

Bioinformatik

Z-Box Algorithmus
Preprocessing eines Strings

Ulf Leser

Wissensmanagement in der
Bioinformatik



Drei Anwendungen

- Sequenzierung
 - Assembly von Teilsequenzen
- cDNA Clustering
 - All-against-all Sequenzvergleiche
- Funktionale Annotation
 - Schnelle Suche in Sequenzdatenbanken

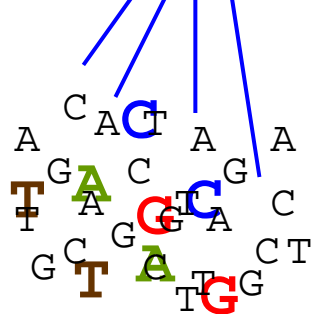
Schritt 3

Primer

Template

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

Polymerase



ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

ACGAACGCGAGTT**A**

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

ACGAACGCGA**G**

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

ACGAACGCGAGTTAGT**T**

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

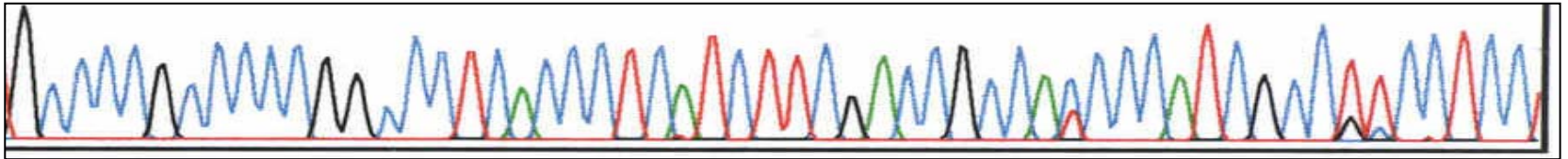
ACGAACGCGAGTTAGTTAG**T**

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

ACGAACGCG**A**

Ergebnis (Zwischenprodukt)

- Signalverarbeitung (Rauschen, ...)



- Übersetzung in **Traces**
 - 4 Arrays, jedes für eine Farbe
 - Intensitätswerte in regelmäßigen Zeitabschnitten
- Theoretisch
 - Peaks entdecken
 - Immer nur eine Farbe
 - Sequenz zuordnen

Abstrakte Formulierung

- **SUPERSTRING**

- Geg.: Menge S von Strings
- Ges.: String T so, dass
 - (a) $\forall s \in S: s \in T$ (s Substring von T)
 - (b) $\forall T'$, für die (a) gilt, gilt: $|T| \leq |T'|$ (T ist minimal)
- NP-vollständig

- **Assembly: Verschärfungen von SUPERSTRING**

- Fehler in Sequenzen (s „ungefähr Substring“ von T)
- Zwei Orientierungen von s möglich

cDNA/EST Bibliotheken

- Erstellung von cDNA Bibliotheken aus Zellen
 - Inhalt abhängig von Gewebe, Entwicklungsstadium, ...
 - Organismusstatus (Krank – Gesund)
- cDNA sequenzieren erzeugt EST
 - Expressed Sequence Tags
 - Ansequenzierte cDNAs
- Wir können also schnell große Mengen von Gensequenzabschnitten erzeugen
 - Aber welchen Genen entspricht das?

