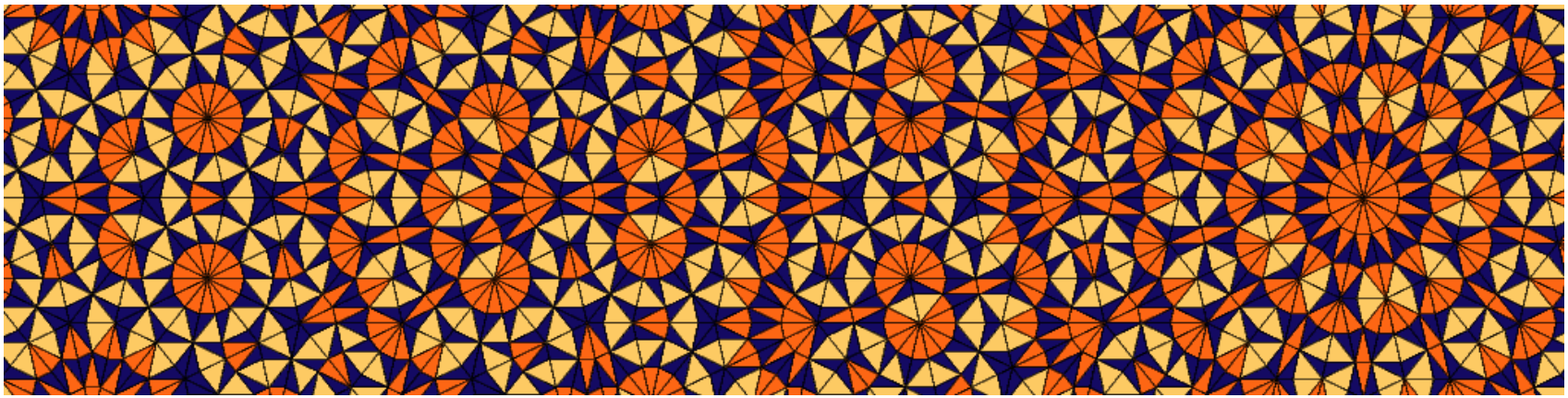
Panorama der Mathematik und Informatik

19: Algorithmen I/II: Maximum subarray problem / Analoge Algorithmen

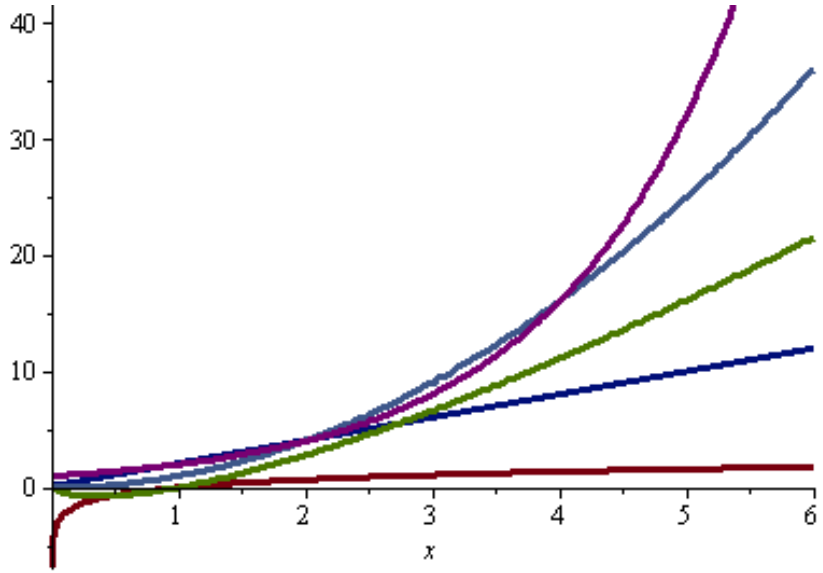
Dirk Frettlöh
Technische Fakultät / Richtig Einsteigen



Algorithmen

Recall “Algorithmen und Datenstrukturen”, “Theoretische Informatik”, “Algorithmen der Informatik”:

- (Theoretische) Rechnermodelle: Registermaschinen, endliche Automaten, ...
- Laufzeiten: $O(\log n), O(n), O(n \log n), O(n^2), \ldots, O(2^n), \ldots$
- Entwurfsmethoden für Algorithmen: Greedy, Divide & Conquer, Branch & Bound...



In den nächsten paar Wochen ein paar ergänzende Ideen:

- Schönes Beispiel: Maximum subarray problem
- Analoge Algorithmen (im Ggs. zu “digital”)
- Googles PageRank^{TM}
- Bildkompression: png, jpeg
- Schnelle Multiplikation
- Schon gesehen: Kryptographie: RSA, Diffie-Hellman, PGP

Maximum subarray problem

Gegeben eine endliche Folge (array) $a_1, \ldots, a_n$.

Setze $f(i,j) = a_i + \cdots + a_j$ $(1 \leq i \leq j \leq n)$.

