Informatik II: Algorithmen und Datenstrukturen SS 2017

Vorlesung 12b, Mittwoch, 19. Juli 2017
(String Matching, Teil 2)

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

Blick über die Vorlesung heute

- Inhalt
  - KMP Algorithmus — Fortsetzung von gestern
  - Karp-Rabin (KR) Algorithmus — Prinzip + Beispiele
  - Plagiatserkennung — Anwendung von KR
  - ÜB12: Implementieren Sie den Karp-Rabin Algorithmus und benutzen Sie ihn zur automatischen **Plagiat**serkennung

Edi-Tier 1/3

- Woher stammt es und was hat es uns voraus?
  - "Eins ist klar: es kommt aus dem großen Welt-Edi-Tor"
  - "Es verträgt mehr Wodka"
  - "Das Edi-Tier stammt sicher vom Edi-Fant ab"
  - "Ein Kumpel von mir heißt Eddie und der hat einen Hund"
  - "Hier ist ein Foto: https://goo.gl/yvHDCJ"
  - "The Edi-Tier is called Leafcutter Ant and it's fascinating"

    "In the Zoo Burkart, Lörracher Straße, there is a very big colony. Its amazing to watch the ants work. They transport leafs, deposit waste on their waste site, farming their mushrooms. Its worth visiting just to see the ants."

Edi-Tier 2/3

- Blattschneiderameisen (Leafcutter ants)
  - Eine faszinierende Tierart, die uns einiges voraus haben
  - Sie sind bekannt für die grünen Blätter, die sie durch die Gegend tragen (bis zum 20-fachen ihres Körpergewichtes)
  - Die Blätter sind aber nicht für sie selber, sondern werden in vielen Arbeitsschritten zu kleinen Kügelchen verarbeitet, mit denen die Ameisen dann schließlich einen **Pilz** füttern
  - Von diesem Pilz ernähren sich dann die Larven der Ameisen

  Die Ameisen betreiben also komplexe **Landwirtschaft**, und sie tun das seit mindestens 15 Millionen Jahren

  Menschen seit gerade mal 10.000 Jahren

Edi-Tier 3/3

- Schädlingsbekämpfung
  - Forscher haben sich lange gewundert / gefragt, warum die Ameisen keine Probleme mit Schädlingen haben

    Aber sie haben: ohne die Ameisen, wird der Pilz in relativ kurzer Zeit von (anderen) Schimmelpilzen dahingerafft
  - Einigen Forschern ist aufgefallen, dass manche von den Ameisen weißes Zeug auf dem Rücken tragen
  - Das sind Bakterien, die u.a. **Antibiotika** produzieren, gegen die Schädlinge, die für den kultivierten Pilz gefährlich sind

    Insbesondere sind da viele Antibiotika dabei, die auch wir Menschen dann im 20ten Jahrhundert mal entdeckt haben
  - Die Bakterien helfen außerdem beim **Düngen** des Pilzes

Karp-Rabin Algorithmus 1/11

- Grundidee
  - 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 Stück Text im aktuellen Fenster ebenso
  - Verschiebt man das Fenster um eins nach rechts, lässt sich die neue Zahl leicht aus der alten berechnen

$n = 12$ Text: 5 7 2 8 3 0 3 5 4 8 2 6

$m = 3$ Pattern: 2 8 3

× $283 \neq 572$ $-5 \cdot 100 \;(=72)$, $\cdot 10 \;(=720)$, $+8 \;(=728)$

× $283 \neq 728$ $-7 \cdot 100 \;(=28)$, $\cdot 10 \;(=280)$, $+3 \;(=283)$

✓ $283 = 283$

Karp-Rabin Algorithmus 2/11

$b = 10$:

$283 = 2 \cdot 10^2 + 8 \cdot 10^1 + 3$

$\quad\quad\quad\; b^{m-1} \quad\; b^{m-2} \quad b^0$

- Rechnen mit diesen "Zahlen"
  - Pro Fensterverschiebung braucht man nur konstant viele Rechen-Operationen auf diesen Zahlen
  - Wenn wir mit den Zahlen in konstanter Zeit operieren könnten, wäre die Laufzeit also O(n)