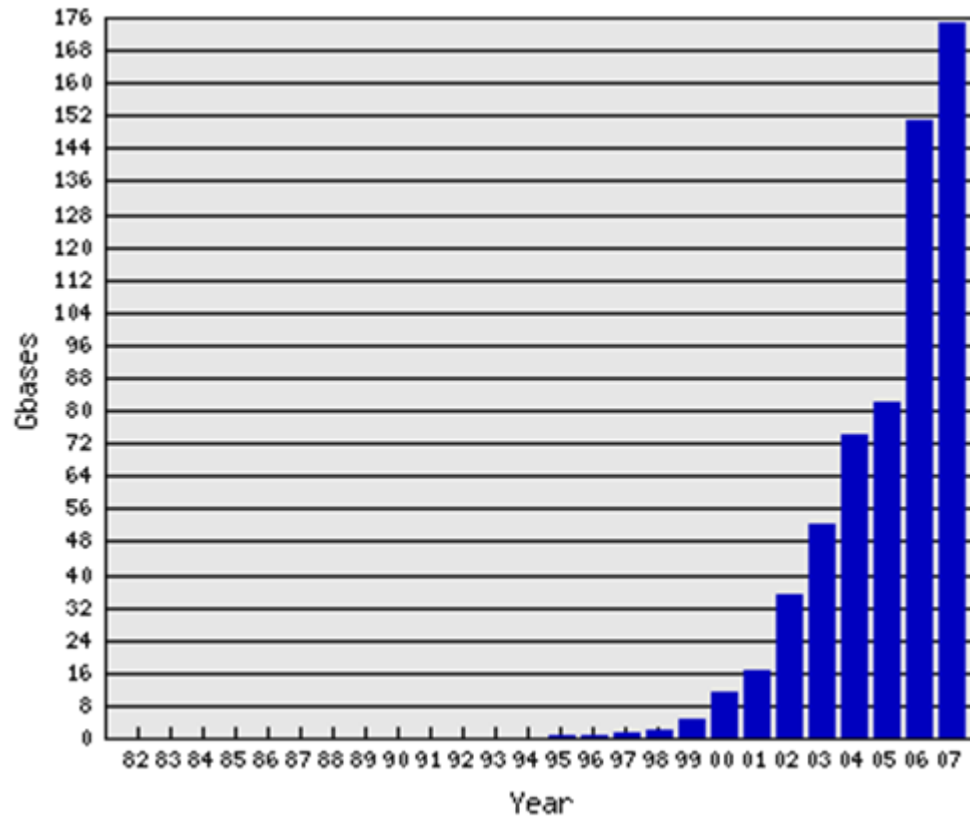
EST Clustering

- Berechnung der Gene durch Clustering
- Typisches Verfahren
 - Schritt 1: Berechnung der Überlappung / Ähnlichkeit aller Sequenzpaare
 - Schritt 2: Berechnung von Sequenzclustern durch Bildung der transitiven Hülle
 - Schritt 3: Zuordnung isolierter Sequenzen zu Clustern mit geringerer Ähnlichkeitsanforderung

3. Funktionale Annotation

- Sequenzen bestimmen Funktionen
 - Gensequenzen => Proteinsequenz
 - Proteinsequenzen => Struktur
 - Struktur => Funktion
- Grundannahme der Bioinformatik
 - Gleiche Sequenzen – gleiche Funktion
 - Sehr ähnliche Sequenzen – sehr ähnliche Funktion
 - Etwas ähnliche Sequenzen – verwandte Funktion?
 - (Stimmt nicht immer)
- Insbesondere wichtig: Comparative Genomics

Problemdimension



Quelle: EMBL, Genome Monitoring Tables, Stand 2007

Exaktes Matching

- Gegeben: P (Pattern) und T (Text)
 - Trivialerweise verlangen wir $|P| \leq |T|$
 - In der Regel nehmen wir an $|P| \ll |T|$
- Gesucht: Sämtliche Vorkommen von P in T
- Beispiel
 - Auffinden der Erkennungssequenzen von **Restriktionsenzymen**

Eco RV - GATATC

```
tcagcttactaattaaaaattctttctagtaagtgctaagatcaagaaaaataaattaaaaataatggaacatggcacattttcctaaactcttcacagattgctaatga
ttattaattaaagaataaatggttataatttttatggtaacggaatttcctaaaatattaattcaagccatggaatgcaataagaaggactctgttaattgggtact
atccaactcaatgcaagtggaaactaagttgggtattaatactctttttacatatatatgtagttattttaggaagcgaaggacaatttcactctgctaataaagggattac
atatttatttttgtgaatataaaaaatagaaagtatggtatcagattaaacttttgagaaaggtaagatgaagtaaagctgtatactccagcaataagttcaaataggc
gaaaaactttttaataacaaaagttaataatcattttgggaattgaaatgtcaaagataattacttcacgataagtagttgaagatagtttaaattttctttttgtatt
acttcaatgaaggtaacgcaacaagattagagtatatatggccaataaggtttgctgtaggaaaattattctaaggagatacgcgagaggggcttctcaaatttattcaga
gatggatggttttagatgggtgggttaagaaaagcagatttaaatccagcaaaactagaccttaggtttattaagcgaggcaataagttaattggaattgtaaaagatat
cctaattcttcttcatttggtaggggaaaactagtttaacttcttaccocatgcagggccataggggtcgaatacgcactgtcactaagcaaaaggaaaaatgtgagtgtagact
ttaaaccatttttattaatgactttagagaatcatgcatttgatgttactttcttaacaatgtgaacatatttatgcgattaagatgagttatgaaaaaggcgaatata
tattcagttacatagagattatagctgggtctattcttagttataggacttttgacaagatagcttagaaaataagattatagagcttaataaaagagaacttcttggaa
tagctgcctttgggtgcagctgtaattggctattgggtatggctccagcttactgggttaggttttaatagaaaaattccocatgattgctaattatatctatcctattgagaa
caacgtgcgaagatgagtggaatgggttcatatttaactgctgggtgctatagtagttatccttagaaagatatataaatctgataaagcaaaatcctggggaaaatat
tgctaactgggtgctggtaggggtttggggatgggattatctcctctacaagaaatttgggtggttactgatataataatagagaaaaaataataaagatgat
```

