Kapitel 8
Methoden für Ähnlichkeitsanfragen

Skript zur Vorlesung: Index- und Speicherungsstrukturen für Datenbanksysteme
Wintersemester 07/08, LMU München

Übersicht

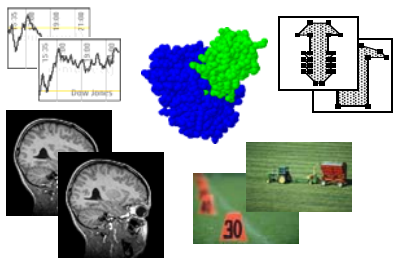
8.1 Feature-Basierte Ähnlichkeit

8.1.1 Das Prinzip der feature-basierten Ähnlichkeit

– Grundidee der Feature-Transformation

- Erwünscht: effiziente Ähnlichkeitssuche in Datenbanken
- Meist ist Effizienz ohne Einsatz von Indexstrukturen nicht zu verwirklichen
- Prinzipien:
  - Entwickle nicht für jedes der einzelnen Anwendungsgebiete/ Ähnlichkeitsmaße spezielle Indexstrukturen
  - Versuche mit wenigen Arten von Indexstrukturen möglichst viele Anwendungsgebiete abzudecken
- Indexstrukturen für:
  - Multidimensionale Vektoren
  - Allgemeine metrische Daten (beliebige Objekte, auf denen eine metrische Distanzfunktion definiert ist)

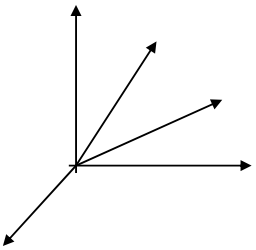
- Extrahiere charakteristische (numerische) Eigenschaften („Features“) aus den Objekten



Anwendungsgebiete (Objektraum)

Feature-Transformation

Histogramm-Bildung
Moment-Invarianten
Rechteck-Abdeckung
Sektoren-Ermittlung
Fourier-Transformation Kurvatur
usw.



Feature-Vektor-Raum

- Wichtigste Eigenschaft der Feature-Transformation:
  - Ähnlichkeit der Objekte entspricht geringem Abstand der Feature-Vektoren

  => Ähnlichkeitsanfragen im Objektraum entsprechen Nachbarschaftsanfragen im Feature-Raum

  => Unterstützung durch geeignete multidimensionale Indexstrukturen

8.1.2 Feature-Räume und Distanzen

– Allgemeiner Feature-Raum

Ein Feature-Raum ist ein Tupel Φ = (Dom, dist) mit

- Dom ist ein Wertebereich (Domain)
- dist ist eine Distanzfunktion, d.h. es gilt
  - Reflexivität $\forall x,y \in Dom: dist(x,y) = 0 \Leftrightarrow x=y$
  - Positiv-Definitheit $\forall x,y \in Dom, x \neq y: dist(x,y) > 0$
  - Symmetrie $\forall x,y \in Dom: dist(x,y) = dist(y,x)$

– Metrischer Raum

Ein metrischer Raum ist ein Tupel Φ_M = (Dom, dist) mit

- (Dom,dist) ist ein allgemeiner Feature-Raum
- dist erfüllt zusätzlich die
  - Dreiecksungleichung $\forall x,y,z \in Dom: dist(x,z) \leq dist(x,y) + dist(x,z)$

– Vektorraum

Ein (euklidischer) Vektorraum der Dimension *d* (*d*-dimensionaler Vektorraum) ist ein Tupel Φ_E = (Dom, dist) mit

- (Dom,dist) ist ein metrischer Raum
- Dom = $\mathbb{R}^d$

– Feature-Transformation

Eine Feature-Transformation ist eine Abbildung

T: OBJ $\rightarrow$ (Dom,dist)

die jedem Objekt $o \in$ OBJ aus dem Objektraum ein Objekt aus dem Wertebereich Dom zuordnet.

Die Distanz im Objektraum wird durch die Distanz im Feature-Raum repräsentiert, d.h.

