Abgabe 9

Anna Tamina von Stietcorn, Emanuel Reinking Florian, Marcel Giller

---

Aufgabe 1:
Code: Siehe Anhang
Wie schneiden, Ihre Implementierungen im Vergleich ab?
Naive Suche: 326387400ns
Suche nach Rabin Karp: 184919000ns
$\frac{326387400ns}{184919000ns} = ca. 1.7$
Folglich ist der Rabinkarp algorithmus ca. 1,7 mal schneller als der naive Algorithmus.

Aufgabe 2:
(a) Der Algorithmus von Rabin-Karp l¨asst sich leicht auf mehrere Suchmuster verallgemeinern. Gegeben eine Zeichenkette s und Suchmuster t1, . . . , tk, bestimme die erste Stelle in s, an der eines der Muster t1, . . . , tk vorkommt. Beschreiben Sie, wie man den Algorithmus von Rabin-Karp f¨ur diese Situation anpassen kann. Was ist die heuristische Laufzeit Ihres Algorithmus (unter der Annahme, dass Kollisionen selten sind)?

Antwort: Der algorithmus muss angepasst werden. Sei k die Länge des längsten Strings, Sei p die Anzahl der Patterns, sei n die Länge des Textes.

1. für einjedes pattern in der Eingabe muss ein Passender Hashwert ermittelt werden. Die Laufzeit der initialisierung steigt. Sei k das längste Pattern, dann dauert die initialisierung im Worstcase $O(k * p)$
2. Beim linearen durchgehen des Strings wird der Hashwert der als Vergleich dient mit jedem neuen Index um den wert der ersten Stelle verkleinert. dies geschieht in $O(1)$ und ändert sich nicht im Vergleich.
3. Aber um zu überprüfen, ob die Hashwerte übereinstimmen, muss jedes mal jeder Hashwert der Patterns mit dem neuen Hashwert des Textes verglichen werden. Diese Änerung ist statt $O(1 * n)$ ist es nun $O(p * n)$
4. Auch die Spanne, die Überprüft werden soll, ändert sich bei jedem pattern. Somit muss auch die Spanne bei jedem Index für Jedes Element geändert werden. Der Rolling Hash wird dort also immer verändert. Ein schritt nach vorne ist also $O(n * p * patternsizedifferenz)$
$\rightarrow k * p + p * n + n * p * patternsizedifferenz = O(n * p * k * patternsizedifferenzerential) -$ da $n * p$ ausschlaggebend für die anderen parameter ist.

b)
sense = 41
sensebility = 15
sensible = 20
Sense ist der gewinner

Übung 9

Aufgabe 3:

$\Sigma = C, G, T, A$
$s = C, T, T, G, G, A, T, T, A$
$t = T, T, A$

a) naiver Algorithmus:

```
C T T G G A T T A
T T A

C T T G G A T T A
→ T T A

C T T G G A T T A
 → T T A

C T T G G A T T A
  → T T A

C T T G G A T T A
   → T T A

C T T G G A T T A
    → T T A

C T T G G A T T A
     → T T A
```

⇐ gefunden

b) Rabin-Karp:

A: 0
T: 1
G: 2
C: 3

$h = (o_1 \cdot d^{l-1} + o_2 \cdot d^{l-2} \dots o_n \cdot d^0) \bmod 5$

$d = |\Sigma| = 4$
$l = |t| = 3$

$h(t) = (1 \cdot 4^2 + 1 \cdot 4^1 + 0 \cdot 4^0) \bmod 5$
$= 0$

$h(x_1) = (3 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0) \bmod 5$
$= (53) \bmod = 3$
$|\Sigma|^l = 4^3$

Rausgefallendes Element $\| o_i \| = C = 3$

Reingekommendes Element $\| o_{i+l} \| = G = 2$

$h(o_{i+1} \dots o_{i+l}) = (|\Sigma| \cdot h \cdot (o_i \dots o_{i+l-1}) - |\Sigma|^l \cdot \| o_i \| + \| o_{i+l} \|) \bmod$ …

$\Rightarrow h(x_2) = |(4 \cdot 3) - 4^3 \cdot 3 + 2)| \bmod 5$

$= (12 - 84 + 2) \bmod 5$

$= (-178) \bmod 5$

$= 2$

$\Rightarrow 0 \neq 2$

$\|\theta_i\| = T = 1$

$\|\theta_{i+l}\| = G = 2$

$h(x_3) = (4 \cdot 2 - 4^3 \cdot 1 + 2) \bmod 5$

$= (-116) \bmod 5$

$= 4$

$\Rightarrow 0 \neq 4$

$\|\theta_i\| = G = 2$

$\|\theta_{i+l}\| = A = 0$

$h(x_4) = (4 \cdot 4 - 4^3 \cdot 2 + 0) \bmod 5$

$= (-112) \bmod 5$

$= 3$

$\Rightarrow 0 \neq 3$

$\|\theta_i\| = G = 2$

$\|\theta_{i+l}\| = T = 1$

$h(x_5) = (4 \cdot 3 - 4^3 \cdot 2 + 1) \bmod 5$

$= (-115) \bmod 5$

$= 0$

$0 = 0$

$\Rightarrow$ Untersuchen auf Kollision:

ATT

TAA

$\Rightarrow A \neq T \Rightarrow$ Suche wird fortgesetzt

$\|\theta_i\| = A = 0$

$\|\theta_{i+l}\| = A = 0$

$h(x_6) = (4 \cdot 0 - 4^3 \cdot 0 + 0) \bmod 5$

$= (0) \bmod 5$

$= 0$

0 = 0

=> Untersuchen auf Kollision

TTA
TTA

=> TTA wurde gefunden ✓

c) Knuth-Morris-Pratt

0 1 2 3 4 5 6 7 8
C, T, T, G, G, A, T, T, A

0	1	2
T	T	A
0	1	0

i[0] = C   j[0] = T
=> kein Element vorher
=> i++

i[1] = T   j[0] = T
i++   j++

i[2] = T   j[1] = T
i++   j++

i[3] = G   j[2] = A
j = j - 1

i[3] = G   j[1] = T
j = j - 1

i[3] = G   j[0] = T
=> kein Element vorher
i++

i[4] = G   j[0] = T
=> kein Element vorher
i++

i[5] = A   j[0] = T
=> kein Element vorher
i++

i[6] = T   j[0] = T
i++   j++

i[7] = T   j[1] = T
i++   j++

i[8] = A   j[2] = A

=> gefunden