Gesucht i, j so dass $f(i,j)$ maximal wird. Z.B:

$$-2, 1, -3, 4, -1, 2, 1, -5, 4, -2, 1, 1, -3, 2, 1$$

Hier: Maximal bei $f(4,7) = 4 + (-1) + 2 + 1 = 6$.

Wie finden wir effizient das Maximum?

Zugriff auf a_i und Zuweisungen seien gratis.
Vergleiche und **Additionen** kosten je 1 Einheit.

Versuch 1: Der naive Algorithmus.

Berechne systematisch nacheinander alle $f(i,j)$.
Bestimme den Maximalwert.

```
max:=0;
for j from 1 to n;
  for i from 1 to j;
    z := a_i + ··· + a_j;
    if z>max then max=z;
  end for;
end for;
```

Laufzeitanalyse:

Zahl der **Additionen**:

$$\sum_{j=1}^{n}\sum_{i=1}^{j} j - i = \sum_{j=1}^{n}\sum_{k=0}^{j-1} k$$

$$= \sum_{\ell=0}^{n-1}\sum_{k=0}^{\ell} k = \sum_{\ell=0}^{n-1}\sum_{k=1}^{\ell} k = \sum_{\ell=0}^{n-1} \frac{1}{2}\ell(\ell+1) = \sum_{\ell=1}^{n-1} \frac{1}{2}\ell(\ell+1)$$

$$= \frac{1}{2}\left(\sum_{\ell=1}^{n-1}\ell^2 + \sum_{\ell=1}^{n-1}\ell\right) = \frac{1}{2}\left(\frac{1}{6}(n-1)n(2(n-1)+1) + \frac{1}{2}n(n-1)\right)$$

$$= \frac{1}{6}n^3 - \frac{1}{6}n$$

Zahl der **Vergleiche**: Zahl der Paare (i, j).

$$\binom{n}{2} = \frac{1}{2}n(n+1) = \frac{1}{2}n^2 + \frac{1}{2}n$$

Also **Laufzeit** $O(n^3)$.

Erste Verbesserung:

Versuch 2: Der normale Algorithmus.

Der naive Algorithmus berechnet z.B. $n-1$-mal $a_1 + a_2$.
(für $f(1,2), f(1,3), \ldots, f(1,n)$.)

Also: mit $f(i,j+1) = f(i,j) + a_{j+1}$ berechne die
$f(i,j)$ für $i = 1, 2, \ldots n$: je $n-i$ Additionen. Insgesamt

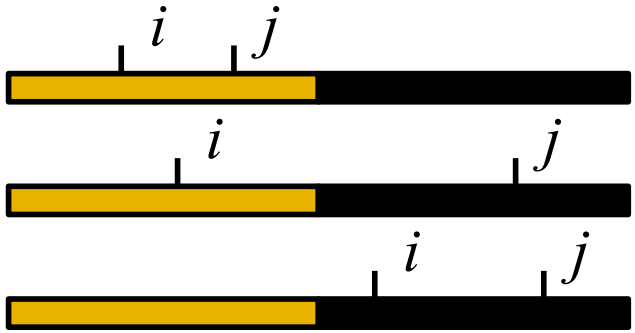
$$\sum_{i=1}^{n} n - i = \sum_{k=1}^{n-1} k = \frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n$$

Additionen und (wie oben) $\frac{1}{2}n^2 + \frac{1}{2}n$ Vergleiche.

Laufzeit n^2, also $O(n^2)$.

Versuch 3: Divide and Conquer.

Sei $n = 2^k$. *Divide*: Löse Problem für



1. $1 \leq i \leq j \leq \frac{n}{2}$
2. $1 \leq i \leq \frac{n}{2} < j \leq n$
3. $\frac{n}{2} \leq i \leq j \leq n$

1. und 3. sind je das ursprüngliche Problem für $\frac{n}{2}$ statt n.

Wenn wir 2. lösen genügen 2 Vergleiche für den Gesamtsieger.

Laufe von $a_{n/2}$ nach links und rechts. Maximiere.

Also: $g(i) = a_i + \cdots + a_{n/2}$, $h(j) = a_{n/2+1} + \cdots + a_j$.

Dann ist $f(i,j) = g(i) + h(j)$. Wir können g und h einzeln maximieren.

Das sind je $\frac{n}{2} - 1$ Vergleiche und $\frac{n}{2} - 1$ Additionen.

Also reichen für 2. bereits $n - 2 + 1$ Additionen und $n - 2$ Vergleiche.

Also $2n - 3$ Operationen.

Was ist mit 1 und 3: Das sind Probleme halber Größe:
Statt $n = 2^k$ nur $\frac{n}{2} = 2^{k-1}$. Rekursion:

$$t(1) = 0,\ t(2^k) = 2t(2^{k-1}) + \underbrace{2 \cdot 2^k - 3}_{2n-3} + \underbrace{2}_{\text{2 Vergleiche}} = 2t(2^{k-1}) + 2 \cdot 2^k - 1$$

Lösung (siehe Übung) $t(2^k) = (2k-1)2^k + 1$ bzw $(n = 2^k)$

$$t(n) = (2 \log n - 1)n + 1$$

Also **Laufzeit** $O(n \log n)$. Aber es geht noch besser:

Versuch 4: Der schlaue Algorithmus

Ansatz: Wir wollen jedes a_k nur einmal lesen: $a_1, a_2, \ldots$

Was müssen wir uns merken? Sicherlich die Lösung `max` für $a_1, \ldots, a_k$.