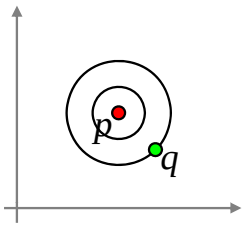
$\forall x,y \in OBJ: dist_{OBJ}(x,y) \equiv dist_{Dom}(T(x),T(y))$

– Distanzmaße in Vektorräumen

- Euklidische Norm (L_2):

  $dist = ((p_1-q_1)^2+(p_2-q_2)^2+...)^{1/2}$

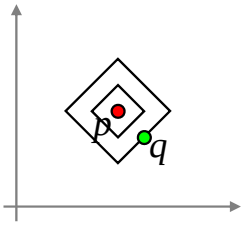
  Natürliches Distanzmaß



- Manhattan-Norm (L_1):

  $dist = |p_1-q_1|+|p_2-q_2|+...$

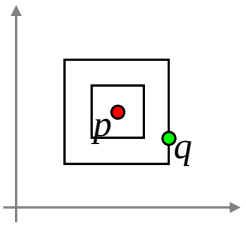
  Die Unähnlichkeiten der einzelnen Merkmale werden direkt addiert.



- Maximums-Norm (L_∞):

  $dist = \max\{|p_1-q_1|, |p_2-q_2|,...\}$

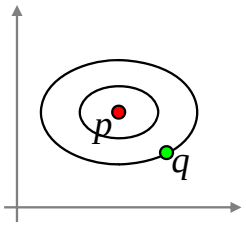
  Die Unähnlichkeit des am wenigsten ähnlichen Merkmals zählt.



- Verallgemeinerung L_p-Abstand: $dist_p = (|p_1-q_1|^p + |p_2-q_2|^p + ...)^{1/p}$
- Gewichtete Euklidische Norm:

  $dist = (w_1(p_1-q_1)^2+w_2(p_2-q_2)^2+...)^{1/2}$

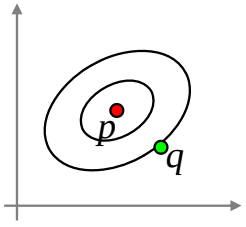
  Häufig sind die Wertebereiche der Merkmale deutlich unterschiedlich.
  Beispiel: Merkmal $M_1 \in [0.01 \ldots 0.05]$
  Merkmal $M_2 \in [3.07 \ldots 22.2]$
  Damit M1 überhaupt berücksichtigt wird, muss es höher gewichtet werden.



- Quadratische Form:

  $dist = ((p - q)\ \mathbf{M}\ (p - q)^{\mathbf{T}})^{1/2}$

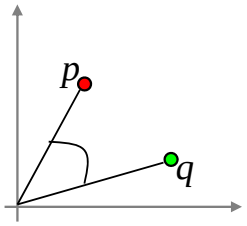
  Bisherige Abstandsmaße gewichten Merkmale nur getrennt.
  Besonders bei Farbhistogrammen müssen verschiedene Merkmale gemeinsam gewichtet werden.



- Cosinus-Distanz

  dist = cos(winkel(p,q))

  Berechnet den Cosinus des Winkels. Meist für sehr hochdimensionale Feature-Vektoren (z.B. Texte).



– Bemerkungen

- Jeder Vektorraum ist ein metrischer Raum. Jeder metrische Raum ein allgemeiner Feature-Raum.
- Sprechweise meist: „Feature-Raum“ statt (euklidischer) Vektorraum.
- Transformation der komplexen Objekte meistens in metrische Räume wegen der Dreiecksungleichung (Performanz!!!).

