# Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

## Vorlesung 13, Montag, 31. Januar 2011

(String Matching)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

# Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Ihre Erfahrungen mit dem Ü12 (Editierdistanz)
  - Das ist heute die vorletzte Vorlesung
  - Die Klausur ist immer noch am 30. März 14:00 – 16:30 Uhr
- String Matching
  - Das was man für std::string::find machen muss
  - Wir lernen drei Algorithmen dazu kennen
    - den trivialen und zwei ziemlich clevere
  - **Übungsblatt 13:**
    - Laufzeit für die ersten beiden, Programm für den dritten

# Ihre Erfahrungen mit dem Ü12 (Edit.dist)

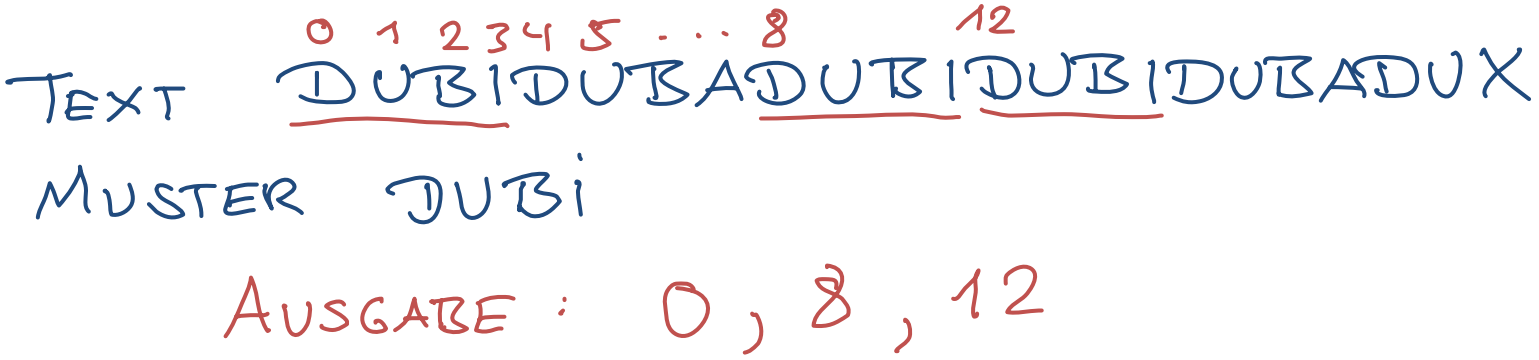
UNI FREIBURG

- Zusammenfassung von Ihrem Feedback **[31.1 14:48]**
  - Das Verfahren haben fast alle gut verstanden
  - Und fanden es auch relativ leicht zu implementieren
  - Das Schwierigste war die Folge der Operationen
  - Den Beweis fanden die meisten schwierig und haben ihn aus Zeitgründen weggelassen

# String Matching — Definition

UNI FREIBURG

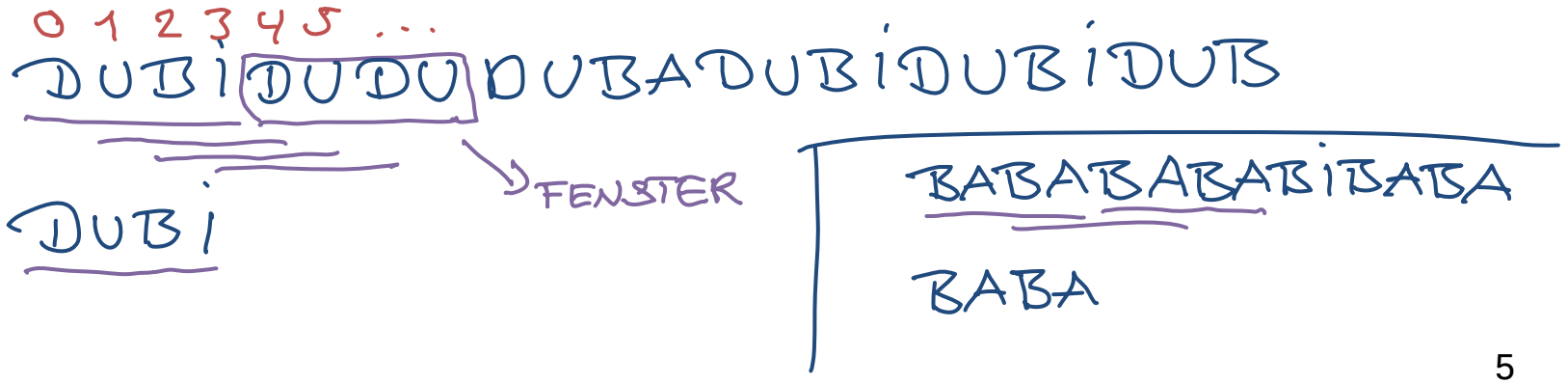
- Definition
  - Gegeben zwei Zeichenketten / strings:
    - ein (typischerweise langer) Text (engl. text)
    - ein (typischerweise kurzes) Muster (engl. pattern)
  - Finde alle Vorkommen des Musters im Text
- Notation
  - die Länge des Textes nennen wir n
  - die Länge des Musters nennen wir m



