Bioinformatik

Z-Box Algorithmus – Lineares Stringmatching

Silke Trißl

Wissensmanagement in der Bioinformatik

Zeichenketten

- Definition
  Ein String S ist eine von links nach rechts angeordnete Liste von Zeichen eines Alphabets Σ
  - *|S| ist die Länge des Strings*
  - *Positionen in S sind 1,...,|S|*
    - *Wir zählen ab 1 (Java: „index wird hierbei von null an gezählt.")*
  - *S(i) beschreibt das Zeichen an der Position i im String S*
  - *S[i..j] ist der Substring, welcher an Position i beginnt und an Position j endet*
  - *S[i..j] ist ein leerer String, falls i > j*
  - *S[1..i] heißt Präfix von S bis zur Position i*
  - *S[i..] ist das Suffix von S, welches an Position i beginnt*
  - *Echte Präfixe und echte Suffixe umfassen nicht den gesamten String S und sind nicht leer*

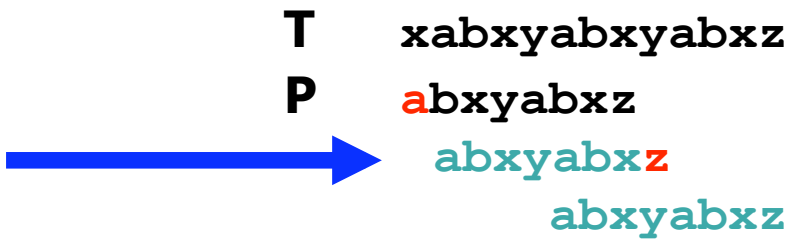
Naiver Ansatz

1. P und T an Position 1 ausrichten
2. Vergleiche P mit T von links nach rechts
   - Zwei ungleiche Zeichen ⇒ Gehe zu 3
   - Zwei gleiche Zeichen
     - P noch nicht durchlaufen ⇒ Verschiebe Pointer nach rechts, gehe zu 2
     - P vollständig durchlaufen ⇒ Merke Vorkommen von P in T
3. Verschiebe P um ein Zeichen nach rechts
4. Solange Startposition <= |T|-|P|, gehe zu 2

```
T   ctgagatcgcgta
P   gagatc
     gagatc
      gagatc
       gagatc
        gagatc
         gatatc
          gatatc
           gatatc
```

Optimierungsidee

- Anzahl der Vergleiche reduzieren
  - P um mehr als ein Zeichen verschieben
  - Aber nie soweit, dass ein Vorkommen von P in T nicht erkannt wird
- Beobachtung: Zeichen



- Substring in T muss mit a beginnen
- Nächstes a in T erst an Position 6 – springe 4 Positionen
- Vorkommen von Buchstaben in T kann während des Vergleichs ab Position 2 gelernt werden

Z-Algorithmus

- Zerlegung des Problems in zwei Phasen
  - Preprocessing: Lerne möglichst viel über die Struktur der beiden Strings
  - Search: Nutze das gelernte, um Vergleiche zu sparen
- Auch das Preprocessing muss schnell sein
  - Forderung kann aufgehoben werden, wenn bei vielen Suchen P oder T gleich ist
    - Beispiel: Suche Promotersequenz in vielen Gensequenzen

Z-Algorithmus: Preprocessing

- Im Folgenden: S
  - (wird gleich aus P und T zusammengesetzt)
- Definition
  - *Sei i>1. Dann ist $Z_i(S)$ die Länge des längsten Substrings x von S mit*
    - *x=S[i..i+|x|-1]* *(x startet an Position i in S)*
    - *S[i..i+|x|-1] = S[1..|x|-1]* *(x ist auch Präfix von S)*
  - *x ist die Z-Box von S an Position i mit Länge $Z_i(S)$*