Konkurrenten für `max` sind (a) $f(i,j)$ mit $k < i$, und (b) $f(i,j)$ mit $i \leq k < j$.

Über (a) können wir noch nichts wissen. Bei (b) wissen wir aus Versuch 3, dass das maximale $m(k)$ der $g(i) = a_i + \cdots + a_k$ entscheidend ist.

Also, beim Lesen von a_{k+1}:

$$m(k+1) = \max\{m(k) + a_{k+1},\ a_{k+1}\} \quad \text{(ein Vergleich)}$$

und

$$\texttt{max}_{neu} = \max\{\texttt{max}_{alt},\ m(k+1)\} \quad \text{(noch ein Vergleich)}$$

Ist a_n abgearbeitet enthält `max` die Lösung.

In jedem Schritt k nur eine Addition und 2 Vergleiche ($2 \leq k \leq n$).

Laufzeit $3(n-1) = 3n - 3$, also $O(n)$.

n	naiv	normal	D&C	schlau
4	19	15	9	9
16	815	255	97	45
64	45759	4095	641	189
1024	179 481 599	1 048 575	18 433	3096

Siehe auch wikipedia: maximum subarray problem, Algorithmus von Kadane.

Analoge Algorithmen

Sortieren einer Liste $a_1, \ldots, a_n$ (Wertebereich unbekannt, keine Parallelisierung,...) kostet $O(n \log n)$.

Auf einem Digitalrechner! Was ist damit: $\texttt{SpaghComp}^{\text{TM}}$

- Schneide n Spaghetti auf Längen $a_1, \ldots, a_n$ zu
- Nimm sie in die Hand und stell sie senkrecht auf einen Tisch
- Lockere die Hand
- Nimm Spagehtti von oben nach unten weg, notiere die Werte

Laufzeit: (mit Vor- und Nachbereitung, also Pre- und Postprocessing)

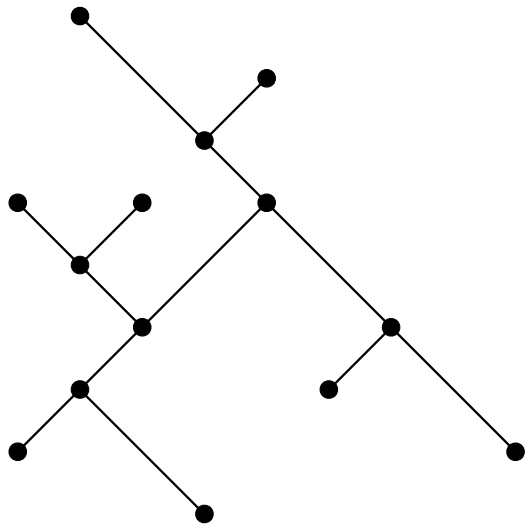
- Zuschneiden: 1 min pro Spaghetti
- Sortieren: 1 sec
- Ablesen: 10 sec pro Wert

Linear! Ab irgendeinem (astronomisch hohen) n schlägt `SpaghComp`$^{\text{TM}}$ Quicksort!

Längste Wege in einem Baum

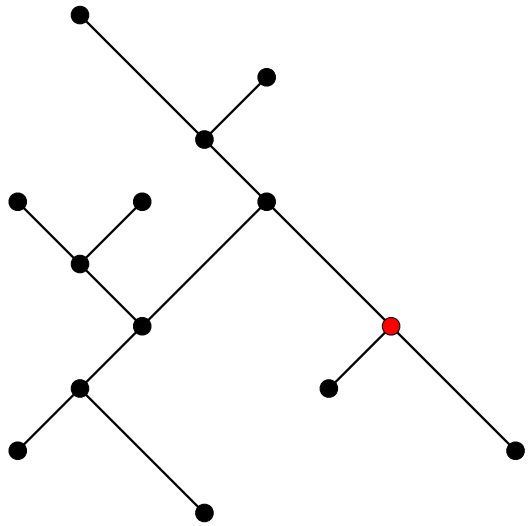
Gegeben ein Baum, dessen Kanten Längen (Kosten,...) haben.

