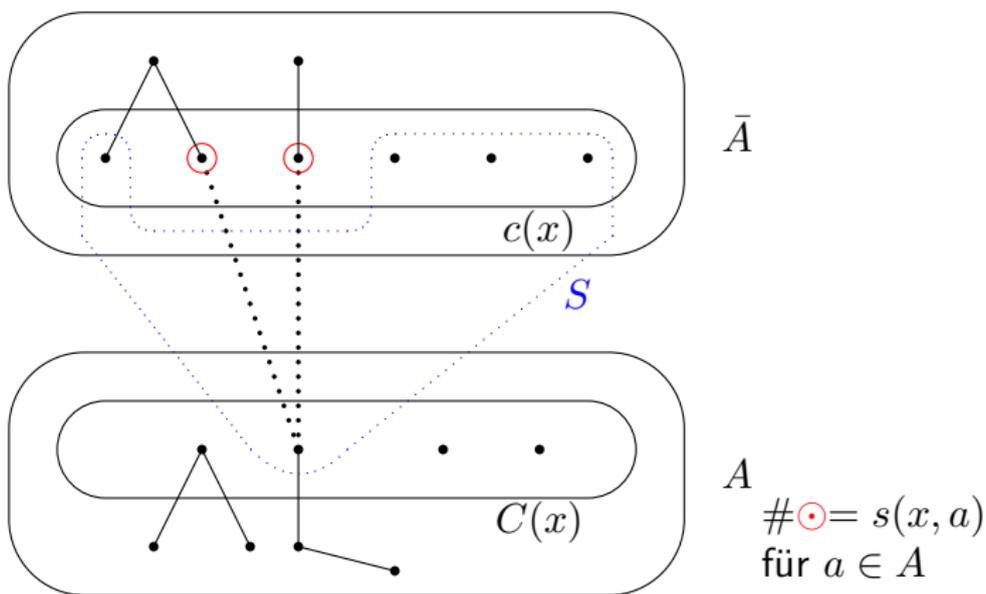


## Beweis (Forts.):

Fall 2:  $s(x, a) < s$ .

Fall 2.1:  $|c(x)| \geq p$ .



## Beweis (Forts.):

Sei  $S := c(x) \cup \{a\} \setminus$  die Elemente, die in  $s(x, a)$  gezählt werden. Der Gegenspieler antwortet nun so, dass das Ergebnis das kleinste Element in  $S$  wird. Ist  $w$  ein Blatt in  $T_A$  unter  $x$ , so ist  $\text{little}(w) = A - \{a\}$ . Der Entscheidungsbaum  $T$  wird also gemäß folgender Regeln gestutzt (sei  $y$  der aktuelle Knoten und seien  $e$  und  $f$  die beiden Elemente, die in  $y$  verglichen werden):

2.1: falls  $e, f \in S$ , dann bleibt  $y$  erhalten

2.2: andernfalls sei o.B.d.A.  $e \in A \setminus S$  oder  $f \in \bar{A} \setminus S$ ; ersetze  $y$  mit seinem Unterbaum durch das Kind von  $y$  und dessen Unterbaum, das der Antwort auf  $e < f$  entspricht.

Da überhalb des kritischen Knoten  $x$  keine zwei Elemente in  $S$  verglichen wurden, muss der Unterbaum von  $T_A$  unterhalb von  $x$  Tiefe  $\geq |S| - 1$  haben.

## Beweis (Forts.):

Zusammen mit Phase 1 ergibt sich eine Tiefe von  $T_A$ :

$$\begin{aligned} &\geq n - r - |c(x)| + |S| - 1 \\ &\geq n - r - |c(x)| + |c(x)| + 1 - (s - 1) - 1 \\ &= n - r - s + 1 \\ &= n - p \end{aligned}$$

## Beweis (Forts.):

Fall 2.2:  $|c(x)| < p$ . Sei  $S := C(x)$ .