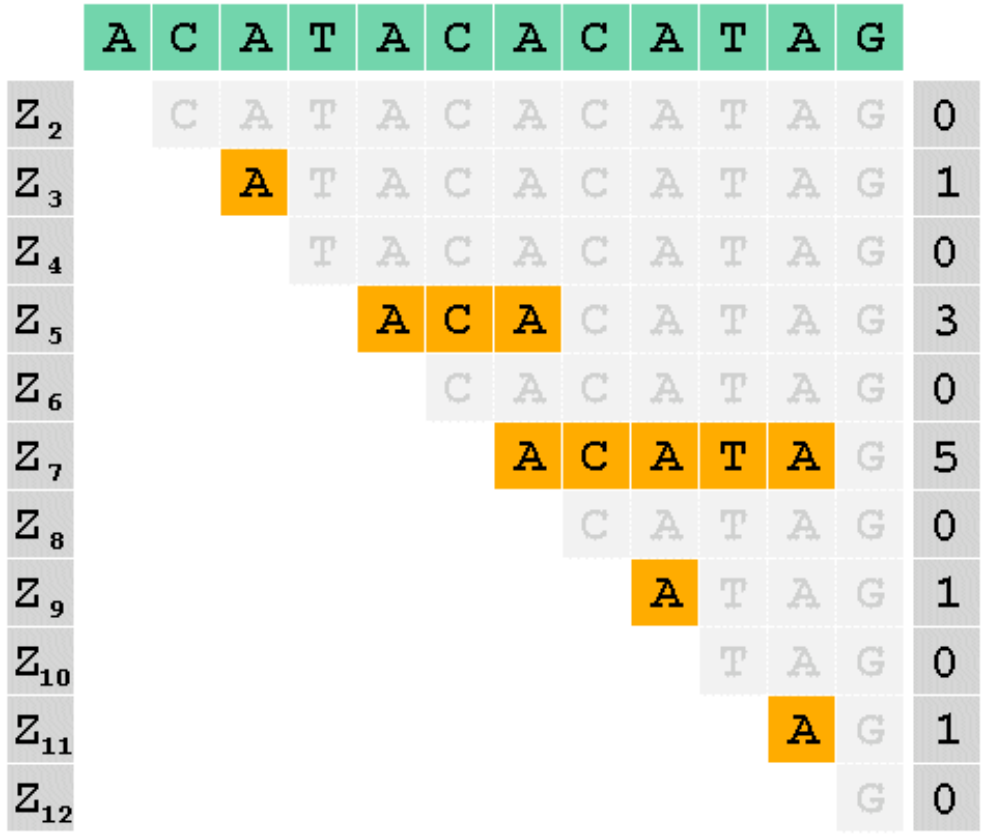
Beispiele

```
S = aabcaabxaaz
      1(a)
       0
        0
          3(aab)
           1(a)
             0
               0
                 2(aa)
                   1(a)
                     0
```

```
S = aaaaaa
```

```
S = baaaaa
```

Beispiel 2


A C A T A C A C A T A G
Z2 C A T A C A C A T A G 0
Z3 A T A C A C A T A G 1
Z4 T A C A C A T A G 0
Z5 A C A C A T A G 3
Z6 C A C A T A G 0
Z7 A C A T A G 5
Z8 C A T A G 0
Z9 A T A G 1
Z10 T A G 0
Z11 A G 1
Z12 G 0

Linearer Stringmatching Algorithmus

- Annahme: Z-Boxen lassen sich in O(|S|) berechnen
  - Wie zeigen wir später
- Verwendung der Z-Boxen für String Matching

```
S := P||'$'||T;                          // ($ ∉ Σ)
compute Z-Boxes for S;
for i = |P|+2 to |S|
       if (Z_i(S)=|P|) then
              print i-|P|-1;  // P in T at position i
       end if;
end if;
```

- Komplexität
  - Schleife wird |S|-mal durchlaufen => O(m)

Berechnung der Z-Boxen

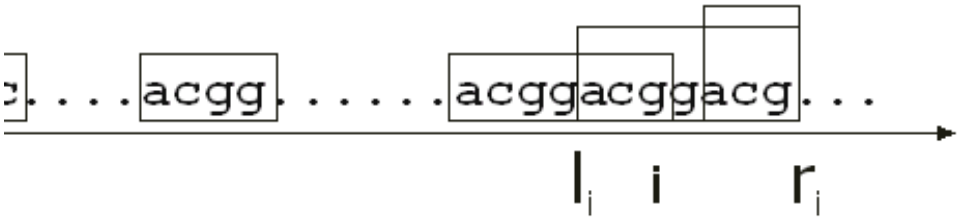
- Naiver Algorithmus braucht $O(|S|^2)$

```
for i = 2 to |S|
        Z_i := 0;
        j := 1;
        while (((S[j] = S[i + j – 1])) and (j <= |S|))
                Z_i := Z_i + 1;
                j := j + 1;
        end while;
end for;
```

