Informatik II: Algorithmen und Datenstrukturen SS 2013

Vorlesung 12a, Dienstag, 9. Juli 2013

(String Matching, Naiver Algorithmus + Rabin-Karp)

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 Ü11 (Editierdistanz)
  - Offizielle Evaluation dieser Veranstaltung
- String Matching
  - Finde alle Vorkommen eines (in der Regel kurzen) Musters in einem (in der Regel langen) Text
  - Heute: Naiver Algorithmus + Rabin-Karp
  - Morgen: Knuth-Morris-Pratt
  - Übermorgen: Donnerstag
  - Ü12, Aufgabe 1: Evaluation, dafür gibt's **10 Ü-Punkte** !

Erfahrungen mit dem Ü11 (Editierdistanz)

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 9. Juli 16:00
  - Aufgabe 1 (Programm) war gut machbar
  - Aufgabe 2 (verschiedene Wege) war schön zum Knobeln
  - Keine Code-Vorlage diesmal … es gab nichts zum Vorlegen
  - Schlechtes Gewissen, weil mitverantwortlich für meine Frustration gefühlt
  - Wetter war zu schön, Hochzeitsfeier in der Familie, Klausurzulassung schon erreicht, Prokrastination lässt grüßen, auf's Hardware-Praktikum konzentriert
  - "Ich gehör zu denen, die das dann für die Klausur machen, sorry, Ü-Aufgaben werden aber spätestens dann gemacht !"

Musterlösung Ü11, Aufgabe 2, Teil 2

- Möglichst viele verschiedene Wege

  $x = a \dots a$ (k mal) $\quad y = b \dots b$ (n mal) $\quad k \le n$

  – $x = a^k$ (k mal a), $y = b^n$ (n mal b)

  – ED(x, y) = n : k mal replace(a → b), n – k mal insert(b)

  – Beobachtung: Es gibt viel Spielraum, an welchen Positionen man die insert Operationen macht

  – Genauer: es gibt so viele Möglichkeiten, wie es verschiedene Tupel $(i_1, \dots, i_k)$ gibt mit $1 \le i_1 < i_2 < \dots < i_k \le n$

  Das sind die Positionen der k replace Operationen, daraus ergeben sich dann die Positionen der n – k insert Operat.

  das ist gerade $\binom{n}{k}$ = # k-elementiger Teilmengen von $\{1, \dots, n\}$

  Für die Aufgabe $n + k = 9$

  $\binom{8}{1} = 8$, $\binom{7}{2} = \frac{7 \cdot 6}{2 \cdot 1} = 21$, $\binom{6}{3} = \frac{6 \cdot 5 \cdot 4}{3 \cdot 2 \cdot 1} = 20$, $\binom{5}{4} = 5$

  MAX

Offizielle Evaluation der Vorlesung

UNI FREIBURG

- Bitte den Bogen **bis Ende der Woche** abgeben
  - Ich werde das Feedback dann nächste Woche (= letzte Vorlesungswoche) zusammenfassen und besprechen !
  - Sie bekommen dafür 10 wunderschöne Punkte
  - Schreiben Sie dazu einfach in Ihre erfahrungen.txt, dass Sie den Evaluationsbogen ausgefüllt haben (wenn es so ist)
  - Nehmen Sie sich bitte genug Zeit für das Ausfüllen
  - Die Freitextkommentare sind für uns am interessantesten
  - Seien Sie bitte ehrlich und möglichst konkret
  - **Abgabe bitte bis Sonntag diese Woche (14. Juli)**

    ... und machen Sie es doch am allerbesten heute noch !

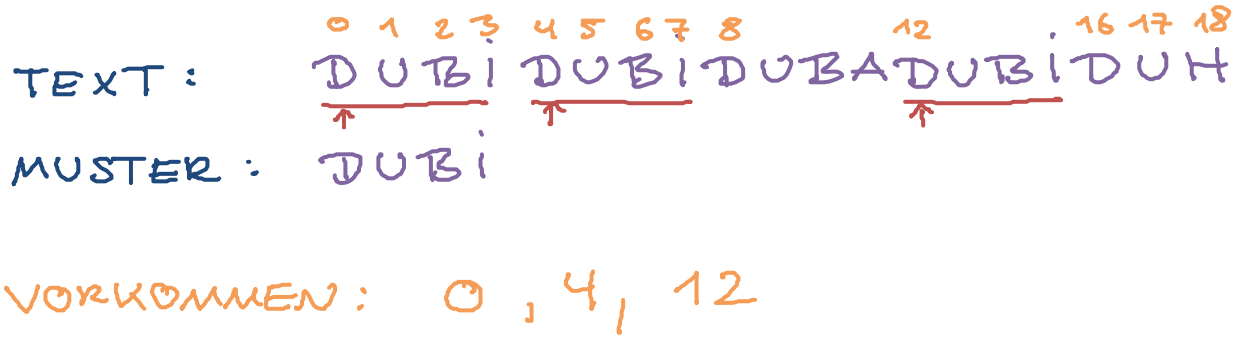
String Matching 1/2

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: wir benutzen durchgängig n für die Länge des Textes und m für die Länge des Musters