Die Regeln für Phase 2 sind in diesem Fall so, dass der Algorithmus als Antwort das Maximum von  $S$  bestimmt. Damit ergibt sich für die Tiefe von  $T_A$ :

$$\begin{aligned} &\geq n - r - |c(x)| + |S| - 1 \\ &\geq n - r - (p - 1) + r - 1 \\ &= n - p. \end{aligned}$$

## Beweis (Forts.):

Insgesamt ergibt sich also: Jeder Pfad in  $T_A$  von  $x$  zu einem Blatt hat mindestens die Länge  $n - p$ . Also enthält jedes  $T_A$  mindestens  $2^{n-p}$  Blätter (von  $T$ ).

Alle  $T_A$ 's zusammen enthalten  $\geq \binom{n}{i} 2^{n-p}$  Blätter von  $T$ , wobei jedes Blatt von  $T$  höchstens  $n - i + 1$  mal vorkommt: Sei  $w$  Blatt von  $T$ , dann ist  $\text{little}(w)$  eindeutig bestimmt und es muss, falls  $T_A$  das Blatt  $w$  enthält,  $\text{little}(w) \subseteq A$  sein.

Für das Element in  $A \setminus \text{little}(w)$  gibt es  $\leq n - i + 1$  Möglichkeiten. Damit ist die Anzahl der Blätter von  $T \geq \frac{1}{n-i+1} \binom{n}{i} 2^{n-p}$  und

$$\text{Höhe}(T) \geq \log \left[ \binom{n}{i} \frac{2^{n-p}}{n-i+1} \right].$$





L. Hyafil:

*Bounds for selection*

SIAM J. Comput. **5**, pp. 109–114 (1976)



Sam Bent, John W. John:

*Finding the median requires  $2n$  comparisons*

Proc. 17th Annual ACM Symposium on Theory of Computing,  
pp. 213–216 (1985)



John Welliaveetil John:

*A new lower bound for the set-partitioning problem*

SIAM J. Comput. **17**, pp. 640–647 (1988)



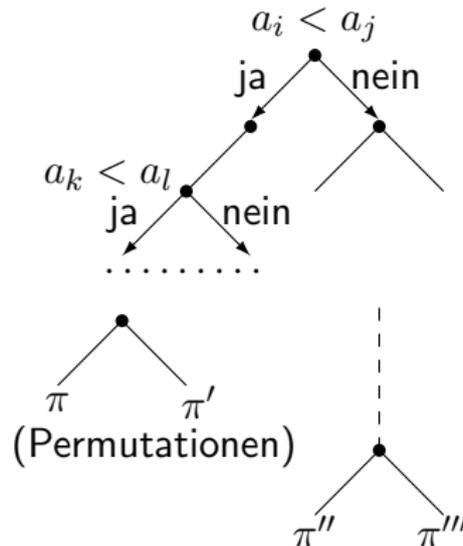
Dorit Dor, Uri Zwick:

*Selecting the median*

SIAM J. Comput. **28**(5), pp. 1722–1758 (1999)

## 7. Untere Schranke für (vergleichsbasiertes) Sortieren

Gegeben  $n$  Schlüssel, Queries  $a_i < a_j$ . Dies entspricht einem Entscheidungsbaum:



**Beobachtung:** Jede Permutation  $\pi \in S_n$  kommt in mindestens einem Blatt des Entscheidungsbaums vor. Da  $|S_n| = n!$ , folgt, dass die Tiefe des Entscheidungsbaums

$$\geq \log_2 n! .$$

**Stirling'sche Approximation:**  $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

Also

$$\begin{aligned} \text{Tiefe} &\geq \log_2 \left[ \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \right] = n \log_2 n - n \log_2 e + \frac{1}{2} \log_2 (2\pi n) \\ &= n \log_2 n - \mathcal{O}(n) . \end{aligned}$$

## Satz 91

*Jeder vergleichsbasierte Sortieralgorithmus benötigt für das Sortieren von  $n$  Schlüsseln mindestens*

$$n \log_2 n - \mathcal{O}(n)$$

*Vergleiche.*

## 8. Bucketsort im Schnitt

Gegeben seien  $n$  zufällig und gleichverteilt gewählte Schlüssel im Intervall  $]0, 1]$ . Um diese zu sortieren:

