

## Vorlesungsmitschrieb

# Algorithmen und Berechenbarkeit

## Vorlesung 04

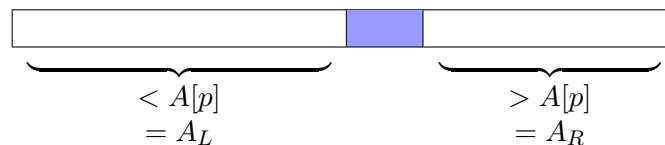
Letztes Update: 2017/11/13 - 23:28 Uhr

### Las-Vegas-Algorithmus: Randomisierter Quicksort-Algorithmus

Eingabe A: [1, 2, 3, ..., n]

Wähle p aus Eingabe zufällig gleichverteilt

Füge p an der richtigen Stelle ein



Einfügen eines Elements an der *richtigen* Stelle

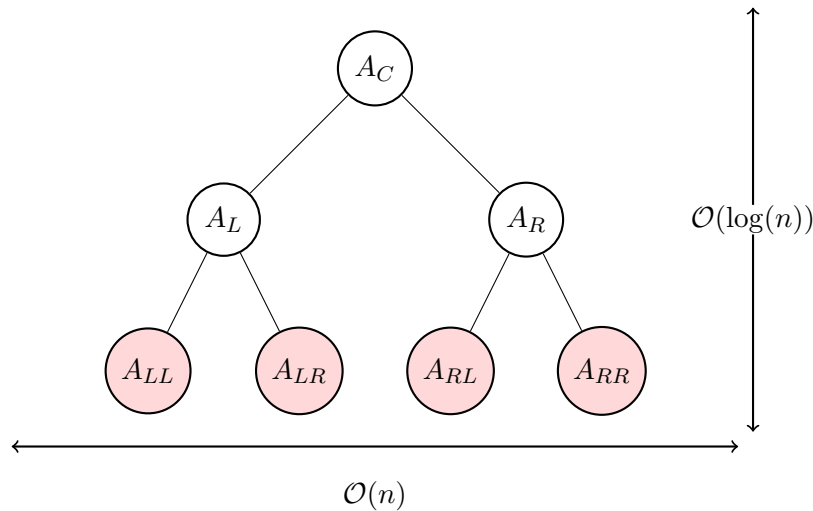
Der Algorithmus wird dann für die linke Seite  $RQS(A_L)$  bzw.  $RQS(A_R)$  rekursiv aufgerufen. Um  $p$  *zufällig gleichverteilt* aufzurufen, können verschiedene Ansätze gewählt werden. Zum Beispiel nimmt man immer das mittige Element ( $\frac{n}{2}$ ) oder man berechnet den Median (das klappt in  $O(n)$ ) und wählt das entsprechende Element aus. Beide Fälle sind suboptimal.

### Versuch einer Analyse

Zuerst stellt man sich die folgende **Frage**: Was ist die erwartete Länge von  $A_L$  bzw.  $A_R$ ?

$$E(|A_L|) = E(|A_R|) = \sum_{i=0}^{n-1} \frac{1}{n} \cdot i = \frac{1}{n} \cdot \frac{(n-1) \cdot n}{2} = \frac{n-1}{2}$$

Daraus folgt, dass die Tiefe des Rekursionsbaums  $\mathcal{O}(\log(n))$  sein muss.



Die Gesamtlaufzeit ergibt sich somit als  $\mathcal{O}(n \cdot \log(n))$

Dieser Ansatz der Analyse kann nicht richtig sein, denn folgender Quicksort-Algorithmus hätte mit derselben Folgerung dieselbe Laufzeit, was aber nicht stimmt.

```
Eingabe A: [1, 2, 3, ..., n]
  Wähle p
    mit 50% Wahrscheinlichkeit als kleinstes Element
    mit 50% Wahrscheinlichkeit als größtes Element
  Füge p an der richtigen Stelle ein
```

Man wählt  $p$  also entweder als erstes (kleinstes) oder letztes (größtes) Element, und startet den Algorithmus rekursiv erneut. Nach obiger Rechnung ergäbe sich auch hier wieder eine Länge von

$$E(|A_l|) = E(|A_r|) = \frac{1}{2} \cdot 0 + \frac{1}{2}(n-1) = \frac{n-1}{2}$$

Da bei einem rekursiven Aufruf aber jeweils wieder  $n-1$  Elemente in der Eingabe enthalten sind, kann das also nicht stimmen.