Naiver Ansatz

1. P und T an Position 1 ausrichten
2. Vergleiche P mit T von links nach rechts
 - Zwei ungleiche Zeichen \Rightarrow Gehe zu 3
 - Zwei gleiche Zeichen
 - P noch nicht durchlaufen \Rightarrow Verschiebe Pointer nach rechts, gehe zu 2
 - P vollständig durchlaufen \Rightarrow Merke Vorkommen von P in T
3. Verschiebe P um ein Zeichen nach rechts
4. Wenn P noch nicht über $|T|-|P|$ hinaus, gehe zu 2

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

Optimierungsideen

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

1. Beobachtung: Zeichen

T xabxyabxyabxz
P abxyabxz
 abxyabxz
 abxyabxz

Nächstes a in T
an Position 6

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

Optimierungsideen 2

2. Beobachtung: Substrings

T xabxyabxyabxz
P abxyabxz
 abxyabxz
 abxyabxz

- Nächstes abx in T an Position 6
- abx doppelt in P

- Algorithmus kann sich interne Struktur von P merken
 - $P[1..3] = P[5..7]$
 - $P[1..3]$ kommt nicht vor Position 5 wieder in P vor
- Vergleich findet: $P[1..7] = T[2..8]$
- Daher muss $P[1..3] = T[6..8]$, und zwischen 2 und 6 kann in T kein Treffer für P liegen
 - 4 Zeichen weit schieben, erst ab Position 4 in P weiter vergleichen

Inhalt dieser Vorlesung

- Z-Boxen
- Exaktes Stringmatching mit Z-Boxen
- Berechnung von Z-Boxen

Z-Algorithmus

- Zerlegung des Problems in **zwei Phasen**
 - **Preprocessing**: Lerne möglichst viel über die Struktur der beiden Strings
 - **Search**: Nutze das Gelernte, um
 - P weiter nach rechts verschieben zu können
 - Bei einem Vergleich von P mit Substring von T nicht an Position 1 von P starten zu müssen
 - [Das passiert im Z-Box Algorithmus nur implizit]
- Auch das Preprocessing muss schnell sein
 - Forderung kann aufgehoben werden, wenn bei vielen Suchen P oder T gleich ist (Datenbankproblem)

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	
Z ₂		C	A	T	A	C	A	C	A	T	A	G	0
Z ₃			A	T	A	C	A	C	A	T	A	G	1
Z ₄				T	A	C	A	C	A	T	A	G	0
Z ₅					A	C	A	C	A	T	A	G	3
Z ₆						C	A	C	A	T	A	G	0
Z ₇							A	C	A	T	A	G	5
Z ₈								C	A	T	A	G	0
Z ₉									A	T	A	G	1
Z ₁₀										T	A	G	0
Z ₁₁											A	G	1
Z ₁₂												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 (Zi(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

