Neuere Erkenntnisse der rautavistischen Informatik

Lutz Donnerhacke

Rautavistik*

- die bewußte Ausführung bewußt sinnfreier Handlungen aus eigenem Antrieb,
- die bewußt sinnfreie Nutzung vorhandener Einrichtungen aus eigenem Antrieb,
- die bewußte Nutzung vorhandener sinnfreier Einrichtungen aus eigenem Antrieb,
- die bewußte Erschaffung sinnfreier Einrichtungen aus eigenem Antrieb,
- die bewußte Erschaffung sinnfrei nutzbarer Einrichtungen mit dem Hintergedanken, sie bewußt sinnfrei zu nutzen,

*$<lat.>$ Rauten sehen

Rautavistik *cont*

- bewußte Erschaffung sinnfrei nutzbarer Einrichtungen mit dem Hintergedanken, sie zu sinnfreier insbesondere bewußt sinnfreier Nutzung freizustellen,
- bewußter Austausch bewußt sinnfreier Information aus eigenem Antrieb, bewußte Aufnahme von Information mit dem Hintergedanken, sie bewußt sinnfrei zu nutzen,
- Weitergabe von Information mit dem Hintergedanken, sie bewußt sinnfrei nutzen zu lassen,
- bewußte Weitergabe von Information aus eigenem Antrieb mit dem Hintergedanken, sie sinnfrei nutzen zu lassen,
- bewußte Veranlassung von bewußt rautavistischen Handlungen

!!! ACHTUNG !elf

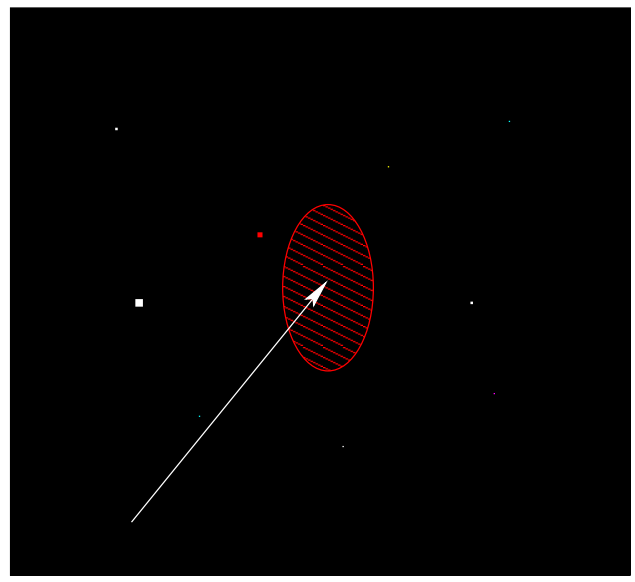
Rautavistik $\neq$ BWL

Rautavistik		BWL
bewußt	$\Leftrightarrow$	unbewußt
sinnfrei	$\Leftrightarrow$	unsinnig

Jedoch: BWL unterliegt keiner doppelten Negation!

Astronomisches Institut

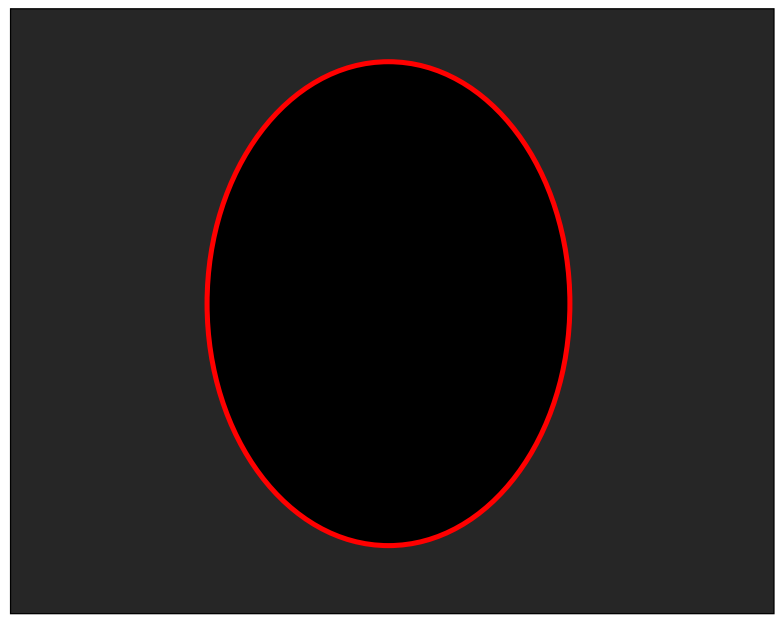
»Very deep space astronomy«



Beobachtung von Röntgenstahlung im langwelligen Infrarot

. . . oft negative Wellenlängen

Praktikumsversuch



Hier durch Rotverschiebung auf Kurzwelle beobachtbar

Informatik

- Klassische rautavistische Informatik $\Rightarrow$ Pascal
- Neuere rautavistische Informatik $\Rightarrow$ Haskell
- Vergleich