8.2 Algorithmische Paradigmen zur Anfragebearbeitung

8.2.1 Übersicht

- Typen von Ähnlichkeitsanfragen
  - Bereichsanfragen
  - Nächste-Nachbarn-Anfragen
- Algorithmische Paradigmen
  - Naive (sequentielle) Suche
    - Für alle *n* Objekte der Datenbank wird das Anfrage-Prädikat (d.h. meist eine Distanzberechnung zum Anfrageobjekt) ausgewertet
    - Kosten: O(*n* · Kosten für die Auswertung des Anfrage-Prädikats)
  - Indexbasierte Suche
  - Mehrstufige Anfragebearbeitung

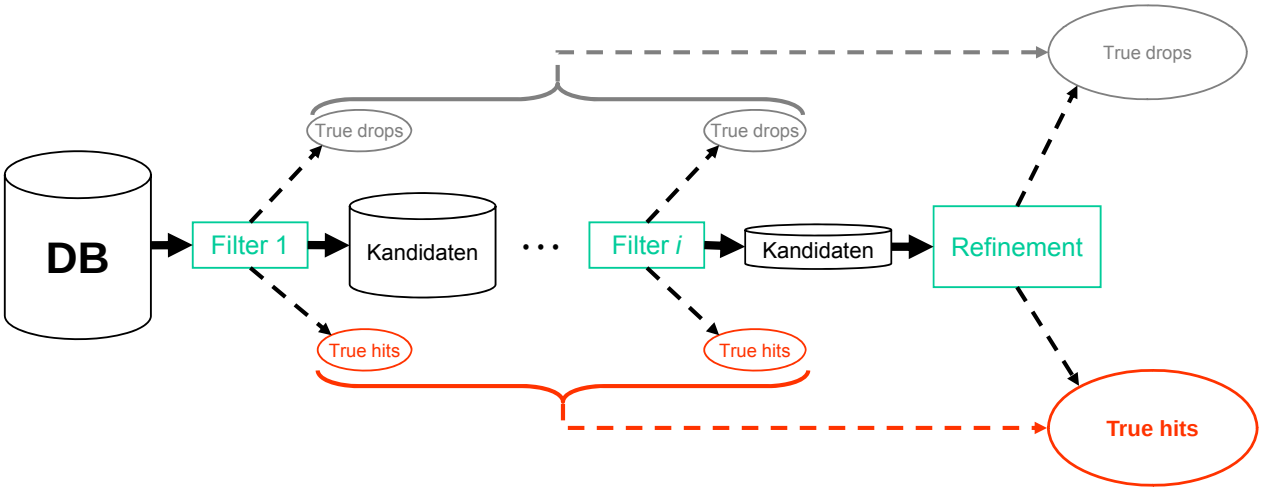
8.2.3 Mehrstufige Anfragebearbeitung

- Idee:
  - Verwendetes Distanzmaß ist sehr teuer (z.B. Edit-Distanz) oder nicht feature-basiert (z.B. Überlappungsfläche von Polygonen)
  - Vektorraum ist sehr hochdimensional (Curse of Dimensionality)
  - Benutze ein feature-basiertes (meist niedrig-dimensionales) Distanzmaß als Filterschritt (Filterdistanz)
    - Filterdistanz sollte billiger sein als exakte Distanz (entsprechend niedrig-dimensional => Dimensionsreduktion?)
    - Werte Anfrage-Prädikat (Distanzberechnung) mit Filterdistanz aus
    - Ergebnisse sind noch keine exakten Treffer sondern Kandidaten
    - Kandidatenmenge sollte möglichst klein sein (Filterselektivität)
    - Filterselektivität

$$\sigma_F = \frac{\#\text{Kandidaten}}{n}$$

    - Verfeinerung: für die Kandidaten wird das exakte Distanzmaß berechnet, was i.A. teurer, dafür selektiver als der Filterschritt ist

– Mehrstufige Anfragebearbeitung:
  - Ein oder mehrere (kaskadierende) Filterschritte schränken die Kandidatenmenge sukzessive ein
  - Verfeinerungsschritt testet auf Korrektheit der Kandidaten



