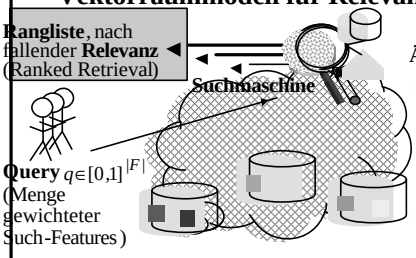
2 Vektorraummodell und IRS-Architektur

Grundprinzipien:
- **Featureraum**: Wörter in Dokumenten werden auf Terme reduziert.
- **Dokumentenmodell:** Jedes Dokument di wird als Vektor $\in [0,1]^{|F|}$ repräsentiert, wobei dij das Gewicht des j-ten Terms in di angibt.
- **Anfragemodell:** Anfragen sind Vektoren qi $\in [0,1]^{|F|}$
- **Relevanz:** Suchresultatsranking basiert auf einer Ähnlichkeitsfunktion im Vektorraum $[0,1]^{|F|}$
- **Crawling:** Das Web wird entlang von Hyperlinks traversiert, um Dokumente zu analyiseren und zu indexieren.
- **Indexierung:** Zu jedem Term wird eine Liste von Dokumenten-Ids (z.B. URLs) mit dem jeweiligen Gewicht in einem „invertierten File" (Suchbaum oder Hash-File) angelegt.
- **Anfrageverarbeitung:** Anfragen werden zerlegt in Index-Lookups für Einzelterme, um Trefferkandidatenlisten zu bestimmen.

Vektorraummodell für Relevanz-Ranking



Verwendete Ähnlichkeitsfunktionen sind z.B.:

$$sim(d_i,q) := \sum_{j=1}^{|F|} d_{ij}\, q_j$$ (Skalarprodukt)

$$sim(d_i,q) := \frac{\sum_{j=1}^{|F|} d_{ij}\, q_j}{\sqrt{\sum_{j=1}^{|F|} d_{ij}^2 \sum_{j=1}^{|F|} q_j^2}}$$ (Cosinus-Maß)

Termgewichtung in Dokumenten

Betrachtet werden die folgenden Werte (für N Dokumente und M Terme):
- tf_{ij}: Häufigkeit (term frequency) des Terms ti in Dokument dj
- df_i: Anzahl der Dokumente mit Term ti (doc. frequency)
- idf_i: N / df_i (inverse document frequency)
- cf_i: Häufigkeit von ti in allen Dokumenten (corpus frequency)

(ggf. mit separater Berücksichtigung von Termen in title u.ä.)

Grundprinzip:
Das Gewicht w_{ij} von Term ti in Dokument dj sollte mit tf_{ij} und mit idf_i monoton wachsen.

→ erster Ansatz: wij = tf_{ij} * idf_i **(tf-idf-Formel)**
Ggf. sollten die Gewichte w_{ij} wie folgt zu ω_{ij} normiert werden:

$$w_{ij} := w_{ij} / \sqrt{\sum_k w_{kj}^2}$$

Variationen der Termgewichtung mit tf und idf

Empirische Resultate zeigen, daß in der Regel die tf- und idf-Werte normalisiert und/oder gedämpft sein sollten.

Normalisierung tf-Werte: $tf_{ij} := \frac{tf_{ij}}{\max_k tf_{kj}}$

Gedämpfte tf-Werte: $tf_{ij} := (1 + \log tf_{ij})$

Gedämpfte idf-Werte: $idf_i := \log \frac{N}{df_i}$

Häufige Variante: $w_{ij} := \frac{tf_{ij}}{\max_k tf_{kj}} \log \frac{N}{df_i}$ **tf*idf-Formel**

$w_{ij} := w_{ij} / \sqrt{\sum_k w_{kj}^2}$

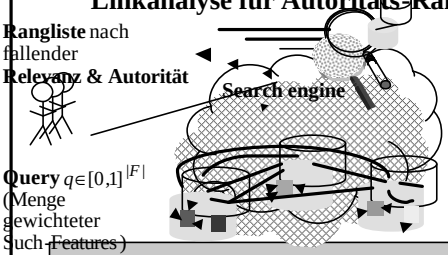
Termgewichtung in Anfragen

Je nach Anfrage-Interface und Benutzerkategorie kommen z.B. die folgenden Gewichtungen der Terme in einer Anfrage qj in Frage:

- $w_{ij} = (0.5 + \frac{0.5 tf_{ij}}{\max_k tf_{lj}}) \times \log \frac{N}{df_i}$

- wij = 1 falls ti in qj vorkommt, 0 sonst
- wij = 1 / k
  falls qj k (konjunktiv verknüpfte) Terme enthält und ti der an l-ter Stelle in qj aufgeführt ist

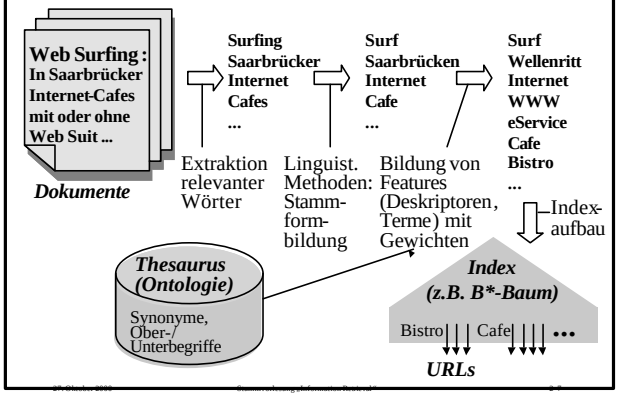
Linkanalyse für Autoritäts-Ranking



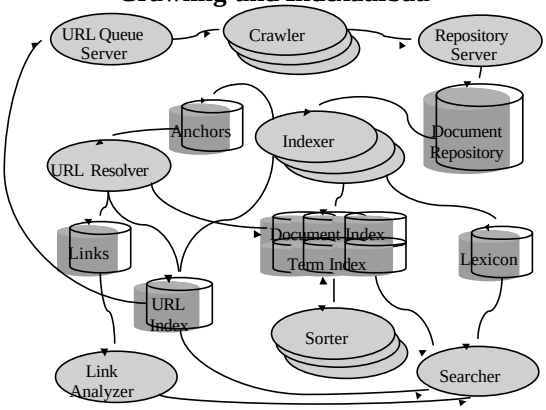
+ **Analysiere In-Degree und Out-Degree von Webseiten:**
  Autoritätsrang (d_i) :=
  Stationäre Besuchswahrscheinlichkeit [d_i] bei Random Walk auf dem Web

Gesamtrang = (ad hoc) gewichtete Summe von Relevanz und Autorität

Inhaltserschließung von Dokumenten und Indexierung



Crawling und Indexaufbau



Datenstrukturen einer Web-Suchmaschine

- **Document Repository**: alle traversierten HTML-Dokumente in komprimierter Form
- **Lexicon:** alle vorkommenden Stammformen jeweils mit TermId
- **DocumentIndex:** sortiertes/indexiertes File mit Einträgen der Form (DocId, Terms (TermId, #Hits, tf-Value, Hits (PosFontTagEncoding)))
- **TermIndex:** sort./index. invertiertes File mit Einträgen der Form (TermId, #Docs, idf-Value, Docs (DocId, #Hits, Hits(...)))
- **Anchors:** alle Hyperlinks in der Form (SourceDocId, TargetDocId, AnchorText)
- **URLIndex:** sortiertes/indexiertes File mit Einträgen der Form (URL, DocId, PtrToRepository) z.B. sortiert nach Checksum(URL)

