2. Hilfsmittel und Grundbegriffe (z.T. Wiederholung!)

2.1. *Asymptotik* als Werkzeug zur Analyse von Algorithmen

- Wie hängt der Zeit- und Speicherplatzbedarf eines Algorithmus von der Größe des zu lösenden Problems ab?
- Assoziiere mit jedem Problem eine positive ganze Zahl n, die ein Maß für die Größe der zu verarbeitenden Eingabedaten ist; z.B. bezeichnet n beim Sortierproblem die Anzahl der zu sortierenden Elemente
- worst case: Verhalten eines Algorithmus im schlimmsten Fall
- average: Verhalten eines Algorithmus im Durchschnitt, d.h. im Mittel über alle möglichen Konfigurationen von Eingabedaten
- Angabe der von einem Algorithmus aufgewendeten Arbeit als Funktion in Abhängigkeit von der Größe n der Eingabe
- Betrachte Wachstumsverhalten dieser Funktion: asymptotische Zeit- und Speicherplatzkomplexität
- $\mathbb{N}_0$ Menge der nicht-negativen ganzen Zahlen
- $\mathbb{R}_0$ Menge der nicht-negativen reellen Zahlen
- X stehe entweder für $\mathbb{N}_0$ oder $\mathbb{R}_0$. Sei g: $X \to X$.

Zweck für Algorithmen

- Asymptotische Laufzeit im schlechtesten Fall
- Asymptotischer Platzbedarf im schlechtesten Fall
- Interessant für große Eingaben

Zweck für Probleme

- Obere Schranke durch Angabe eines effektiven Algorithmus

Bachmann-Landau'sche "O"-Notation

- Definition von O(), "groß Oh":
  $O(g) := \{f: X \to X \mid \exists\, c > 0, \exists\, x_0 \in X, \forall\, x \geq x_0: f(x) \leq c \cdot g(x)\}$
  Wir sagen, daß $f \in O(g)$, oder daß f höchstens so schnell wächst wie g(x) für $x \to \infty$.
- Definition von $\Omega()$, "groß Omega":
  $\Omega(g) := \{f: X \to X \mid \exists\, c > 0, \exists\, x_0 \in X, \forall\, x \geq x_0: f(x) \geq c \cdot g(x)\}$
  Wir sagen, daß $f \in \Omega(g)$, oder daß f mindestens so schnell wächst wie g(x) für $x \to \infty$.
- Definition von $\Theta()$, "Theta":
  $\Theta(g) := O(g) \cap \Omega(g)$
  Wir sagen, daß $f \in \Theta(g)$, oder daß f das gleiche Wachstumsverhalten hat wie g(x) für $x \to \infty$.
- Definition von o(), "klein Oh":
  $o(g) := \{f: X \to X \mid \lim_{x \to \infty} \frac{f(x)}{g(x)} = 0\}$
  Wir sagen, daß $f \in o(g)$, oder daß f langsamer wächst als g(x) für $x \to \infty$.
- Notation: In der Literatur wird häufig = anstelle von $\in$ benutzt, wie in $x = O(x^2)$. Dieses = besitzt keine der Standardeigenschaften der Gleichheitsrelation - es ist weder kommutativ noch transitiv $\Rightarrow$ $O(x^2) = x$ nicht zulässig, und aus $x = O(x^2)$ und $x^2 = O(x^2)$ folgt nicht $x = x^2$.
- O() ist keine Funktion, sondern eine Menge von Funktionen!

Bemerkung: Eine andere Schreibweise für $f(n) = \Theta(g(n))$ ist $f(n) \asymp g(n)$. $\asymp$ stellt offensichtlich eine Äquivalenzrelation dar. Die Äquivalenzklasse von $f(n)$ heißt **Wachstumsrate** von $f(n)$.

Es gelten die folgenden Rechenregeln.

Satz 1.2 *(a)* $f_1 = O(g)$ *und* $f_2 = O(g)$, *dann gilt* $f_1 + f_2 = O(g)$.

(b) $f_1 = O(g_1)$ *und* $f_2 = O(g_2)$, *dann gilt* $f_1 \cdot f_2 = O(g_1 \cdot g_2)$.

(c) $g_1 \leq g_2$ *und* $f = O(g_1)$, *dann gilt* $f = O(g_2)$.

(d) $f = O(g)$ *und* $g = O(h)$, *dann gilt* $f = O(h)$.

(e) $f_1 = O(g_1)$ *und* $f_2 = O(g_2)$, *dann gilt* $f_1 + f_2 = O(\max\{g_1, g_2\})$.

(aus Schnell 2001)

Von Interesse sind 'effiziente' Verfahren in dem folgenden Sinne.

Definition 1.3 *Ein Algorithmus heißt* **polynomial**, *wenn es ein Polynom in der Länge des Input gibt, das den Aufwand des Algorithmus abschätzt. Die übrigen Algorithmen heißen* **exponentiell**.

Ermittlung von oberen Schranken:

Ermittlungsverfahren

- Direkt ersichtlich
- Iterative Algorithmen: Summation der Laufzeiten
- Rekursive Algorithmen: Auflösen von Rekurrenzen
- Aber: Korrekte Rekurrenz aufstellen

Drei Verfahren zum Auflösen einer Rekurrenz

- Substitutions-Methode
- Iterations-Methode
- Master-Methode

Bemerkung: Im Deutschen wird statt „Rekurrenz" teilweise auch der Begriff „Rekursionsgleichung" verwendet.

(aus Breimann & Vahrenhold 2001)

Substitutions-Methode für obere Schranken:

Ablauf des Verfahrens:

- Obere Schranke raten oder über den Daumen peilen
- Per Induktion zeigen, daß Schranke korrekt

Ein Beispiel

- $T(n) = 2 * T(\lfloor \frac{n}{2} \rfloor) + n$ mit $T(0) = 0$
- Es gelten $T(1) = 1, T(2) = 4, T(3) = 5, T(4) = 12, \ldots$
- Also auch $T(n) \leq 2 * n \log n$ mit $n \in \{2, 3, 4\}$
- Induktionsschritt:
  $$\begin{aligned} T(n) = 2 * T(\left\lfloor \tfrac{n}{2} \right\rfloor) + n &\leq 2 * (2 * \left\lfloor \tfrac{n}{2} \right\rfloor \log \left\lfloor \tfrac{n}{2} \right\rfloor) + n \\ &\leq 2 * n * \log \left\lfloor \tfrac{n}{2} \right\rfloor + n \leq 2 * n * \log n - 2 * n * \log 2 + n \\ &\leq 2 * n * \log n \end{aligned}$$
- Obere Schranke gilt mit $c = 2$ und $n_0 = 2$

Iterations-Methode:

Ablauf des Verfahrens:

- Einsetzen, einsetzen, einsetzen, einsetzen, ...
- Geschickt umformen und Lösung sehen

Ein Beispiel

- $$\begin{aligned} T(n) = 3 * T(\left\lfloor \tfrac{n}{4} \right\rfloor) + n &= n + 3 * (\left\lfloor \tfrac{n}{4} \right\rfloor + 3 * T(\left\lfloor \tfrac{n}{16} \right\rfloor)) = \ldots \\ &\leq n + \tfrac{3n}{4} + \tfrac{9n}{16} + \tfrac{27n}{64} + \ldots + \hat{c} * 3^{\log_4 n} \\ &\leq n * \sum_{i=0}^{\infty} \left(\tfrac{3}{4}\right)^i + \hat{c} * n^{\log_4 3} \leq n * \tfrac{1}{1-\frac{3}{4}} + \hat{c} * n \leq (4 + \hat{c}) * n \end{aligned}$$
- Dabei $\hat{c}$ abhängig von $T(1)$
- Insgesamt also $T(n) \in \mathcal{O}(n)$ mit $c = 4 + \hat{c}$ und $n_0 = 1$
- Insbesondere: $\mathcal{O}(n * \log n)$ keine scharfe obere Schranke

"Master-Methode":

Master-Theorem: Seien $a \geq 1, b > 1$ konstant und $f(n)$ eine Funktion sowie $T(n)$ eine Rekurrenz mit

$$T(n) = a * T\left(\frac{n}{b}\right) + f(n)$$

wobei $\frac{n}{b}$ als $\left\lfloor \frac{n}{b} \right\rfloor$ oder $\left\lceil \frac{n}{b} \right\rceil$ interpretiert wird. Dann gelten:

1. $\exists\ \epsilon > 0 : f(n) \in \mathcal{O}\left(n^{\log_b a - \epsilon}\right) \Rightarrow T(n) \in \mathcal{O}\left(n^{\log_b a}\right)$
2. $f(n) \in \Theta(n^{\log_b a}) \Rightarrow T(n) \in \Theta(n^{\log_b a} * \log n)$
3. $\exists\ \epsilon > 0, \exists\ c < 1, \exists\ n_0 \in \mathbb{N} : \forall\ n \geq n_0 :$
   $f(n) \in \Omega(n^{\log_b a + \epsilon}) \wedge a * f\left(\frac{n}{b}\right) \leq c * f(n) \Rightarrow T(n) \in \Theta(f(n))$