- Damit wäre nichts gewonnen
  - $O((m+n)^2) + O(m) = O(m^2)$

Vorarbeiten

- Definition
  - *Sei i > 1. Dann ist*
    - r_i *der maximale Endpunkt aller Z-Boxen, die bei oder vor i beginnen*
    - l_i *ist die Startposition der längsten Z-Box, die bei* r_i *endet*
- l_i ist eindeutig, da an jeder Position nur eine Z-Box beginnt
- $S[l_i..r_i]$ ist die Z-Box, die die Position i von S enthält, am weitesten nach rechts reicht und am längsten ist



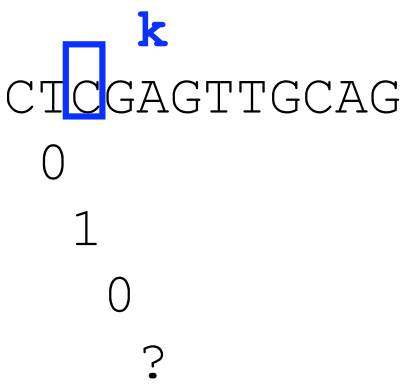
Lineare Berechnung der Z_i Werte

- Trick
  - Verwenden von bereits bekannten Z_i zur Berechnung von Z_k ($k > i$)
- Grundaufbau
  - Lineares Durchlaufen des Strings (Laufvariable k)
  - Kontinuierliches Vorhalten der aktuellen Werte $l=l_k$ und $r=r_k$
  - Größe der Z-Box an Position k ergibt sich mit konstantem Aufwand
- Induktive Erklärung
  - Induktionsanfang: Position k=2
    - Berechne Z_2.
    - Wenn $Z_2 > 0$, setze $r=r_2$ $(=2+Z_2-1)$ und $l=l_2$ $(=2)$, sonst $r=l=0$
  - Induktionsschritt: Position k>2
    - Bekannt sind r, l und $\forall j<k$: Z_j

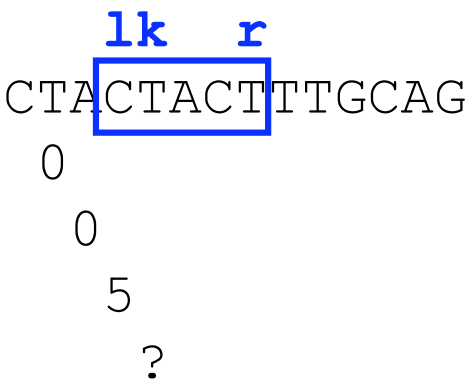
Z-Algorithmus, Fall 1

- Möglichkeit 1: k > r
  - D.h., dass es keine Z-Box gibt, die k enthält
  - Wir wissen nichts über den Bereich in S ab k
  - Dann gehen wir primitiv vor
    - Berechne Z_k durch Zeichen-für-Zeichen Matching
    - Wenn $Z_k>0$, setze $r=r_k$ und $l=l_k$