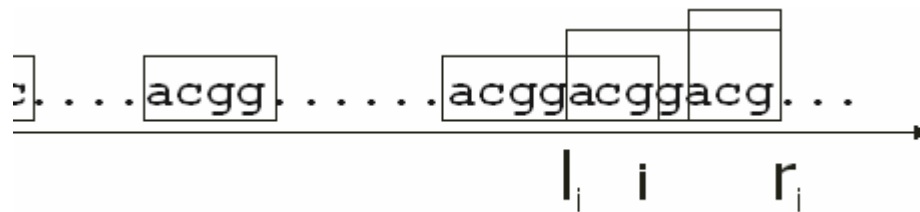
- Naiver Algorithmus zur Berechnung der Z-Boxen braucht $O(|S|^2)$

```
for i = 2 to |S|
    Zi := 0;
    j := 1;
    while ((S[j] = S[i + j - 1]) and (j <= |S|))
        Zi := Zi + 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 eindeutig, da an jeder Position nur eine Z-Box beginnt
 - Im Grunde aber egal – kann Startposition irgendeiner Z-Box sein, die bei r endet
- $S[l_i..r_i]$ ist die Z-Box, die die Position i von S enthält, am weitesten nach rechts reicht und am längsten ist



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

k
CTCGAGTTGCAG
0
1
0
?

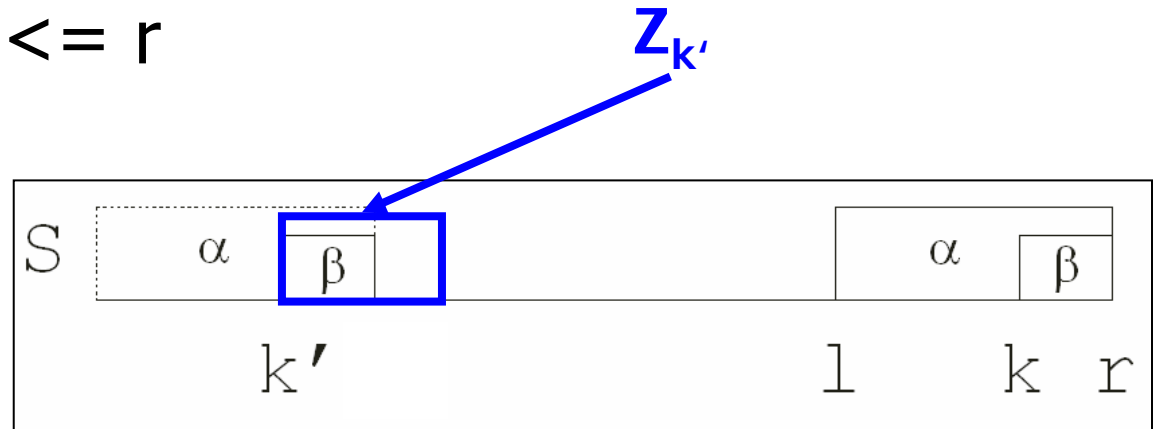
Gegenbeispiel

lk r
CTACTACTTTGCAG
0
0
5
?

Z-Algorithmus, Fall 2

- Möglichkeit 2: $k \leq r$

- Die Situation:



- Also

- Z-Box Z_l ist Präfix von S
 - Substring $\beta = S[k..r]$ 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