# String Matching — Naiver Algorithmus

UNI FREIBURG

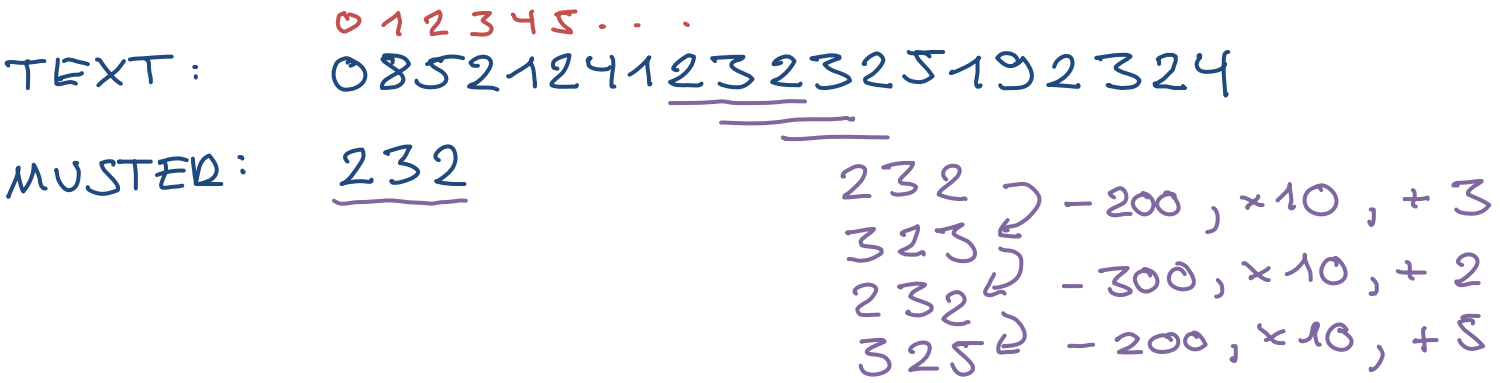
- Beschreibung des Algorithmus
  - Gehe den Text von links nach rechts durch
  - Prüfe an jeder Stelle ob das Muster passt, indem man es Buchstabe für Buchstabe mit dem Text dort vergleicht
  - Den jeweiligen Ausschnitt (der Größe m) aus dem Text nennen wir Fenster (engl. window)
  - Laufzeit offensichtlich $O(n \cdot m)$
    - im besten Fall auch schneller → Übungsaufgabe



# String Matching — Rabin-Karp

UNI FREIBURG

- Idee des Algorithmus
  - Wir schieben wieder ein Fenster der Größe m über den Text
  - Und schauen an jeder Stelle ob es zu dem Muster passt
  - Nehmen wir an die Buchstaben sind die Ziffern 0..9
  - Dann kann man das Muster als (große) ganze Zahl auffassen
  - Und das Fenster ebenso
  - Wenn wir das Fenster um eins nach rechts verschieben, kann man die Zahl für das neue Fenster leicht aus der Zahl für das alte Fenster berechnen:



