Randomisierte Algorithmen

Die Verwendung von Zufallszahlen kann manche Algorithmen ganz erheblich beschleunigen! Es ist zwar ein eigener Forschungsgegenstand, ob und wie man geeignete Zufallszahlen überhaupt produzieren kann, aber wir abstrahieren davon und führen einfach eine Operation WÄHLE x ZUFÄLLIG AUS $\{1, \ldots, n\}$ oder WÄHLE x ZUFÄLLIG AUS $\{0, 1\}$ ein mit der offensichtlichen Semantik.

1. Anwendung: Quicksort

Bei Quicksort haben wir zwei Möglichkeiten der *Randomisierung:*

1) Man kann das Pivot-Element in jedem Schritt zufällig wählen.

2) Man kann die Eingabe zu Beginn „zufällig“ permutieren.

Beiden Techniken ist gemeinsam, dass das Ergebnis jedenfalls immer korrekt ist - unabhängig von den Zufallszahlen!

Primzahltest

Die Überprüfung, ob eine Zahl eine Primzahl ist, ist theoretisch mit dem AKS-Test deterministisch in polynomieller Zeit möglich. Wesentlich effizienter ist aber ein probabilistischer Primzahltest.

Es gibt Tests (z.B. den Fermat-Primzahltest), die für jede große Zahl ausgeben: Ist keine Primzahl oder Ist wahrscheinlich Primzahl. Dabei ist die Aussage im ersten Fall immer korrekt, im zweiten Fall ist ein Irrtum mit einer festgelegten Wahrscheinlichkeit erlaubt.

Selbst wenn ein solcher Test eine Fehlerwahrscheinlichkeit von 25% hat, ist er durchaus brauchbar. Man kann leicht nachrechnen, dass bereits mit 5 Wiederholungen die Fehlerquote auf 0,1% gesenkt werden kann; wie sieht es mit 100 Wiederholungen aus?

Bei 100 Wiederholungen ist die Fehlerwahrscheinlichkeit so gering, dass man die Möglichkeit eines Fehlers praktisch ohne Risiko ausschließen kann.

Verstärkung der Erfolgswahrscheinlichkeit

Die eben mehr oder weniger intuitiv durchgeführte Verbesserung eines probabilistischen Algorithmus durch Wiederholung kann man formal so beschreiben:

L sei die zu erkennende Sprache, und es liege ein probabilistischer Algorithmus für L vor, der folgendes Verhalten zeigt:

Input $x \in L$:	Der Algorithmus gibt JA aus.
Input $x \notin L$:	Der Algorithmus gibt NEIN mit W'keit $\geq \frac{1}{2}$ aus.

Beachte: Das bedeutet, dass eine NEIN-Antwort des Algorithmus immer zuverlässig ist, während die JA-Antworten nicht sicher sind.

Nun wiederholen wir den Algorithmus n mal mit immer wieder neu gewählten Zufallszahlen. Wir schließen $x \in L$, falls n mal Antwort JA gegeben wurde, $x \notin L$, sonst.

Auch hier ist eine NEIN-Antwort immer korrekt, während eine JA-Antwort mit Wahrscheinlichkeit $\leq \frac{1}{2^n}$ falsch sein kann.

Verstärkung bei zweiseitigem Fehler

Die Verbesserung durch Wiederholung scheint auf den ersten Blick nur wegen der Einseitigkeit des Fehlers zu funktionieren.

Tatsächlich kann man aber auch bei zweiseitigem Fehler einen ähnlichen Effekt durch Wiederholung erzielen. Die Rechnung wird hier allerdings ein bisschen komplizierter.

Beispiel: Der ursprüngliche Algorithmus produziere das richtige Ergebnis mit Wahrscheinlichkeit $\frac{2}{3}$. Wir wiederholen dreimal und nehmen als Ergebnis die Mehrheitsentscheidung. Also geben wir das richtige Ergebnis mit Wahrscheinlichkeit $(\frac{2}{3})^3 + 3 \cdot (\frac{2}{3})^2 \cdot \frac{1}{3} = \frac{20}{27}$ aus.

Bitte überprüfen Sie, ob Sie das Zustandekommen dieser Rechnung verstanden haben.

Allgemein kann man zeigen, dass die t-fache Wiederholung eines probabilistischen Algorithmus, der mit Wahrscheinlichkeit $> (1 - \delta)$ das korrekte Ergebnis liefert, maximale Fehlerwahrscheinlichkeit δ' hat, wobei $\delta' = \left[2 \cdot \sqrt{\delta(1-\delta)}\right]^t$ ist (für $0 < \delta < \frac{1}{2}$).

Monte Carlo Algorithmen

Bei probabilistischen Algorithmen werden oft zwei Konzepte unterschieden. Das erste dieser Konzepte stellen die

Monte Carlo Algorithmen

dar.

Das charakteristische Merkmal dieser Sorte probabilistischer Algorithmen ist die Tatsache, dass sie ein falsches Ergebnis liefern dürfen. Allerdings sollte die Wahrscheinlichkeit, das korrekte Ergebnis zu liefern, möglichst hoch sein.

Die Wahrscheinlichkeit ist hierbei immer zu verstehen bzgl. der Gleichverteilung bei dem zugrunde gelegten Zufallsexperiment.

Als ein Beispiel für diese Art der probabilistischen Algorithmen haben wir gerade die Primzahltests behandelt.

Las Vegas Algorithmen

sind die andere Sorte probabilistischer Algorithmen. Hier sind keine Fehler erlaubt. Allerdings besteht bei solchen Algorithmen im Normalfall die Option, keine Ausgabe zu machen.

Was ist bei diesen Algorithmen dann noch zufällig?

Zum einen kann die Laufzeit eines Las Vegas Algorithmus von den verwendeten Zufallswerten abhängen. Darüberhinaus kann es sein, dass der Algorithmus keine Ausgabe produziert und folglich (mit neuen Zufallswerten) wiederholt werden muss.

Welches Ziel verfolgt man eigentlich in diesen Fällen durch die Verwendung des Zufalls?

Antwort: Die erwartete Rechenzeit soll minimiert werden!