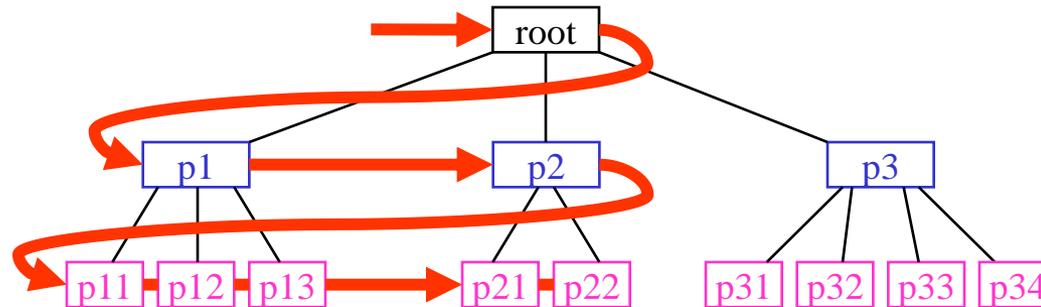


– Algorithmus mit Index: Breitensuche



- Abgesehen von MINMAXDIST-Abschätzung stehen Punktdistanzen erst auf Blatt-Ebene zur Verfügung
 - Viele Zugriffe auf Directoryseiten
 - Die erste Punktdistanz hätte viele dieser Zugriffe schon verhindern können
- Speicherintensiv
 - Worst-case: gesamte letzte Directory-Ebene muss im RAM gehalten werden

– 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
 - » $\text{MINDIST}(q, p.\text{getRegion}()) \leq \text{pruningdist}$
 - 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 $\text{MINDIST} \leq \text{pruningdist}$ in APL einfügen
 - Ändert sich pruningdist werden Seiten mit $\text{MINDIST} > \text{pruningdist}$ alternativ:
 - » aus APL entfernt
 - » als gelöscht markiert
 - » ohne explizite Markierung später ignoriert

– Algorithmus:

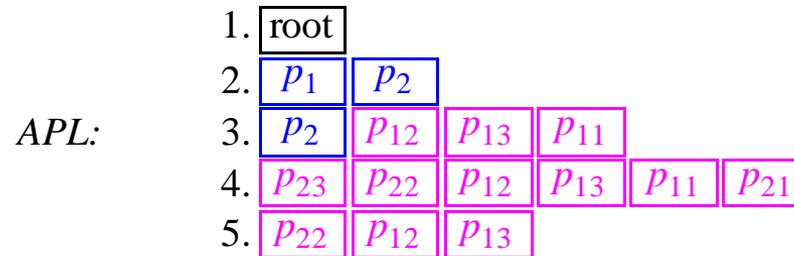
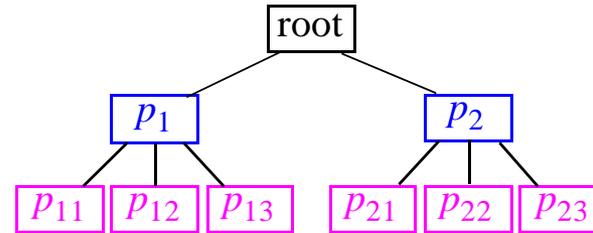
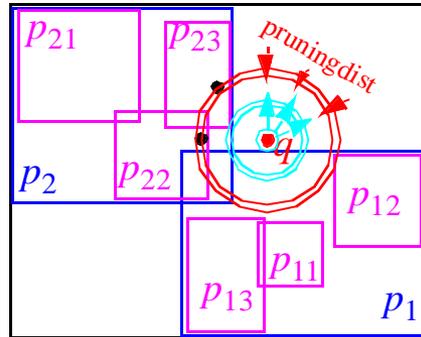
Globale Variablen: stopdist = $+\infty$; pruningdist = $+\infty$;

```
NN-Index-HS(pa, q)           // pa = Diskadress z.B. der Wurzel des Indexes
  result =  $\emptyset$ ;
  apl = LIST OF (dist:Real, da:DiskAdress) ORDERED BY dist ASCENDING
  apl = [(0.0, pa)]
  WHILE NOT apl.isEmpty() AND apl.first().dist  $\leq$  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))  $\leq$  pruningdist THEN
          apl.insert(MINDIST(q, p.getRegion(i)), p.childPage(i));
  RETURN result;
```

• Beispiel



• 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 unterste Directorylevel in APL stehen
 - » Dieser Fall is allerdings unwahrscheinlicher als bei Breitensuche
 - » Speicherkomplexität $O(n)$ (Tiefensuche $O(\log n)$)
- Optimalität des Verfahrens

[Berchtold, Böhm, Keim, Kriegel. ACM Smp. 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 $\text{MINDIST}(q, s) > \text{NN-Distanz}(q)$
 - **Lemma 1:** Ein korrekter NN-Algorithmus muss mind. die Seiten s laden, die $\text{MINDIST}(q, s) \leq \text{NN-Distanz}(q)$ erfüllen.