Ablauf des Verfahrens

- Schreibe $T(n)$ entsprechend der Notation im Master-Theorem
- Wende das Master-Theorem an, falls möglich

Weitere Bemerkungen zu asymptotischen Schranken:

Komplexitätsanalyse von Algorithmen

- Oft auch untere Schranken Ω und genaue Schranken Θ gesucht
- Vorgehen analog zu oberen Schranken

Komplexitätsanalyse von Problemen

- Obere Schranke durch Angabe eines effektiven Algorithmus
- Untere Schranke durch allgemeine Betrachtung des Problems
- Nicht: Untere Schranke eines beliebigen Algorithmus

Allgemein

- Asymptotische Analyse wesentlich in der Alg. Geometrie
- Erfordert oft Kenntnis der Mathematik
- Oft scharfe Schranken gesucht
- Laufzeitverhalten und Speicherplatzbedarf interessant

Beispiel *Sortieren* (vgl. Informatik I)

Berechnungsmodell beim Sortieren: nur binäre Vergleiche erlaubt

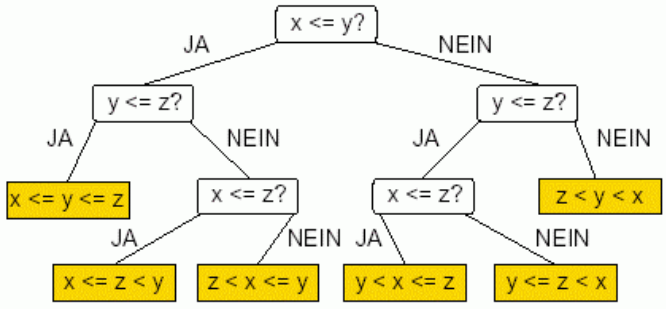
Untere Schranke:

Satz: Jeder Sortieralgorithmus, der zum Sortieren von n Elementen nur binäre Vergleiche benutzt, benötigt im durchschnittlichen und im ungünstigsten Fall $\Omega(n \log n)$ Operationen.

Beweis:

Idee: Sortieren $\approx$ Suche nach Permutationen

- Darstellung des Algorithmus als binärer Baum.
- Interne Knoten $\approx$ binäre Vergleiche.
- Blätter $\approx$ Permutationen.



Beobachtung: (Durchschnittliche) Tiefe des Baumes $\rightarrow$ Laufzeit.

Lemma: Die durchschnittliche Tiefe eines Baumes mit k Blättern ist mindestens $\log_2 k$. (Induktion)

Beweis des Satzes:

- Anzahl Blätter $\geq$ Anzahl Permutationen $= n!$.
- Lemma $\Rightarrow$ durchschnittliche Tiefe $\geq \log_2 n!$.
- Stirlingsche Formel: $n! \approx \sqrt{2 * \pi * n} * \left(\frac{n}{e}\right)^n$ (vgl. Forster I)
- $\log_2 n! > \log_2 \left(\frac{n}{e}\right)^n = n * \log_2 n - n * \log_2 e \in \Omega(n \log n)$
- Pfad von der Wurzel zu einem Blatt entspricht den Vergleichen.
- Durchschnittliche Tiefe ergibt durchschnittliche Laufzeit und Laufzeit im ungünstigsten Fall.

Mergesort: ein "divide-and-conquer"-Verfahren

Algorithmus: Seien $k \in \mathbb{N}, k \geq 2, n = |\text{Array}|$.

- Teile zu sortierendes Array in k Teile der Größe $\leq \left\lceil \frac{n}{k} \right\rceil$.
- Sortiere jedes Teil-Array rekursiv.
- Füge sortierte Teil-Arrays zusammen.

Laufzeit:

- Aufteilen: $\mathcal{O}(|\text{Array}|)$.
- Rekursives Sortieren: ???
- Zusammenfügen: $\mathcal{O}(|\text{Array}|)$

Rekurrenz: Sei $c \in \mathbb{R}, c > 0$.

$$T(n) = k * T\left(\left\lceil \frac{n}{k} \right\rceil\right) + c * n \in \mathcal{O}(n * \log_k n)$$

Auflösen der Rekursion:

Bemerkung: Rekurrenz im allgemeinen nur

$$T(n) \leq k * T\left(\left\lceil \frac{n}{k} \right\rceil\right) + c * n$$

1. Möglichkeit: Master-Methode

$a = k, b = k \Rightarrow \log_a b = 1 \Rightarrow f(n) \in \Theta(n) \Rightarrow T(n) \in \Theta(n * \log n)$

2. Möglichkeit: Iterations-Methode

$$T(n) \leq k * T\left(\left\lceil \frac{n}{k} \right\rceil\right) + c * n \leq c * n + k * \left(c * \left\lceil \frac{n}{k} \right\rceil + k * T\left(\left\lceil \frac{n}{k^2} \right\rceil\right)\right) \leq \ldots$$

$$\leq \sum_{i=1}^{\lceil \log_k n \rceil} k^i * c * \left(\frac{n}{k^i} + 1\right) \leq (\lceil \log_k n \rceil - 1) * 2 * c * n + c * n + k^{\lceil \log_k n \rceil} * c$$

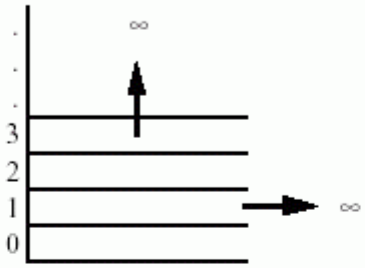
$$\leq 2 * c * n * \log_k n + c * n + c * k * n \leq \hat{c} * n * \log_k n \quad \text{mit} \quad n > k$$

2.2. Unser *Berechnungsmodell* für die Algorithmische Geometrie:

keine Turingmaschine oder Integer-Maschine,
sondern die "Real RAM"
(RAM = random access machine)

- Berechnungsmodell spezifiziert die ausführbaren primitiven Operationen und ihre Kosten.
- Primitive Operationen sind die Operationen, für die wir fixe Kosten veranschlagen, auch wenn diese Kosten von Operation zu Operation variieren können.
- Entscheidung für ein Modell: Kompromiß zwischen Realität und mathematischer Handhabbarkeit, indem ein Schema gewählt wird, das auf der einen Seite die wesentlichen Charakterzüge der benutzten Rechner widerspiegelt, auf der anderen Seite jedoch noch genügend einfach ist, um eine sorgfältige Analyse zu ermöglichen.
- Welche Art von Modell ist geeignet für geometrische Anwendungen?
- Fundamentale Objekte: Punkte im $\mathbb{R}^d$
- Die Wahl des Koordinatensystems soll die Laufzeit eines geometrischen Algorithmus nicht signifikant beeinflussen.
- Eine Menge von n Punkten kann bzgl. eines gewählten kartesischen Koordinatensystems in Zeit proportional zu n repräsentiert werden.

- Random-Access-Machine (RAM):
  - RAM verfügt über einen Speicher unbegrenzter Kapazität, d.h. über eine abzählbar unendliche Menge von Speicherzellen, adressiert durch 0, 1, 2, … ; jede Speicherzelle kann eine ganze Zahl beliebiger Länge speichern:



  - RAM verfügt über eine Recheneinheit und wird durch ein Programm gesteuert.
  - Auf jede Speicherzelle kann in konstanter Zeit zugegriffen werden (random access).
  - Arithmetische Operationen können in konstanter Zeit durchgeführt werden, unabhängig von der Größe der Operanden.

- *Real RAM* kann in jeder Speicherzelle eine reelle Zahl speichern. Folgende primitive Operationen können auf der *Real RAM* in konstanter Zeit durchgeführt werden:
  1. Die arithmetischen Operationen (+ , – , · , /).
  2. Vergleiche zwischen zwei reellen Zahlen (< , ≤ , = , ≠ , ≥ , >).
  3. Indirekte Speicheradressierung (nur ganzzahlige Adressen).

  Optional für Anwendungen, die sie gebrauchen:
  4. k-te Wurzeln, trigonometrische Funktionen, exp, log, analytische Funktionen.

