Exact String Matching with Z-Array

Wittawat Jitkrittum

30 May 2017

Gatsby Tea Talk

Exact String Matching

- Given: a pattern P, long string T.
- Find all occurrences of P in T.

- Many applications
  - Find subsequences of DNA.

    P = “gtcc”, T = “...ctggtccactgtccactgg...”
  - “ctrl + f” in a web browser.

Naive Algorithm

- P = “ab”, T = “aabaabe”. Let m := len(P), n := len(T).

index	1	2	3	4	5	6	7
T	a	a	b	a	a	b	e
k=1	a	b					
k=2		a	b				
k=3			a	b			
k=4				a	b		
k=5					a	b	
k=6						a	b

¿

- At iteration k, check [T(k), .., T(k+m-1)], and P.
- Complexity: O(m*n).
- Does not share information across iterations.

Z-Array

- Let S be a string of length u.
- **Z-Array**: Z(k) = Length of the longest substring of S starting at k and matches a prefix of S, for k > 1.
- S = “aaabc”

index	1	2	3	4	5
S	a	a	a	b	c
Z(2) = 2	a	a	b	c	
Z(3) = 1	a	b	c		
Z(4) = 0	b	c			
Z(5) = 0	c				

- Can be constructed in O(u) time. Linear-time!

Z-Array for String Matching

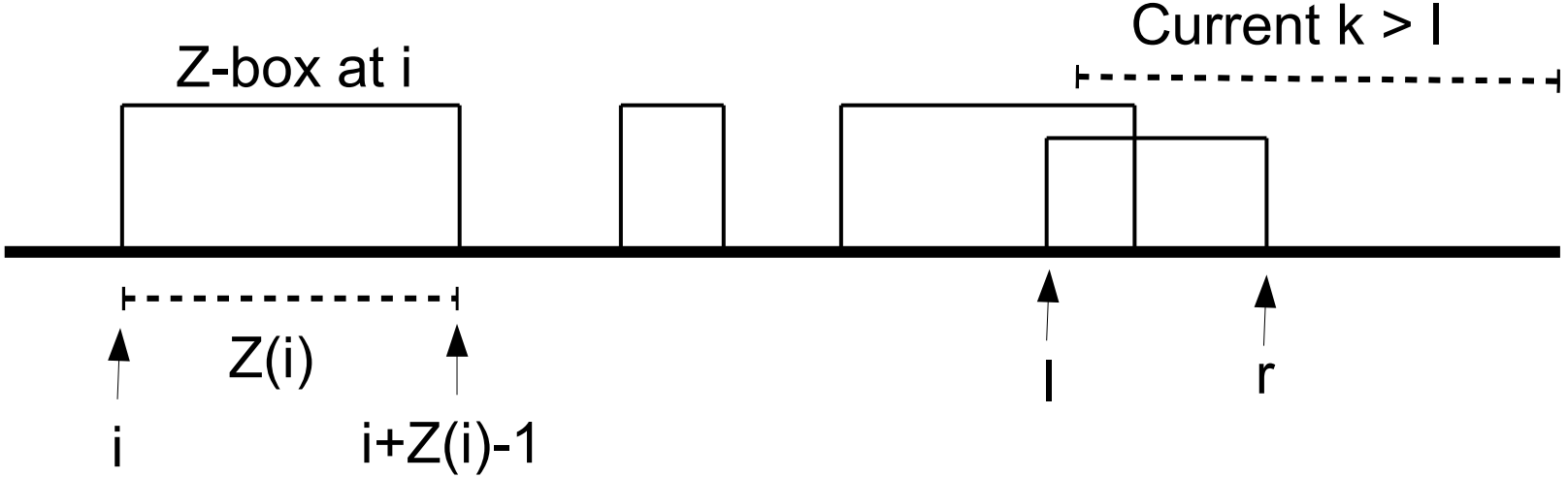
- Let S = P$T, where $ = character appearing in neither P nor T.
- P = “ab”, T = “aabaabe”. Let m := len(P), n := len(T).

index	1	2	3	4	5	6	7	8	9	10
S	a	b	$	a	a	b	a	a	b	e
Z(4) = 1	a	a	b	a	a	b	e			
Z(5) = 2	a	b	a	a	b	e				
Z(6) = 0	b	a	a	b	e					
Z(7) = 1	a	a	b	e						
Z(8) = 2	a	b	e							

- P occurs at k where Z(k) = m.
- So, string matching can be done in O(m + n).