Pascal	*Haskell*
imperative fordernd	**functional** funktionierend
eager früh	**lazy** faul
impure verdreckt	**pure** Freibier
procedural arbeitend	**descriptive** beschreibar
$\Rightarrow$ nachvollziehbar	$\Rightarrow$ Voodoo

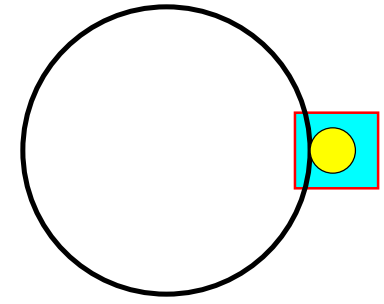
Also: Einsatz von Haskell verwirrt mehr

Klassische Algorithmen

Bubble Sort

```
bubbleSort :: Ord a => [a] -> [a]
bubbleSort xs | xs == xs' = xs
              | otherwise = bubbleSort xs'
   where xs' = bubbleUp xs
         bubble (x1:x2:xs) | x1 <= x2  = x1 : bubble (x2 : xs)
                           | otherwise = x2 : bubble (x1 : xs)
         bubble xs = xs
```

- Nicht offensichtlich in der Funktion
- Lustige Namen
- Üblicherweise per Fixpunkt Kombinator
- Optimal für ringförmige Turingmaschine mit einem Register



Random Sort

```
randomSort :: Ord a => [a] -> [a]
randomSort = head . filter isSort . iterate shuffle
```

- `iterate shuffle` = Menge aller zufälligen Permutationen

$\Rightarrow$ Erstes Element der sortierten Permutationen, die durch unablässiges Vertauschen entstehen

- Maximale Laufzeit unbestimmt: Gut
- Leider minimale Laufzeit $O(1)$: Schlecht

Random Sort *Details*

```
isSort :: Ord a => [a] -> Bool
isSort (x1:xs@(x2:_)) | x1 <= x2  = isSort xs
                      | otherwise = False
isSort _  = True
```

Definition Eine Inversion ξ über[†] einem Tupel ζ ist das Paar $(\underline{i}, i)$, so daß gilt $\underline{i} < i \Rightarrow \zeta_i < \zeta_{\underline{i}}$

- Triviale Suche nach erster Inversion, dann Abbruch
- Übungsaufgabe: Korrektheitsbeweis von `bubbleSort`
- Laufzeit linear, maximal aber $O(n)$

[†] $\frac{\xi}{\zeta}$

Random Sort *Details*

```
shuffle :: [a] -> [a]
shuffle = sortBy r
  where r _ _ | fst . random $ unsafePerformIO newStdGen = LT
              | otherwise                                 = GT
```

- Klassisch undurchsichtig
- Verwendet seinerseits einen Sortieralgorithmus
- `unsafePerformIO` erfindet Welten, um Zufall zu generieren
- Inhärentzbeziehung zwischen Chaos und Quantenmechanik
- Computer brauchen eine Umgebung, um zu funktionieren
- Optimizer kann `r` in `const` überführen
- Rekursion führt zu Absturz

Laufzeitpessimierung

Problem `isSort` ist maximal linear und bricht viel zu früh ab

Ursache Erste gefundene Nachbarschaftsinversion führt zum Abbruch

Verunsicherung Können Inversionen auch übersehen worden sein?