- 1 Initialisiere Buckets  $b_1, \dots, b_n$
- 2 Lege  $a_i$  in Bucket  $\lceil n \cdot a_i \rceil$
- 3 Sortiere Schlüssel innerhalb eines jeden Buckets
- 4 Konkatiniere die Buckets

### Satz 92

*Wird zum Sortieren ein Sortieralgorithmus mit einer Zeitkomplexität von  $\mathcal{O}(n^2)$  verwendet, dann hat obiger Bucketsort-Algorithmus im Durchschnitt eine **lineare** Laufzeit.*

## Beweis:

Sei  $n_i$  die Anzahl der Elemente im Bucket  $b_i$  (nach Schritt 2). Die Schritte 1,2 und 4 benötigen zusammen Zeit  $\mathcal{O}(n)$ . Die Zeit für Schritt 3 beträgt:

$$\mathcal{O}\left(\sum_{i=1}^n n_i^2\right).$$

Die Erwartungswert für  $\sum n_i^2$  lässt sich folgendermaßen abschätzen:

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^n n_i^2\right] &\leq c \sum_{1 \leq j < k \leq n} \Pr[a_j \text{ und } a_k \text{ enden im gleichen Bucket}] \\ &= c \left( \sum_{1 \leq j < k \leq n} \frac{1}{n} \right) = \mathcal{O}(n).\end{aligned}$$



## 9. Quicksort

Wie bei vielen anderen Sortierverfahren (Bubblesort, Mergesort, usw.) ist auch bei Quicksort die Aufgabe, die Elemente eines Array  $a[1..n]$  zu sortieren.

Quicksort ist ein Divide-and-Conquer-Verfahren.

**Divide:** Wähle ein **Pivot-Element**  $p$  (z.B. das letzte) und partitioniere  $a[l..r]$  gemäß  $p$  in zwei Teile  $a[l..i - 1]$  und  $a[i + 1..r]$  (durch geeignetes Vertauschen der Elemente), wobei abschließend  $a[i] = p$ .

**Conquer:** Sortiere  $a[l..i - 1]$  und  $a[i + 1..r]$  rekursiv.

## Algorithmus:

```
proc qs( $a, l, r$ )  
  if  $l \geq r$  then return fi  
  #ifndef Variante 2  
    vertausche  $a[\text{random}(l, r)]$  mit  $a[r]$   
  #endif  
   $p := a[r]$   
   $i := l; j := r$   
  repeat  
    while  $i < j$  and  $a[i] \leq p$  do  $i++$  od  
    while  $i < j$  and  $p \leq a[j]$  do  $j--$  od  
    if  $i < j$  then vertausche  $a[i]$  und  $a[j]$  fi  
  until  $i = j$   
  vertausche  $a[i]$  und  $a[r]$   
  qs( $a, l, i - 1$ )  
  qs( $a, i + 1, r$ )
```

### **Bemerkung:**

Der oben formulierte Algorithmus benötigt pro Durchlauf für  $n$  zu sortierende Schlüssel in der Regel  $n + 1$  Schlüsselvergleiche.

Lediglich wenn die beiden Indizes zusammenfallen (was nur bei einem Pivotduplikat passieren kann), benötigt er nur  $n$  Vergleiche.

Durch geschicktes Einstreuen von **if**-Abfragen kann man in jedem Fall mit  $n - 1$  Schlüsselvergleichen auskommen.

## Komplexität von Quicksort:

- Best-case-Analyse: Quicksort läuft natürlich am schnellsten, falls die Partitionierung möglichst ausgewogen gelingt, im Idealfall also immer zwei gleichgroße Teilintervalle entstehen, das Pivot-Element ist dann stets der Median.

