Algorithmen und Datenstrukturen

Suchen in Zeichenketten

Aufgabenstellung

- Suche einer Zeichenkette in einem langen Text
- Klassische Suchfunktion vieler Programme
- Gesuchte Zeichenkette ist fest ohne Platzhalter
- Text ist oft größer als Hauptspeicher
- Textlänge sei n, Musterlänge sei m: m << n
- Erweitert: Reguläre Ausdrücke => Automaten

Schrittweise Probieren

- Einfachster Ansatz
- Teste an jeder Position, ob der Suchtext paßt
- Wenn nicht, versuche es einen Schritt weiter
- Einfach zu implementieren
- Laufzeit O(m*n)

Schrittweise probieren

```
find' k [] = False
find' k xs@(_:xs')
    | startsWith k xs    = True
    | otherwise          = find' k xs'
startsWith :: Eq a => [a] -> [a] -> Bool
startsWith (x:xs) (y:ys)  =  x == y &&
                                 startsWith xs ys
startsWith xs     _          = null xs
```

Knuth-Morris-Pratt

- Ausnutzung des bereits analysierten Textes
- Verschiebung des Suchmusters
- Keine Rückschritte bei der Verarbeitung
- Jedes Zeichen nur einmal ansehen
- Aufbau als Automat möglich
- Laufzeit O(n + m)
- Suchzeit unabhängig vom Suchmuster

Knuth-Morris-Pratt

```
findKMP :: Eq a => [a] -> [a] -> Bool
findKMP key xs = kmp key xs
  where
    kmp ks@(k:ks') (x:xs')
      | k == x       = kmp ks'                  xs'
      | otherwise  = kmp (rewind x ks)   xs'
    kmp ks _ = null ks
```

Knuth-Morris-Pratt

```
rewind x rest = findmatch (take len key ++ [x]) key
    where  len = length key - length rest

findmatch ms ks
      | startsWith ms ks    = drop (length ms) ks
      | otherwise           = findmatch (tail ms) ks
```

Diese Funktionen sind komplett vorausberechenbar.

Boyer-Moore

- Umkehrung der Suchrichtung
- Funktionsweise wie Knuth-Morris-Pratt
- Erzeugung großer Sprünge
- Rückwärtsbewegung im Text nötig
- Laufzeit: O(n/m)
- Je größer das Muster, desto schneller die Suche