String Matching 2/2

UNI FREIBURG

- Motivation
  - **Jeder** Editor hat eine "Find" Funktion
  - **Jede** Programmiersprache hat in ihrer "String" Klasse Methoden dafür

    Java: String.indexOf(String pattern, int fromThisPosition)

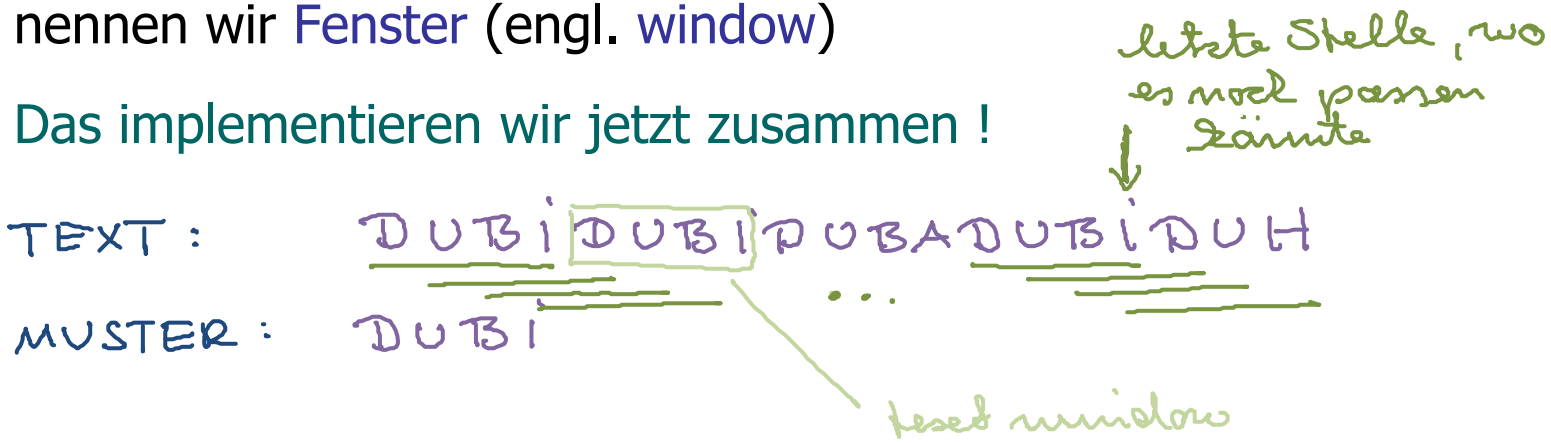
    C++: std::string.find(std::string str, size_t fromThisPosition)

    Damit bekommt man das nächste Vorkommen ab einer bestimmten Position … durch wiederholtes Aufrufen bekommt man so natürlich auch alle Vorkommen
  - Wenn man alle Vorkommen durch etwas ersetzen will (find & replace), muss man die Vorkommen erst mal alle finden

Naiver Algorithmus 1/2

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)
  - Das implementieren wir jetzt zusammen !



Naiver Algorithmus 2/2

n = Länge Text
m = Länge Muster

UNI FREIBURG

- Laufzeit
  - Im schlechtesten Fall (worst case): $\Theta(n \cdot m)$
  - Im besten Fall (best case): $\Theta(n)$ ($m \leq n$)

Beispiel worst case:

AAAAA... AAA

DUDADUDIDUDADUDI... DUDA

Beispiel best case:

Text = Muster

XYZXYQFVWGRBM AB

XYZXYABVWGRBM AB

Rabin-Karp Algorithmus 1/5

- 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

immer nur drei Operationen UNABHÄNGIG von der Länge m des Musters

base = 10

TEXT : 5 7 2 8 3 9 6 5 4 3 1 2 8 3 4 ...

MUSTER: 2 8 3

10^2 = base^{m-1}

572 ↓ − 5 × 100, × 10, + 8

728 ↓ − 7 × 100, × 10, + 3

283 ↓ − 2 × 100, × 10, + 9

839

Rabin-Karp Algorithmus 2/5

UNI FREIBURG

- Laufzeit
  - Wenn wir mit den Zahlen in konstanter Zeit operieren könnten, wäre die Laufzeit nur $O(n)$
  - Aber diese Zahlen können sehr groß werden
  - Wir rechnen mit den Zahlen **modulo M**
  - Muster und Fenster sind dann Zahlen aus $\{0, ..., M - 1\}$
  - 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

Rabin-Karp Algorithmus 3/5

in der Praxis VIEL größer viele später

UNI FREIBURG

- Beispiel

TEXT : 5728303548826 ...

