

## Vorlesungsmitschrieb

# Algorithmen und Berechenbarkeit

## Vorlesung 01

Letztes Update: 2017/10/20 - 16:56 Uhr

### Organisatorisches

#### Übung

- Anmeldung über das Campus
- Übungsabwicklung über Ilias
- 14 -tägig
- Anmeldung zwischen 10/20/2017 - 10/27/2017 ab 16 Uhr
- Gruppengröße: 3-4 Personen
- Mehr Informationen auf der Website

#### Schein

- 50% der Punkte aus den Übungen
- Mindestens 50% der Aufgaben votieren
- Bestehen von 2 MC-Tests

Pro Übungsblatt wird in der Regel eine Woche Bearbeitungszeit eingeräumt. Wer Aufgaben abgibt, „votiert“ automatisch für diese Aufgaben. Falls die Aufgaben im Zweifel unzureichend vorgerechnet werden, gibt es 0 Punkte für das gesamte Blatt.

### Randomisierte Algorithmen

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

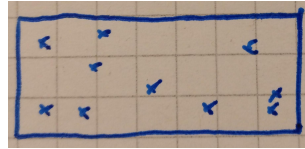
Randomisierte Algorithmen werden nach „dem Verbrauch von Zufall“ bewertet, denn Zufall zu erzeugen, ist nicht so ohne Weiteres möglich (Zufall zu „erzeugen“ ist **teuer**). Im Gegensatz dazu bewertet man „normale“ Algorithmen nach Platz- und Zeitbedarf.

Wir haben bisher die folgenden Gattungen kennengelernt:

- **Las Vegas:** Die Laufzeit hängt vom Zufall ab, die Korrektheit nicht.
- **Monte Carlo:** Die Laufzeit hängt nicht vom Zufall ab, die Korrektheit schon.

## Closest Pair (Randomisiert, Las Vegas)

Gegeben seien  $n$  Punkte im  $\mathbb{R}^2$



**Ziel:** Finde das Paar mit dem kleinsten Abstand zueinander.

### Naiver Ansatz

- Betrachte  $P_1$  und berechne Distanzen zu  $P_2, P_3, \dots, P_n$
- Betrachte  $P_2$  und berechne Distanzen zu  $P_3, P_4, \dots, P_n$

Das würde eine Laufzeit von

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

ergeben. Ein Prozessor mit 1 GHz Taktfrequenz schafft 1.000.000 Instruktionen pro Sekunde. Das führt zu folgender Tabelle:  $O(n^2)$  ist für das Problem nicht praktikabel.

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	$1000000^2 \cdot 10^{-9}$	$10^3$	15 min

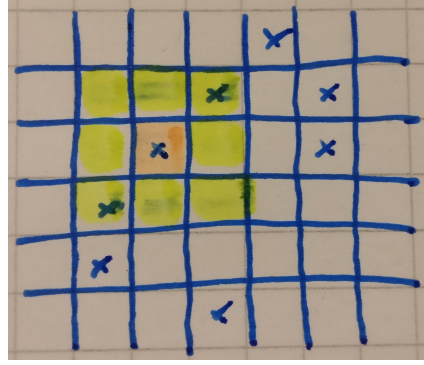
### Randomisierter Algorithmus für CP

Ein randomisierter Algorithmus mit *erwarteter Laufzeit* (Varianten: Glück Pech) hat eine Laufzeit von  $O(n)$ . Er wird aus folgendem inkrementellen Ansatz hergeleitet:

Zuerst wird die Datenmenge  $P = \{P_1, P_2, P_3, \dots, P_n\}$  betrachtet. Sei nun  $\delta_i$  die CP-Distanz in der Menge  $P$ . Es kommt nun ein weiterer Punkt  $P_{i+1}$  hinzu.

Eine einfache Möglichkeit, den neuen geringsten Abstand zu finden, wäre, wenn man die Abstände vom neuen Punkt zu allen anderen Punkten vergleicht und überprüft, ob es einen kleineren Abstand als das bisherige  $\delta_i$  gibt.

Besser wäre jedoch Folgendes: Angenommen,  $\delta_i$  ist bestimmt. So kann nun ein Gitter erzeugt werden, das eine Maschenweite (Breite der Zeilen/Spalten) von genau  $\delta_i$  hat.



Das Einfügen folgt einem einfachen Ablauf

1. Lokalisiere  $P_{i+1}$  im Gitter (orange)
2. Inspiziere alle Punkte im Feld von  $P_{i+1}$  sowie alle Felder um  $P_{i+1}$  herum (gelb)
  - a)  $\delta_{i+1} = \delta_i \rightarrow$  Nichts mehr tun  $\rightarrow O(1)$
  - b)  $\delta_{i+1} < \delta_i \rightarrow$  Baue Gitter mit neuer Maschenweite  $\delta_{i+1} \rightarrow O(n^2)$  im schlimmsten Fall

Da es auch Fälle geben kann, in denen  $P_{i+1}$  immer einen kleineres  $\delta$  hat als davor (Liste nach größtem Abstand geordnet), empfiehlt es sich, die Punkte immer in zufälliger Reihenfolge „einzufügen“. So bleibt eine Wahrscheinlichkeit von  $\frac{2}{i+1}$  dafür, dass der nächste Punkt ein CP-Punkt ist.

Die erwarteten Kosten des Einfügens sind

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

Damit lässt sich der Erwartungswert berechnen zu

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

**Lemma 1** Die Wahrscheinlichkeit, beim Einfügen von  $P_{i+1}$  das Gitter neu aufbauen zu müssen, ist  $< \frac{2}{i+1}$ .

[**Beweis**] Das Gitter muss genau dann neu aufgebaut werden, wenn  $P_{i+1}$  einer der beiden Punkte ist, welche das CP in der Menge der ersten  $i+1$  Punkte bestimmen. Jeder der ersten  $i+1$  Punkte ist mit gleicher Wahrscheinlichkeit der  $P_{i+1}$ .

Falls CP eindeutig:

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

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

### Closest Pair mit deterministischem Algorithmus

Closest Pair kann in  $O(n \cdot \log(n))$  berechnet werden. Es kann sogar gezeigt werden, dass CP mindestens  $\Omega(n \cdot \log(n))$  braucht.

Deterministisch geht CP vergleichsbasiert nicht besser als  $\Omega(n \cdot \log(n))$

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

[**Beweis**] Man weiß, dass Elementuniqueness (geg  $n$  Zahlen; entscheide, ob eine Zahl doppelt vorkommt)  $\Omega(n \cdot \log(n))$  braucht. Falls CP deterministisch vergleichsbasiert besser als  $o(n \cdot \log(n))$  gelöst werden könnte, so könnte man auch Elementuniqueness in dieser Zeit lösen (Zahl  $i \rightarrow \text{Punkt } (i, i) \in \mathbb{R}^2$ ). Allerdings gilt das nur in einem anderem Rechenmodell (Randomisierung und Abrundung).

---

## Anhang

### Zufallsvariable und Erwartungswert

Sei  $Y$  eine Zufallsvariable, zum Beispiel *Augenzahl beim Wurf* mit einem normalen und fairen Würfel.

Der Erwartungswert berechnet sich zu

$$E[X] = \sum \text{Ereignis} \cdot P(\text{Ereignis})$$

Für  $Y$  also:  $\frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 4 + \frac{1}{6} \cdot 5 + \frac{1}{6} \cdot 6 = 3,5$