Algorithmen und Berechenbarkeit: Vorlesung 01 - Mitschrift

Letztes Update: 2018/03/09 - 17:52 Uhr

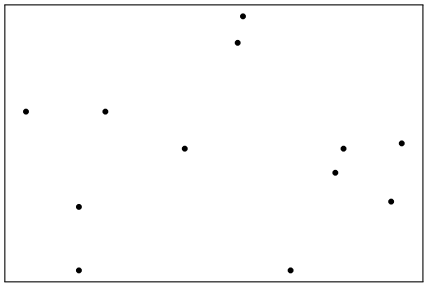
Randomisierte Algorithmen

Randomisierte Algorithmen sind Algorithmen, die Probleme unter Einbeziehung einer Zufallsquelle (z.B. Münzwurf, Zufallsgenerator) lösen. In manchen Fällen sind diese Algorithmen einfacher und effizienter als ihre deterministischen Pendants.

Randomisierte Algorithmen werden nach *dem Verbrauch von Zufall* bewertet (im Gegensatz zu „normalen“ Algorithmen, die nach Platz- und Zeitbedarf bewertet werden). Die „Generierung“ echten Zufalls ist jedoch **teuer**.

Closest-Pair

Gegeben sei eine Menge P von Punkten im $\mathbb{R}^2$. Finde $p_1, p_2 \in P$ für die $|p_1, p_2|$ (*euklidische Distanz*) minimal.



Ansatz 1: Naiv

Man betrachtet P_1 und berechnet die Distanzen zu $P_2, P_3, \ldots, P_n$, danach betrachtet man P_2 und berechnet die Distanzen zu $P_3, P_4, \ldots, P_n$. Dies wird für alle verbleibenden Punkte gemacht. Es ergibt sich hieraus eine Laufzeit von

$$\mathcal{O}\left(\sum_{i=1}^{n-1}\right) = \mathcal{O}\left(\frac{(n-1)\cdot n}{2}\right) = \mathcal{O}(n^2)$$

und den tabellarisch aufgelisteten Laufzeiten.

n	Rechnung		Ergebnis
$= 100$	$100^2 \cdot 10^{-9}$	$= 10^{-5}$	$10\,\mu s$
$= 1000$	$1000^2 \cdot 10^{-9}$	$= 10^{-3}$	$1\,ms$
$= 10000$	$10000^2 \cdot 10^{-9}$	$= 10$	$10\,s$
$= 1000000$	$1000000^2 \cdot 10^{-9}$	$= 10^3$	$10\,min$

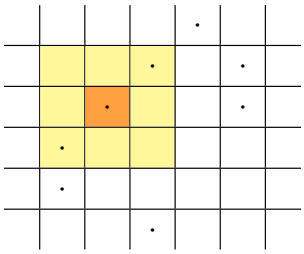
1-GHz-Prozessor (1.000.000.000 Instruktionen pro Sekunde) für $\mathcal{O}(n^2)$

Ansatz 2: Randomisiert (Las-Vegas)

Ein randomisierter Ansatz des Closest-Pair-Problems hat eine *erwartete* Laufzeit von $\mathcal{O}(n)$ und funktioniert folgendermaßen:

Sei P die Punktemenge mit $P_1, P_2, P_3, \ldots, P_n$ und sei δ_i die aktuelle Closest-Pair-Distanz. Für das **Einfügen** eines neuen Punktes P_{i+1} gilt allgemein: Die neue Closest-Pair-Distanz erhält man, wenn man die Abstände vom neuen Punkt zu allen bisherigen Punkten berechnet und überprüft, ob es einen noch kürzeren Abstand zweier Punkte gibt als δ_i.

Optimiert werden kann das Einfügen durch die folgende Überlegung: Da δ_i bekannt ist, kann nun ein Gitter mit einer Maschenweite (Breite der Zeilen/Spalten) von δ_i erzeugt werden.