    Aber diese Zahlen können sehr groß werden

    Bei Basis b und einem Muster der Länge m bis zu b^m

    Dafür braucht man $\log_2 b^m = m \cdot \log_2 b$ Bits

    $\log_2 b = 8$

    Für b = 256 (ASCII) und m = 10 sind das schon 80 Bits … zu viel für z.B. ein int auf einem 64-Bit Rechner

Karp-Rabin Algorithmus 3/11

- Hashwerte
  - Statt Zahl **x** betrachten wir Hashwert **h(x) = x mod M**

    Die Hashwerte sind dann aus dem Bereich $\{0, ..., M - 1\}$
  - Bei einem Match sind sicher auch die Hashwerte gleich

    Aber gleiche Hashwerte bedeuten nicht unbedingt Match

    Wenn M groß ist, ist es allerdings unwahrscheinlich, dass ungleiche Zahlen auf denselben Wert abgebildet werden

    Das kennen wir ja schon vom Hashing (Kollisionen)
  - Wenn die Hashwerte gleich sind, überprüfen wir wie beim naiven Algorithmus Buchstabe für Buchstabe

Karp-Rabin Algorithmus 4/11

- Beispiel

Text: 5 7 2 8 3 0 3 5 4 8 2 6

Pattern: 2 8 3 $\qquad h(283) = 3$

Modulus: M = 5 $\qquad , \; h(x) = x \bmod 5$

In der Praxis benutzt man viel größere Werte für M

$h(572) = 2 \neq 3 = h(283)$ ⇒ SICHER kein Match

$h(728) = 3 = 3 = h(283)$ ⇒ Kandidat für ein Match, aber Nachprüfen der ursprünglichen Zahl ($O(m)$ Zeit) → KEIN Match

$h(283) = 3 = 3 = h(283)$

⇒ Kandidat für Match und es ist auch einer (wieder $O(m)$ Zeit)

Karp-Rabin Algorithmus 5/11

- Laufzeit mit Modulus $n = \#\text{Text}$, $m = \#\text{Pattern}$
  - Im schlechtesten Fall: $O(n \cdot m)$

    Wenn Hashwert für Muster und Fenster immer gleich, obwohl das Muster gar nicht überall passt

    Bei guter Wahl von M **unwahrscheinlich** … Folie 11+16
  - Im besten Fall: $O(n)$

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

    Bei guter Wahl von M **wahrscheinlich** … Folie 11+16

Karp-Rabin Algorithmus 6/11

z.B. $b = 10$

$15127 \bmod 10 = 7$

$23417 \bmod 10 = 7$

- Wahl der Basis b und des Modulus M
  - Wir hätten gerne, dass es unwahrscheinlich ist, dass **x ≠ y** aber **h(x) = h(y)**, für h(x) = x mod M
  - Dazu sollte M möglichst groß sein und keinen Teiler mit b gemeinsam haben … Negativbeispiel dazu:

    Wenn b durch M teilbar, würde nur letzte "Stelle" zählen
  - Wir wählen deshalb im Programm b = 257

    Kleinste Primzahl, die größer ist als jeder ASCII Code
  - Wir können dann M im Prinzip beliebig wählen

$m = 4,\ b = 257$

$$\text{doof} = 100 \cdot \underbrace{257^3}_{b^{m-1}} + 111 \cdot \underbrace{257^2}_{b^{m-2}} + 111 \cdot \underbrace{257^1}_{b^1} + 102 \cdot \underbrace{1}_{b^0}$$

Ascii 100 111 111 102

Karp-Rabin Algorithmus 7/11

- Rechnen modulo M
  - Es gelten folgende Rechenregeln (für jedes $M \in \mathbf{N}$)