- Typen von Problemen:
  - Teilmengenselektionsprobleme
    Wähle aus einer Menge von gegeben Objekten eine Teilmenge von Objekten aus, die eine bestimmte Eigenschaft erfüllen.
    Bsp.: Finde für eine gegebene Menge von n Punkten das Paar mit kleinstem Abstand oder die Eckpunkte der konvexen Hülle.
  - Berechnungsprobleme
    Berechne für eine gegebene Menge von Objekten den Wert eines geometrischen Parameters dieser Menge. Die durch das Berechnungsmodell bereitgestellten primitiven Operationen müssen mächtig genug sein, so daß derartige Berechnungen möglich sind.
    Bsp.: Um die Distanz zweier Punkte berechnen zu können, muß man nicht nur irrationale Zahlen darstellen, sondern auch Wurzeln ziehen können.
  - Entscheidungsprobleme
    Jedem Teilmengenselektions- und Berechnungsproblem ist auf natürliche Art und Weise ein Entscheidungsproblem zugeordnet:
    a) Erfordert ein Teilmengenselektionsproblem $\mathcal{T}$ die Bestimmung einer Teilmenge einer gegebenen Menge S, die eine bestimmte Eigenschaft $\mathcal{P}$ erfüllt, so erfordert das assoziierte Entscheidungsproblem D($\mathcal{T}$) eine ja/nein Antwort auf eine Frage des Typs: "Erfüllt S' die Eigenschaft $\mathcal{P}$?", wobei S' $\subseteq$ S.
    b) Erfordert ein Berechnungsproblem $\mathcal{B}$ die Berechnung eines Parameters α, so erfordert das assoziierte Entscheidungsproblem D($\mathcal{B}$) eine ja/nein Antwort auf eine Frage des Typs: "Ist $\alpha \geq \alpha_0$?", wobei α_0 eine Konstante ist.

2.3. Transformierbarkeit von Problemen

- Wie effizient kann ein Algorithmus höchstens sein, der ein gegebenes Problem löst? → Finde eine untere Schranke für die Zeitkomplexität des Problems!
- Beispiel: $\Omega(n \cdot \log n)$ ist untere Schranke des Sortierproblems, falls nur binäre Vergleiche als Elementaroperationen zugelassen
- Durch *Transformation eines Problems* (auch *Reduktion* genannt) kann die Komplexität eines zu betrachtenden Problems zur Komplexität eines bereits bekannten Problems in Beziehung gesetzt werden.
- Seien $\mathcal{A}$ und $\mathcal{B}$ zwei Probleme, die folgendermaßen zueinander in Beziehung stehen:
  1. Die Eingabe für Problem $\mathcal{A}$ wird in eine passende Eingabe für Problem $\mathcal{B}$ konvertiert.
  2. Problem $\mathcal{B}$ wird gelöst.
  3. Die Ausgabe von Problem $\mathcal{B}$ wird in eine korrekte Lösung für Problem $\mathcal{A}$ transformiert.

  Man sagt, daß Problem $\mathcal{A}$ auf Problem $\mathcal{B}$ transformiert wird. Wenn die obigen Transformationsschritte 1 und 3 zusammen in Zeit $O(\tau(n))$ durchgeführt werden können, so heißt $\mathcal{A}$ $\tau(n)$-transformierbar auf $\mathcal{B}$, geschrieben als $\mathcal{A} \rightarrow_{\tau(n)} \mathcal{B}$.
- Transformierbarkeit ist i.a. keine symmetrische Relation. Sind $\mathcal{A}$ und $\mathcal{B}$ gegenseitig transformierbar, so nennen wir $\mathcal{A}$ und $\mathcal{B}$ äquivalent.

- Transformation unterer Schranken:
  Die untere Zeitschranke für Problem $\mathcal{A}$ sei in $\Omega(T_u(n))$, d.h. es wird mindestens $\Omega(T_u(n))$ Zeit benötigt, um $\mathcal{A}$ zu lösen. Ist $\mathcal{A} \rightarrow_{\tau(n)} \mathcal{B}$, wobei $\tau(n) \in o(T_u(n))$, so ist auch die untere Zeitschranke für Problem $\mathcal{B}$ in $\Omega(T_u(n))$.
- Transformation oberer Schranken:
  Problem $\mathcal{B}$ sei in $O(T_o(n))$ Zeit lösbar. Ist $\mathcal{A} \rightarrow_{\tau(n)} \mathcal{B}$, so ist Problem $\mathcal{A}$ in $O(T_o(n) + \tau(n))$ Zeit lösbar.

- $D(\mathcal{A})$ sei das zu einem Problem $\mathcal{A}$ assoziierte Entscheidungsproblem $\Rightarrow D(\mathcal{A}) \rightarrow_{O(n)} \mathcal{A}$, denn:
  1. Ist $\mathcal{A}$ ein Berechnungsproblem, so muß keine Eingabekonvertierung vorgenommen werden, und die Lösung von $\mathcal{A}$ wird in konstanter Zeit O(1) mit dem durch $D(\mathcal{A})$ vorgegebenen Wert verglichen.
  2. Ist $\mathcal{A}$ ein Teilmengenselektionsproblem, wird die Eingabe S von $D(\mathcal{A})$ als Eingabe von $\mathcal{A}$ verwendet, es ist also keine Eingabekonvertierung notwendig. In O(n) Zeit wird überprüft, ob die Lösung von $\mathcal{A}$ mit der durch $D(\mathcal{A})$ vorgegebenen Teilmenge $S' \subseteq S$ übereinstimmt.

  $\Rightarrow$ Bei der Suche nach einer unteren Schranke für ein Problem $\mathcal{A}$ kann man sich auf das assoziierte Entscheidungsproblem $D(\mathcal{A})$ beschränken.

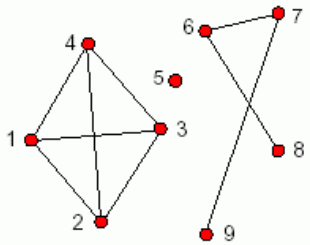
2.4. Grundlagen aus der Graphentheorie

Definition: Ein Graph ist ein Paar $G = (V, E)$ mit $E \subseteq V \times V$.

- V = Menge der Knoten ($V(G)$, vertices).
- E = Menge der Kante ($E(G)$, edges).

Terminologie:

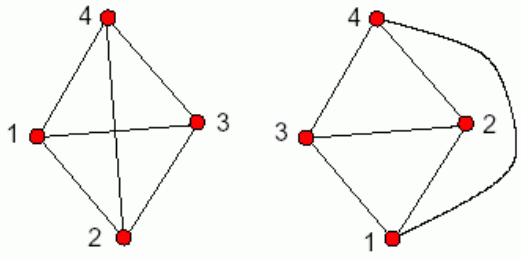
- $v \in V$ inzident mit $e = (v_1, v_2) \in E \iff v = v_1 \ \vee \ v = v_2$.
- $e \in E$ adjazent zu $v \in V \iff v$ inzident mit e.
- $v, w \in V$ benachbart $\iff (v, w) \in E \ \vee \ (w, v) \in E$.
- G heißt ungerichtet, wenn gilt: $(v, w) \in E \iff (w, v) \in E$.

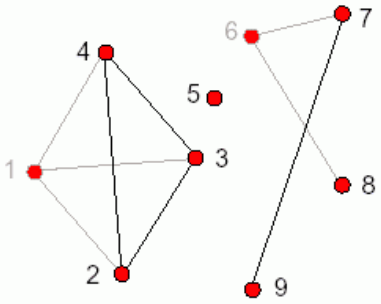


Definition: Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ heißen isomorph ($G_1 \simeq G_2$), wenn es eine Bijektion $\varphi : V_1 \to V_2$ gibt, sodaß gilt:

$$(v, w) \in E_1 \iff (\varphi(v), \varphi(w)) \in E_2$$

Definition: Ein Teilgraph $G_{|V'}$ eines Graphen $G = (V, E)$ entsteht durch Einschränkung von V auf V' und durch Einschränkung von E auf die in $V' \times V'$ enthaltenen Kanten.



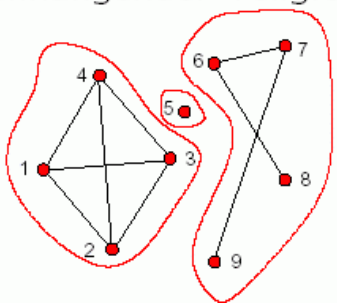


Definition: Ein Pfad ist ein Graph $G = (V, E)$ mit $V = (v_1, \ldots, v_n)$ und $E = ((v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n))$.

Definition: Ein Pfad $G = (V, E)$ mit $V = (v_1, \ldots, v_n)$ und $E = ((v_1, v_2), (v_2, v_3), \ldots, (v_n, v_1))$ heißt Zykel.

Definition: Ein Graph heißt zusammenhängend, wenn je zwei Knoten durch einen Pfad verbunden sind.

Definition: Eine Zusammenhangskomponente eines Graphen G ist ein maximaler zusammenhängender Teilgraph von G.

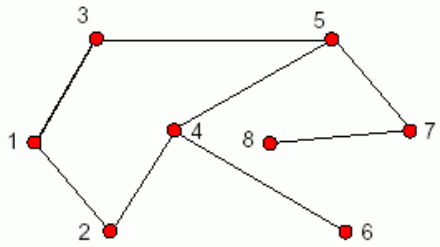


Elementare Algorithmen für Graphen:

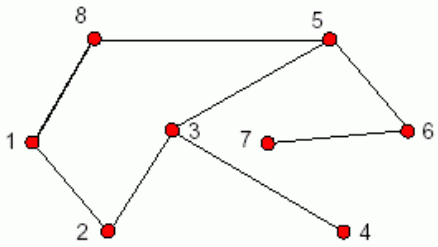
Exploration: Besuche jeden Knoten genau einmal.

