Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

1. Dezember 2016

Algorithmen & Datenstrukturen Lösungen zu Blatt P10 HS 16

Lösung P10.1 *Teilarray finden.*

Die Lösung ist als Robin-Karp Algorithmus bekannt. In groben Zügen wurde der Algorithmus schon auf dem Aufgabenblatt beschrieben und wir beschreiben die Details hier.

Zuerst müssen wir `cToK` $= c^k \bmod m$ berechnen indem wir `cToK = (cToK * c) % m` ausgehend von `cToK = 1` k mal ausführen, was $\mathcal{O}(k)$ Zeit kostet. Es ist wichtig in jedem Schritt Modulo auszuführen um Überläufe zu vermeiden. Wir können $c^k \bmod m$ nicht für jeden Index von A, den wir betrachten, neu zu berechnen.

Dann berechnen wir den Hash `HB` $= H(B)$, indem wir beispielsweise die Formel zur Aktualisierung[1] k mal anwenden. Wir berechnen den Hash `HA` $= H(a_0 \dots a_{k-1})$ in derselben Weise. Nun vergleichen wir für alle möglichen Startpositionen von B in A, $i = 0, \dots, n-k$, den aktuellen Hash `HA` mit `HB`. Wenn beide gleich sind, prüfen wir `A[i+j]==B[j]` für $j = 0, \dots, k-1$ und geben Index i aus, falls alle Elemente gleich sind. Dann inkrementieren wir i und bewegen das Hash-Fenster mit der Formel aus der Aufgabenbeschreibung um eins nach rechts.

Wir erwarten nur sehr wenige Kollisionen (die vorgeschlagenen Parameter führen zu etwa 20 Kollisionen am Judge). Daher benötigen nur sehr wenig Zeit um die "falschen" Übereinstimmungen zu überprüfen.

Das Problem kann auch mit dem Knuth-Morris-Pratt Algorithmus (ursprünglich für Textsuche entwickelt) gelöst werden, wenn die Werte hinreichend klein sind (kleiner als 1000 in unserem Fall). Wir werden diesen Ansatz hier nicht beschreiben und überlassen die Ausarbeitung dem interessierten Leser. Sie können eine gute Beschreibung des Algorithmus auf Wikipedia finden.

Lösungen

Auf der Vorlesungswebseite finden Sie eine Lösung, welche eine Laufzeit von $\mathcal{O}(n+kp)$ benötigt, wobei p die Anzahl der gefundenen Teilarrays ist. Die Lösung enthält weitere Kommentare zur Implementierung.

[1] Wir verwenden 0 als ein neutrales Element.

Daten

Der Test `judge1` enthält mehrere kleine Instanzen mit Spezialfällen, Test `judge2` zusätzlich lange, sich wiederholende Sequenzen, wie zum Beispiel $A = 0^{100000}10^{100000}$ und $B = 0^{10000}10^{10000}$. Eine Lösung mit Laufzeit in $\mathcal{O}(nk)$ würde sehr lange brauchen, um zu erkennen, dass es kein passendes Teilarray gibt (selbst wenn der Algorithmus rückwärts laufen würde). `judge3` enthält zusätzlich lange, zufällige Sequenzen, die einige Kollisionen bei den Hashwerten erzeugen (wahrscheinlich auch für andere Parameter c und m, falls m nicht viel grösser gewählt wird).

Hinweise zu den Abgaben. Ein häufiges Problem bei den Abgaben war das Aktualisieren der Hash-Werte (Berechnung des nächsten Fensters). Wie im Aufgabenblatt erwähnt kann man beliebig oft modulo m rechnen. Sie können also so oft modulo m rechnen, dass es nie zu einem Überlauf des Integer Datentyps kommt. Zum Beispiel kann modulo bei jeder Berechnung von $c^k \bmod m = (\dots(((cc \bmod m)c \bmod m)c \bmod m)\dots)c \bmod m$ verwendet werden. Ausserdem muss beim Aktualisieren der Hash-Funktion beim Subtrahieren beachtet werden, dass das Resultat positiv modulo m ist.

Die Fehlersuche ist bei Hash-Funktionen leider etwas umständlich, da die Hash-Werte keine leicht ersichtliche Bedeutung haben.