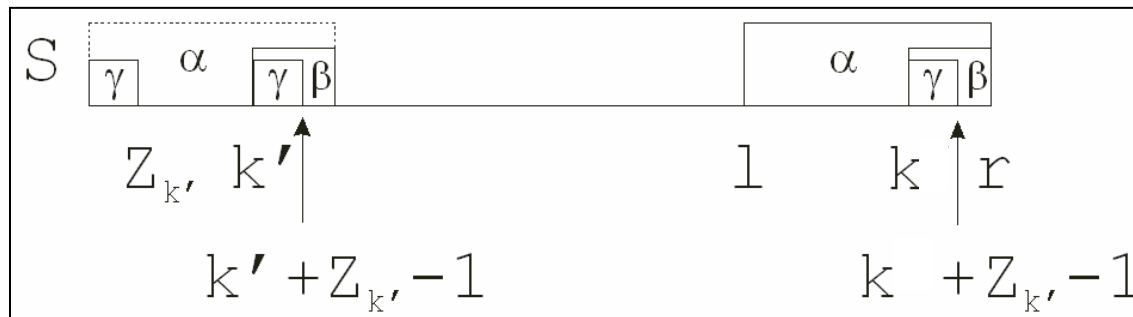
Z-Algorithmus, Fall 2.1

- Fallunterscheidung

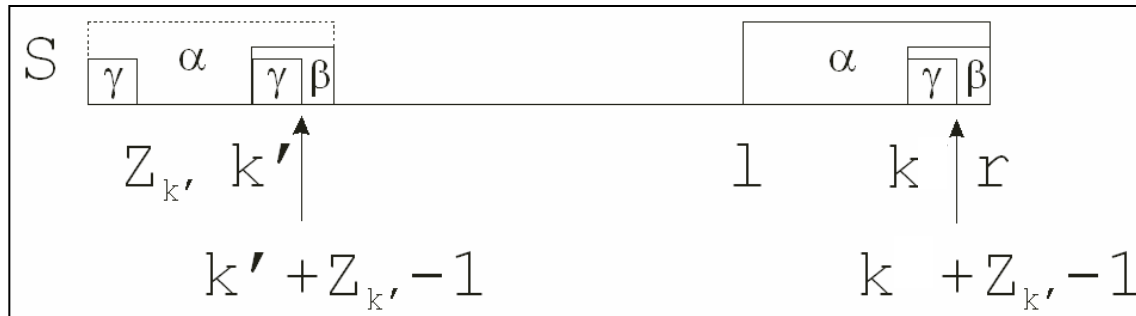
- $Z_{k'} < |\beta| = r - k + 1$

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:

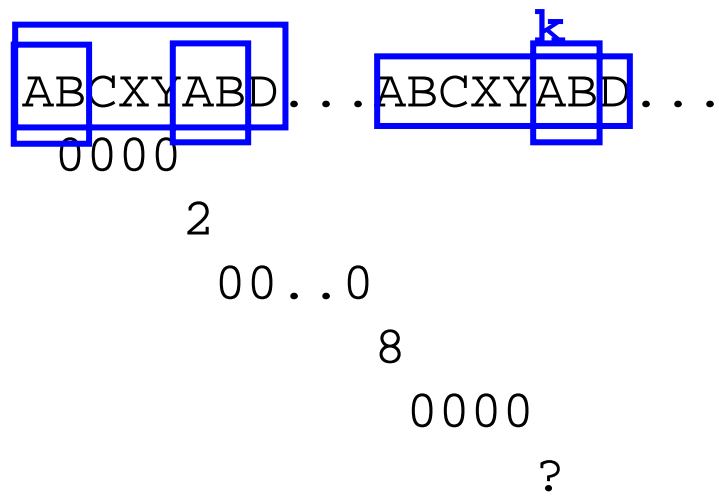
$Z_k = Z_{k'}$; r und l unverändert



Beispiel



Beispiel



$$\beta = |ABD|; k' = 6; Z_6 = 2 < |\beta|$$

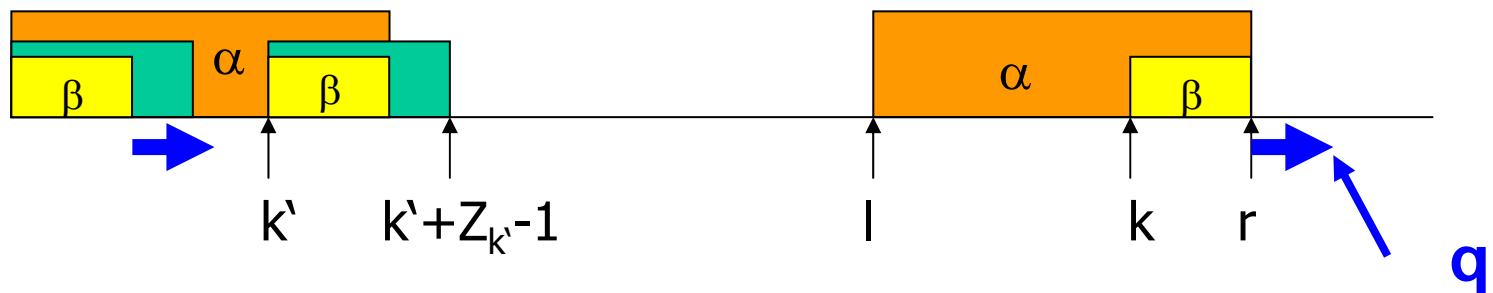
Z-Algorithmus, Fall 2.2

- $Z_{k'} \geq |\beta|$: 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 aber nichts über $S[r+1]$ – dieses Zeichen wurde noch nie betrachtet

Matche Zeichen für Zeichen $S[r+1..]$ mit $S[|\beta|+1..]$

Sei der erste Mismatch an Position q

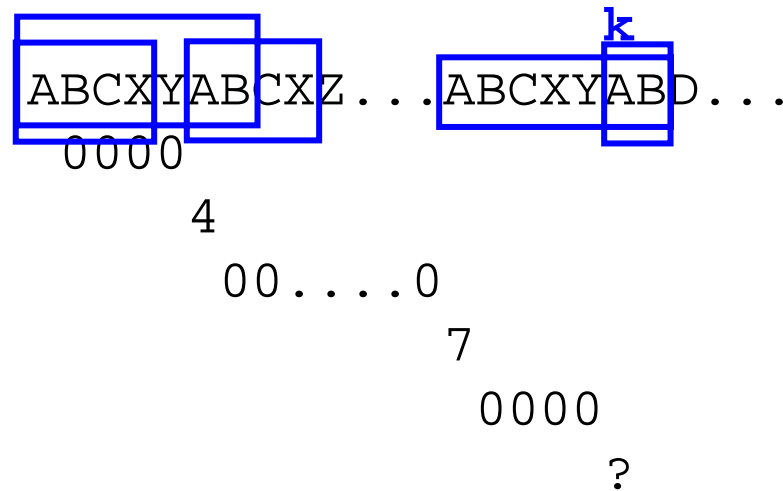
Dann: $Z_k = q - k$; $r = q - 1$; wenn $q \neq r + 1$: $l = k$



Beispiel



Beispiel



$$\beta = |AB|; k' = 6; Z_6 = 4 > |\beta|$$

Algorithmus