- Speichere einen Knoten v in einer Queue.
- Solange Queue nicht leer ist:
  - Besuche ersten Knoten in der Queue.
  - Entferne ersten Knoten aus der Queue.
  - Füge noch nicht besuchte Nachbarn der Queue hinzu.

FIFO: breadth first search.



LIFO: depth first search.



Bestimmung der Zusammenhangskomponenten:

- Exploration von beliebigem unmarkierten Knoten.
- Dabei: Gleiche Markierung aller besuchten Knoten.
- Wiederholung, bis alle Knoten markiert.

Weitere Problemstellungen:

- Mehrfacher Zusammenhang.
- Topologische Sortierung.
- Test auf Zykelfreiheit.
- (Minimaler) Spannbaum.
- Kürzeste Wege.
- ...

2.5. Geometrische Objekte

Punkte:

Punkte des d-dimensionalen Euklidischen Raums bezeichnen wir mit Koordinatentupeln $p = (p_1, ..., p_d)$, $q = (q_1, ..., q_d), \ldots \in \mathbb{R}^d$, bezüglich eines festen kartesischen Koordinatensystems. Mit den Koordinaten des Punktes p wird auch der Vektor identifiziert, der vom Nullpunkt des Koordinatensystems zum Punkt p führt.

Seien $p, q \in \mathbb{R}^d$, $\lambda \in \mathbb{R}$.
Aus der Linearen Algebra sind folgende Operationen bekannt:

- Komponentenweise Addition:
  $p + q = (p_1, ..., p_d) + (q_1, ..., q_d) := (p_1 + q_1, ..., p_d + q_d)$
- Skalarmultiplikation:
  $\lambda p = \lambda(p_1, ..., p_d) := (\lambda p_1, ..., \lambda p_d)$
- Skalarprodukt:
  $< p, q >:= p_1 q_1 + \cdots + p_d q_d = \sum_{i=1}^{d} p_i q_i$
  Zwei Vektoren $p, q \in \mathbb{R}^d$ stehen senkrecht aufeinander, wenn gilt: $< p, q >= 0$.

Strecken:

Strecken sind Punktmengen der Form

$$\overline{pq} := \{(1-\lambda)p + \lambda q \mid \lambda \in [0,1]\},$$

siehe Abbildung 1.1

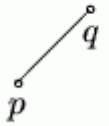


Abbildung 1.1: Strecke zwischen p und q

Geraden:

Geraden (in der Zweipunktform) sind Punktmengen der Gestalt

$$g := \{(1-\lambda)p + \lambda q \mid \lambda \in \mathbb{R}\},$$

siehe Abbildung 1.2

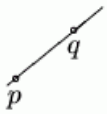


Abbildung 1.2: Gerade durch p und q

Eine Gerade in der Ebene $\mathbb{R}^2$ hat die Eigenschaft, daß sie die Ebene in zwei *Halbebenen* aufteilt.

Halbgeraden (*Strahlen*): analog mit $\lambda \geq 0$.

Hyperebenen:

Sei $v \in \mathbb{R}^d$, $v \neq 0$, $\alpha \in \mathbb{R}$. Die Menge

$$h := \{p \in \mathbb{R}^d \mid < p, v >= \alpha\}$$

heißt *Hyperebene*.

Eine Hyperebene ist damit ein $(d-1)-$dimensionaler affiner Unterraum des $\mathbb{R}^d$. Um die geometrische Bedeutung des Vektors v und der Konstante α zu verstehen, machen wir folgende Betrachtung in der Ebene (Abbildung 1.3).

Bekanntermaßen gilt $< p, v > = \cos\varphi \, \|p\| \, \|v\|$, und der Cosinussatz besagt $\cos\varphi = \|p'\|/\|p\|$. Ein Punkt p liegt also genau dann auf der Hyperebene, wenn gilt $< p, v > = \|p'\| \, \|v\| = \alpha \Leftrightarrow \|p'\| = \alpha/\|v\|$. Also besteht die Hyperbene aus den Punkten p, deren Projektion auf v eine *feste*

Länge hat. Ist v ein Einheitsvektor, ist diese Länge gleich α. v selbst ist ein *Normalvektor* zur Hyperebene. Im $\mathbb{R}^3$ ist eine Hyperbene eine 'übliche' Ebene, senkrecht (orthogonal) zu v.

Die Hyperebene h teilt den $d-$dimensionalen Raum in zwei abgeschlossene *Halbräume*: $h^- := \{p \in \mathbb{R}^d \mid < p, v > \leq \alpha\}$ und $h^+ := \{p \in \mathbb{R}^d \mid < p, v > \geq \alpha\}$. Es gilt: $h^- \cap h^+ = h$ und $h^- \cup h^+ = \mathbb{R}^d$ (Abbildung 1.4).

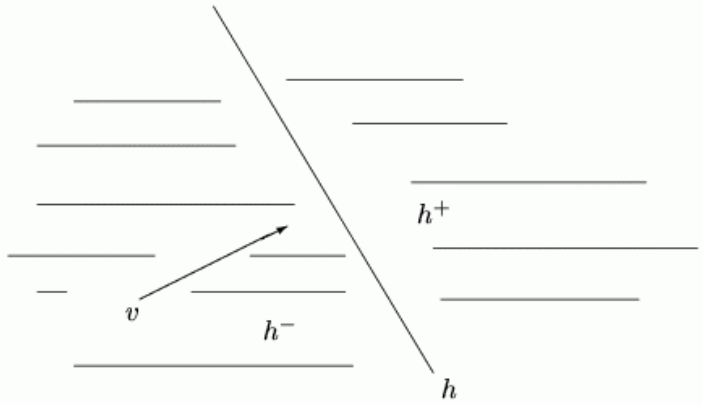


Abbildung 1.4: Die beiden Halbebenen von h im $\mathbb{R}^2$

Polygonzüge und *Polygone*:

Ein *Polygonzug* (der Länge n) ist eine Folge von Strecken $\overline{p_0p_1}$, $\overline{p_1p_2}$, ..., $\overline{p_{n-1}p_n}$. Ein Polygonzug ist geschlossen, wenn $p_0 = p_n$, wir sprechen dann auch von einem *Polygon*. Bei einem *einfachen* Polygonzug sind keine Selbstüberschneidungen erlaubt (Abbildung 1.5).

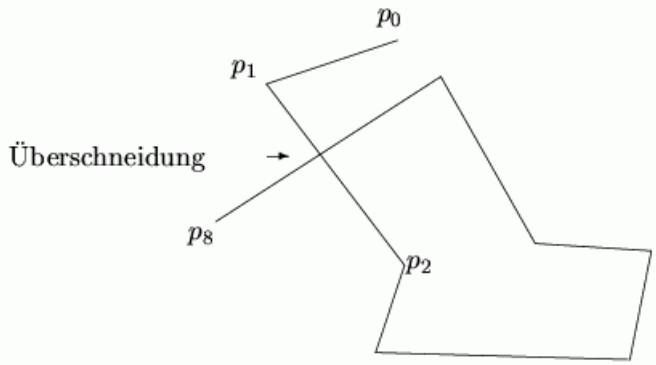


Abbildung 1.5: Selbstschneidender Polygonzug

Es gilt der grundlegende
Jordansche Kurvensatz für ebene, geschlossene Polygone:
Jedes geschlossene, einfache Polygon in der Ebene unterteilt die Ebene in zwei disjunkte *Polygongebiete*, ein inneres und ein äußeres Polygongebiet.

Punkte im Inneren können dadurch charakterisiert werden, dass die Anzahl der Schnittpunkte zwischen den Kanten des Polygons und einem Strahl, der von dem Innenpunkt ausgeht, ungerade ist.

Ein *Polygonnetz* ist eine Menge *M* von endlich vielen geschlossenen, ebenen und einfachen Polygonen mit folgenden Eigenschaften:

- die inneren Polygongebiete von je 2 Polygonen aus *M* haben keine gemeinsamen Punkte
- je 2 Polygone aus *M* haben entweder keinen Punkt oder eine Ecke oder eine ganze Kante gemeinsam
- jede Kante eines Polygons aus *M* gehört zu höchstens 2 Polygonen
- die Menge aller Kanten, die nur zu einem Polygon aus *M* gehören, ist entweder leer (*M* heißt dann "geschlossen") oder bildet selbst ein einziges, geschlossenes, einfaches Polygon.