Beispiel

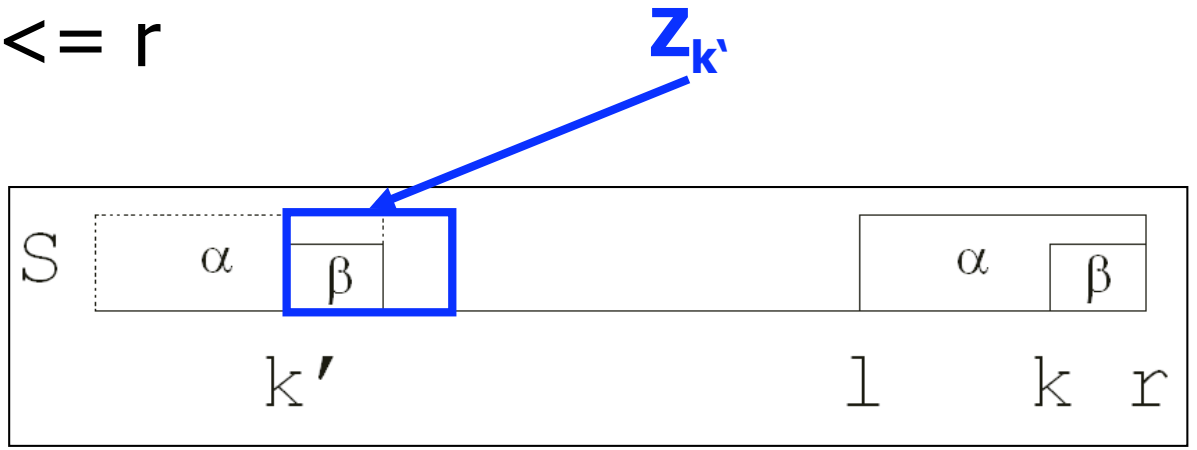


Gegenbeispiel



Z-Algorithmus, Fall 2

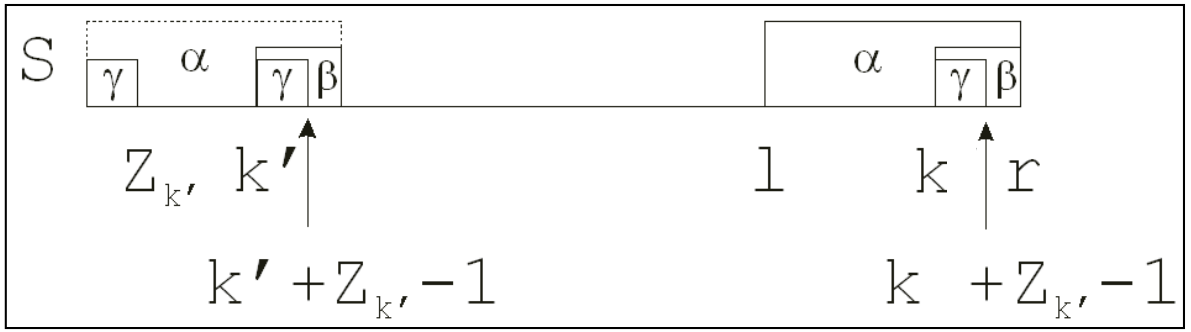
- Möglichkeit 2: k <= r
  - Die Situation:



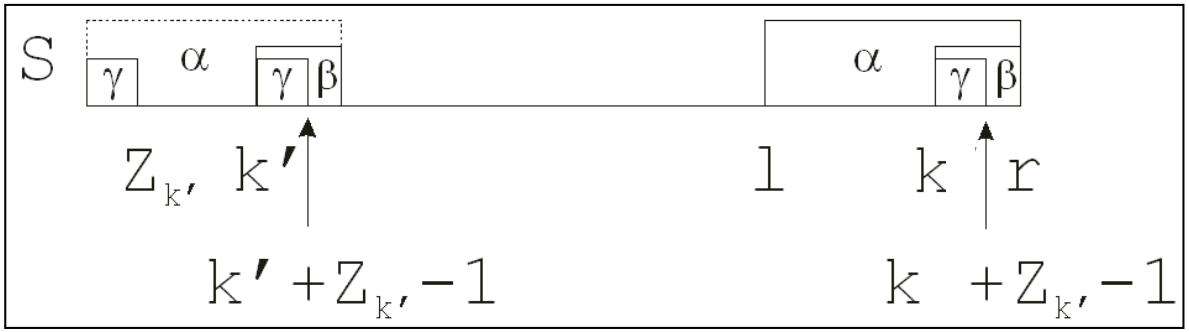
  - Also
    - Z-Box Z_l ist Präfix von S
    - Substring β=S[k..r] kommt auch an Position k'=k-l+1 von S vor
    - Was wissen wir über diesen Substring? Natürlich: $Z_{k'}$
    - $Z_{k'}$ und Z_k können aber länger oder kürzer als $|\beta|$=r-k+1 sein
    - S[r+1..] kennen wir noch nicht; S[k'+1..] schon