Anzahl der Schlüsselvergleiche:

$$\sum_{i=1}^{\log n} (n + 1) = (n + 1) \log n \approx n \log n$$

- Worst-case-Analyse: Z.B. bei einer aufsteigend sortierten Eingabe.

Anzahl der Schlüsselvergleiche:

$$\Omega(n^2)$$

- Average-case: Da die Laufzeit von Quicksort sehr stark von den Eingabedaten abhängt, kann man die Frage stellen, wie lange der Algorithmus “im Mittel“ zum Sortieren von  $n$  Elementen braucht. Um jedoch überhaupt eine derartige Analyse durchführen zu können, muss man zuerst die genaue Bedeutung von “im Mittel“ festlegen. Eine naheliegende Annahme ist, dass alle möglichen Permutationen der Eingabedaten mit gleicher Wahrscheinlichkeit auftreten.

## Satz 93

Die durchschnittliche Anzahl von Schlüsselvergleichen von Quicksort beträgt unter der Annahme, dass alle Permutationen für die Eingabe gleichwahrscheinlich sind, höchstens

$$C_n = 2(n+1)(H_{n+1} - 1) \approx 2n \ln n - 0.846n + o(n) \approx 1.386n \log n$$

wobei  $H_n := \sum_{i=1}^n i^{-1}$  die  $n$ -te *Harmonische Zahl* ist.

### Beweis:

(Variante mit  $n - 1$  Vergleichen pro Durchlauf)

Sei  $C_n$  die Anzahl der Vergleiche bei einem Array der Länge  $n$ .

$$C_0 = C_1 = 0.$$

$$C_n = n - 1 + \frac{1}{n} \sum_{j=1}^n (C_{j-1} + C_{n-j})$$

## Beweis (Forts.):

Da

- i) (in beiden Varianten) das  $j$ -kleinste Element bestimmt wird und
- ii) auch für die rekursiven Aufrufe wieder alle Eingabepermutationen gleichwahrscheinlich sind:

$$\Rightarrow C_n = n - 1 + \frac{2}{n} \sum_{j=0}^{n-1} C_j;$$

$$nC_n = n^2 - n + 2 \sum_{j=0}^{n-1} C_j;$$

$$(n - 1)C_{n-1} = (n - 1)^2 - (n - 1) + 2 \sum_{j=0}^{n-2} C_j;$$

## Beweis (Forts.):

$$nC_n - (n-1)C_{n-1} = 2n - 1 - 1 + 2C_{n-1};$$

$$nC_n = 2n - 2 + (n+1)C_{n-1}; \quad / \frac{1}{n(n+1)}$$

$$\begin{aligned} \frac{C_n}{n+1} &= 2 \frac{n-1}{n(n+1)} + \frac{C_{n-1}}{n} = \\ &= 2 \frac{n-1}{n(n+1)} + 2 \frac{n-2}{(n-1)n} + \frac{C_{n-2}}{n-1} = \\ &= 2 \frac{n-1}{n(n+1)} + \dots + \frac{C_2}{3} = \\ &= 2 \left[ \sum_{j=2}^n \frac{j-1}{j(j+1)} \right] = \end{aligned}$$

## Beweis (Forts.):

$$\begin{aligned} &= 2 \left[ \sum_{j=2}^n \left( \frac{j}{j(j+1)} - \frac{1}{j(j+1)} \right) \right] = \\ &= 2 \left[ H_{n+1} - \frac{3}{2} - \sum_{j=2}^n \left( \frac{1}{j} - \frac{1}{j+1} \right) \right] = \\ &= 2 \left[ H_{n+1} - \frac{3}{2} - \left( \frac{1}{2} - \frac{1}{n+1} \right) \right] = \\ &= 2 \left[ H_{n+1} - 2 + \frac{1}{n+1} \right] \end{aligned}$$