– Zusammenpassen von Filter und Refinement
  - Idealfall: Filterdistanz ist obere oder untere Schranke (upper/ lower bound) der exakten Distanz => es kann garantiert werden, dass keine exakten Treffer verloren gehen (no false dismissals/drops)
  - Sonst: Ergebnisse u.U. nicht vollständig!!!
  - Lower Bounding Filter F_{LB}

$$\forall x, y \in DB : dist_{F_{LB}}(F_{LB}(x), F_{LB}(y)) \leq dist(x, y)$$

    – Konservative Approximation (d.h. Obermenge) der Ergebnismenge
    – Objekte können evtl. bereits aufgrund der Filterdistanz als exakte Fehltreffer (true drops) identifiziert werden
  - Upper Bounding Filter F_{UB}

$$\forall x, y \in DB : dist_{F_{UB}}(F_{UB}(x), F_{UB}(y)) \geq dist(x, y)$$

    – Progressive Approximation (d.h. Untermenge) der Ergebnismenge
    – Objekte können evtl. bereits aufgrund der Filterdistanz als exakte Treffer (true hits) identifiziert werden

8.3 Bereichsanfragen

– Allgemeines
  - Eigenschaften
    – Benutzer gibt Anfrageobjekt *q* und maximale Distanz ε vor
    – Ergebnis enthält alle Objekte, die höchstens eine Distanz von ε zu *q* haben
  - Formal

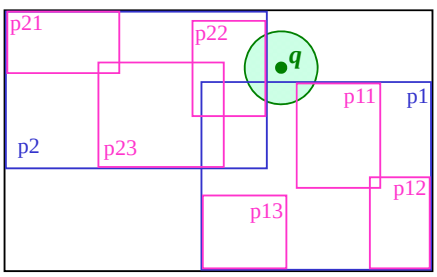
$$RQ(q,\varepsilon) = \{o \in DB \mid dist(q,o) \leq \varepsilon\}$$

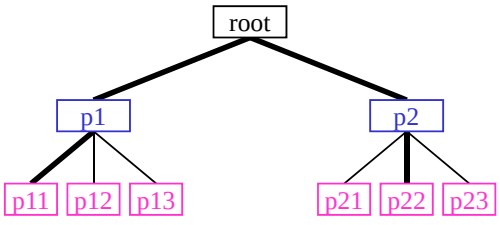
– Basisalgorithmus (sequential scan)

```
RQ-SeqScan(DB, q, ε)
    result = ∅;
    FOR i=1 TO n DO
        IF dist(q, DB.getObject(i)) ≤ ε THEN
            result := result ∪ getObject(i);
    RETURN result;
```


– Algorithmus mit Index: Tiefensuche

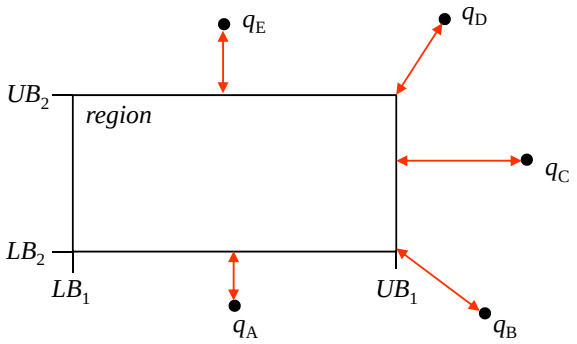
```
RQ-Index(pa, q, ε)          // pa = page address z.B. der Wurzel des Indexes
    result = ∅;
    p := pa.loadPage();
    IF p.isDataPage() THEN
        FOR i=0 TO p.size() DO
            IF dist(q, p.getObject(i)) ≤ ε THEN
                result := result ∪ getObject(i);

    ELSE                        // p ist Directoryseite
        FOR i=0 TO p.size() DO
            IF MINDIST(q, p.getRegion(i)) ≤ ε THEN
                result := result ∪ RQ-Index(p.childPage(i), q, ε)
    RETURN result;
```





- MINDIST
  - Test, ob Queryregion sich mit Seitenregion schneidet
  - Minimale Distanz zwischen Anfragepunkt und allen Punkten der Seitenregion (=> Lower Bound!!!)
  - Beispiel: Berechnung der MINDIST für L_2-Norm