Lösung

- Gründlichere Suche
- Kontrolle des Kontrollierten
- Prüfung entfernterer Inversionen

Alternative `isSort`-Implementierungen

```
isSort xs@(_:xs') = all id $! zipWith (<=) xs xs'
isSort _          = True
```

- Verdeutlichung des Wirkprinzips
- Erzeugung *aller* Nachbarschaftsvergleiche
- Sonderbehandlung nur für 0-elementige Menge
- Minimale Laufzeit linear, keine Nachbarschaftsinversion übersehen
- Verwendung des Sequenzoperators nötig, da Haskell faul

Alternative `isSort`-Implementierungen *cont*

```
isSort xs = null [ (i, j) |
                   i <- [0     .. length xs - 2],
                   j <- [i + 1 .. length xs - 1],
                   xs !! j < xs !! i ]
```

- Erzeugung *aller* Vergleiche
- Literale Übernahme der Definition
- Quadratische Laufzeit möglich !
- Quadratischer Platzbedarf möglich !
- Statt `null` erzwingt `0 == length` mehr Berechnung
- Neues Tätigkeitsfeld: *Kampf dem faulen Computer !*

Alternative `isSort`-Implementierungen *cont*

```
isSort xs = sort xs == xs
```

- Gänzlich andere Taktik
- Verweis auf Implementation von Dritten
- Minimale Laufzeit ist die der Referenz
- Möglichkeit der verdeckten Rekursion $\Rightarrow$ Tautologie
- Gewaltige Möglichkeiten, mit doppeltem `randomSort` Laufzeit zu potenzieren

Sortieren in konstanter Zeit

```
lossySort :: Ord a => [a] -> [a]
lossySort (x':x'':xs) = [min x' x'', max x'' x']
lossySort xs = xs
```

- `lossySort` sortiert in **konstanter** Zeit
- Algorithmus ist *rautavistisch*, weil nutzlos
- Sonderfälle `length xs < 3` vernachlässigt

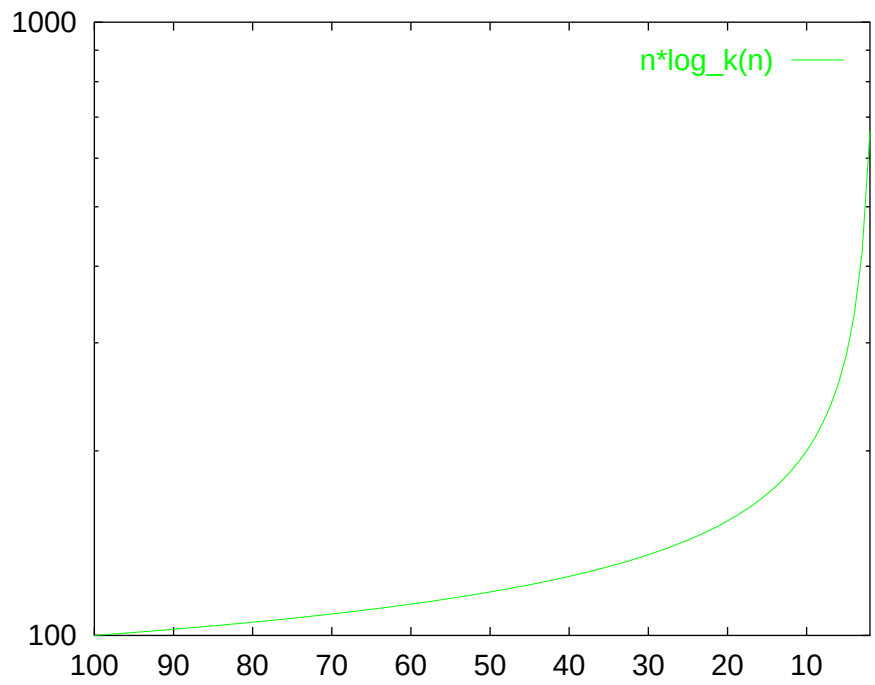
Sortieren in linearer Zeit

```
mergeSort :: Ord a => [a] -> [a]
mergeSort [ ] = [ ]
mergeSort [x] = [x]
mergeSort xs  = mergeSort a `merge` mergeSort b
  where (a,b) = pair xs
        pair (x:x':xs) = let (a,a') = pair xs in (x:a, x':a')
        pair x = (x,[])
        merge (x:xs) (y:ys) | x <= y = x : merge xs (y:ys)
                            | True   = y : merge (x:xs) ys
        merge x y = x ++ y
```

- Mergesort hat Laufzeit $O(n \log n)$
- Mergesort teilt in *zwei* Haufen und beherrscht diese

