Exaktes Pattern-Matching

Tutorium Algorithmen &
Datenstrukturen für Bioinformatik

Utz Pape; Adrian Haß

Boyer Moore Algorithmus

- Bad-Character-Rule (BC)
  - -Beispiel
  - -Vorverarbeitung
- Good-Suffix-Rule (GS)
  - -Beispiele
  - -Vorverarbeitung
- Anwendung beider Regeln

Ext. Bad Char. - Rule

- Gegeben:

T Text; P Pattern: |T|=m;|P|=n

Ext. Bad Char. - Rule

- Gegeben:

T Text; P Pattern: |T|=m;|P|=n

- Alignment: T[p...p+n]~P

Ext. Bad Char. - Rule

- Gegeben:

T Text; P Pattern: |T|=m;|P|=n

- Alignment: T[p...p+n]~P
- 1. Mismatch: T[k] != P[i]

Ext. Bad Char

- Suche rechtestes Auftreten von T[k] in P*, dass links von i liegt.*

eBC-Rule

```
   0         1
   12345678901
T: ACgAgACAggT
     X|
P: gACA
```


eBC-Rule

```
   0         1
   12345678901
T: ACgAgACAggT
       X|
P:   gACA
```

eBC-Rule

```
    0          1
    12345678901
T:  ACgAgACAggT
        ||||
P:      gACA
```


Pattern-Vorverarbeitung eBC-Rule

Σ={A,C,g,T} P=gACA

Liste mit Positionen(←) in P:

- A:{4,2}
- C:{3}
- G:{1}
- T:{}

eBC-Rule

Abfrage:

Durchsuche Liste von T[k] nach Wert kleiner i!

Strong Goog Suffix - Rule

- Suche nächstes Auftreten P[i-1..n] (Suffix) in P*, wobei der nächste Buchstabe != P[i] ist.*

Strong Goog Suffix - Rule

- Suche nächstes Auftreten P[i-1..n] (Suffix) in P*, wobei der nächste Buchstabe != P[i] ist.*

Falls nicht vorhanden:

Strong Goog Suffix - Rule

- Suche nächstes Auftreten P[i-1..n] (Suffix) in P*, wobei der nächste Buchstabe != P[i] ist.*

Falls nicht vorhanden:

- Suche Prefix von P (P[1..l]), der Suffix von P[n-l..n] entspricht.

sGS-Rule

1.Fall: Wir finden P[i..n] in P ganz

```
    0        1
    12345678901
 T: gTACgATgACT
       X||
 P: CgATgA
```


sGS-Rule

1.Fall: Wir finden P[i..n] in P ganz

```
    0        1
    12345678901
 T: gTACgATgACT
       ||||||
 P:    CgATgA
```

sGS-Rule

2.Fall: Suchen Prefix(P) = Suffix(P)

```
    0        1
    123456789012
T:  gTACTgATTgAC
       X|||
P:   gATTgA
```


sGS-Rule

2.Fall: Suchen Prefix(P) = Suffix(P)

```
    0        1
    123456789012
T:  gTACTgATTgAC
          ||||||
P:        gATTgA
```


Pattern-Vorverarbeitung sGS-Rule

Beobachtung:

- P[i..n] soll gleich P[k..(k+n-i)] sein

Dreht man P um, so erhält man

- P[1..(n-i)] soll gleich P[l..(l+n-i)] sein

$\rightarrow$ Z-Algorithmus

Anwendung beider Regeln

- Nehmen immer max{eBC,sGS,1}

Beide Regeln

```
P: CgACCg
T: ACAgACTgTCACggACCgACCg
```

- Sprungliste P:
  A:{3};g:{6,2};C:{5,4,1};T:{}
- Prefix-Suffix:
  geschultes Auge...

Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
       X
P: CgACCg
```


Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
      X
P: CgACCg

GS:1
BC:1
            →1
```


Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
        X
P:  CgACCg
```

Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
         X
P:  CgACCg

GS:1
BC:6 (T kommt nicht vor)
              →6
```


Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
              X||
P:        CgACCg
```


Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
            X||
P:        CgACCg

GS:4
BC:1
              →4
```


Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
              ||   X
P:            CgACCg
```

Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
              ||   X
P:            CgACCg

GS:1
BC:1
              →1
```


Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
                X|||||
P:              CgACCg
```


Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
               X|||||
P:             CgACCg

GS:4
BC:1
              →4
```


Beide Regeln

```
T: ACAgACTgTCACggACCgACCg
                     ||||||
P:                   CgACCg

              MATCH!
```

ENDE