

$h(l, r)$ = hash della sottostr. $[l, r]$

$h(l, r) \stackrel{?}{=} h(1, r) - h(1, l-1)$ non proprio

a a a a ... a

$b = 31$



$$h(l,r) = \frac{h(l,r) - h(l,l-1)}{b^{l-1}} \quad \text{(in modulo)}$$

$$h(l_1, r_1) \stackrel{?}{=} h(l_2, r_2)$$

confronto le due frazioni (in modulo)

- alt., "shifto" tutti gli hash moltiplicando per b^n

$\frac{1}{b^{l-1}} \rightarrow b^{n-l+1}$ ⇒ non serve calcolare inversi

- precalcolo le potenze di b in un array

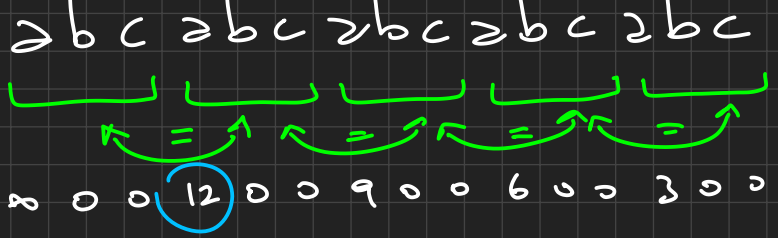
- periodi di una stringa

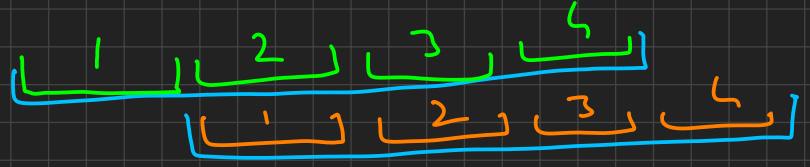
abcabca??

- fisso l e trovo i blocchi lunghi l
- controllo se i blocchi sono uguali

ho $O(n/l)$ blocchi

$$\sum_{l=1}^{n} O(n/l) = O(n \log(n))$$


abc abc abc abc abc
?
=
=
=
=
12


1
2
3
4
1
2
3
4


1
=
1

- Controllo se una sottostringa è palindroma

a b c d e d c f g

→ ∘ ∘ ∘ ∘ ∘
← ∘ ∘ ∘ ∘ ∘

a b c d e h c f g

→ ∘ ∘ ∘ ∘ ∘
← ∘ ∘ ∘ ∘ ∘

2 non funziona

c d e d c

$h(1,2) \stackrel{?}{=} h'(4,5)$

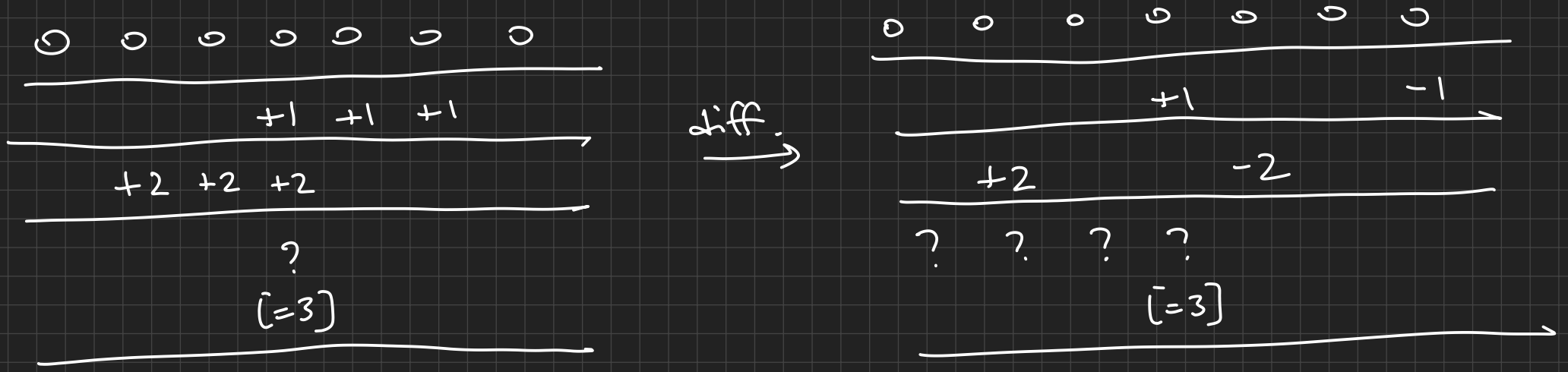
h →
h' ←

$h(1,5) = h'(1,5)$

- point update?
  - per ogni posizione, salvo l'hash del prefisso
  - se aggiorno un carattere, devo aggiornare un suffisso (si può fare con segment tree lazy) [range add]
  - risposta a una query:
    - hash di una sottostringa: differenza di due hash prefissi [point query]

range add, point query → point add, range query

- basta un fenwick



- torniamo al problema iniziale
  - diff. tra hash di prefissi consecutivi = hash di un singolo carattere (. bi)