Sortieren in linearer Zeit *cont*

- Aufteilung in k Haufen führt zur Laufzeit von $O(n \log_k n)$



$\Rightarrow$ n-Wege Mergesort mit Laufzeit von $\boxed{O(n \log_n n) = O(n)}$

- Wichtigstes Optimierungsergebnis der Rautavistik im Jahre 2002

$\Rightarrow$ Prüfungsstoff

Deutlich Endliche Graphen

Definition Ein DEG $\delta := (\nu, \phi)$ mit:

$$\begin{aligned} \iota \in \phi &\Rightarrow \iota = (\alpha, \beta) : \alpha, \beta \in \nu \\ \Xi, \underline{\zeta} \in \nu &\Rightarrow \Xi^{\iota} = \underline{\zeta}^{\iota} \\ \gamma^{-1} = \overline{\gamma}^{1} &\Leftarrow \gamma, \overline{\gamma} \in \phi \\ \widehat{\xi}, \overline{\xi} \in \nu &\Rightarrow (\overline{\xi}, \widehat{\xi}) \in \phi \end{aligned}$$

```
firstVertex :: DEG a -> a
firstVertex = head . fst
```

`firstVertex` löst die folgenden Probleme: Breitensuche, minimaler Spanbaum, teuerster Pfad, Auffindung des Handelsreisenden, kürzester Pfad, Zusammenhang, Tiefensuche und Gordischer Knoten.

DEG *cont*

Satz `firstVertex` läuft in $\mathsf{O}(1 + \sin n \cdot \sqrt{\frac{n}{n}})$

Auffällige Vorbemerkungen

- OBdA genügt eine Beschränkung auf die ariden Gebiete von τ
- ζ kann nie verschwinden, auch wenn $\overline{\overline{\Xi K}} = \sum_\iota \xi_\iota \kappa^\iota$
- *Kovariante Summenbildung erfolgt ab sofort implizit, d.h.* $\sum_\ldots$ *kann immer entfallen, solange die Indicies paarweise auftreten*
- **Lemma** Irreduzible Komposita im Phasenraum sind konträr

  Beweis aus lineare Algebra II bekannt

Testfolie

- _ geht das? Absatz und

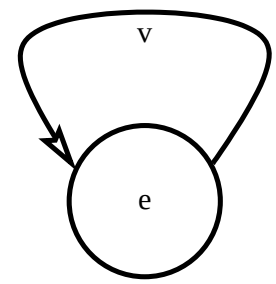
- Pfeile $\boxed{\text{Test}}$ $\underline{ul}\overline{ol}\frac{1}{2}\sqrt{n}$

- Vor Präsentation unbedingt entfernen!

$$\begin{array}{cc} a & b \\ \hline c & d \\ e & f \end{array}$$

Beweis *cont*

- Somit gilt an jedem dritten $\omega \in \Omega'$
- $|\nu| = 1$ und damit $|\phi| = 1$, also ist jeder DEG folgender Form



- q.e.d.

Bäume

Definition Ein Baum B ist ein ((D)E)G. Der Startpunkt heißt **Wurzel**, die Verbindungen nennt man **Zweige**, die Zusammenstöße von Zweigen nennt man **Knoten** und die Enden **Blätter**.

- Typische Informatiker malen Bäume bunt an
- Bunt- und Filzstifte entlassen gefährliche Gifte in die Umgebung
- Im Herbst besondere Gefahr: *Laubzüge*

$\Rightarrow$ **Sofortige Beendingung der Geldverschwendung !**

Drittmittel in umweltbewußte Rautavistenhände !

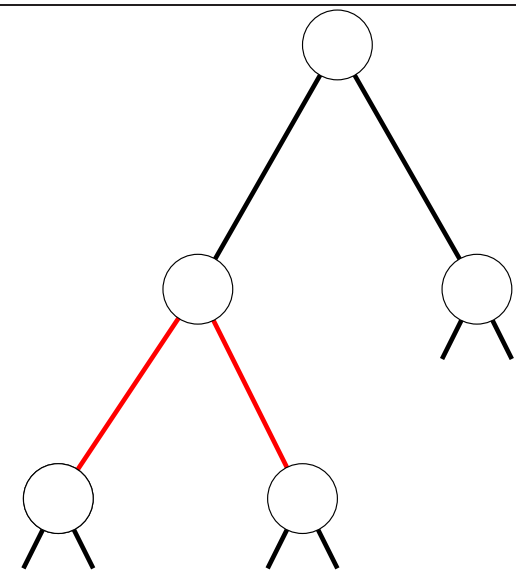
Farbbäume

Definition Ein Farbbaum CB ist ein Baum, der gefärbt ist. Färbungen sind entweder an den Knoten oder an den Zweigen möglich.

Corrolar Für die $\sqrt{\text{Wurzel}}$färberung ist jeder selbst verantwortlich.