Z-Algorithmus, Fall 2.1

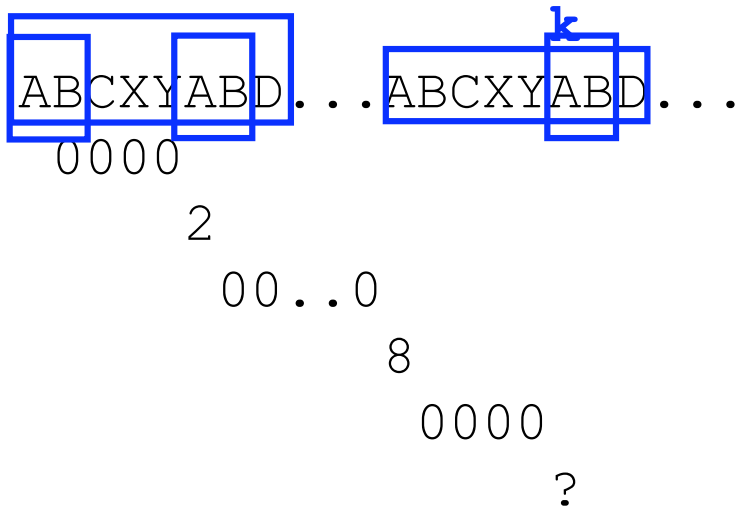
- Fallunterscheidung
  - $Z_{k'} < |\beta| = r-k+1$
    Dann ist das Zeichen an $k'+Z_{k'}$ ein Mismatch bei der Präfixverlängerung. Dann ist das Zeichen $S(k+Z_{k'})$ der gleiche Mismatch. Also:
    $Z_k = Z_{k'}$; r und l unverändert



Beispiel



Beispiel

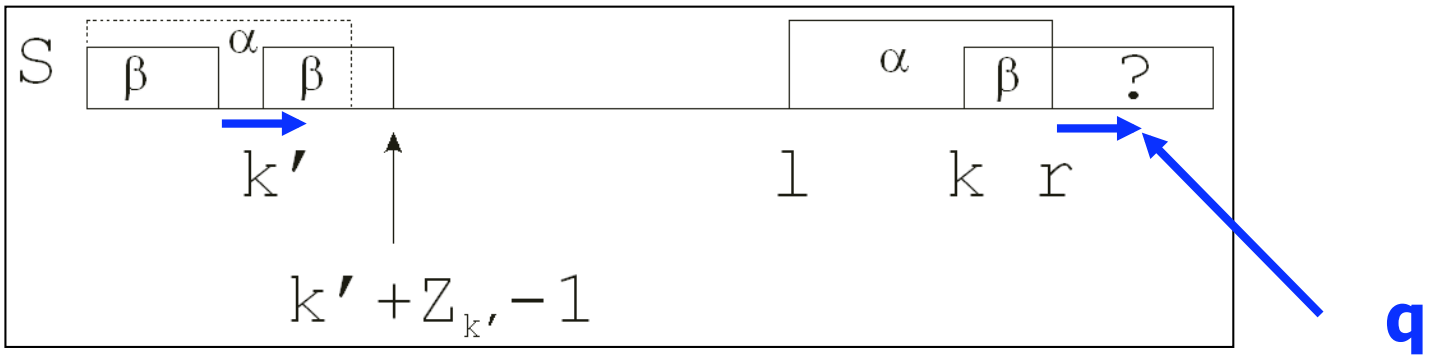


β=|ABD|; k'=6; Z_6=2<|β|

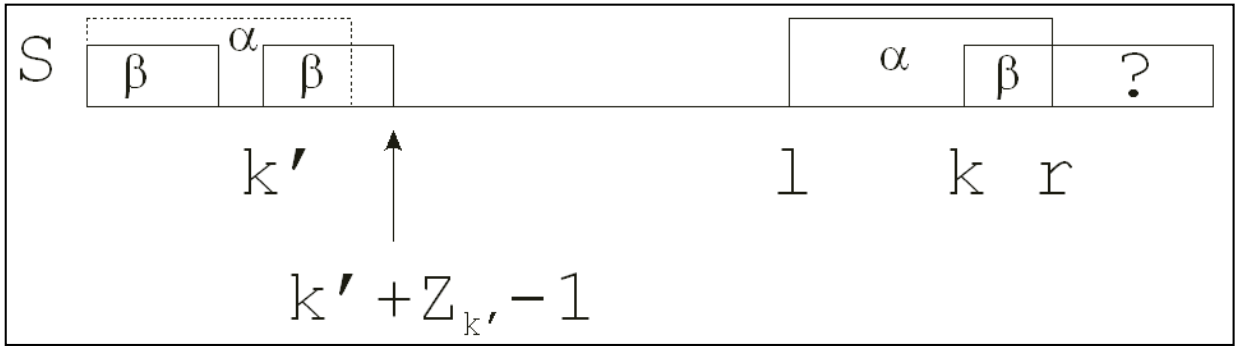
Z-Algorithmus, Fall 2.2

- $Z_{k'} \geq |\beta|$: Dann ist β ein Präfix von S
  - ... dass sich vielleicht noch verlängern lässt
  - Wenn $Z_{k'} > |\beta|$, dann wissen wir: $S(|\beta|+1)=S(k'+|\beta|)$
  - Wir wissen nichts über S(r+1) – dieses Zeichen wurde noch nie betrachtet