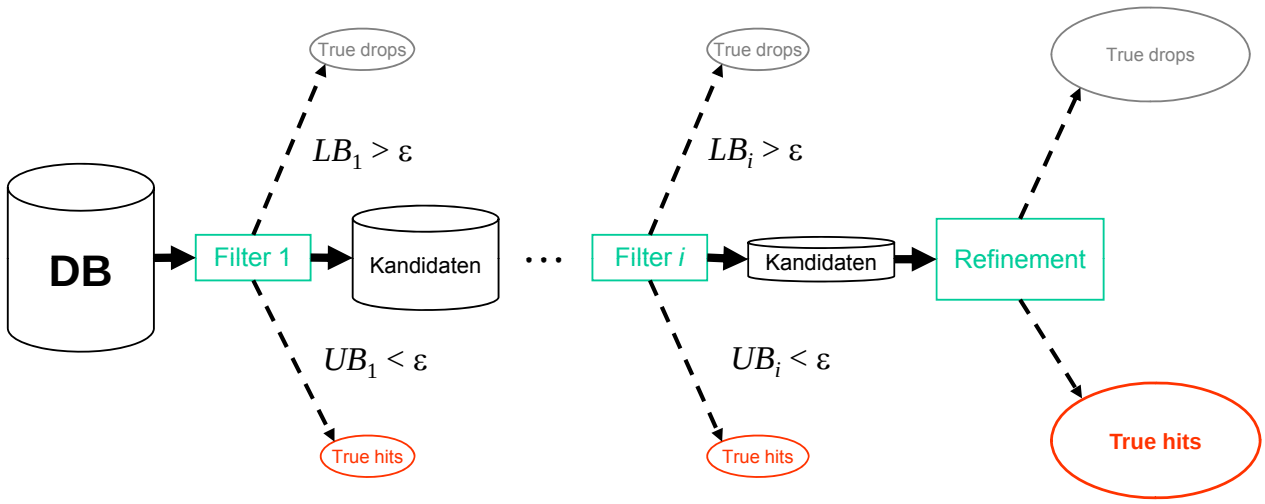
$$\text{MINDIST}(q, region) = \sqrt{\sum_{0<i\le d} \begin{cases} (region.LB_i - q_i)^2 & \text{if} \quad q_i \le region.LB_i \\ 0 & \text{if} \quad region.LB_i \le q_i \le region.UB_i \\ (q_i - region.UB_i)^2 & \text{if} \quad region.UB_i \le q_i \end{cases}}$$



– Mehrstufiger Algorithmus: Filter-/Refinement

- Lower Bounding Filterdistanz LB
- Upper Bounding Filterdistanz UB

```
RQ-MultiStep(DB, q, ε)
    result = ∅;
    candidates = ∅;

    // Filter
    FOR i=1 TO n DO
        IF UB(q, DB.getObject(i)) ≤ ε THEN
            result := result ∪ getObject(i);
        ELSE IF LB(q, DB.getObject(i)) ≤ ε THEN
            candidates := candidates ∪ getObject(i);

    // Refinement
    FOR i=1 TO candiates.size() DO
        IF dist(q, candidates.getObject(i)) ≤ ε THEN
            result := result ∪ candidates.getObject(i);
    RETURN result;
```



$LB_1 \leq LB_2 \leq \ldots \leq LB_i$

$UB_1 \geq UB_2 \geq \ldots \geq UB_i$

– Oft nur Lower Bounding Distanzen verwendet

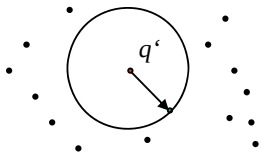
=> Anzahl der Kandidaten größer, da keine true hits