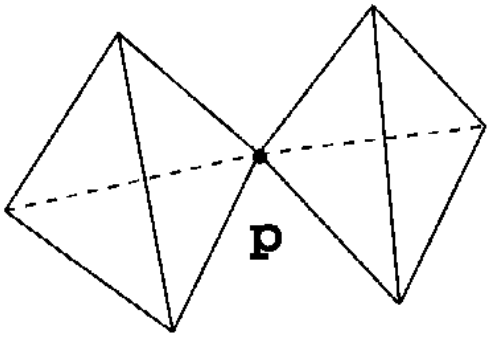
Polygonnetze spielen eine wichtige Rolle in der Computergrafik und in physikalischen Simulationen! (Bsp. Finite-Elemente-Methode)

Def. *Polyeder*:

Ein Polygonnetz *M* heißt Polyeder, wenn gilt:

- *M* ist geschlossen (d.h. jede Kante gehört zu genau 2 Polygonen)
- *M* ist zusammenhängend
- jede Ecke gehört zu einer endlichen, zyklisch geordneten Menge von Polygonen, in der aufeinanderfolgende Polygone jeweils eine zur Ecke gehörende Kante gemeinsam haben

Beispiel, wo die dritte Bedingung verletzt ist:



Die inneren Polygongebiete von *M* heißen auch *Facetten* oder Seitenflächen des Polyeders.
Der Abschluss der Vereinigung aller Facetten heißt die *Oberfläche* (surface) des Polyeders.

Die Polyeder sind die 3D-Verallgemeinerungen der ebenen, einfachen, geschlossenen Polygone.
Insbes. gilt das 3D-Analogon des Jordanschen Kurvensatzes:

Jedes Polyeder teilt den Raum in zwei disjunkte Bereiche, das Innere und das Äußere.

Das Innere kann wieder dadurch charakterisiert werden, dass die Anzahl der Schnitte eines Strahls, der von einem Innenpunkt ausgeht, mit der Oberfläche ungerade ist.

Das Innere eines Polyeders ist also vollständig durch seine Oberfläche definiert

Höherdimensionale Analoga zu Polyedern: *Polytope*

topologisch "einfachste" nichttriviale Polytope: *Simplices*

0-Simplex: Punkt
1-Simplex: Strecke
2-Simplex: Dreieck
3-Simplex: Tetraeder
...

math. Def.: *m*-Simplex = konvexe Kombination von *m*+1 affin unabhängigen Punkten im R^m.

simpliziale Zerlegung (a.: Triangulierung) einer Punktmenge *P*: endliche Menge von Simplices mit den Eigenschaften

- jeder Punkt gehört zu mindestens einem der Simplices
- der Schnitt zweier Simplices ist entweder leer oder eine gemeinsame *k*-dim. Facette (ein Simplex niedrigerer Dimension)
- die Menge aller Facetten auf dem Rand der Triangulierung bildet ein konvexes Polytop (Polygon im R^2, Polyeder im R^3).

koordinatenbasierte Def. des Simplex:

- $p_i := (p_{i1}, p_{i2}, \dots, p_{id}) \in \mathbb{R}^d$, i = 0,…,d
- Unter dem von den Punkten p_i, i = 0,…,d, aufgespannten Simplex versteht man die Menge

$$S(p_0,p_1,\cdots,p_d) := \left\{ \sum_{i=0}^{d} \alpha_i \cdot p_i : \quad \alpha_i \geq 0, \quad \sum_{i=0}^{d} \alpha_i = 1 \right\}$$

- Das vorzeichenbehaftete Volumen von $S(p_0,p_1,\dots,p_d)$ ist gegeben durch $\frac{1}{d!} \cdot \Delta(p_0,p_1,\cdots,p_d)$, wobei

$$\Delta(p_0,p_1,\cdots,p_d) = \begin{vmatrix} p_{01} & p_{02} & \cdots & p_{0d} & 1 \\ p_{11} & p_{12} & \cdots & p_{1d} & 1 \\ \vdots & & & \vdots & \vdots \\ p_{d1} & p_{d2} & \cdots & p_{dd} & 1 \end{vmatrix}$$

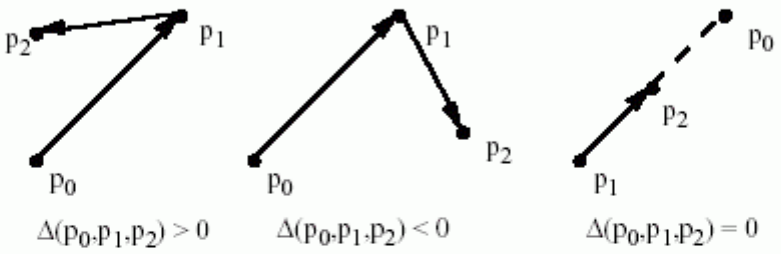
Beweis siehe Forster, Analysis 3, S. 50

2.6. Einfache Tests

Orientierungstest als Beispiel eines "geometrischen Primitivs" (Baustein geometrischer Algorithmen):

- Gegeben: $p_0, p_1, p_2 \in \mathbb{R}^2$, $p_i := (p_{ix}, p_{iy})$

- Liegt p_0 zur Linken oder zur Rechten der gerichteten Linie von p_1 nach p_2 oder auf der Geraden durch p_1 und p_2, d. h. bilden p_0, p_1, p_2 eine Linksdrehung, eine Rechtsdrehung oder sind sie kollinear?



- Diese Frage wird beantwortet durch das Vorzeichen der Determinanten $\Delta(p_0, p_1, p_2)$: Seien e_x, e_y, e_z die Einheitsvektoren entlang der Koordinatenachsen $\Rightarrow$

$$(p_1 - p_0) \times (p_2 - p_0) = \begin{vmatrix} p_{1x} - p_{0x} & p_{1y} - p_{0y} \\ p_{2x} - p_{0x} & p_{2y} - p_{0y} \end{vmatrix} \cdot e_z$$

$$= \begin{vmatrix} p_{0x} & p_{0y} & 1 \\ p_{1x} & p_{1y} & 1 \\ p_{2x} & p_{2y} & 1 \end{vmatrix} \cdot e_z$$

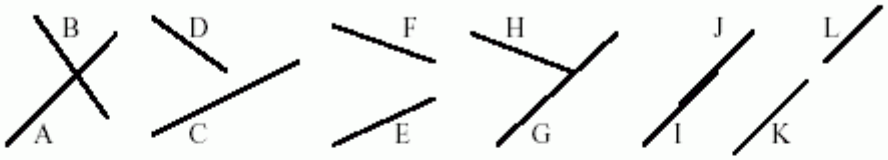
$$\Rightarrow \Delta(p_0, p_1, p_2): \begin{cases} < 0: \text{ Rechtsdrehung} \\ = 0: \text{ kollinear} \\ > 0: \text{ Linksdrehung} \end{cases}$$

- Prädikate:
  leftTurn(p_0, p_1, p_2) := ($\Delta(p_0, p_1, p_2) > 0$)
  rightTurn(p_0, p_1, p_2) := ($\Delta(p_0, p_1, p_2) < 0$)
  collinear(p_0, p_1, p_2) := ($\Delta(p_0, p_1, p_2) = 0$)

- Übertragung des Orientierungstests auf $\mathbb{R}^d$:
  Die Orientierung einer Folge von d+1 Punkten $p_0, p_1, \ldots, p_d \in \mathbb{R}^d$ ist positiv, null oder negativ und entspricht dem Vorzeichen der Determinante $\Delta(p_0, p_1, \ldots, p_d)$. Ist die Orientierung null, so liegen alle Punkte in einer (d–1)-dimensionalen Hyperebene.

Linienschnitt-Test:

- Teste, ob sich zwei Liniensegmente $\overline{p_0p_1}$ und $\overline{p_2p_3}$, gegeben durch $p_i := (p_{ix}, p_{iy}) \in \mathbb{R}^2$, schneiden.
- degenerierte Fälle:

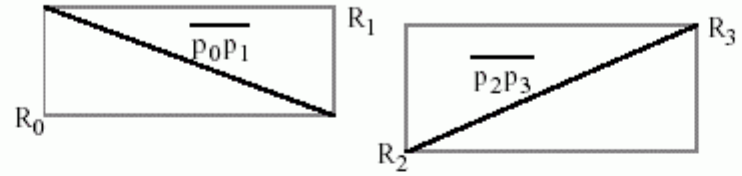


- naive Lösung (für nicht zu Punkte degenerierte Segmente) basiert auf der Schnittpunktberechnung der zwei zugehörigen Geraden:
  1. Überprüfe, ob die zwei Liniensegmente parallel sind (dies ist eine notwendige Vorsichtsmaßnahme, bevor man versucht, den Schnittpunkt zu berechnen). Falls dem so ist, liegt eine degenerierte Konfiguration vor, die zu einem der folgenden drei Spezialfälle gehört: nicht kollinear, kollinear nicht-überlappend, kollinear überlappend.
  2. Berechne den Schnittpunkt der zugehörigen Geraden (dies kann für fast parallele Geraden zu numerischen Problemen führen).
  3. Überprüfe, ob der Schnittpunkt auf beiden Liniensegmenten liegt.