# String Matching — Rabin-Karp

- Laufzeit
  - Wenn wir mit den Zahlen in konstanter Zeit operieren könnten, wäre die Laufzeit nur $O(n)$
  - Aber diese Zahlen können sehr groß werden
- Zusätzlicher Trick
  - Wir rechnen mit den Zahlen modulo einer Primzahl $p$
  - Muster und Fenster sind dann Zahlen aus $\{0, \ldots, p-1\}$
  - Wenn das Muster zu einem Fenster passt, sind die Zahlen gleich, aber wenn es nicht passt, können sie auch gleich sein (auch wenn es "unwahrscheinlich" ist)
  - Wenn die Zahlen gleich sind (aber nur dann), überprüfen wir wie beim naiven Algorithmus Buchstabe für Buchstabe
  - Laufzeit jetzt → Übungsaufgabe

# String Matching — Rabin-Karp



```
          0 1 2 3 4 5 . . .
TEXT      5 7 8 2 3 2 3 2 5 4 2 3 2 4 2 1
MUSTER    2 3 2
```

$p = 5$

$578 \bmod 5 = 3$
$232 \bmod 5 = 2$ $\neq$

$782 \bmod 5 = 2$
$232 \bmod 5 = 2$ $=$ , aber kein Match

$\vdots$

$232 \bmod 5 = 2$
$232 \bmod 5 = 2$ $=$ → AUSGABE: 3

$x \cdot y \bmod p$
$= (x \bmod p)(y \bmod p)$
Stichwort: $\mathbb{Z}/p\mathbb{Z}$