Für welches Knotenpaar i, j wird die Gesamtlänge der Kanten dazwischen maximal? (Gesamtlänge ist eindeutig wg. Baum)

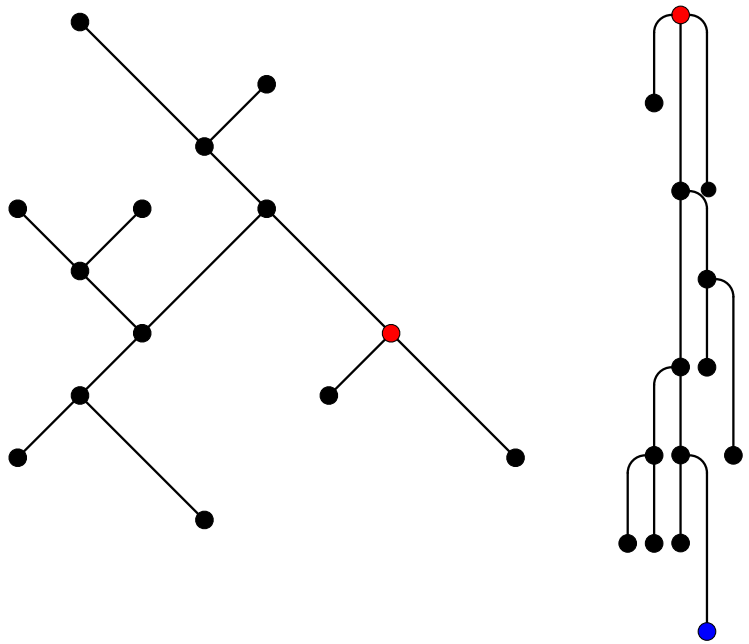


Analoger Algorithmus:

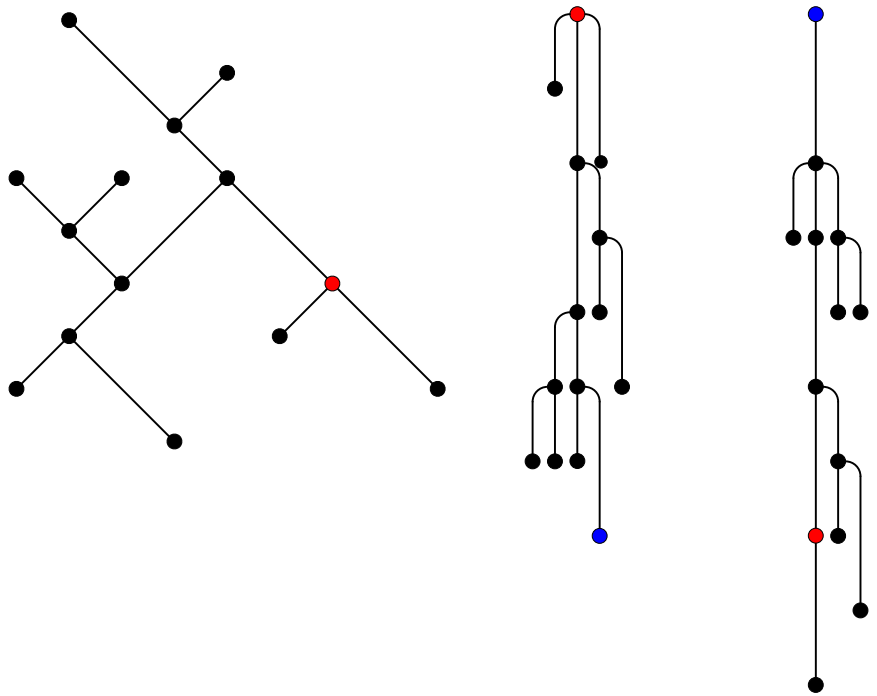
- Realisiere den Baum mit Schnüren entsprechender Längen
- Hebe einen Knoten an
- Erfasse den untersten Knoten, hebe diesen an (lass den ersten Knoten los)
- Lösung: oberster Knoten — unterster Knoten



- Realisiere den Baum mit Schnüren entsprechender Längen
- Hebe einen Knoten an
- Erfasse den untersten Knoten, hebe diesen an (lass den ersten Knoten los)
- Lösung: oberster Knoten — unterster Knoten



- Realisiere den Baum mit Schnüren entsprechender Längen
- Hebe einen Knoten an
- Erfasse den untersten Knoten, hebe diesen an (lass den ersten Knoten los)
- Lösung: oberster Knoten — unterster Knoten



Laufzeit: auch linear! In der Zahl der Kanten, mit Pre-Processing. (Ohne Pre-Processing: Laufzeit konstant)

Kürzeste Wege in einem Graphen

Gegeben ein zusammenhängender Graph, dessen Kanten Längen haben.

Für ein gegebenes Knotenpaar i, j, welcher Weg von i nach j hat die kürzeste Gesamtlänge?

- Realisiere den Graph mit Schnüren entsprechender Längen
- Fasse mit der einen Hand Knoten i, mit der anderen Knoten j
- Strammziehen
- Lösung: der Weg der nur aus straff gespannten Schnüren besteht

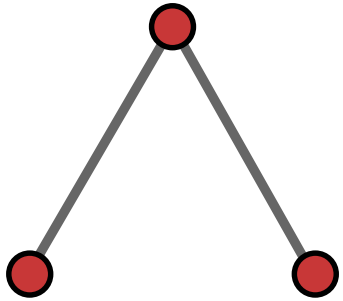
Lineare Laufzeit! (bzgl. n=Zahl der Kanten) Digitalrechner brauchen $O(n^2)$.