- Soll nur das Entscheidungsproblem gelöst werden, kann man sich den Aufwand zur Berechnung des Schnittpunktes sparen.

Effizienterer Linienschnitt-Test:

1. Containertest:
   Schneiden sich die Container der beiden Segmente?
   Der Container eines geometrischen Objekts das kleinste achsenparallele Rechteck, das dieses Objekt enthält.

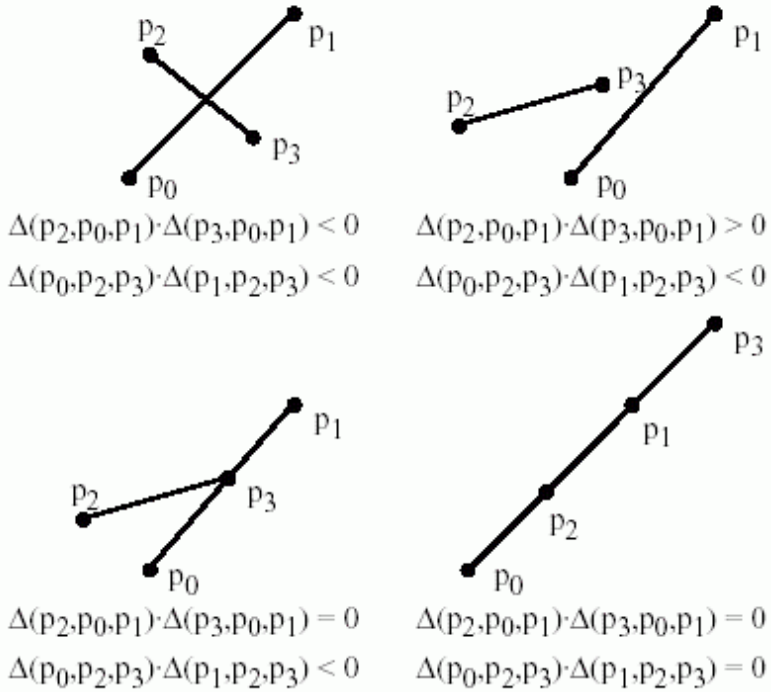


   Container des Liniensegments $\overline{p_0p_1}$: Rechteck (R_0,R_1) mit unterem linken Eckpunkt $R_0 := (r_{0x}, r_{0y})$ und oberem rechten Eckpunkt $R_1 := (r_{1x}, r_{1y})$, wobei
   $r_{0x} := \min(p_{0x}, p_{1x})$, $r_{1x} := \max(p_{0x}, p_{1x})$,
   $r_{0y} := \min(p_{0y}, p_{1y})$, $r_{1y} := \max(p_{0y}, p_{1y})$.
   Analog: Container (R_2,R_3) des Liniensegments $\overline{p_2p_3}$.
   Die beiden Container (R_0,R_1) und (R_2,R_3) schneiden sich genau dann, wenn das Prädikat

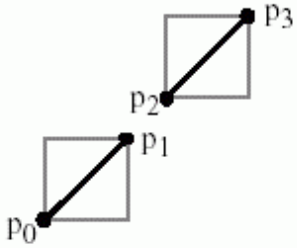
$$(r_{1x} \geq r_{2x}) \wedge (r_{3x} \geq r_{0x}) \wedge (r_{1y} \geq r_{2y}) \wedge (r_{3y} \geq r_{0y})$$

   wahr ist.

2. Schnittest:
   Schneiden sich die beiden Container, so werden die beiden Segmente auf Schnitt überprüft. Idee: Zwei Liniensegmente schneiden sich, falls die beiden Endpunkte des einen Liniensegments jeweils auf verschiedenen Seiten der durch das andere Liniensegment festgelegten Geraden oder auf dieser Geraden liegen. Dies trifft genau dann zu, wenn das Prädikat
   $(\Delta(p_0,p_2,p_3) \cdot \Delta(p_1,p_2,p_3) \leq 0) \wedge (\Delta(p_2,p_0,p_1) \cdot \Delta(p_3,p_0,p_1) \leq 0)$
   wahr ist.



Der Fall, daß $\overline{p_0p_1}$ und $\overline{p_2p_3}$ kollinear sind, sich aber nicht überlappen, wird bereits durch den Containertest ausgeschlossen:



- Diese Tests funktionieren auch für vertikale Segmente.
- Zwei Liniensegmente $\overline{p_0p_1}$ und $\overline{p_2p_3}$ mit Containern (R_0,R_1) und (R_2,R_3), die wie oben definiert sind, schneiden sich genau dann, wenn das folgende Prädikat wahr ist:

$$\begin{aligned}\text{intersect}(\overline{p_0p_1}, \overline{p_2p_3}) := (\quad & (r_{1x} \geq r_{2x}) \wedge (r_{3x} \geq r_{0x}) \\ \wedge\ & (r_{1y} \geq r_{2y}) \wedge (r_{3y} \geq r_{0y}) \\ \wedge\ & (\Delta(p_0,p_2,p_3) \cdot \Delta(p_1,p_2,p_3) \leq 0) \\ \wedge\ & (\Delta(p_2,p_0,p_1) \cdot \Delta(p_3,p_0,p_1) \leq 0)\ )\end{aligned}$$

(aus Hinrichs 2001)

Geschickte Bearbeitung von mehr als 2 Liniensegmenten: folgt später.

Element-Test für Punktmengen *W*:

es soll für ein beliebiges $x \in \mathrm{R}^n$ entschieden werden, ob *x* in *W* liegt oder nicht.

Satz:
In der Real RAM werde für einen Vorzeichentest der Art

$$c_1x_1 + \ldots + c_nx_n + d < 0 \; ? \qquad (*)$$

nur ein Rechenschritt veranschlagt. Die Menge *W* habe *m* Zusammenhangskomponenten. Dann benötigt jeder Algorithmus für den Elementtest für *W* im schlimmsten Fall mindestens $\log_2 m$ viele Schritte.

Beweis:

Sei *A* ein Algorithmus für den Elementtest von *W*.
E sei der lineare Entscheidungsbaum von *A*:
Blätter = mögliche Ausgänge,
innere Knoten = Tests der Form (*) in allen möglichen Rechenabläufen von *A*.
Es reicht, zu zeigen: *E* hat mindestens so viele Blätter wie W Zusammenhangskomponenten hat.

Für jedes Blatt *b* sei
$A_b = \{ x \mid A$ terminiert bei Eingabe von *x* im Blatt *b*$\}$.

A_b ist Durchschnitt von endlich vielen offenen und abgeschlossenen Halbräumen – den Lösungsräumen von Ungleichungen der Form (*) bzw. ihren Komplementen.

Diese Halbräume sind konvex $\Rightarrow A_b$ ist konvex, also zusammenhängend.

Für alle Punkte *x* in A_b trifft *A* dieselbe Entscheidung
$\Rightarrow A_b$ liegt ganz in *W* oder ist mit *W* disjunkt.

Da der Algorithmus für jede Eingabe terminiert, gilt außerdem:

$$\mathbb{R}^n = \bigcup_{b \text{ Blatt}} A_b,$$

und es folgt: *W* entspricht genau einer bestimmten Teilmenge *B* aller Blätter von *E*, also

$$
\begin{aligned}
W &= \mathbb{R}^n \cap W \\
&= \bigcup_{b \text{ Blatt}} A_b \cap W \\
&= \bigcup_{b \in B} A_b
\end{aligned}
$$

jede der Mengen A_b ist zusammenhängend $\Rightarrow$
W kann höchstens soviele Zusammenhangskomponenten haben, wie B Elemente hat $\Rightarrow$

da Anzahl *aller* Blätter $\geq |B|$, folgt die Behauptung.

Problem "ε-Closeness":

Gegeben seien n reelle Zahlen $x_1, ..., x_n$ und ein $\varepsilon > 0$. Gefragt ist, ob es Indices $i \neq j$ gibt, so dass $|x_i - x_j| < \varepsilon$ ist.

Satz:
Das Problem ε-Closeness hat (unter den Voraussetzungen des vorigen Satzes) die Zeitkomplexität $\theta(n \log n)$.

Beweis:
ε-Closeness ist ein Elementtest-Problem für die Menge

$$W = \{ (x_1, ..., x_n) \in \mathrm{R}^n \mid i \neq j \Rightarrow |x_i - x_j| \geq \varepsilon \}.$$

Man zeigt zunächst:
Die Zusammenhangskomponenten von W sind genau die Mengen

$$W_\pi = \{ (x_1, ..., x_n) \in W \mid x_{\pi(1)} < x_{\pi(2)} < ... < x_{\pi(n)} \},$$

wobei π alle Permutationen der n Indices durchläuft.

Seien p, $q \in W_\pi$, es gelte $p_i \geq p_j + \varepsilon$.

