CC52V
Bases de datos multimedia

Prof. Benjamin Bustos

Capítulo 3
Algoritmos de búsqueda por similitud

3.1 Conceptos básicos

- En el capítulo anterior
  - Definición formal de similitud para distintas aplicaciones
  - Eficacia de la búsqueda por similitud
- A continuación
  - Métodos eficientes para encontrar los objetos similares a una consulta
  - Eficiencia de la búsqueda por similitud
    - Índices y algoritmos de búsqueda

2

3.1 Conceptos básicos

- Transformación en vectores característicos
  - Extracción de atributos numéricos de los objetos multimedia
  - Característica importante de la función de transformación
    - Similitud entre objetos corresponde a la cercanía entre puntos en el espacio vectorial
  - Se pasa desde un problema de búsqueda por similitud en el espacio original a buscar puntos cercanos en un espacio vectorial

3

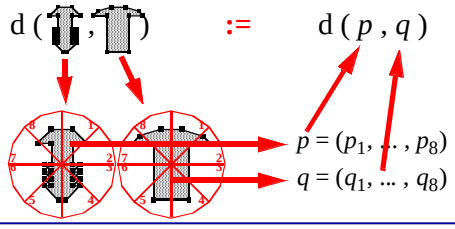
3.1 Conceptos básicos

- Transformación en vectores característicos
  - Distintas funciones de distancia
    - Manhattan, euclidiana, forma cuadrática, etc.
  - Búsquedas eficientes
    - *Índices multidimensionales*
- En caso que no haya una función de transformación que tenga sentido
  - Aprovechar las propiedades métricas de la función de distancia para indexar el espacio
    - *Índices métricos*

4

3.1 Conceptos básicos

- Método basado en atributos del objeto
  - La similitud entre objetos es directamente la distancia entre sus vectores característicos
  - No se necesita ningún refinamiento: Objetos recuperados desde el índice son el resultado final



5

3.1 Conceptos básicos

- Método basado en filtrar-y-refinar
  - La *verdadera* distancia entre objetos puede ser
    - Una métrica
    - Otra distancia basada en atributos
    - Una medida de similitud no métrica
  - Distancia verdadera (en general) *costosa de calcular*
  - Para responder una consulta se utiliza un *arquitectura multi-step*
    - Objetos recuperados desde el índice no son el resultado, sino sólo candidatos

6

3.1 Conceptos básicos

- Método basado en filtrar-y-refinar
  - El conjunto de candidatos debe ser pequeño con respecto al tamaño de la BD
    - *Selectividad del filtro*

$$\sigma_F = \frac{\#\text{ de candidatos}}{\#\text{ de objetos en la BD}}$$

    - Para los candidatos se mide la distancia verdadera con el objeto de consulta (es más cara, pero más selectiva)

3.1 Conceptos básicos

- Método basado en filtrar-y-refinar
  - Filtrado en cascada
    - Primer filtro determina candidatos con el índice
    - Filtros siguientes reducen el conjunto de candidatos
    - Paso de refinamiento determina rectitud de la respuesta
  - Caso ideal para filtrar-y-refinar
    - Distancia filtro es cota inferior (*lower bound*) de la distancia verdadera
      - Está garantizado que no se perderá ningún objeto relevante (no hay falsos negativos)

3.1 Conceptos básicos

- Método basado en filtrar-y-refinar
  - Si el filtro *no es cota inferior*
    - Pueden haber falsos negativos
      - No se garantiza que el resultado esté completo
    - Se puede mostrar experimentalmente que el *recall* es alto, i.e., que no se pierden “muchos” objetos relevantes

9

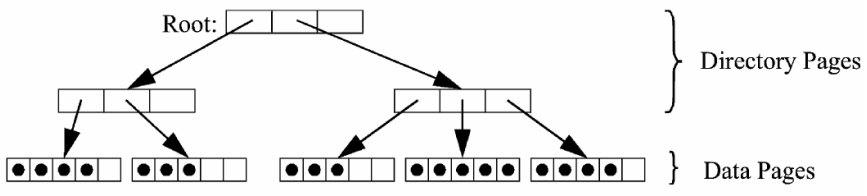
3.2 Índices multidimensionales

- Principios generales [BBK01]
  - En general son árboles balanceados
  - Cada nodo corresponde a
    - Una página de disco
    - Una región del espacio
  - Existen dos tipos de páginas (nodos)
    - Páginas de datos: corresponden a las hojas y contienen puntos del espacio
    - Páginas de directorio: son nodos internos y almacenan entradas de directorio
      - Punteros a los hijos
      - Describen la región espacial de los hijos

10

3.2 Índices multidimensionales

- Estructura básica



11

3.2 Índices multidimensionales

- Páginas lógicas vs. páginas físicas
  - Idea original: utilizar una página física del disco
    - Unidad de información más pequeña que se puede transferir desde el disco a memoria principal
  - Usualmente las páginas físicas son muy pequeñas o el tamaño es desconocido
    - Leer páginas consecutivas de disco es órdenes de magnitud más rápido que accesos aleatorios
    - *Paging artificial*: página lógica definida por el usuario (varias páginas físicas adyacentes)