Beweis: Angenommen eine Seite s mit $\text{MINDIST}(q, s) \leq \text{NN-Distanz}(q)$ 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 $\text{MINDIST}(q, s') < r := \text{MINDIST}(q, s)$. Alle Seiten, die nach Entnahme von s in APL eingefügt werden, sind entweder Kindseiten von s oder Kindseiten von Seiten s'' mit $\text{MINDIST}(q, s'') \geq r$. 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 $\text{MINDIST} \geq r$.

□

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

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

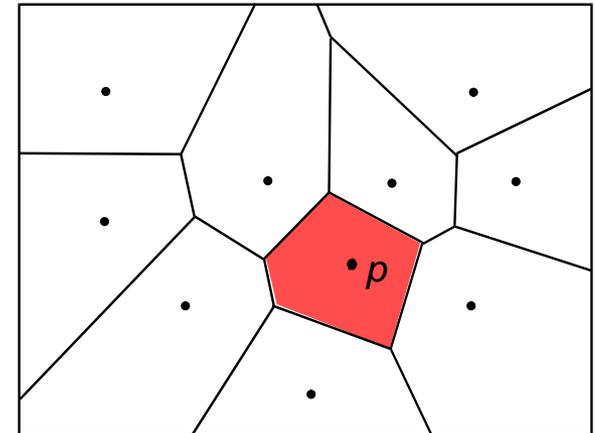
□

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

– Hybrider Algorithmus: Voronoi-Diagramme

[Berchtold, Ertl, Keim, Kriegel, Seidl. Proc. Int. Conf. Data Engineering (ICDE), 1998]

- Nur für Vektordaten!!!
- Idee:
 - Berechne für jeden Punkt p den Teil des Datenraumes in dem p der nächste Nachbar ist (Voronoi-Zellen)
 - Speichere Voronoi-Zellen in DB
 - NN-Anfrage entspricht Punktanfrage mit q auf den Voronoi-Zellen
=> Punkt p , in dessen Voronoi-Zelle q liegt, ist der NN von q
- Problem:
 - Voronoi-Zellen sind konvexe Polygone
 - Ab $d > 2$ sehr komplex (große Anzahl Eckpunkte)
- Lösung: Approximation der Voronoi-Zellen (z.B. mit MBR)
 - Nur Filterschritt, da MBRs sich überlappen können, und q in mehrere dieser MBRs liegen kann
=> Verfeinerung der Kandidaten mit exakten Punktdistanzen



– Mehrstufiger Algorithmus: Filter/Refinement

- Allgemeines

- Algorithmen verwenden meist nur LB-Filter
- Bei mehreren Filterschritten: $\text{dist}_{\text{Filter1}} \leq \text{dist}_{\text{Filter2}} \leq \dots$
- Unterschied zu Bereichsanfragen:
 - » RQs können durch einfache Hintereinanderschaltung der Filterschritte und der Verfeinerung sequentiell ausgewertet werden.
 - » Bei NN-Anfragen (NNQ) nicht so leicht möglich. Grund: Der (erste) Filter ist nicht in der Lage eine NN-Kandidatenmenge (als kleine Teilmenge der Datenbank) zu identifizieren, die garantiert den exakten NN enthält. (Unter Beschränkung auf den LB-Filter)
 - » Bei geeigneter Filterdistanz ist es aber wahrscheinlich, dass exakter NN unter den ersten NN-Kandidaten des Filterschritts ist.
 - » **Rückmeldung** der im Refinement ermittelten Distanzen an den Filterschritt um aufgrund des Filters Objekte auszuschließen.