### Korrekte Analyse des Random Quicksort-Algorithmus

Wir gehen zunächst von folgender **Annahme** aus: Die zu sortierenden Elemente sind  $s_1, s_2, s_3, \dots, s_n$  und außerdem gilt  $s_1 < s_2 < s_3 < \dots < s_n$ . Die Zufallsvariable wird festgelegt als

$$X_{ij} = \begin{cases} 1 & \text{falls während des Alorithmus } s_i \text{ mit } s_j \text{ verglichen} \\ 0 & \text{sonst} \end{cases}$$

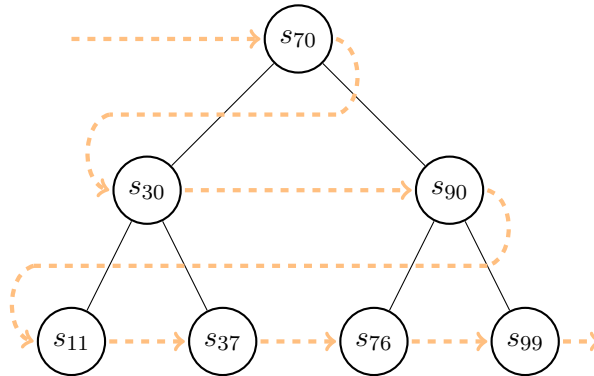
Die Gesamtlaufzeit des Random Quicksort-Algorithmus ist damit

$$\begin{aligned}
\sum_{i < j} X_{ij} &= \sum_{i=1}^{n-1} \cdot \sum_{j=i+1}^n X_{ij} \\
E\left(\sum_{i < j} X_{ij}\right) &= \sum_{i < j} E(X_{ij}) \quad (\text{Linearität des Erwartungswerts}) \\
&= \sum_{i < j} (0 \cdot P(X_{ij} = 0) + 1 \cdot P(X_{ij} = 1)) \\
&= \sum_{i < j} \underbrace{P(X_{ij} = 1)}_{P_{ij}}
\end{aligned}$$

Als Nächstes soll nun  $P_{ij}$  berechnet werden, wobei folgende Überlegungen zu beachten sind:

- Elemente mit großer Rangdifferenz werden nur mit geringer Wahrscheinlichkeit miteinander verglichen, da viele Chancen bestehen, dass sie im Rekursionsbaum getrennt werden.
- Elemente mit kleiner Rangdifferenz werden mit größerer Wahrscheinlichkeit miteinander verglichen.
- Elemente mit der Rangdifferenz von 1 müssen miteinander verglichen werden.

In folgendem Ablauf des Random Quicksorts entspricht ein Knoten einem Pivotelement im entsprechenden Aufruf.



Die erzeugte Permutation ist  $\pi = s_{70}, s_{30}, s_{90}, s_{11}, s_{37}, s_{76}, s_{99}$

*Anmerkung:  $\pi$  ist nicht zufällig gleichverteilt, da manche Permutationen gar nicht vorkommen können.*

Als Nächstes stellt sich die **Frage**, wie es sich auf die Laufzeit auswirkt, wenn  $s_i$  nicht verglichen wird.

Dann taucht ein Element  $s_k$  mit  $s_i < s_k < s_j$  als Pivotelement vor  $s_i$  und vor  $s_j$  in  $\pi$  auf, denn  $s_i$  wird genau dann mit  $s_j$  verglichen, wenn  $s_i$  oder  $s_j$  das erste Element aus  $s_i, s_{i+1}, \dots, s_j$  in  $\pi$  ist. Außerdem gilt: Mit gleicher Wahrscheinlichkeit ist jedes der Elemente  $s_i, s_{i+1}, \dots, s_j$  das erste, das in  $\pi$  auftaucht.

Die Wahrscheinlichkeit, dass  $s_i$  mit  $s_j$  verglichen wird, beträgt also

$$P_{ij} := P(s_i \text{ wird mit } s_j \text{ verglichen}) = \frac{2}{j - i + 1}$$

Damit kann nun die Laufzeit eingeordnet werden:

$$\begin{aligned}
E\left(\sum_{r < j} X_{ij}\right) &= \sum_{i=1}^{n-1} \cdot \sum_{j=i+1}^n E(X_{ij}) \\
&= \sum_{i=1}^{n-1} \cdot \sum_{j=i+1}^n P_{ij} \\
&= \sum_{i=1}^{n-1} \cdot \sum_{j=i+1}^n \underbrace{\frac{2}{j-i+1}}_{\frac{2}{2} + \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \dots} \\
&\leq \sum_{i=1}^{n-1} \cdot \sum_{l=2}^n \frac{2}{l} \\
&= \sum_{i=1}^{n-1} 2 \cdot \sum_{l=2}^n \frac{1}{l} \\
&= \sum_{i=1}^{n-1} 2 \cdot \mathcal{O}(H_n) \\
&= \sum_{i=1}^{n-1} 2 \cdot \mathcal{O}(\log(n)) \\
&= \mathcal{O}(n \cdot \log(n))
\end{aligned}$$

Anmerkung:  $H_n$  beschreibt die  $n$ -te harmonische Zahl.