$$\Rightarrow C_n = 2(n+1) \left( H_{n+1} - 2 + \frac{1}{n+1} \right);$$

Mit  $H_n \approx \ln n + 0.57721 \dots + o(1)$

$$\Rightarrow C_n \approx 2n \ln n - 4n + o(n) \approx 1.386n \log n$$



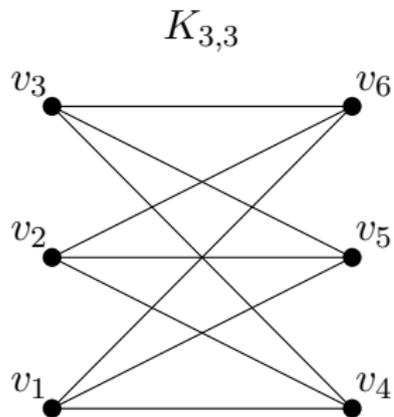
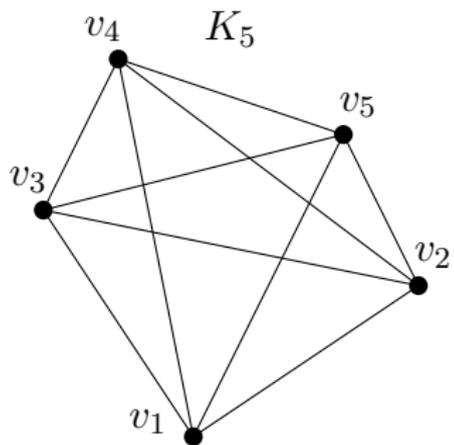
# Kapitel IV Minimale Spannbäume

## 1. Grundlagen

Ein Graph  $G = (V, E)$  besteht aus einer Menge  $V$  von Knoten und einer Menge  $E$  von Kanten. Wir werden nur endliche Knoten- (und damit auch Kanten-) Mengen betrachten. Die Anzahl der Knoten bezeichnen wir mit  $n$  ( $|V| = n$ ), die Anzahl der Kanten mit  $m$  ( $|E| = m$ ). Eine **gerichtete** Kante mit den Endknoten  $u$  und  $v$  wird mit  $(u, v)$ , eine **ungerichtete** mit  $\{u, v\}$  notiert. Eine Kante  $(v, v) \in E$  (bzw.  $\{v, v\}$ ) heißt **Schlinge**. Falls  $E$  eine Multimenge ist, spricht man von **Mehrfachkanten**. Kanten  $(u, v)$  und  $(v, u)$  heißen **antiparallel**. Graphen ohne Schlingen und Mehrfachkanten heißen **einfache** Graphen (engl. **simple**). Für einfache ungerichtete Graphen gilt daher:

$$E \subseteq \binom{V}{2} := \{X \subseteq V, |X| = 2\}$$

## Graphische Darstellung:



Ist  $E \subseteq V \times V$ , dann heißt  $G$  gerichteter Graph (engl. digraph).



Der zu  $G$  gehörige ungerichtete Graph ist  $G' = (V, E')$ .  $E'$  erhält man, indem man in  $E$  die Richtungen weglässt und Mehrfachkanten beseitigt.

Sei  $v \in V$ . Unter der Nachbarschaft

$N(v) := \{w; (v, w) \text{ oder } (w, v) \in E\}$  eines Knotens  $v$  versteht man die Menge der direkten Nachbarn von  $v$ . Der Grad eines Knotens ist definiert als:

$$\deg(v) = \begin{cases} |N(v)| & ; \text{ falls } G \text{ ungerichtet und} \\ \text{indeg}(v) + \text{outdeg}(v) & ; \text{ falls } G \text{ gerichtet.} \end{cases}$$

Dabei ist  $\text{indeg}(v)$  die Anzahl aller Kanten, die  $v$  als Endknoten, und  $\text{outdeg}(v)$  die Anzahl aller Kanten, die  $v$  als Anfangsknoten haben.

**Beobachtung:** Für einfache (gerichtete oder ungerichtete) Graphen gilt

$$\sum_{v \in V} \deg(v) = 2|E|.$$

### Korollar 94

*In jedem (einfachen) Graphen ist die Anzahl der Knoten mit ungeradem Grad eine gerade Zahl.*