8.4 Nächste-Nachbarn-Anfragen

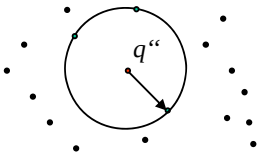
8.4.1 Nächste-Nachbarn-Anfrage

– Allgemeines
  - Eigenschaften
    – Benutzer gibt Anfrageobjekt *q* vor
    – Ergebnis enthält das Objekt, das die geringste Distanz zu *q* hat
    – Mehrdeutigkeiten müssen sinnvoll behandelt werden (mehrere nächste Nachbarn, oder nichtdeterministisch ein Objekt)
  - Formal

$$NN(q) = \{o \in DB \mid \forall x \in DB : dist(q,o) \leq dist(q,x)\}$$



Eindeutiges Ergebnis



Mehrdeutiges Ergebnis

– Basisalgorithmus (sequential scan): nichtdeterministisch

```
NN-SeqScan(DB, q)
    result = ∅;
    pruningdist = +∞;
    FOR i=1 TO n DO
        IF dist(q, DB.getObject(i)) ≤ pruningdist THEN
            result := getObject(i);
            pruningdist = dist(q, DB.getObject(i));
    RETURN result;
```

– Algorithmus mit Index: Einfache Tiefensuche
  - Unterschied zur Range-Query
    – Nächste-Nachbar kann beliebig weit vom Anfragepunkt weg liegen
    – Gestalt der Query zunächst unbekannt
    – Es kann zunächst nicht anhand der Seitenregion entschieden werden, ob eine Seite gebraucht wird
    – Ob eine Seite gebraucht wird, hängt auch von dem Inhalt der anderen Seiten ab

- Kennt man NN-Distanz, würde Range-Query ausreichen
- Kennt man ein beliebiges Objekt, kann man dessen Abstand als obere Schranke für die NN-Distanz nutzen
- Kennt man mehrere Objekte, kann man den geringsten Abstand als obere Schranke für die NN-Distanz nutzen

• Umformulierung des RQ-Algorithmus:
- Verwende als ε die kleinste Distanz zu den bisher gefundenen Nachbarn

```
Globale Variable: pruningdist = +∞;

NN-Index-Simple-TS(pa, q)          // pa = page address z.B. der Wurzel des Indexes
    result = ∅;
    p := pa.loadPage();
    IF p.isDataPage() THEN
        FOR i=0 TO p.size() DO
            IF dist(q, p.getObject(i)) ≤ pruningdist THEN
                result := getObject(i);
                stopdist = dist(q, p.getObject(i));
    ELSE                               // p ist Directoryseite
        FOR i=0 TO p.size() DO
            IF MINDIST(q, p.getRegion(i)) ≤ pruningdist THEN
                result := NN-Index-Simple-TS(p.childPage(i), q)
    RETURN result;
```


– Algorithmus mit Index: Prioritätssuche nach [HS 95]

[Hjaltason, Samet. Proc. Int. Symp. on Large Spatial Databases (SSD), 1995]

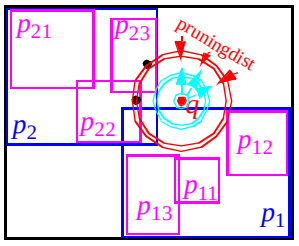
• Statt rekursivem Durchlauf: Liste der aktiven Seiten (active page list APL)
- Seite p ist aktiv genau dann, wenn folgende Bedingungen erfüllt sind:
  - » p wurde noch nicht geladen
  - » Elternseite von p wurde bereits geladen
  - » MINDIST(*q*, p.getRegion()) ≤ pruningdist
- APL wird mit Wurzel des Indexes initialisiert
- Seiten in APL nach MINDIST zum Anfrageobjekt aufsteigend sortiert
- Algorithmus entnimmt immer die erste Seite aus APL (mit kleinster MINDIST)
- Entnommene Seite wird geladen und verarbeitet: („verfeinert“)
  - » Datenseiten werden wie bisher verarbeitet
  - » Directoryseiten: Kindseiten mit MINDIST ≤ pruningdist in APL einfügen
- Ändert sich pruningdist werden Seiten mit MINDIST > pruningdist alternativ:
  - » aus APL entfernt
  - » als gelöscht markiert
  - » ohne explizite Markierung später ignoriert