    $(a \cdot b) \bmod M = ((a \bmod M) \cdot (b \bmod M)) \bmod M$

    $(a + b) \bmod M = ((a \bmod M) + (b \bmod M)) \bmod M$

    Beweis: gute Aufgabe zur Mathe-Phobie-Überwindung !
  - Das heißt, wir können bei einem größeren Ausdruck das **mod M** auch schon auf Teilterme anwenden

    Das ist wichtig, damit die Zahlen nicht zu groß werden

Karp-Rabin Algorithmus 8/11

10111011 mod 2^4
= 1011

z.B. bei einem 64-Bit unsigned und in C++
macht der Überlauf „automatisch" mod 2^{64}

- Berechnung Hashwert in Java, C++, Python
  - In Java und C++ kann man statt einem expliziten mod M einfach den Überlauf beim Rechnen mit int nutzen

$283 = 2 \cdot 100 + 8 \cdot 10 + 3 \Leftarrow b = 100$
$= (2 \cdot 10 + 8) \cdot 10 + 3$

```
int hash_value = 0;
for (int j = 0; j < pattern.size(); j++) {
    hash_value = 257 * hash_value + pattern[j];
}
```

  - In Python können Zahlen beliebig groß werden, dort braucht man also ein explizites mod M

```
hash_value = 0
for c in pattern:
    hash_value = (257 * hash_value + c) % M
```

Karp-Rabin Algorithmus 9/11

$-38 \bmod 10$
$= -38 + 4 \cdot 10 = 2$

$715 \bmod 10$
$= 715 + (-71) \cdot 10 = 5$

- Modulo bei negativen Zahlen
  - Für eine beliebige ganze Zahl x ist mathematisch einfach

    x mod m = min { x + q · m : q ∈ **Z** und x + q · m ≥ 0 }

    Damit ist immer x mod m ∈ {0, …, m – 1}

    Zum Beispiel: 24 mod 10 = 4 und –4 mod 10 = 6
  - In Java und C++ ist aber – x % m = – (x % m)

    Zum Beispiel: 24 % 10 = 4 und –4 % 10 = –4

    Wenn man es für Java oder C++ so macht, wie auf der Folie vorher, braucht man sich darum nicht zu kümmern
  - In Python ist x % m = x mod m … insbes. –4 % 10 = 6

Karp-Rabin Algorithmus 10/11

- Erweiterung auf mehrere Patterns
  - Nehmen wir jetzt an, wir haben **k** Patterns und wollen alle Vorkommen **aller** dieser Patterns in einem Text finden
  - Eine typische Anwendung dafür ist die Plagiatserkennung

    Patterns = Sätze / Fragmente aus einer Originalarbeit

    Text = vermeintliches Plagiat
  - Wir könnte jetzt einfach einen unserer bisherigen Algorithmen **k** mal anwenden, für jedes Pattern einmal

    Die Laufzeit würde sich dann um dem Faktor k erhöhen, das dauert für große k (viele Patterns) sehr lange

    Für das ÜB12, Aufgabe 2 ist k immerhin gleich 288

UNI FREIBURG

Karp-Rabin Algorithmus 11/11

- Erweiterung von Karp-Rabin für **k** Patterns
  - Speichere die Patterns in einer Map, mit dem Hashwert als Schlüssel und dem Pattern als Wert
  - Für jedes Textstück (aus dem aktuellen Fenster), schauen wir dann, ob es Patterns mit demselben Hashwert gibt

    Das geht mit einer geeignete Map in O(1) Zeit
  - Falls ja, prüfen wir für jedes solche Pattern nach, ob es passt, wie beim naiven Algorithmus

    Für M $\gg$ k wird es selten vorkommen, dass der Hashwert übereinstimmt, aber das Pattern trotzdem nicht matched
  - Die Laufzeit ist dann **O(n + Gesamtlänge der Matches)**

Literatur / Links

- Karp-Rabin Algorithmus
  - Wikipedia

    http://en.wikipedia.org/wiki/Rabin-Karp-Algorithmus
  - Originalarbeit von Rabin & Karp:

    Richard Karp und Michael Rabin
    Efficient Randomized Pattern Matching
    1987 IBM Journal of Research and Development