$\Rightarrow q_i \geq q_j + \varepsilon$

$\Rightarrow \forall t \in [0; 1]$: $(1-t)p_i + t\, q_i \geq (1-t)(p_j + \varepsilon) + t(q_j + \varepsilon)$
$= (1-t)p_j + t\, q_j + \varepsilon$

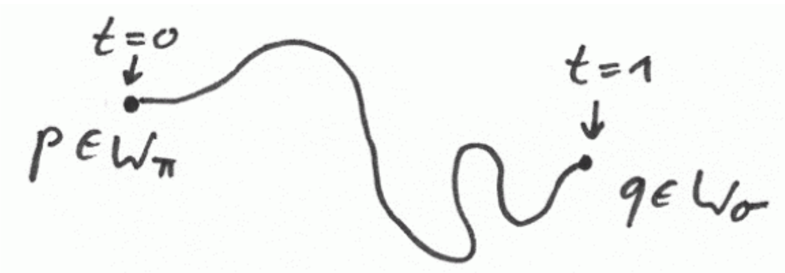
$\Rightarrow$ die gesamte Strecke pq liegt in W_π

$\Rightarrow W_\pi$ ist konvex

$\Rightarrow W_\pi$ ist zusammenhängend.

Seien nun $p \in W_\pi$, $q \in W_\sigma$
für verschiedene Permutationen $\pi \neq \sigma$

$\Rightarrow \exists(i; j)$: $p_i < p_j$ und $q_i > q_j$.
Sei $f(t) = (f_1(t), ..., f_n(t))$ eine Parametrisierung eines beliebigen Weges von p nach q ($t \in [0; 1]$):



(Wege sind stetig!)

Dann gilt:
$f_i(0) - f_j(0) = p_i - p_j < 0$
$f_i(1) - f_j(1) = q_i - q_j > 0$.
Mit dem Zwischenwertsatz für stetige Funktionen folgt:
$\exists\, t \in (0; 1)$: $f_i(t) = f_j(t)$
$\Rightarrow f(t) = (f_1(t), ..., \underset{\uparrow}{f_i(t)}, ..., \underset{\uparrow}{f_j(t)}, ..., f_n(t)) \notin W$ (mit $f_i(t) = f_j(t)$)

$\Rightarrow$ die W_π sind die paarweise disjunkten Zusammenhangskomponenten von W.
Anzahl: $n!$

$\Rightarrow$ mit dem vorigen Satz folgt:
Die Zeitkomplexität liegt in $\Omega(\log n!)$, und damit in $\Omega(n \log n)$.

Andererseits: Eine Lösung von "ε-Closeness" hat man, wenn man die n Zahlen zuerst sortiert und dann in linearer Zeit die aufeinanderfolgenden Paare prüft ($x_i - x_{i+1} \geq \varepsilon$?).

$\Rightarrow$ obere Schranke $O(n \log n)$

$\Rightarrow$ insgesamt $\theta(n \log n)$.

Für $\varepsilon \to 0$ ergibt sich eine entsprechende Aussage für das Problem "**Elementeindeutigkeit**" (*element uniqueness*):

gegeben sind n reelle Zahlen $x_1, ..., x_n$, gefragt ist, ob es Indices $i \neq j$ gibt mit $x_i = x_j$.

Korollar:
Das Problem "Elementeindeutigkeit" hat (unter den Voraussetzungen des vorigen Satzes) die Zeitkomplexität $\Theta(n \log n)$.
jetzt Beispiele für Transformation von Problemen:

Korollar:
Auch bei Zugrundelegung eines Berechnungsmodells, das (wie in den beiden vorangehenden Sätzen) für einen Vorzeichentest der Art "$c_1x_1 + ... + c_nx_n + d < 0$?" nur einen Rechenschritt veranschlagt, hat *Sortieren* die Zeitkomplexität $\Theta(n \log n)$.

Beweis: Könnte man schneller sortieren $\Rightarrow$ das Problem "ε-Closeness" könnte schneller gelöst werden, im Widerspruch zum vorigen Satz.

Problem "**nächste Nachbarn**" (*all nearest neighbours*):
Für jeden Punkt einer n-elementigen Punktmenge aus R^d soll ein nächster Nachbar (bzgl. euklidischer Distanz) aus dieser Punktmenge bestimmt werden.

Satz:
Das Problem "nächste Nachbarn" hat (unter denselben Voraussetzungen wie die vorigen Sätze) die Zeitkomplexität $\Omega(n \log n)$.

Beweis:
Annahme: Berechnung der nächsten Nachbarn geht schneller
$\Rightarrow$ schnellere Lösung von "ε-Closeness" :

seien $x_1, ..., x_n \in R$ und $\varepsilon > 0$ gegeben,
bilde die Punkte $p_i = (x_i, 0, 0, ..., 0) \in R^d$,
bestimme für jeden den nächsten Nachbarn.

Dichtestes Paar (p_i , p_j) mit

$$d(p_i , p_j) = \min d(p_s , p_t),$$

wobei das Minimum über alle (s, t) mit $1 \leq s \neq t \leq n$ erstreckt wird,
kann dann in der Zeit $O(n)$ gefunden werden
$\Rightarrow$ teste, ob $d(p_i , p_j) = |x_i - x_j| < \varepsilon$
$\Rightarrow$ "ε-Closeness" in Zeit schneller als $n \log n$ gelöst.

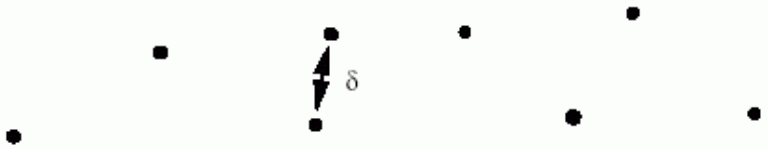
Korollar:
Problem "**closest pair**": Ein dichtestes Paar von n Punkten im R^d zu finden, hat die Zeitkomplexität $\Omega(n \log n)$.

2.7. Metriken (Distanzmaße) für geometrische Probleme

- "Paar mit minimalem Abstand" Problem:
  Bestimme für eine gegebene Menge S von n Punkten in der Ebene das Paar von Punkten mit dem kleinsten Abstand $\delta = \delta(S)$. Wir messen Abstände in der Metrik d_k, für beliebiges $k \geq 1$, oder d_∞, definiert durch

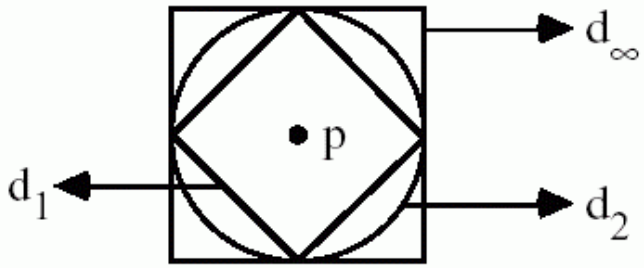
$$d_k((x',y'),(x'',y'')) = \sqrt[k]{|x'-x''|^k + |y'-y''|^k}$$

$$d_\infty((x',y'),(x'',y'')) = \max(|x'-x''|,|y'-y''|)$$



- Spezialfälle von besonderem Interesse:
  Manhattan-Metrik d_1
  Euklidische Metrik d_2
  Maximum-Metrik d_∞

- "Kreise" mit Radius 1 um einen gegebenen Punkt p:



- Im folgenden bezeichne d eine d_k-Metrik ($1 \leq k \leq \infty$).

2.8. Totale Ordnung von Punkten im Raum

- Häufig müssen geometrische Algorithmen Punkte bezüglich einer vorgegebenen totalen Ordnung vergleichen oder bearbeiten; z.B. vergleichen die im folgenden vorgestellten Algorithmen Punkte bezüglich ihrer x- oder y-Koordinaten. Problem: Mehrere Punkte mit gleichen x- oder y-Koordinaten.
- Annahme: Es gibt keine zwei Punkte mit gleichen x- oder y-Koordinaten.
- Diese Einschränkung ist zulässig, da Punkte mit gleichen x- oder y-Koordinaten auf beliebige, aber konsistente Weise total geordnet werden können, z.B. durch die lexikographische Ordnung $\leq_x$ bzw. $\leq_y$:

$$(x', y') \leq_x (x'', y'') \;:\Leftrightarrow\; (x' < x'') \vee ((x' = x'') \wedge (y' \leq y''))$$
$$(x', y') \leq_y (x'', y'') \;:\Leftrightarrow\; (y' < y'') \vee ((y' = y'') \wedge (x' \leq x''))$$

Verallgemeinerung auf Dimensionen > 2 ist klar.