- longest repeating substring

cababababc

- come calcolo Z senza Z-alg., usando hashing? $(O(n \log(n)))$

abcababcab

∞ 0 0 2 0 5 0 0 2 0

- binary search per trovare la lungh. massima

- come confronto due sottostringhe (determino se $h(l_1, r_1) \lessgtr h(l_2, r_2)$? $O(\log(n))$
  - trovo il max prefisso uguale con binary search
  - confronto il carattere successivo (che è diverso)

"se non funziona hashing, non funziona 2 (a meno di fattori $\log(n)$"

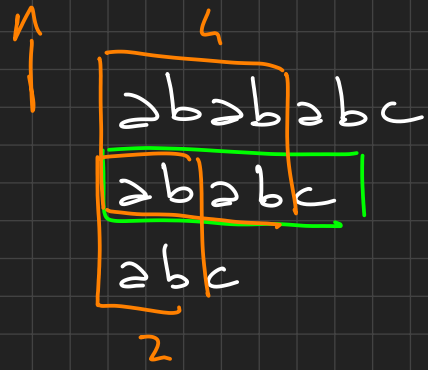
- DBF / suffix array mi danno l'ordinamento dei suffissi.

cababababc

l_4

- fisso l_4

  come faccio a trovare l_2 "ottimale"?

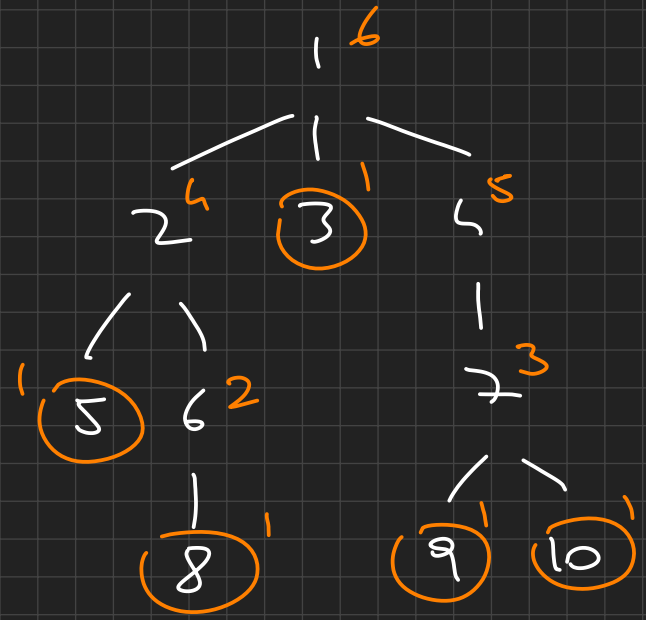


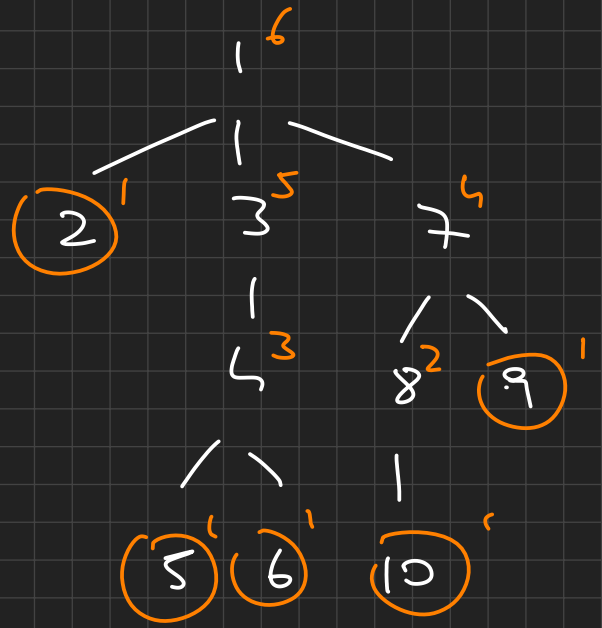
per assurdo, qui esistono suffissi con un prefisso "lungo" in comune ≥ 5

dovrebbe iniziare con "ababc"

falso per l'ordinamento

stessa cosa





map <vector<int>, int> f;

- f associa a ogni vector un int

{} → 1

{1} → 2

{1,1} → 3

{1,2} → 4

{3} → 5

{1,4,5} → 6

DCDDDCCADA

ACADDCCADA

DBADDCCBDC

DBADDCCADA

ABADDCCADC

1

2

4

8

16

valore totale dei cerchi:

$2(1+2+4+16)$

si calcola in $O(1)$

trovo la somma per colonne

A	18		11	
B	0	- - -	0	
C	0		20	
D	13		0	