## Zero-Knowledge-Proof

Zunächst soll eine **Anwendung** des "Zero-Knowledge-Proofs" beschrieben werden: Wie könnte eine Bank beim Onlinebanking beweisen, dass sie weiterhin die Bank ist (die betrachtete Website weiterhin zur Bank gehört)?

Ein möglicher **Ansatz** besagt, dass die Bank (im Folgenden auch einfach **Alice**) ihrem Kunden (**Bob**) beweisen muss, dass sie ein Geheimnis kennt, welches nur Alice bekannt sein kann, allerdings ohne das Geheimnis zu verraten.

Außerdem sollen die folgenden **Eigenschaften** erfüllt sein:

- Falls Alice das Geheimnis nicht kennt, wird das mit hoher Wahrscheinlichkeit von Bob erkannt.
- Alice gibt nichts über das Geheimnis preis, was Bob nicht alleine rausfinden könnte.

Das Protokoll basiert auf dem Graphisomorphie-Problem (Isomorphismus zwischen zwei öffentlichen Graphen  $G_1$  und  $G_2$ ).

Alice kann  $G_1$  und  $G_2$  wie folgt erzeugen: Sie wählt  $G_1$  zufällig und außerdem eine zufällige Permutation  $\Phi$  der Knoten von  $G_1$ , was dann  $G_2$  definiert.  $\Phi$  bleibt geheim, wohingegen  $G_1$  und  $G_2$  veröffentlicht werden.

### Protokoll, durch welches Alice Bob überzeugt, P zu kennen:

Alice permutiert  $G_j$  wobei  $j \in \{1, 2\}$  mit einer zufälligen Permutation  $\pi$  zu einem Graphen  $H$ . Dabei stellt sie sicher, dass  $H \neq G_1$  und  $H \neq G_2$ . Außerdem sendet sie  $H$  an Bob.

Bob möchte sich davon überzeugen, dass Alice die Isomorphie zwischen  $G_1$  und  $G_2$  kennt, indem er  $k \in \{1, 2\}$  zufällig wählt und Alice bittet, die Isomorphie zwischen  $G_k$  und  $H$  offenzulegen. Falls nun Alice das Geheimnis  $\Phi$  kennt, so kann sie **immer**  $\Phi$  bzw.  $\Phi \circ \pi$  zurückgeben. Falls Alice  $\Phi$  nicht kennt und Bob den Graphen wählt, aus dem Alice  $H$  generiert hat, kann sie leicht antworten, falls jedoch Bob den anderen Graphen gewählt hat, müsste sie das Graphisomorphie-Problem lösen, um die korrekte Antwort zu geben.

Bei  $x$ -maliger Wiederholung ist die Wahrscheinlichkeit, dass Bob immer den richtigen Graphen erwischt, aus dem  $H$  erzeugt wurde

$$\frac{1}{2^x}$$

Daraus folgt, dass die Wahrscheinlichkeit, dass Bob erkennt, dass Alice  $\Phi$  nicht kennt,

$$1 - \frac{1}{2^x}$$

beträgt.

**Theorem 1** *Alice verrät nichts über  $\Phi$ .*

[**Beweis**] Bob lernt Isomorphie zwischen z.B.  $G_1$  und einer zufälligen Permutation von  $G_1$ . Das hätte er auch selber herausfinden können.