12

3.2 Índices multidimensionales

- La mayoría de los índices utilizan un tamaño de pagina único, para evitar problemas como la administración del espacio libre
- Capacidad de páginas de datos y directorio

$$C_{data} = \left\lfloor \frac{|\text{página}| + |\text{overhead}|}{|\text{registro}|} \right\rfloor \qquad C_{dir} = \left\lfloor \frac{|\text{página}| + |\text{overhead}|}{|\text{entrada directorio}|} \right\rfloor$$

- Las páginas no se utilizan un 100% para dejar espacio a nuevas entradas. Se define generalmente una utilización mínima (e.g., 40%)

13

3.2 Índices multidimensionales

- *Storage utilization (su)*: utilización promedio de una página (%)
- Capacidad efectiva de una página

$$C_{eff,Data} = su_{Data} \cdot C_{Data}$$

- Cada registro (punto en el espacio) se almacena en una sola página de datos (no hay duplicados)

14

3.2 Índices multidimensionales

- Región espacial: garantizan que puntos cercanos se almacenarán en lo posible en la misma página de datos o subárbol
- Diferentes opciones para forma de la región
  - Hiperesfera
  - Hipercubo
  - Cuboide multidimensional
  - Cilindro multidimensional
  - Combinación de los anteriores

15

3.2 Índices multidimensionales

- Regiones de páginas jerárquicamente organizadas siempre deben estar contenidas completamente en la región de su padre
  - Condición para la rectitud del resultado (aproximación conservativa, e.g., propiedad de cota inferior)
- Índices multidimensionales son dinámicos
  - Operaciones de inserción y borrado eficientes ($O(\log n)$)

16

3.2 Índices multidimensionales

- Procedimiento típico de inserción
  - Buscar una página de datos adecuada para el objeto a insertar (se inserta donde debería encontrarse)
  - Insertar el objeto
  - Si se excede la capacidad de la página (*overflow*), dividir la página en dos (*split*)
  - Modificar las representación de la región en el nodo padre
  - Si el número de subárboles almacenados excede la capacidad de una página de directorio, dividir el nodo padre. Proceder recursivamente hacia arriba en el árbol
  - Si la raíz se divide, el árbol crece en un nivel

17

3.3 Ejemplo: R-tree

- R-tree
  - Antonin Guttman, SIGMOD 1984 [Gut84]
- Estructura de datos R-tree
  - Árbol balanceado similar a un B-tree
  - Estructura dinámica: inserciones, borrados, etc.
  - Permite buscar objetos espaciales en una área determinada

18

3.3.1 Estructura del R-tree

- Base de datos espacial
  - Tuplas: representan objetos espaciales
  - Cada tupla posee un identificador único
- Hojas del R-tree poseen registros de la forma

$$(I,\ \text{identificador–tupla})$$

  - I es un rectángulo n-dimensional

$$I = (I_0, I_1, \ldots, I_{n-1})$$

    - *n*: número de dimensiones
    - I_x: intervalo [*a*,*b*] que describe extensión del objeto a lo largo de la dimensión *x*

19

3.3.1 Estructura del R-tree

- Nodos internos del R-tree contienen entradas de la forma

$$(I, \text{puntero–a–hijo})$$

  - *I* cubre todos los rectángulos del nodo hijo
- Parámetros
  - *M*: máximo número de entradas en un nodo
  - *m*: mínimo número de entradas en un nodo

$$\text{Importante: } m \leq \frac{M}{2}$$

20

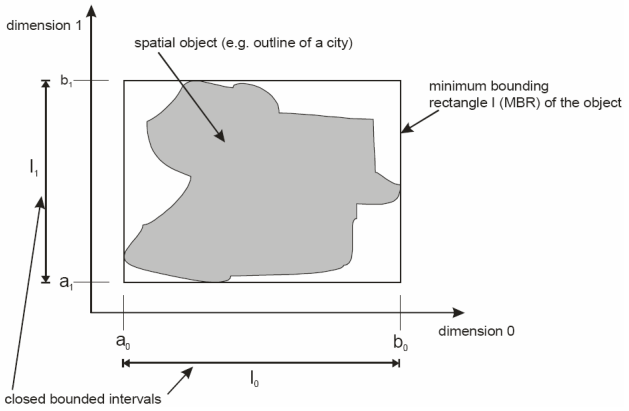
3.3.1 Estructura del R-tree

- Propiedades de un R-tree
  - Todo nodo externo (hojas) contiene entre *m* y *M* registros, excepto si es la raíz
  - Para cada registro en una hoja, *I* es el menor rectángulo que contiene al objeto *n*-dimensional
    - Minimum Bounding Rectangle (MBR)
  - Cada nodo interno contiene entre *m* y *M* hijos, excepto si es la raíz
  - Para cada entrada de un nodo interno, *I* es el MBR que contiene espacialmente los rectángulos en el nodo hijo
  - La raíz contiene a lo menos dos hijos, excepto si es una hoja
  - Todas las hojas están al mismo nivel