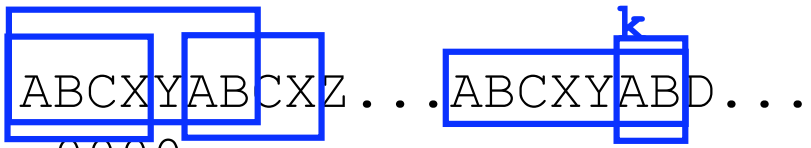
Matche Zeichen-für-Zeichen S[r+1..] mit S[|β|+1..]
Sei der erste Mismatch an Position q
Dann: Z_k=q-k; r=q-1; wenn q≠r+1: l=k



Beispiel



Beispiel



0000

4

00....0

7

0000

?

β=|AB|; k'=6; Z_6=4>|β|

Algorithmus

```
match Z_2;
set l,r;
for k = 3 to |S|
        if k>r then
                match Z_k;
                set r,l;
        else
                k' := k-l+1;
                b := r-k+1;                          // This is |β|
                if Z_k' <b then
                        Z_k := Z_k';
                else
                        match S[r+1.. ] with S[b+1.. ] until q;
                        Z_k := q-k; r := q-1;
                        if q!=r+1 then l := k;
                end if;
        end if;
end for;
```

Komplexität

- Der Algorithmus berechnet alle Z_i-Werte in O(|S|)
- Beweis
  - Idee: Abschätzung der maximalen Anzahl von Matches und Mismatches
  - Beides ist kleiner-gleich |S|
  - Damit ist das Preprocessing O(|S|)
  - Und damit ist der gesamte Algorithmus O(|S|)

Fazit

- Z-Box Algorithmus
  - Findet alle Vorkommen von P in T
  - Berechnung Z Werte für P$T in linearer Zeit
  - Danach alle Vorkommen in linearer Zeit
  - Komplexität O(m+n)
- Als Worst-Case ist das bereits optimal
- Aber
  - Boyer-Moore: Average Case sublinear
  - Knuth-Morris-Pratt: Elegant erweiterbar zu vielen P

12345678901234567890 1
abxyabxz$xabxyabxyabxz

```
k' := k-l+1; b := r-k+1;
Z_k := q-k; l := k; r := q-1;
```

k	Bemerkung	Z_k	l	r
2	Induktionsanfang	0	0	0
3	k>r; Neues Matching, 1 Mismatch	0	0	0
4	k>r; Neues Matching, 1 Mismatch	0	0	0
5	k>r; Neues Matching, 3 Matches, 1 Mismatch	3	5	7
6	$6 \leq 7$; $k'=2;b=2$; $Z_2=0$; Also $Z_{k'}<b$, damit $Z_k=Z_{k'}$	0	5	7
7	$7 \leq 7$; $k'=3;b=1$; $Z_3=0$; Also $Z_{k'}<b$, damit $Z_k=Z_{k'}$	0	5	7
8	8>7; Neues Matching, 1 Mismatch	0	5	7
9	9>7; Neues Matching, 1 Mismatch	0	5	7
10	10>7; Neues Matching, 1 Mismatch	0	5	7
11	11>7; Neues Matching, 7 Matches, 1 Mismatch	7	11	17
12	$12 \leq 17$;$k'=2;b=6,Z_2=0$; $Z_{k'}<b$, damit $Z_k=Z_{k'}$	0	11	17
13	$13 \leq 17$,$k'=3;b=5;Z_3=0$; $Z_{k'}<b$, damit $Z_k=Z_{k'}$	0	11	17
14	$14 \leq 17$,$k'=4;b=4;Z_4=0$; $Z_{k'}<b$, damit $Z_k=Z_{k'}$	0	11	17
15	$15 \leq 17$;$k'=5;b=3;Z_5=3$; Also $Z_{k'} \geq b$; matche S[18..] mit S[4..]; 5 Matches und Erfolg			