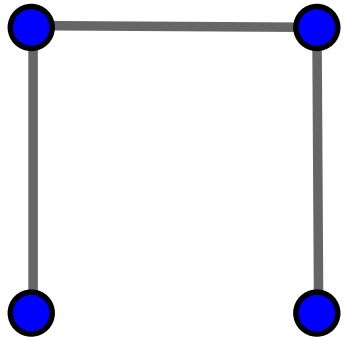
Geht so auch "längster Weg" (ohne Schleifen)?
Das ist NP-vollständig!

Steinerbäume

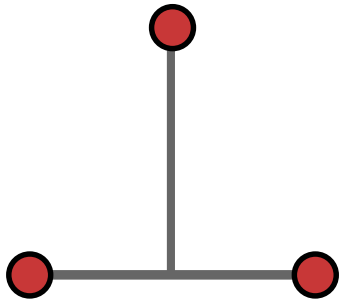
Gegeben n Punkte in der Ebene. Vervollständige das zu einem Graphen mit der niedrigsten Kanten-Gesamtlänge.

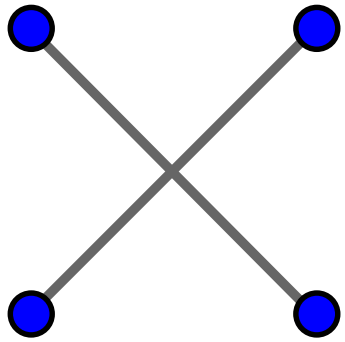
"Länge" heißt hier wirkliche (euklidische) Länge.
Es dürfen Knoten hinzugefügt werden.

1. Versuch: 

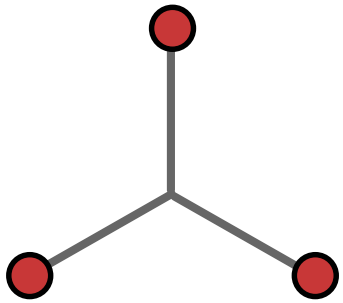


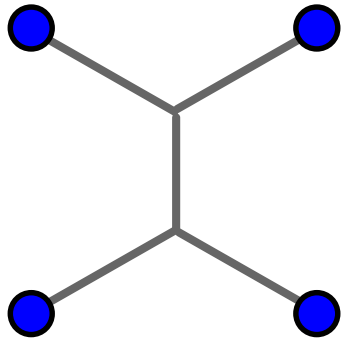
Gesamtlänge 2 bzw 3. Nicht optimal.

2. Versuch: 



Gesamtlänge $1 + \frac{\sqrt{3}}{2} = 1,866...$ bzw $2\sqrt{2} = 2,828....$ Nicht optimal.

3. Versuch: 



Gesamtlänge $\sqrt{3} = 1,732...$ bzw $1 + \sqrt{3} = 2,732....$ Optimal.

Dieses Problem ist auch NP-hart. Aber man weiß über die optimale Lösung:

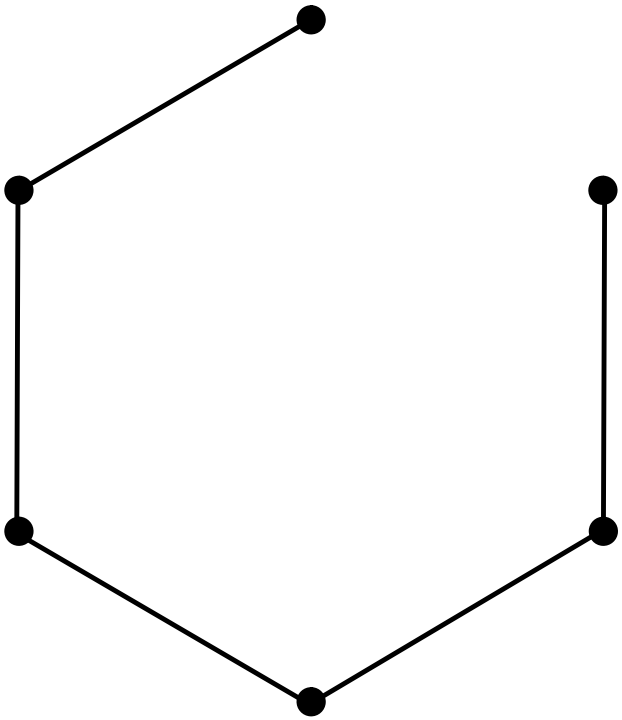
- Die Lösung ist immer ein Baum.
- Alle Knoten haben Grad 1, 2 oder 3.
- Hinzugefügte Knoten haben alle Grad 3.
- Falls Grad 3, dann alle Winkel 120°.