Z-box

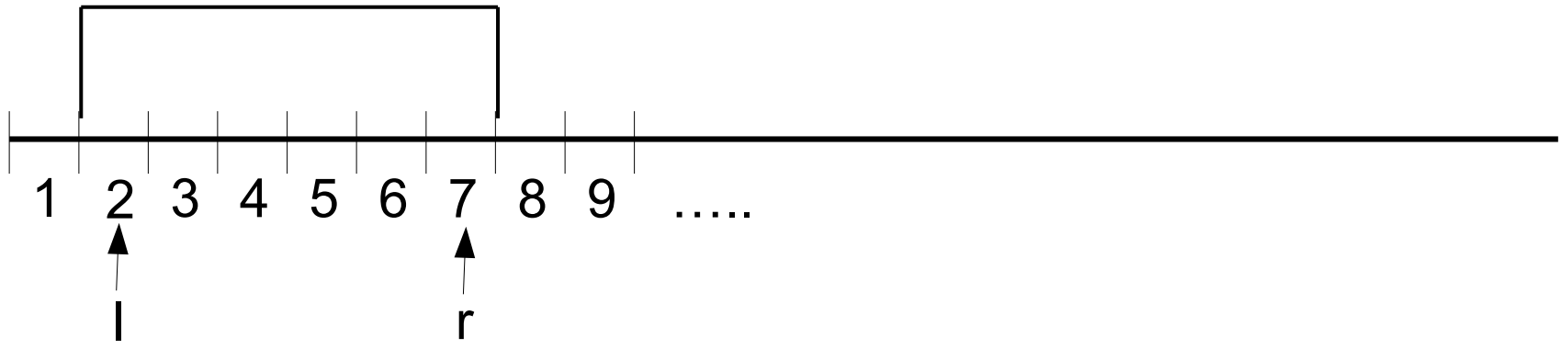
- Will iteratively compute Z(k) given
  - Z(2), .., Z(k-1) and
  - boundaries of the right-most Z-box.
- **Z-box** = Z-box at i is the substring starting at i and continuing to i+Z(i)-1. Only defined for Z(i) > 0.
- Let l and r be the boundaries of the right-most Z-box. Right-most means largest right index.



Z Algorithm (1)

Initialization:

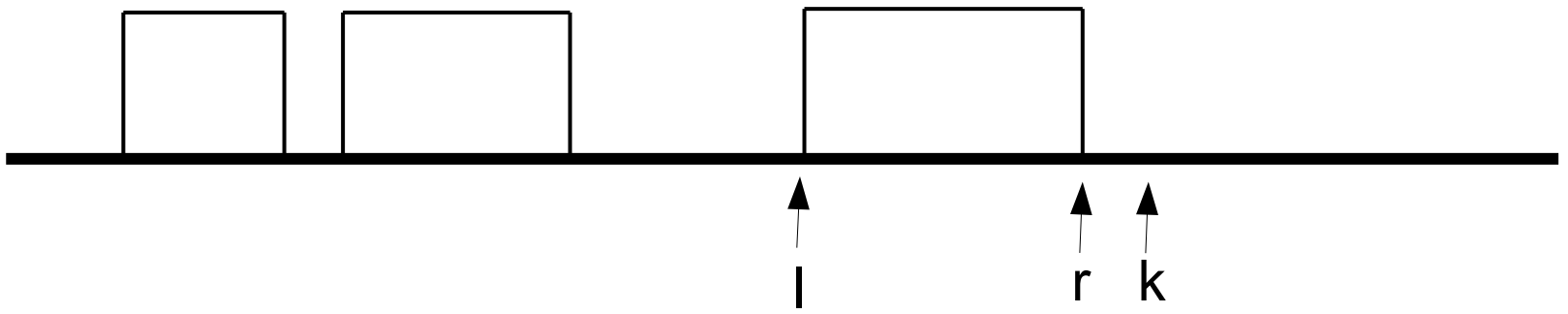
- Set Z(2) = longest prefix of S[2..] and S.
- If Z(2) > 0, set l := 2, r := l + Z(2) – 1.



- Otherwise, l = r = 0.

Z Algorithm (2)

- **Input**: Z(2), .., Z(k-1), and [l, r].
- Determine Z(k) and update [l, r].
- **Case 1**: if k > r,