Komponenten einer Web-Suchmaschine

- **URL Queue Server**: verwaltet Prioritätsliste (noch) zu traversierender Links (z.B. basierend auf In-degree)
- **Crawler (Robot, Spider):** holt Dokumente unter Beachtung von Nebenbedingungen (Filetyp, Robot Exclusion Protocol, usw.)
- **RepositoryServer:** verwaltet DocumentRepository
- **Indexer (incl. Parser, Stemmer):** analysiert Dokumente und erzeugt Einträge in Lexicon, Anchors und DocumentIndex
- **URLResolver:** übersetzt URLs in DocIds (SourceDocId, TargetDocId, AnchorText)
- **Sorter:** erzeugt TermIndex aus DocumentIndex
- **LinkAnalyzer:** berechnet Autoritäts-Ranking aufgrund von Links
- **Searcher:** wertet Anfragen durch Lexicon-, Index- und ggf. Thesaurus-Lookups aus und berechnet Resultats-Ranking aufgrund von tf (evtl. separat für title u.ä.), idf und Autoritätsrang; holt Kandidatendokumente und prüft ggf. weitere Bedingungen

Exkurs: Effizientes Sortieren auf Sekundärspeicher

Annahmen: Dateigröße S Blöcke, Hauptspeichergröße M Blöcke

1) Bilden von sortierten Teilsequenzen (Runs):
   Lade M Blöcke in Hauptspeicher, sortiere intern und schreibe Run in temporäres File.
2) Mischen der Runs:
   Mische M Runs zu einem längeren Run und schreibe in temp. File.
3) Iteriere 2), bis ein einziger Resultat-Run entsteht.

Platten-I/O-Kosten:

- 2S I/Os, um S/M Runs der Größe M zu bilden.
2) 2S I/Os, um S/M^2 Runs der Größe M^2 zu bilden.
3) insgesamt i-1 Mischphasen bis $S/M^i \leq 1$, also $\log_M S \leq i$

Gesamtkosten: $\text{ceil}(\log_M S - 1)2S + 2S$ Platten-I/Os
(bzw. $\leq 4S$ bei $M \geq \sqrt{S}$)

Dimensionen typischer Web-Suchmaschinen

- > 200 Mio. Dokumente
  > 1 Terabyte Rohdaten
- > 10 Mio. Terme
  > 1 Terabyte Index
- > 50 Mio. Anfragen pro Werktag
  < 1 Sek. mittlere Antwortzeit
- < 30 Tage Indexaktualität
  > 200 Webseiten pro Sek. Crawling (> 1 MByte/s)

Typische (High-End-) Serverkonfiguration:
20 SMPs mit jeweils

- 10 CPUs
- 12 GBytes RAM
- 300 GBytes RAID-Plattenplatz

Vereinfachter Query Processor

1) Entferne Stoppwörter aus der Anfrage und
   reduziere alle anderen Wörter auf Terme (in Stammform)
2) Transformiere Anfrage q in die Form
   [(t11 AND t12 AND ... AND $t1k_1$) OR ... OR } q1 ...
   (tr1 AND tl2 AND ... AND trk_r)] } qr
   AND NOT t(r+1) AND NOT ... AND NOT t(r+s)
3) for i=1 to r do
   Finde DocId-Listen für ti1 bis tik_i und
   berechne Vereinigung Vi dieser DocIds
   Setze den RSV von d ∈ Vi auf sim(d,qi)
   Berechne Vereinigung V der DocId-Mengen V1, ..., Vr
4) Sortiere alle d ∈ V absteigend nach ihren RSVs
5) Hole Dokument d ∈ V in Sortierfolge
   - ggf. mit einem heuristischen Abbruchkriterium -
   und eliminiere Dokumente d, in denen einer oder mehrere
   der Terme t(r+1) bis t(r+s) vorkommen

Beispiel für Vektorraummodell

d1: Trainer Daum hat nie Drogen genommen.
Er hat weder Kokain noch andere Drogen genommen.

d2: Manager Hoeness beschuldigt Daum des Drogenkonsums.
Der frühere Bundestrainer Beckenbauer bedauert Daum.

d3: Fussballverein Leverkusen entlässt Trainer Daum.
Völler übernimmt Training.

d4: Daums Haarprobe weist Konsum von Drogen nach.
Der deutsche Fussball distanziert sich vom designierten Bundestrainer Daum. Völler bleibt bis auf weiteres Trainer der deutschen Fussballnationalmannschaft.

q: Fussball Trainer Drogen
(Welcher Fussballtrainer nimmt Drogen?)