- An der FSU-Jena existiert ein Antiquariat und eine Manufaktur für $\sqrt{\text{Wurzeln}}$ $\Rightarrow$ **Standortvorteil**
- Konsequenter Einsatz zyklonfreier Farben $\Rightarrow$ **azyklische Bäume**
- Klassisch bekannt ist der Rot-Schwarz-Baum

Rot-Schwarz-Bäume‡



1. Die Wurzel ist schwarz
2. Die Blätter sind schwarz
3. Nach Rot folgt Schwarz im Pfad
4. Jeder Pfad enthält gleich viel Schwarz

- Grundsätzlich mehr schwarze als rote Färbungen
- Färbungen sind wild verstreut, systemlos
- Neuansatz zur Färbung nötig

‡Benannt nach den Erfindern *Herbert Roth*§und *Roy Black*

§Man beachte den Verlust des `h` durch Translation

Kratische Bäume

Definition Ein Baum heißt **demokratisch**, wenn die Werte in den *Blättern* stehen.

Satz Die Werte eines demokratischen Knotens entscheiden abstrakt über die Klassifizierung des Ziels.

Beweis Die Knotenwerte sind unkorreliert zu den tatsächlichen Werten. Jede Suche endet ausschließlich an Blättern. $\Rightarrow$ Der Knoten hat keine Vorteile von einer Falschauskunft.

Kratische Bäume *Cont*

Definition Ein Baum heißt **autokratisch**, wenn die Werte in den *Knoten* stehen.

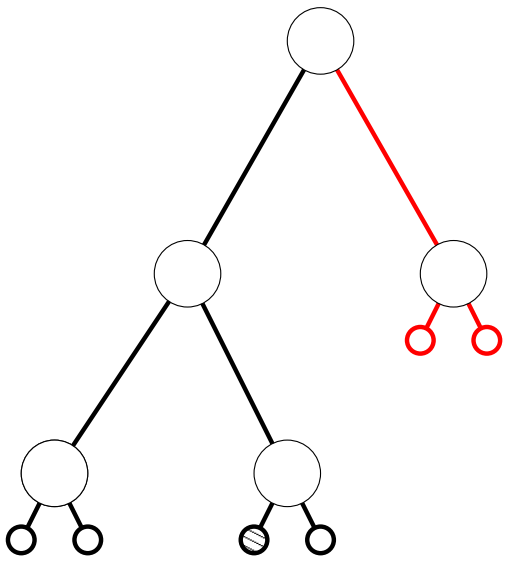
Satz Die Werte eines autokratischen Knotens entscheiden konkret über die Ergebnisse des Ziels.

Beweis Die Knotenwerte sind die tatsächlichen Werten. Jede Suche endet, sobald dieser Knoten das entscheidet. $\Rightarrow$ Der Knoten kann unkontrolliert die Suche beeinflussen.

Kratische Bäume *Cont*

- Demokratische Bäume sind vorzuziehen, weil beständiger
- Abstrakte Entscheidung gestattet Klassifizierung

⇒ Schwarz nach rechts und Rot nach links !



- Schwarz hat immer einfache Mehrheit
- Mehrheit für Rot nur übergangsweise

⇒ Änderung des Systems nötig

- Verschiebung der abstrakten Grenze nach rechts, um mehr Blätter als rot markieren zu können

⇒ **Problem gelöst !**

Baumpflege

- Kratische Bäume sind nicht stabil
- Länger Lagerung führt zu Schäden an den Rändern
- Im linken, roten Grenzbereich treiben neue Blätter
- Am rechten Rand ist mit Fäule zu rechnen

$\Rightarrow$ Bäume regelmäßig benutzen

- Altlasten ausräumen

Bäume im Winter

Satz Demokratische Bäume sind nicht winterhart.

Beweis Bäume werfen im Winter ihre Blätter ab. Da demokratische Bäume ihre Werte in den Blättern halten, sind demokratische Bäume im Winter wertlos. q.e.d.

Bäume bestehen also im Winter nur noch aus Wurzel, Knoten und Zweigen.

Bäume im Winter *Cont*

Lemma Bäume im Winter sind entweder unendlich groß oder fraktal.

Beweis Aus einem Knoten wachen zwei Zweige, die in Knoten enden, weil es keine Blätter mehr gibt. Die Wurzel existiert als Startknoten. Mit vollständiger Induktion kommt man so auf unendlich viele Zweige und Knoten.

Satz Bäume sind im Winter unendlich groß.

Beweis Knoten und Zweige haben eine Mindestgröße, da das Bit unteilbar ist. Bäume sind also niemals fraktal.

Corrolar Biologen sind blind.

Fragen ?