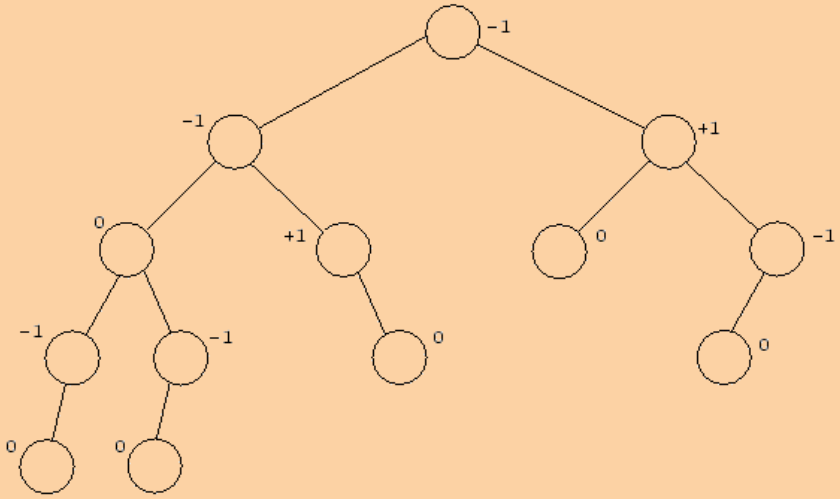
2.9 AVL-Bäume als sucheffiziente Datenstrukturen

AVL-Baum

(benannt nach Adelson-Velskii und Landis, 1962)

Def.:
Ein Knoten eines binären Baumes heißt *ausgeglichen* oder *balanciert*, wenn sich die Höhen seiner beiden Söhne um höchstens 1 unterscheiden.

Def.:
Ein binärer Suchbaum, in dem jeder Knoten ausgeglichen ist, heißt *AVL-Baum*.



Sei *bal* (x) = Höhe des rechten Teilbaums von x minus Höhe des linken Teilbaums von x.

Aufgrund der Ausgeglichenheit ist die Suche in einem AVL-Baum auch im ungünstigsten Fall von der Ordnung $O(\log n)$. Um das zu gewährleisten, muss nach jedem Einfügen oder Löschen die Ausgeglichenheit überprüft werden. Hierzu werden längs des Weges vom eingefügten bzw. gelöschten Element bis zur Wurzel die Balance-Werte abgefragt und durch so genannte *Rotationen* repariert.

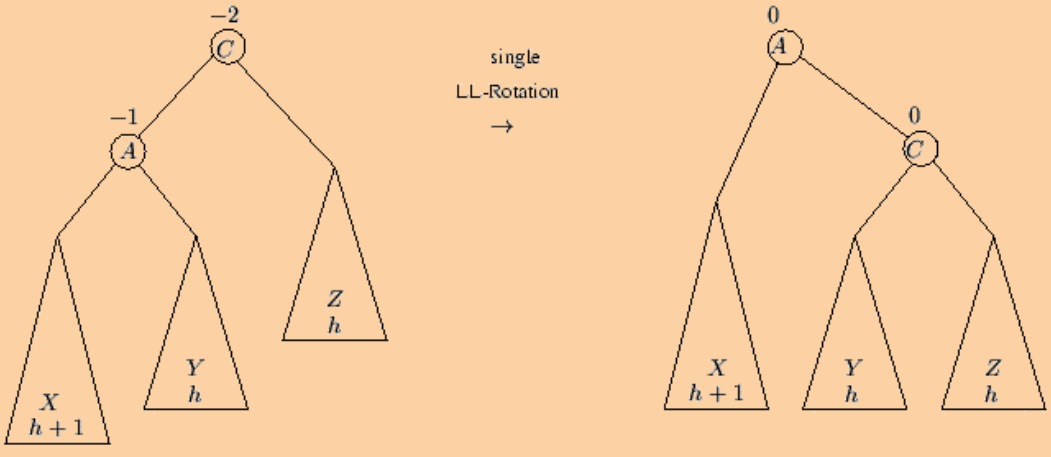
Während das Einfügen eines einzelnen Schlüssels *höchstens eine* Rotation erfordert, kann das Löschen eine Rotation für *jeden* Knoten entlang des Weges zur Wurzel verursachen.

Rotationen für AVL-Baum bei linksseitigem Übergewicht:

Single LL-Rotation

Bei Einfügen in Teilbaum X: Höhe des gesamten Baums vorher und nachher gleich.

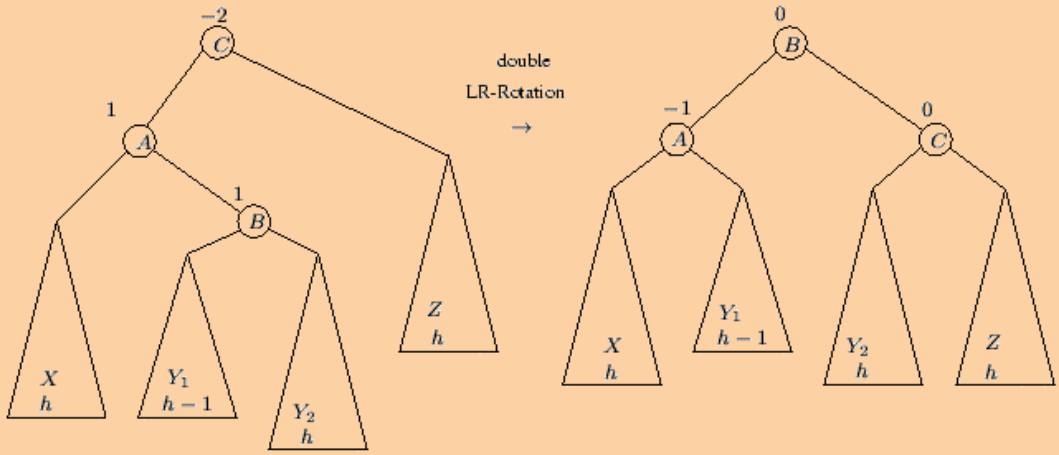
Bei Löschen im Teilbaum Z: Höhe des gesamten Baums vorher und nachher gleich oder nacher um eins kleiner.



Double LR-Rotation

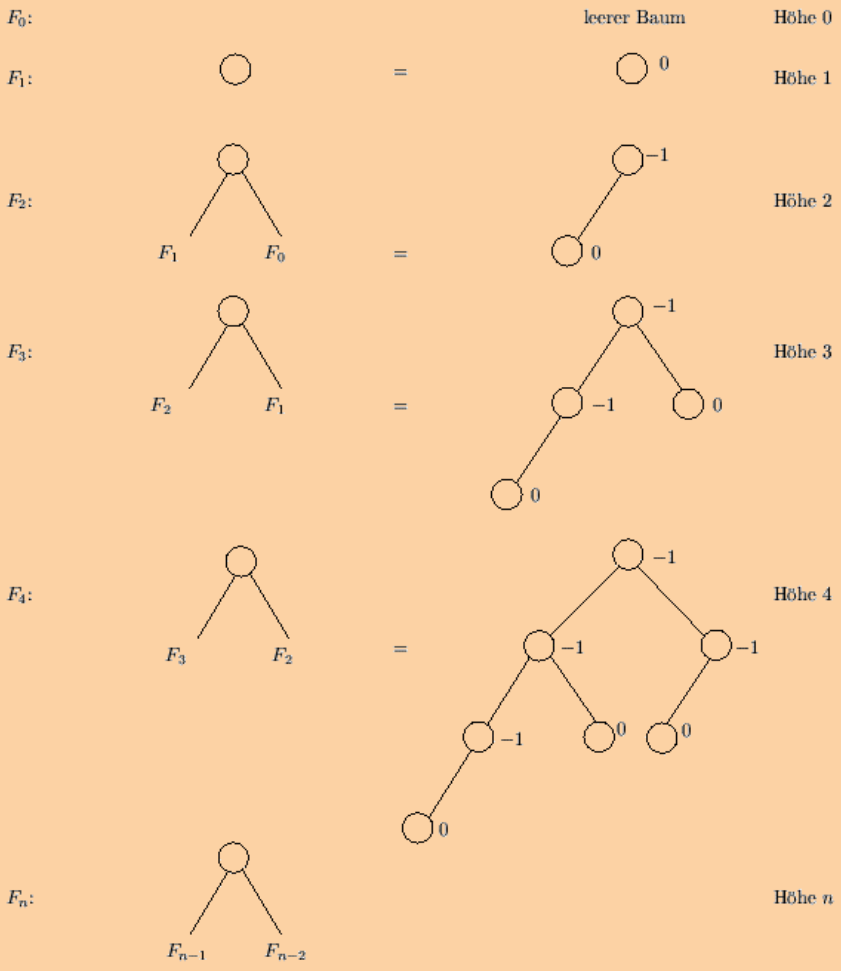
Bei Einfügen in Teilbaum Y_1 oder Y_2: Höhe des gesamten Baums vorher und nachher gleich.

Bei Löschen im Teilbaum Z: Höhe des gesamten Baums nachher um eins kleiner.



(bei rechtsseitigem Übergewicht analog)

Minimal ausgeglichene AVL-Bäume sind *Fibonacci*-Bäume



(aus Vornberger et al. 2000)