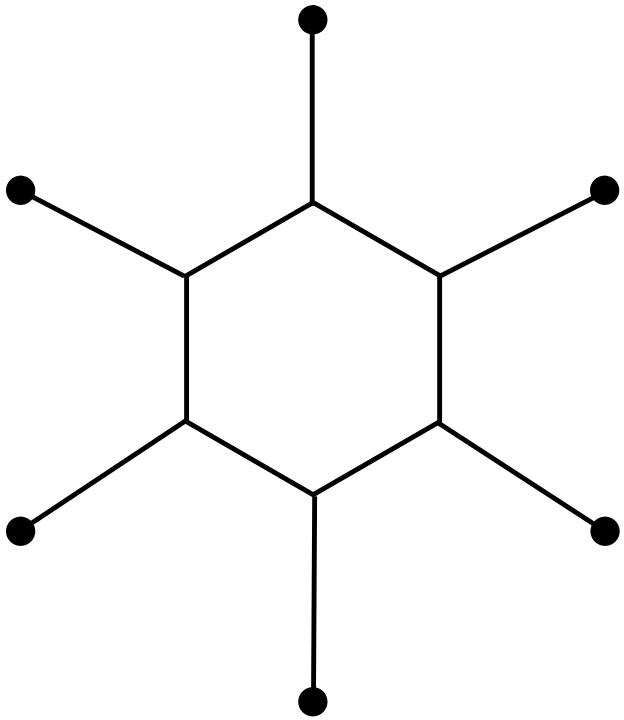
Aber analog geht es effizient (und korrekt?)

Realisiere die n Punkte als Stifte (Nägel) zwischen zwei parallelen Scheiben (Plexiglas). Tauche dies in Seifenlauge. Es bildet sich ein Seifenfilm, der den kürzesten Weg realisiert...

...aber nur bezüglich dieser Topologie (Knoten-Kanten-Struktur des Baumes).

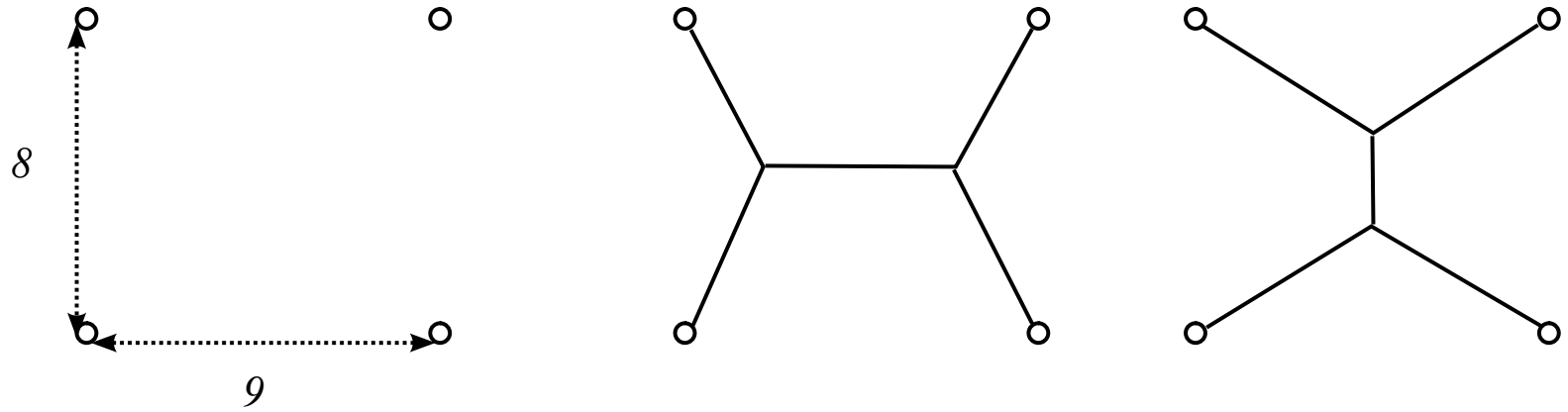
Bsp 1:





Links: optimal. Rechts: Weder optimal, noch ein Baum (kann aber als Seifenfilmergebnis passieren)

Bsp 2:



Ein Rechteck. Mitte: optimale Lösung. Rechts: nicht optimal, kann aber als Seifenfilmergebnis passieren.

Weitere Beispiele:

Standort eines Hochofens: Braucht a Tonnen Erz pro Jahr, b Tonnen Kohle und c Tonnen Kalk. Transport kostet A Euro pro Tonne und Kilometer, bzw B, bzw. C. Die Rohstoffe werden von drei verschiedenen Orten geliefert. Was ist der günstigste Standort für den Hochofen?