MUSTER : 283 → mod M = 3

M = 5

572 mod M = 2 ≠ 3 ⇒ kein Match möglich

728 mod M = 3 = 3 ⇒ Match möglich
jetzt Vergleich wie im naiven Algorithmus ⇒ KEIN MATCH

283 mod M = 3 = 3 ⇒ Match möglich
direkter Vergleich ⇒ MATCH

die direkten Vergleiche (= wie im naiven Alg)
kosten nach wie vor bis zu $\Theta(nm)$
aber hoffentlich selten!

Rabin-Karp Algorithmus 4/5

UNI FREIBURG

- Laufzeit mit Modulus
  - Im schlechtesten Fall: $\Theta(n \cdot m)$

    Das passiert, wenn der Modulus für das Muster und das Textfenster "zufällig" immer gleich ist, obwohl das Muster gar nicht überall passt

    Bei guter Wahl von M unwahrscheinlich ... spätere Folie

  - Im besten Fall: $\Theta(n)$ ($m \leq n$)

    Das passiert, wenn der Modulus für das Muster und für das Textfenster nur dann gleich ist, wenn das Muster auch passt

    Bei guter Wahl von M wahrscheinlich ... spätere Folie

Rabin-Karp Algorithmus 5/5

- Zahlendarstellung und Wahl von M
  - Wir hätten gerne, dass wenn $x \neq y$, dann $h(x) = h(y)$ unwahrscheinlich, für $h(x) = x \bmod M$
  - Seien Muster und Textfenster als Zahlen zur Basis b dargestellt ... in unserem Beispiel war b = 10
  - Wenn b und M einen Teiler gemeinsam haben, dann ist das nicht so unwahrscheinlich
  - Wir wählen deshalb im Programm b als eine Primzahl ≥ 256 = die Anzahl verschiedener ASCII codes
  - Dann tut es irgendein M, insbesondere der implizite Modulus 2^{32} bei Rechnen mit unsigned int mit Überlauf

Rechnen modulo m 1/2

$117 \bmod 10 = 7$

$7 = 117 - \lfloor \frac{117}{10} \rfloor 10$

$= 11$

UNI FREIBURG

- Rechenregeln
  - Wir brauchen (für die Implementierung)

$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m \quad (1)$$

$$(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m \quad (2)$$

  - Dazu muss m nicht prim sein !

Beweis von (1):

$c = a \cdot b \bmod m \Rightarrow c = a \cdot b - q \cdot m$, $q = \lfloor \frac{a \cdot b}{m} \rfloor$

$a \bmod m = a - q_a \cdot m \quad q_a = \lfloor \frac{a}{m} \rfloor$

$b \bmod m = b - q_b \cdot m \quad q_b = \lfloor \frac{b}{m} \rfloor$

$(a \bmod m) \cdot (b \bmod m) = (a - q_a \cdot m) \cdot (b - q_b \cdot m)$

$= a \cdot b - [q_a \cdot b - q_b \cdot a + q_a \cdot q_b \cdot m] \cdot m$

... ∎

Rechnen modulo m 2/2

UNI FREIBURG

- Negative Zahlen
  - Für eine beliebige ganze Zahl x ist mathematisch einfach

    $x \bmod m = \min \{ x + q \cdot m : q \in \mathbf{Z} \text{ und } x + q \cdot m \geq 0 \}$

    Damit ist automatisch x mod m < m, sonst könnte man ja noch mal ein m abziehen
  - Beispiele: 24 mod 10 = 4, 4 mod 10 = 4, -4 mod 10 = 6
  - **ABER:** in Java und in C++ ist (leider) -x % m = - (x % m)
  - Beispiele: 24 % 10 = 4, 4 % 10 = 4, -4 % 10 = -4
  - **Workaround:** wenn wir für negative Zahlen mod haben wollen, einfach ein Vielfaches von m draufaddieren (das ändert nichts mod m), so dass die Zahl positiv wird

Literatur / Links

UNI FREIBURG

- String Matching
  - Mehlhorn/Sanders: gar nichts zu dem Thema!
  - Wikipedia

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

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

    + auf den Englischen Seiten steht wie immer mehr
  - Originalarbeit von Rabin & Karp

    Richard Karp und Michael Rabin

    Efficient randomized pattern matching

    1987 IBM Journal of Research and Development