Beim **Einfügen** eines neuen Punktes wird zuerst der neue Punkt P_{i+1} im Gitter (orange) lokalisiert. Anschließend werden alle Punkte in den angrenzenden Zellen (gelb) mit dem neuen Punkt P_{i+1} verglichen. Gilt

- $\delta_{i+1} = \delta_i$: Ist die neue Closest-Pair-Distanz identisch mit der alten (**guter Fall**), so muss nichts weiter getan werden und es ergibt sich eine Laufzeit von $\mathcal{O}(1)$.
- $\delta_{i+1} < \delta_i$: Ist die neue Closest-Pair-Distanz kleiner als die alte (**schlechter Fall**), so muss das Gitter mit der Maschenweite δ_{i+1} neu aufgebaut werden. Im schlechtesten Fall ergibt sich somit eine Laufzeit von $\mathcal{O}(n^2)$.

Da es auch Fälle geben kann, in denen jeder neu eingefügte Punkt gleichzeitig auch ein neues Closest-Pair bildet, das Gitter also bei jedem Einfügeschritt neu aufgebaut werden muss (Liste, sortiert nach größtem Abstand zueinander), werden die Punkte in *zufälliger Reihenfolge* einzufügt. So bleibt eine Wahrscheinlichkeit von $\frac{2}{i+1}$, dass der nächste Punkt ein Closest-Pair-Punkt ist. Die erwarteten Kosten des Einfügens **pro Punkt** sind

$$\leq \underbrace{\frac{2}{i+1} \cdot \mathcal{O}(i)}_{\text{schlechter Fall}} + \underbrace{\mathcal{O}(1)}_{\substack{\text{guter} \\ \text{Fall}}} = \mathcal{O}(1) + \mathcal{O}(1) = \mathcal{O}(1)$$

Der Erwartungswert für das Einfügen ergibt sich zu

$$E\left[\sum_{i=1}^{n}(\text{Kosten für Einfügen von } P_i)\right] = \sum_{i=1}^{n} E[\text{Kosten für Einfügen von } P_i]$$
$$= \sum_{i=1}^{n} \mathcal{O}(1) = \mathcal{O}(n)$$

Satz: *Die Wahrscheinlichkeit, beim Einfügen eines neuen Punktes P_{i+1} das Gitter neu aufbauen zu müssen, ist* $< \frac{2}{i+1}$.

Beweis: Das Gitter muss genau dann neu aufgebaut werden, wenn der neue Punkt P_{i+1} einer der beiden Punkte ist, welche das Closest-Pair in der Menge der ersten $i+1$ Punkte bestimmen. Jeder der ersten $i+1$ Punkte ist mit gleicher Wahrscheinlichkeit der P_{i+1}. Falls das Closest-Pair eindeutig ist, gilt

$$\Pr(\text{Gitter muss neu aufgebaut werden}) = \frac{2}{i}$$

Wichtig: Wenn der Algorithmus länger braucht, hat das nichts mit der Eingabe zu tun. Der randomisierte Closest-Pair-Algorithmus berechnet **immer** ein korrektes Resultat. □

Ansatz 3: Deterministisch

Ein Closest-Pair kann in $\mathcal{O}(n \cdot \log(n))$ berechnet werden. Es kann sogar gezeigt werden, dass Closest-Pair $\in \Omega(n \cdot \log(n))$, es also gar nicht besser geht.

n	Rechnung	Ergebnis
$= 1000000$	$10^6 \cdot 6 \cdot 10^{-9}$	6 ms

1-GHz-Prozessor für $\mathcal{O}(n \cdot \log(n))$

Beweis: Man weiß, dass Elementuniqueness (gegeben n Zahlen → man prüft, ob eine Zahl doppelt vorkommt) $\Omega(n \cdot \log(n))$ braucht.
Wenn nun Closest-Pair vergleichsbasiert besser als $o(n \cdot \log(n))$ gelöst werden könnte, so könnte man auch Elementuniqueness in dieser Zeit lösen. Das gilt jedoch nur in einem anderen Rechenmodell (Randomisierung und Abrundung). □

Anhang

Zufallsvariable und Erwartungswert

Sei X die Zufallsvariable *Augenzahl beim Wurf* mit einem normalen Würfel. Der Erwartungswert berechnet sich zu

$$\begin{aligned} E[X] &= \sum \text{Ereignis} \cdot \Pr(\text{Ereignis}) \\ &= \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 4 + \frac{1}{6} \cdot 5 + \frac{1}{6} \cdot 6 \\ &= 3,5 \end{aligned}$$