```
match Z2;
set l,r;
for k = 3 to |S|
  if k>r then
    match Zk;
    set r,l;
  else
    k' := k-1+1;
    b := r-k+1; // This is |β|
    if Zk'<b then
      Zk := Zk';
    else
      match S[r+1.. ] with S[b+1.. ] until q;
      Zk := q-k; r := q-1; l := k;
    end if;
  end if;
end for;
```

Komplexität

- Theorem

Der Z-Box Algorithmus berechnet alle Z-Werte in $O(|S|)$

- Beweis

- Der Algorithmus ist mindestens $O(|S|)$ durch die FOR-Schleife
- Wir zählen m = „Anz. Matches“ und m' = „Anz. Mismatches“
 - Erst m' . Wie viele Mismatches gibt es pro k ?
 - Induktionsanfang – Maximal einen
 - Fall 1
 - » Maximal einen
 - Fall 2.1
 - » 0; Es werden überhaupt keine Zeichen verglichen
 - Fall 2.2
 - » Maximal einen
 - Also kann **pro Position in S maximal ein Mismatch** auftreten
 - Also gilt: $m' \leq |S|$

Komplexität 2

- Beweisfortsetzung

- Jetzt m. Wann führt der Algorithmus Matches aus?

- Induktionsanfang – Egal für Komplexität (max. $|S|$)

- Fall 1

- » Nehmen wir x Matches an, dann Mismatch; r wird nach rechts verschoben, Fälle 2.1 und 2.2 werden eintreten

- Fall 2.1

- » Es werden keine Zeichen verglichen

- Fall 2.2

- » Verglichen wird nur rechts von r , und damit rechts vom letzten Match

- Also kann ein Zeichen höchstens einmal einen Match erzeugen

- Also gilt: $m \leq |S|$

- Qed.

Alles zusammen

- Die Z Werte kann man in $O(|S|)=O(m+n)$ berechnen
- Danach muss man in $O(|S|)$ alle passenden Z-Boxen suchen
- Damit löst der Z-Box Algorithmus das exakte Stringmatchingproblem in $O(m+n)$

123456789012345678901
 abxyabxz\$xabxyabxyabxz

$$k' := k-1+1; b := r-k+1;$$

$$Z_k := q-k; l := k; r := q-1;$$

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			

1234567890123456
 aaaat\$aaaaaaaaaaa

$$k' := k-1+1; b := r-k+1;$$

$$Z_k := q-k; l := k; r := q-1;$$

k	Bemerkung	Z_k	l	r
2	Induktionsanfang	3	2	4
3	$k < r$; $k'=2$; $b=2$; $Z_2=3$; $Z_k \geq b$; matche S[5..] mit S[3..]; 1 Mismatch; $q=5$	2	3	4
4	$k \leq r$; $k'=3$; $b=1$; $Z_3=2$; $Z_k \geq b$; matche S[5..] mit S[3..]; 1 Mismatch; $q=5$	1	4	4
5	$k > r$; Neues Matching, 1 Mismatch	0	4	4
6	$k > r$; Neues Matching, 1 Mismatch	0	4	4
7	$k > r$; Neues Matching, 4 Matches, 1 Mismatch	4	7	10
8	$8 \leq 10$; $k'=2$; $b=3$; $Z_2=3$; $Z_k \geq b$; matche S[11..] mit S[4..]; 1 / 1; $q=12$	4	8	11
9	$9 \leq 11$; $k'=2$; $b=3$; $Z_2=3$; $Z_k \geq b$; matche S[12..] mit S[4..]; 1 / 1; $q=12$	4	9	12
10	$10 \leq 12$; ...	4	10	13
..

Fazit

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