Range Query



NN Query



- Im folgenden:
 - Verschiedene Auswertungsstrategien
 - Ein Filterschritt (leicht generalisierbar auf mehrere Filterschritte)
- Auswertung mit Bereichsanfrage
 - [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. Proc. Int. Conf. Very Large Databases (VLDB), 1996]
 - [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. TKDE 10(6), 1998]
 - Idee
 - » Verfeinerungsdistanz ε eines beliebigen Punktes ist obere Schranke für die NN-Distanz
 - » Folge: ist p der NN von q , so gilt $\text{dist}(p, q) \leq \varepsilon$ und $\text{LB}_{\text{Filter}}(p, q) \leq \varepsilon$
 - » Also: $p \in \text{RQ}(q, \varepsilon)$
 - » Gutes ε ist z.B. der NN von q bzgl. der Filterdistanz
 - Prinzip
 - » Auf Filterebene wird zunächst eine NN-Anfrage ausgeführt
 - » Das resultierende Objekt wird verfeinert
 - » Anschließend wird eine Bereichsanfrage (RQ) ausgeführt (mit Index oder ebenfalls mehrstufig)
 - » Auf dem (hoffentlich kleinen) Ergebnis der RQ wird der exakte Test (Refinement) durchgeführt

- Algorithmus

NN-MultiStep-RQ(DB, q)

r = NN-Query auf der Filterdistanz; // beliebig implementierbar

ε = $\text{dist}(q, r)$;

candidates = **RQ-MultiStep**(DB, q, ε);

result = r ;

stopdist = ε ;

// Refinement

FOR EACH $p \in \text{candidates}$ **DO**

IF $\text{dist}(p, q) \leq \text{stopdist}$ **THEN**

 stopdist = $\text{dist}(q, p)$

 result = p ;

RETURN result;

- Vorteil

- » Einfacher Algorithmus

- Nachteil

- » Leistung stark von Filterselektivität abhängig: schlechter Filter => großes ε => große Ergebnismenge der RQ => hohe Kosten für Verfeinerung

- Auswertung mit unmittelbarer Verfeinerung

- Idee

- » Jedes Objekt, das nicht aufgrund des Filters ausgeschlossen werden kann, wird sofort verfeinert
- » Einbau in einen beliebigen NN-Algorithmus, z.B. in NN-Index-Simple-TS (S. 61)

- Algorithmus

```

NN-MultiStep-Simple(pa, q)      // pa = Diskadress z.B. der Wurzel des Indexes
  result = ∅;
  p := pa.loadPage();
  IF p.isDataPage() THEN
    FOR i=0 TO p.size() DO
      IF distFilter(q, p.getObject(i)) ≤ stopdist THEN
        IF dist(q, p.getObject(i)) ≤ stopdist 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)) ≤ stopdist THEN
            result := NN-MultiStep-Simple(p.childPage(i), q)
    RETURN result;
  
```

- Vorteil
 - » Gute Speicherplatzkomplexität (je nach NN-Algorithmus!!!), da keine Kandidaten zwischen gespeichert werden müssen
 - » Einfache Erweiterung eines beliebigen NN-Algorithmus
 - Nachteil
 - » Hohe Verfeinerungskosten (fast alle Punkte), wenn Filter wenig selektiv ist oder NN-Algorithmus langsam konvergiert
-
- **Auswertung nach Priorität**
 - [Seidl, Kriegel. Proc. ACM Int. Conf. Management of Data (SIGMOD), 1998]
 - Auf Filterebene läuft „Ranking Query“ ab (siehe Kapitel 2.4.3)
 - » Funktion getNext(): liefert beim ersten Aufruf den 1. Nachbarn, beim zweiten Aufruf den 2. Nachbarn, usw.
 - » Rufe solange getNext() auf, bis das erhaltene Objekt die aktuelle stopdist überschreitet
 - » Verfeinere das erhaltene Objekt und passe ggf. die stopdist an
 - Vorteil
 - » Beweisbar: Algorithmus optimal bzgl. der Anzahl der Verfeinerungen, d.h. eine minimale Anzahl von Kandidaten werden verfeinert
 - Nachteil
 - » Komplexität des Ranking-Algorithmus (Speicher und/oder Zeit)

– Algorithmus

NN-MultiStep-Optimal(DB, q)

Ranking = initialisiere Ranking bzgl. q auf Filterdistanz // Kapitel 2.4.3

result = \emptyset ;

stopdist = $+\infty$;

REPEAT

p = Ranking.getNext();

filterdist = $\text{dist}_{\text{Filter}}(p, q)$;

IF $\text{dist}_{\text{Filter}}(p, q) \leq \text{stopdist}$ **THEN**

IF $\text{dist}(q, p) \leq \text{stopdist}$ **THEN**

result = p ;

stopdist = $\text{dist}(q, p)$;

UNTIL filterdist > stopdist;

RETURN result;