

■ Boyer-Moore Algorithmus

1 Grundidee

Der **Boyer-Moore** Algorithmus sucht ein **Pattern** P in einem **Text** T, indem er Vergleiche **von rechts nach links** im Pattern macht und bei einem Mismatch das Pattern **weit nach rechts verschiebt**.

Kernideen (Heuristiken):

- **Bad Character Rule** (schlechtes Zeichen)
- **Good Suffix Rule** (gutes Suffix)

→ In der Praxis oft sehr schnell, besonders bei großen Alphabeten und langen Patterns.

2 Voraussetzungen

- Text T und Pattern P sind Strings
 - Vorverarbeitung des Patterns (Tabellen für Verschiebungen)
-

3 Laufzeiten & Eigenschaften

| | | | | | | | | | | | |
|----------------------|------|-------|-------|-------------------------------|----------------------------------|--|--|--|-----------------------------|--|--|
| Eigenschaft | Wert | ----- | ----- | Best / typische Praxis | sehr schnell (sublinear möglich) | | Worst Case O(n·m) (ohne Good-Suffix) | | Speicherbedarf O(Σ + m) | | |
| Backtracking im Text | nein | | | Anwendungsfall String-Suche | | | | | | | |

Hinweis: Mit beiden Heuristiken ist die Worst-Case-Garantie besser, aber die Implementierung wird deutlich komplexer. In Prüfungen wird oft die **Bad Character Rule** fokussiert.

4 Bad Character Rule (Prüfungsfokus)

Idee

Wenn ein Mismatch passiert (Textzeichen passt nicht zum Patternzeichen), kann das Pattern so verschoben werden, dass:

- das mismatching Zeichen im Pattern **unter seiner letzten Vorkommen-Position** liegt
- oder, falls es nicht vorkommt, das Pattern komplett darüber hinweg springt

Dazu wird eine Tabelle `last[char]` berechnet:

- `last[c]` = letzter Index von Zeichen c im Pattern (oder -1)
-

5 Schritt-für-Schritt-Beispiel (Bad Character)

Text:

HERE IS A SIMPLE EXAMPLE

Pattern:

EXAMPLE

Vergleich startet am Pattern-Ende (rechts). Bei Mismatch wird anhand von `last[text_char]` die Verschiebung bestimmt.

Boyer-Moore – Algorithmus in Worten (Grundidee)

Der Boyer-Moore-Algorithmus ist ein String-Suchalgorithmus.

Er sucht ein Muster in einem Text.

Im Gegensatz zu einfachen Suchverfahren vergleicht er nicht von links nach rechts, sondern beginnt den Vergleich am Ende des Suchmusters.

Durch gezieltes Überspringen von Zeichen kann der Algorithmus viele Vergleiche einsparen und ist in der Praxis sehr schnell.\

Boyer-Moore – Suchablauf in Worten

Der Algorithmus richtet das Suchmuster unter dem Text aus.

Der Vergleich beginnt beim letzten Zeichen des Musters.

Stimmen die Zeichen überein, wird nach links weiter verglichen.

Kommt es zu einer Abweichung, wird das Muster nach rechts verschoben.

Die Verschiebung erfolgt anhand vorberechneter Regeln.

Dabei wird das Muster oft um mehrere Zeichen weitergeschoben.

Dieser Vorgang wird wiederholt, bis das Muster gefunden wird

oder das Ende des Textes erreicht ist.\

Bad-Character-Regel – Algorithmus in Worten (sehr prüfungsrelevant)

Tritt ein Zeichenfehler auf, wird das fehlerhafte Textzeichen betrachtet.

Es wird geprüft, ob dieses Zeichen im Suchmuster vorkommt.

Kommt es nicht vor, kann das Muster vollständig an diesem Zeichen vorbeigeschoben

werden.

Kommt es vor, wird das Muster so verschoben,
dass dieses Zeichen im Muster mit dem Textzeichen ausgerichtet ist.
Dadurch werden unnötige Vergleiche vermieden.\

Good-Suffix-Regel – Algorithmus in Worten (fortgeschritten)

Wenn ein Teil des Musters bereits korrekt übereinstimmt,
wird dieser übereinstimmende Suffix betrachtet.

Das Muster wird so verschoben,
dass ein weiterer Vorkommensort dieses Suffixes genutzt wird.
Existiert kein passender Suffix, wird das Muster weiter verschoben.
Auch diese Regel trägt zur Reduktion der Vergleiche bei.\

6 Good Suffix Rule (kurz, der Vollständigkeit halber)

Wenn ein Suffix im Pattern bereits gematcht hat und dann ein Mismatch kommt, verschiebt man das Pattern so, dass:

- das gematchte Suffix erneut passend ausgerichtet wird
- oder ein passender Prefix genutzt wird

→ Diese Regel macht Boyer–Moore noch schneller, ist aber komplexer.

7 Besonderheiten / Prüfungsrelevante Hinweise

- Boyer–Moore vergleicht **von rechts nach links**
 - Typische Prüfungsfrage:
 - *Wie wird die Verschiebung bei Bad Character berechnet?*
 - Häufig reicht eine Implementierung der Bad-Character-Heuristik
-

8 Vor- und Nachteile

Vorteile

- in der Praxis extrem schnell
- wenige Vergleiche (sublinear möglich)
- gut bei großem Alphabet (ASCII/Unicode)

Nachteile

- komplexer als KMP
 - Worst Case kann ohne Good-Suffix schlecht sein
-

Merksatz für die Prüfung

Boyer-Moore sucht von rechts nach links und verschiebt das Pattern mit Bad-Character- und Good-Suffix-Regeln weit nach rechts.

9 Python-Implementierung (Bad Character Heuristik)

```
In [1]: def build_last(pattern):  
    last = {}  
    for i, ch in enumerate(pattern):  
        last[ch] = i  
    return last  
  
def boyer_moore_search(text, pattern):  
    if not pattern:  
        return []  
  
    last = build_last(pattern)  
    n, m = len(text), len(pattern)  
    result = []  
  
    i = 0 # Startindex im Text für das aktuelle Pattern-Fenster  
    while i <= n - m:  
        j = m - 1 # Vergleich startet rechts im Pattern  
  
        while j >= 0 and pattern[j] == text[i + j]:  
            j -= 1  
  
        if j < 0:  
            result.append(i)  
            # Verschiebe um 1 oder um m (hier: 1, um weitere Matches zu finden)  
            i += 1  
        else:  
            bad_char = text[i + j]  
            shift = j - last.get(bad_char, -1)  
            i += max(1, shift)  
  
    return result  
  
# Beispiel  
text = "HERE IS A SIMPLE EXAMPLE"  
pattern = "EXAMPLE"  
print(boyer_moore_search(text, pattern)) # [17]
```