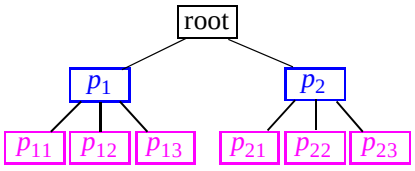
– Algorithmus:

Globale Variablen: pruningdist = +∞;

```
NN-Index-HS(pa, q)              // pa = page address z.B. der Wurzel des Indexes
    result = ∅;
    apl = LIST OF (dist:Real, da:DiskAdress) ORDERED BY dist ASCENDING
    apl = [(0.0, pa)]
    WHILE NOT apl.isEmpty() AND apl.getFirst().dist ≤ pruningdist DO
        p := apl.getFirst().da.loadPage();
        apl.deleteFirst();
        IF p.isDataPage() THEN

            (* wie bisher *)

        ELSE                    // p ist Directoryseite
            FOR i=0 TO p.size() DO
                IF MINDIST(q, p.getRegion(i)) ≤ pruningdist THEN
                    apl.insert(MINDIST(q, p.getRegion(i)), p.childPage(i));
    RETURN result;
```


- Beispiel





APL:

1. root
2. p_1 p_2
3. p_2 p_{12} p_{13} p_{11}
4. p_{23} p_{22} p_{12} p_{13} p_{11} p_{21}
5. p_{22} p_{12} p_{13}

- Eigenschaften
  - Allgemein
    - » Seiten werden nach aufsteigendem Abstand geordnet zugegriffen (blaue Kreise)
    - » pruningdist wird kleiner, sobald nähergelegenes Objekt gefunden (rote Kreise)
    - » Anfragebearbeitung stoppt, wenn beide Kreise sich treffen

- Speicherbedarf
  - » Wie bei Breitensuche kann gesamter unterster Directorylevel in APL stehen
  - » Dieser Fall ist allerdings unwahrscheinlicher als bei Breitensuche
  - » Speicherkomplexität O(n) (Tiefensuche O(log n)

- Optimalität des Verfahrens

  [Berchtold, Böhm, Keim, Kriegel. ACM Symp. Principles of Database Systems (PODS), 1997]

  - Prioritätssuche nach [HS 95] ist optimal bzgl. der Anzahl der Seitenzugriffe
  - Beweis (Überblick):
    - » Lemma 1: jeder korrekte Algorithmus muss mind. die Seiten laden, die von der NN-Kugel um q berührt werden
    - » Lemma 2: das Verfahren greift auf Seiten in aufsteigendem Abstand von q zu
    - » Lemma 3: keine Seite s wird zugegriffen, mit MINDIST(q, s) > NN-Distanz(q)

  - **Lemma 1:** Ein korrekter NN-Algorithmus muss mind. die Seiten s laden, die MINDIST(q, s) ≤ NN-Distanz(q) erfüllen.

    Beweis: Angenommen eine Seite s mit MINDIST(q, s) ≤ NN-Distanz(q) wird nicht geladen. Dann kann diese Seite Punkte enthalten (als Datenseite; Directoryseiten können im entspr. Teilbaum Punkte speichern), die näher am Anfragepunkt liegen als der nächste Nachbar. Der nächste Nachbar ist also nicht als solcher validiert, da über Punkte in einem Teilbaum keine Infos bekannt sind, außer dass sie in der entsprechenden Region liegen.

    □

  - **Lemma 2:** Das Verfahren greift auf die Seiten des Index aufsteigend sortiert nach MINDIST zu.

    Beweis: Die Seiten werden in aufsteigender Reihenfolge aus der APL entnommen. Es muss also nur sichergestellt werden, dass nach Entnahme von Seite s keine Seiten s‘ mehr in APL eingefügt werden, mit MINDIST(q,s‘) < r := MINDIST(q,s). Alle Seiten, die nach Entnahme von s in APL eingefügt werden, sind entweder Kindseiten von s oder Kindseiten von Seiten s‘‘ mit MINDIST(q,s‘‘) ≥ r. Da die Region einer Kindseite in der Region der Elternseite vollständig eingeschlossen ist, ist die MINDIST einer Kindseite nie kleiner als die der Elternseite. Daher haben alle später eingefügten Seiten eine MINDIST ≥ r.