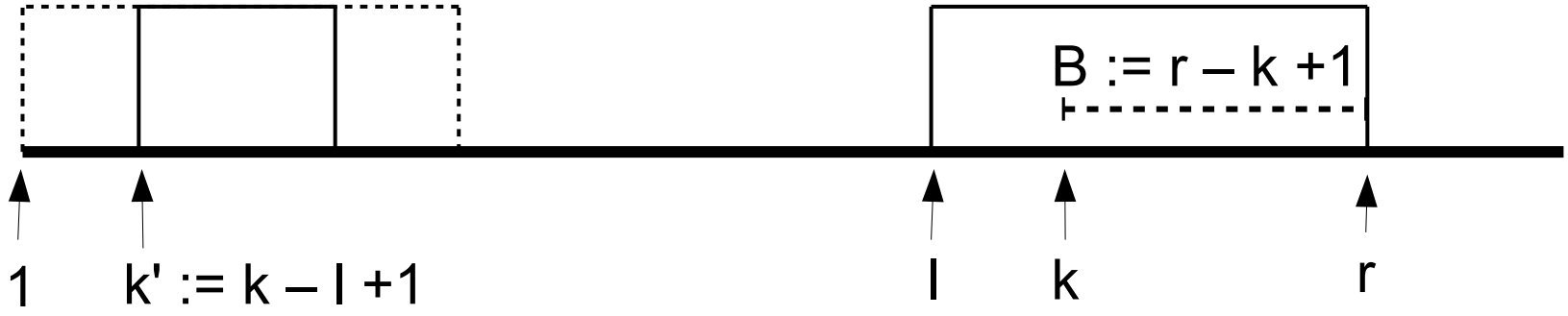


- Compute Z(k) manually i.e., compare S[k..] and S.
- If Z(k) > 0, update l = k and r = k + Z(k) – 1.

Z Algorithm (3)

Case 2: if k <= r

- **Case 2a**: Z(k') < B

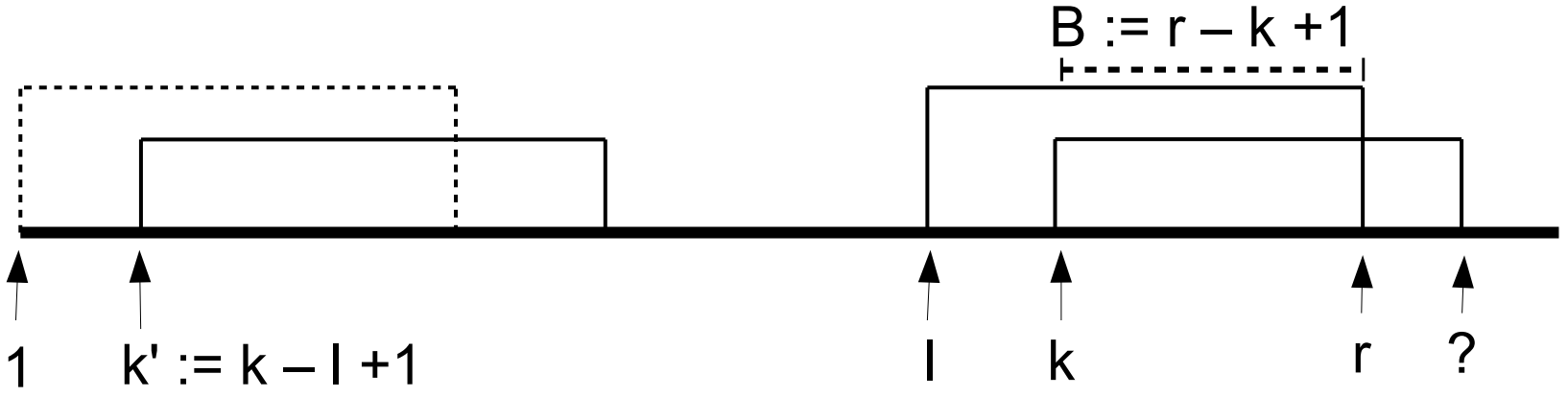


index	1	2 (k')	3	4	..	..	l	k		r	..
S	b	a	a	e	..	..	b	a	a	e	..
Z		0	0	0	..	..	4	?			

- Set Z(k) = Z(k').
- No update to [l, r].

Z Algorithm (4)

Case 2: if k <= r, and **Case 2b**: Z(k') >= B



index	1	2	3 (k')	4	5			l		k		r	
S	a	b	a	b	a	b	...	a	b	a	b	a	c
Z		0	4	0	1			5	0	?			

- Z-box guarantees that S[l..r] = S[1...5], but not beyond.
- S[k..r] must be a prefix. Compare S[r+1...] to S[B+1..].
- If a mismatch occurs at q >= r +1, set Z(k) = q – k + 1.
- Update l = k, r = q-1.

Comments on the Z Algorithm

- Operations under Z-boxes are constant-time.
- Complexity outside Z-box = O(#comparisons) = O(#matches + #mismatches)
- Each iteration contains at most one mismatch. #mismatches = O(u), where u = len(S).
- When matched, Z-box boundary r is extended proportionally to #matches.
- At most u possible values of r. That means #mismatches = O(u).

Thank you