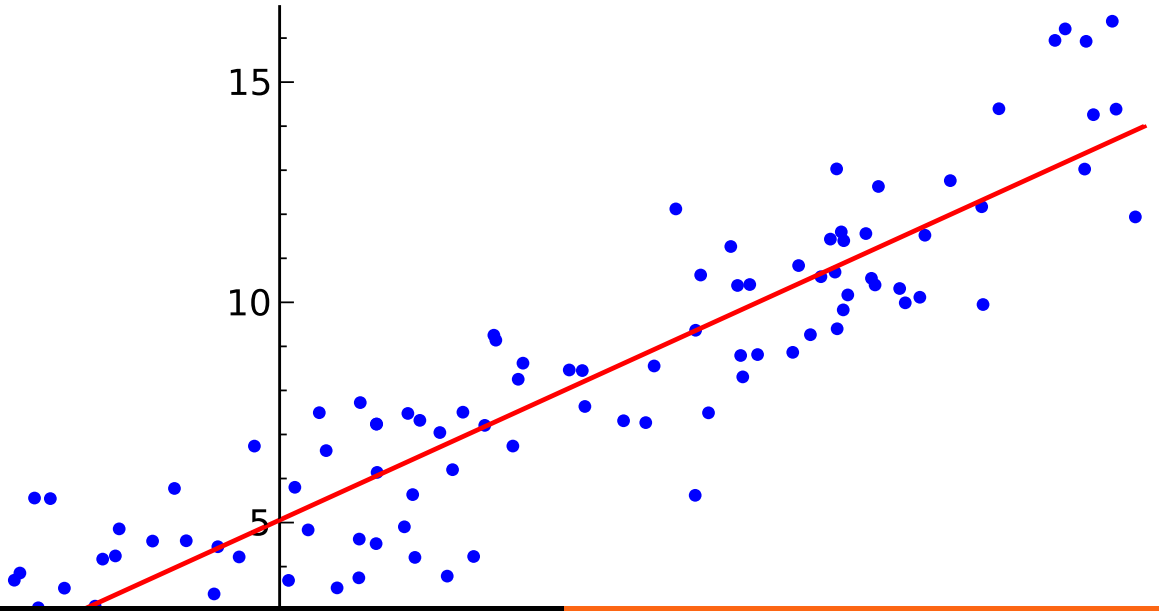
Landkarte, Löcher bei den Herkunftsorten von Erz, Kohle und Kalk. Drei Schnüre mit Gewichten A, B und C von unten durch die Löcher führen, oben an einen Ring binden, loslassen. Ringposition auf der Karte liefert Standort.

Vierte Potenzen: Ein n Meter weit überhängender Balken hängt um $c \cdot \sqrt[4]{n}$ durch.

Ausgleichsgerade:

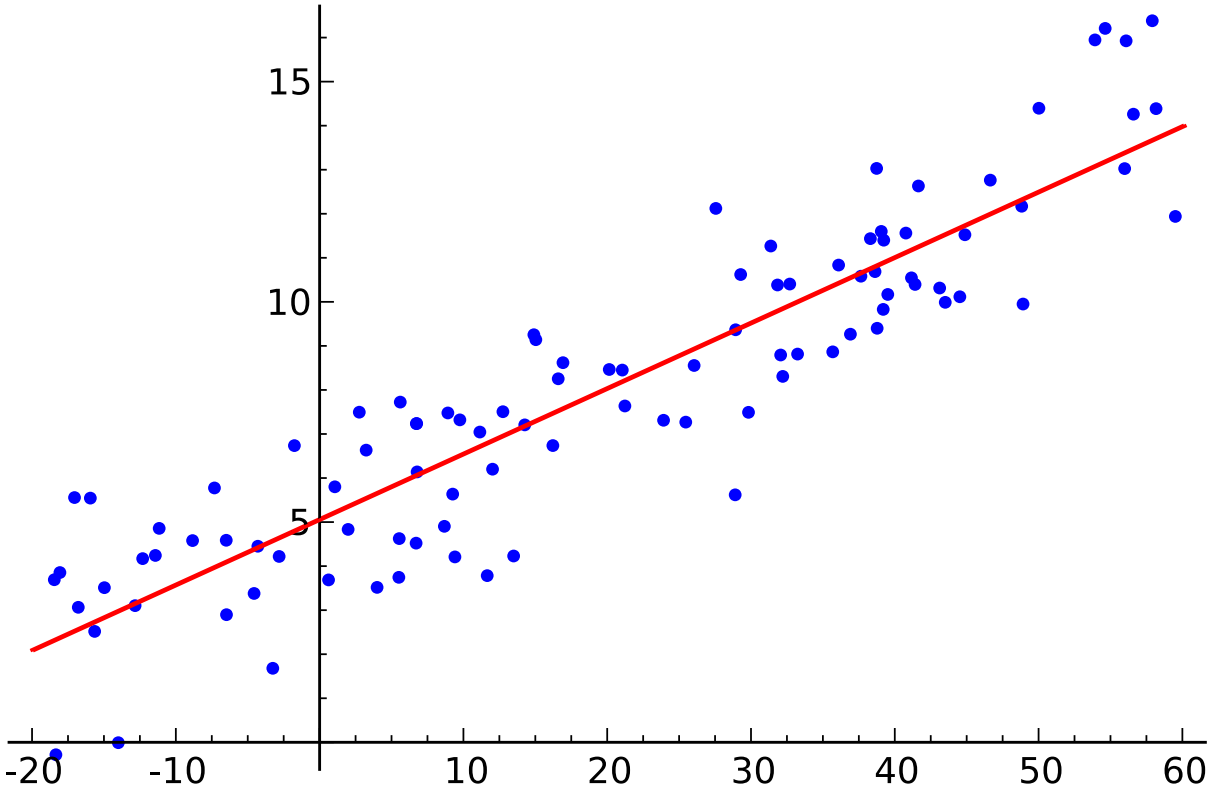
Bei Messung/statistischer Auswertung fallen Daten $(x_1, y_1), \ldots, (x_n, y_n)$ an, die ungenau sein können. Wenn ein zugrundeliegendes Prinzip gesucht wird: **Methode der kleinsten Quadrate**.