    □

  - **Lemma 3:** Das Verfahren greift auf keine Seite s zu, mit MINDIST(q,s) > NN-Distanz(q).

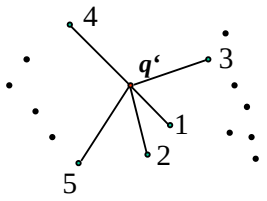
    Beweis: Nach Lemma 2 können nach Zugriff auf Seite s nur Punkte p gefunden werden, mit dist(q, p) > MINDIST(q, s). Wäre vor Zugriff auf s ein Punkt p mit dist(q, p) < MINDIST(q, s) gefunden worden, dann wäre s aus der APL gelöscht worden bzw. der Algorithmus hätte vor der Bearbeitung von p angehalten.

    □

  - Aus Lemma 1-3 ergibt sich, dass der Algorithmus nach [HS 95] optimal bzgl. der Anzahl der Seitenzugriffe ist.

8.4.3 Nächste-Nachbarn-Ranking

– Allgemeines
  - Eigenschaften
    – Benutzer gibt Anfrageobjekt *q* vor und initialisiert damit das Ranking
    – Benutzer kann mehrfach Funktion getNext() aufrufen, die ihm jeweils den 1., 2., usw. Nachbarn von *q* zurück gibt.
    – Mehrdeutigkeiten müssen wiederum sinnvoll behandelt werden
      » Typischerweise nicht-deterministisch: der *k*-te Aufruf ergibt einen der *k*-NN



– Algorithmus mit Index

[Hjaltason, Samet. Int. Symp. Large Spatial Databases (SSD), 1995]

- Alle *k*-NN-Algorithmen können entsprechend erweitert werden
- Problem der rekursiven Algorithmen
  – Nachdem der i-te Nachbar gefunden ist, wird das Ergebnis an die Ranking-Ausgabe übergeben
  – Weitere getNext()-Aufrufe erfordern erneutes rekursives Suchen
- Vorteil der Prioritätssuche
  – Kompletter Zustand des Algorithmus ist in apl und result gespeichert
- Unterschied zum *k*-NN-Algorithmus
  – Unbeschränkte Ergebnisliste *result* in die jeder Punkt einer geladenen Datenseite eingefügt wird (***aufsteigend*** nach Distanz zu *q* sortiert).
  – Keine Pruningdistanz => Kindseiten verfeinerter Seiten in APL einfügen
  – Algorithmus stoppt (für den aktuellen getNext()-Aufruf) sobald erste Seite in APL größere MINDIST zu *q* hat als bestes Element in *result*
  – Dieses Element wird aus result gelöscht und ausgegeben
  – Nächster getNext()-Aufruf arbeitet mit aktuellen APL und *result* weiter
- Hoher Speicherplatzbedarf: im worst case gesamte DB in *result*

- Algorithmus

```
Globale Variablen:
result = LIST OF (dist:Real, obj:Object) ORDERED BY dist ASCENDING;
apl = LIST OF (dist:Real, da:DiskAdress) ORDERED BY dist ASCENDING

NN-Ranking(pa, q)
    result = [(+∞, dummy)];
    apl = [(0.0, pa)];
    WHILE NOT apl.isEmpty() AND apl.getFirst().dist ≤ result.getFirst().dist DO
        p = apl.getFirst().da.loadPage();
        apl.deleteFirst();
        IF p.isDataPage() THEN
            FOR i=0 TO p.size() DO  // Jedes Objekt einfügen
                result.insert( (dist(q, p.getObject(i)),p.getObject(i)) );

        ELSE                                        // p ist Directoryseite
            FOR i=0 TO p.size() DO  // Jede Seite einfügen
                apl.insert( (MINDIST(q, p.getRegion(i)),p.getChildPage(i)) );
    resultObject = result.getFirst().obj;
    result.deleteFirst();
    RETURN resultObject;
```


LMU Database Systems