## Anhang

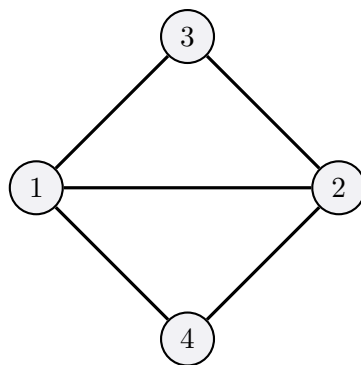
### Graphisomorphie

Gegeben seien zwei Graphen  $G_1(V_1, E_1)$  und  $G_2(V_2, E_2)$ . Graphisomorphie bedeutet, es gibt eine bijektive Funktion  $\phi$ , sodass

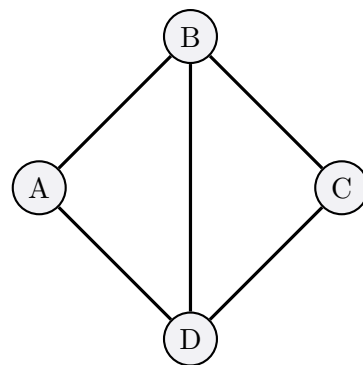
$$\forall a = (v, w) \in E_1 \Leftrightarrow (\phi(v), \phi(w)) \in E_2$$

### Graphisomorphie-Problem

Gegeben seien zwei Graphen  $G_3(V_3, E_3)$  und  $G_4(V_4, E_4)$  und man stellt sich die **Frage**: Wann sind  $G_1$  und  $G_2$  isomorph?



Graph  $G_3$



Graph  $G_4$

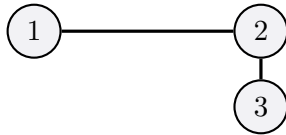
Die folgende Tabelle zeigt die Isomorphie zwischen  $v \in V_3$  und  $\phi(v)$ .

$v \in V_3$  und  $\Phi(v)$  sind nicht isomorph. Festgestellt werden kann dies zum Beispiel über den Knotengrad oder durch die folgende Erkenntnis:

$$\begin{aligned} 2, 4 &\in G_3 \\ A, C &\notin G_4 \end{aligned}$$

| $v \in V_3$ | $\phi(v)$ | $\Phi(v)$ |
|-------------|-----------|-----------|
| 1           | $B$       | $B$       |
| 2           | $D$       | $A$       |
| 3           | $C$       | $D$       |
| 4           | $A$       | $C$       |

Es folgen weitere Beispiele:

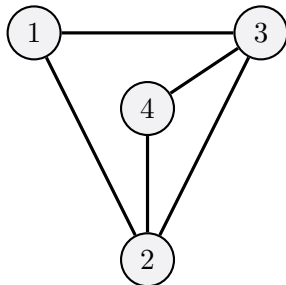


Graph  $G_5$

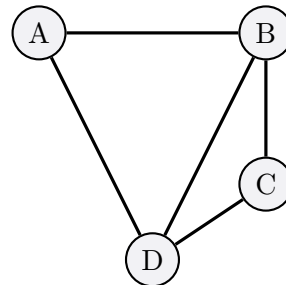


Graph  $G_6$

Die nicht isomorphen Graphen  $G_5$  und  $G_6$

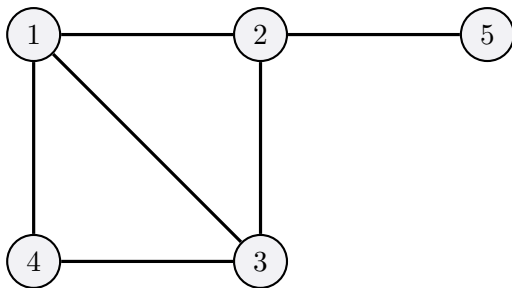


Graph  $G_7$

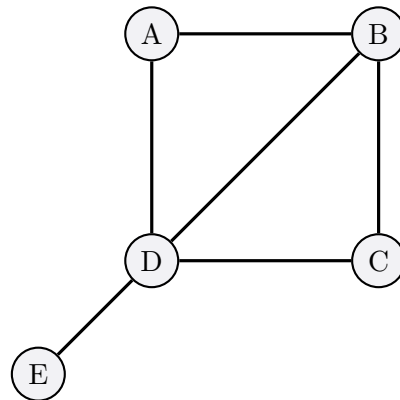


Graph  $G_8$

Die isomorphen Graphen  $G_7$  und  $G_8$



Graph  $G_9$



Graph  $G_{10}$

Die nicht isomorphen Graphen  $G_9$  und  $G_{10}$  ( $D$  hat Grad 4, 2 hat Grad 3)

Anmerkung: Bis zum Jahr 2015 dachten Informatiker, Graphenisomorphie läge in **NP-schwer**. Im Jahr 2015 wurde jedoch gezeigt, dass das Problem der Graphenisomorphie in Quasipoly-nomzeit lösbar ist:

$$\mathcal{O}(2^{O(\log(n))^2})$$