Finde Gerade (Kurve...) so dass die Summe der *quadrierten* Abstände von y_i und $g(x_i)$ minimal wird.



Ausgleichsgerade:

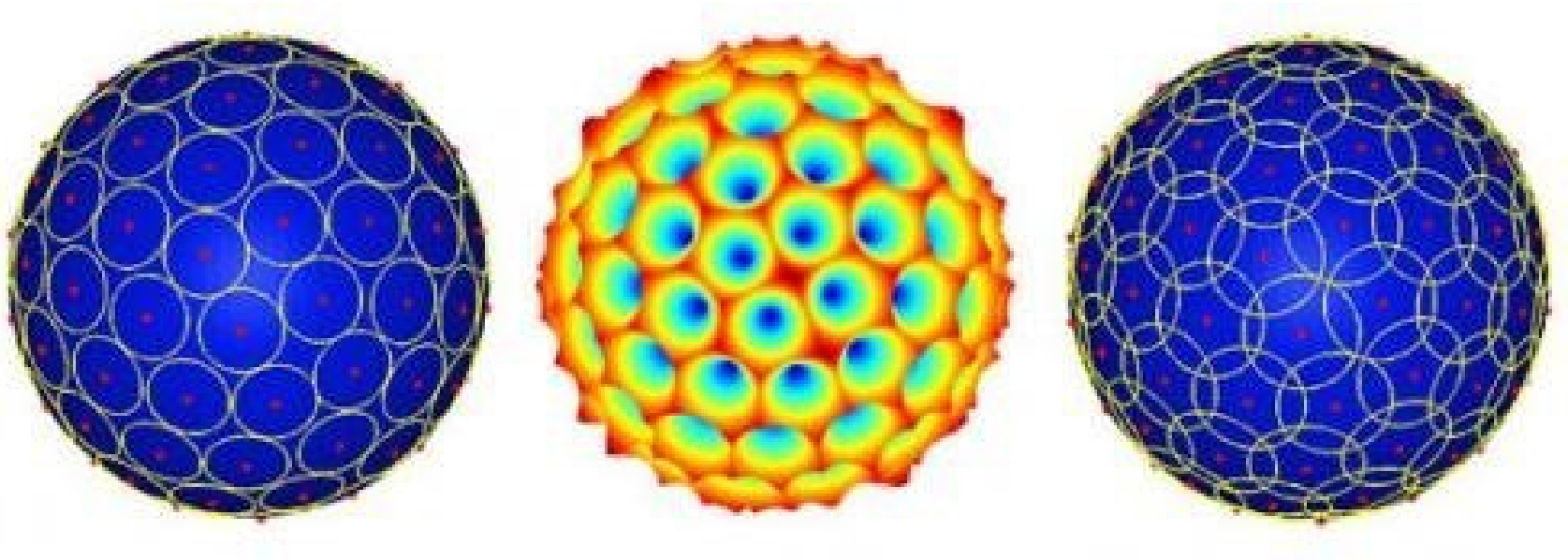
Geht digital auch effizient. Geht analog auch so:



Realisiere Punkte (x_i, y_i) mit Nägeln, Gerade g mit beweglichem (aber staren) Stab. Verbinde jedes (x_i, y_i) mittels Gummibändermit dem Stab. Loslassen. Stab pendelt sich auf Ausgleichsgerade ein.

Das waren Spielzeugbeispiele. Das Prinzip sollte man aber nicht unterschätzen.

- n **Punkte mit maximalem Abstand auf einer Kugel**: realisieren als Pluspole, stoßen sich ab, Physik simulieren liefert gute Antworten.
- **Knotplot:** Stellt "Normalform" von Knoten her durch realisieren als biegsame Bänder, die möglichst gerade sein wollen. Physik simulieren minimiert die Gesamtkrümmung.



[Knotplot Demo]

Zu den Grenzen von Analogcomputern:

- A. Vergis, K. Steiglitz and B. Dickinson: The complexity of analog computation, *Mathematics and Computers in Simulation* 28, (1986) 91-113
- László Lovász: Information and Complexity - how to measure them, in: *The Emergence of Complexity in Mathematics, Physics, Chemistry and Biology* (ed. B. Pullman), Princeton University Press (1996)
- Frank Lee: *Physical Manifestation of NP-Completeness in Analog Computing Devices*, PhD thesis MIT (1999)