# String Matching — Knuth-Morris-Pratt

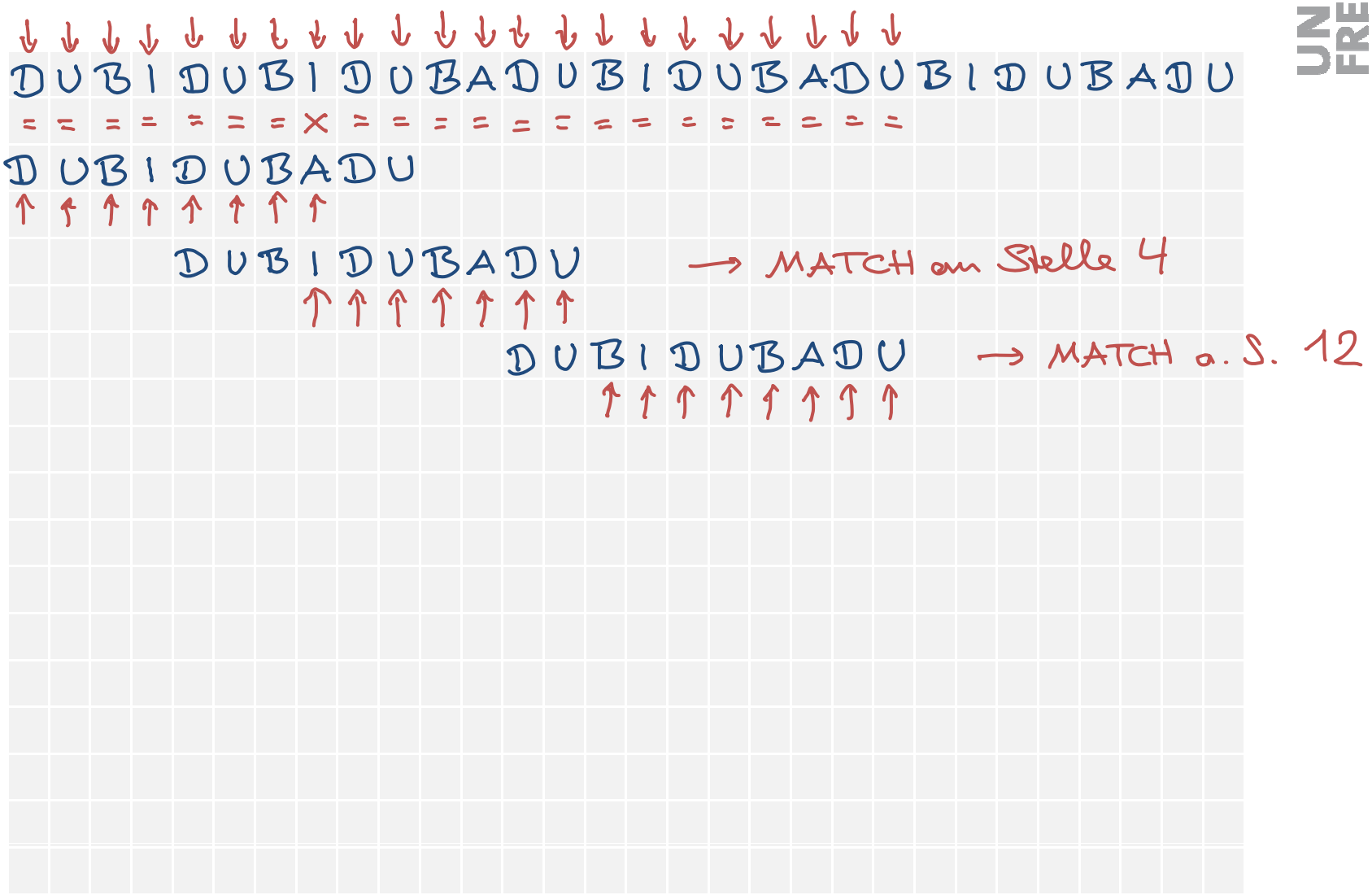
UNI FREIBURG

- Idee des Algorithmus
  - Wenn man die ersten k Zeichen des Musters mit den ersten k Zeichen eines Fensters im Text verglichen hat

  ... und jetzt das Fenster im Text um eins weiter schiebt

  ... dann hat man k-1 Zeichen dieses Fensters schon mal mit dem Muster verglichen
  - Man möchte gerne vermeiden, die nochmal anzuschauen
  - Wie das gehen könnte, sieht man am besten an einem Beispiel → nächste Folie

# String Matching — Knuth-Morris-Pratt



# String Matching — Knuth-Morris-Pratt

- Vorverarbeitung des Musters
  - Für jede Stelle $j \in \{1,\ldots,m\}$ des Musters $P$ berechnen wir **vor**, an welcher Stelle des Musters wir weitermachen, wenn es an Stelle $j+1$ einen Mismatch gibt oder wenn $j = m$
  - Für Stelle $j$ ist das $shift[j] = \min \{ k : P[j-k..j-1] = P[1..k] \}$
  - Das lässt sich leicht in Zeit $O(m)$ berechnen → Üb.aufgabe
  - Man sieht leicht, dass $shift[j-1] < j$

MUSTER
SHIFTS

D U B I D U B A D U
0 0 0 0 1 2 3 0 1 2

A B C D E F
0 0 0 0 0 0

D U D U B D U D U
0 0 1 2 0 1 2 3 4

A A A A A A A
0 1 2 3 4 5 6

# String Matching — Knuth-Morris-Pratt

UNI FREIBURG

- Laufzeit
  - Sei i der Index des aktuellen Zeichens im Text
  - In jeder Iteration erhöhen wir i oder es bleibt gleich
  - Wenn i gleich bleibt, verschieben wir das Muster im Bild (siehe Beispiel Folie 10) um mindestens eins nach rechts
    - die Verschiebung ist gerade $j - shift[j-1] > 0$
  - Da man höchstens n mal nach rechts gehen kann, gibt es also höchstens 2n Iterationen
  - In jeder Iteration gibt es nur konstant viele Operationen
  - Damit ist die Laufzeit O(n)

# String Matching — Geschichte

UNI FREIBURG

- Rabin-Karp Algorithmus
  - Richard Karp und Michael Rabin

    Efficient randomized pattern matching

    1987 IBM Journal of Research and Development

- Knuth-Morris-Pratt Algorithmus
  - Donald Knuth und James Morris und Vaughan Pratt

    Fast pattern matching in strings

    1977 SIAM Journal on Computing

# Literatur / Links

- String Matching
  - In Mehlhorn/Sanders:

    gar nicht!
  - In Wikipedia

    http://de.wikipedia.org/wiki/String-Matching-Algorithmus

    http://de.wikipedia.org/wiki/Rabin-Karp-Algorithmus

    http://de.wikipedia.org/wiki/Knuth-Morris-Pratt-Algorithmus

    + auf den Englischen Seiten steht wie immer mehr
  - Interaktive Demo

    http://tinyurl.com/6hs46c7