21

3.3.1 Estructura del R-tree

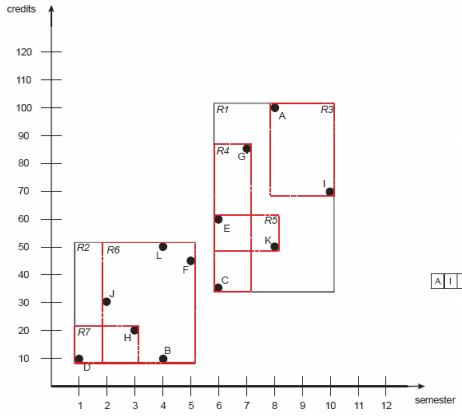
- Ejemplo de objeto espacial



22

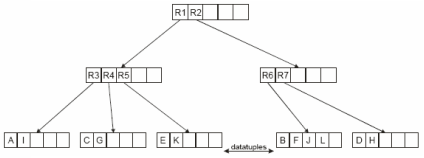
3.3.1 Estructura del R-tree

- Ejemplo de R-tree



Parámetros:
m=2
M=5

Estructura del R-tree



3.3.2 Búsqueda en el R-tree

- Búsqueda (range queries)
  - Algoritmo recorre árbol desde la raíz
  - Puede ser necesario visitar más de un subárbol
    - No se puede garantizar "buen" peor caso

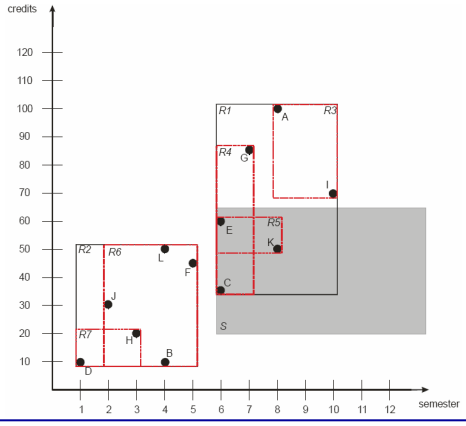
Algorithm **Search.** Given an R-tree whose root node is T, find all index records whose rectangles overlap a search rectangle S

S1 [Search subtrees] If T is not a leaf, check each entry E to determine whether EI overlaps S. For all overlapping entries, invoke **Search** on the tree whose root node is pointed to by Ep

S2 [Search leaf node] If T is a leaf, check all entries E to determine whether EI overlaps S. If so, E is a qualifying record

3.3.2 Búsqueda en el R-tree

- Ejemplo de range query



25

3.3.3 Inserción en el R-tree

- Inserción
  - Similar a insertar en un B-tree
  - Caso especial: overflow

SplitNode lo analizaremos en algunas slides mas…

Algorithm **Insert** Insert a new index entry E into an R-tree

I1 [Find position for new record] Invoke **ChooseLeaf** to select a leaf node L in which to place E

I2 [Add record to leaf node] If L has room for another entry, install E Otherwise invoke **SplitNode** to obtain L and LL containing E and all the old entries of L

I3 [Propagate changes upward] Invoke **AdjustTree** on L, also passing LL if a split was performed

I4. [Grow tree taller] If node split propagation caused the root to split, create a new root whose children are the two resulting nodes

26

3.3.3 Inserción en el R-tree

- Inserción

Algorithm **ChooseLeaf** Select a leaf node in which to place a new index entry E

CL1 [Initialize] Set N to be the root node

CL2 [Leaf check] If N is a leaf, return N.

CL3. [Choose subtree] If N is not a leaf, let F be the entry in N whose rectangle $F.I$ needs least enlargement to include $E.I$ Resolve ties by choosing the entry with the rectangle of smallest area

CL4. [Descend until a leaf is reached.] Set N to be the child node pointed to by $F.p$ and repeat from CL2

Algorithm **AdjustTree** Ascend from a leaf node L to the root, adjusting covering rectangles and propagating node splits as necessary

AT1 [Initialize.] Set $N=L$ If L was split previously, set NN to be the resulting second node

AT2 [Check if done] If N is the root, stop

AT3 [Adjust covering rectangle in parent entry] Let P be the parent node of N, and let E_N be N's entry in P Adjust $E_N.I$ so that it tightly encloses all entry rectangles in N.

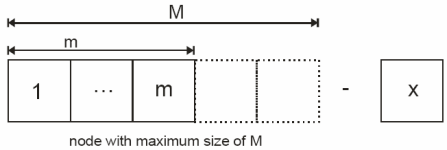
AT4 [Propagate node split upward] If N has a partner NN resulting from an earlier split, create a new entry E_{NN} with $E_{NN}.p$ pointing to NN and $E_{NN}.I$ enclosing all rectangles in NN Add E_{NN} to P if there is room. Otherwise, invoke **SplitNode** to produce P and PP containing E_{NN} and all P's old entries

AT5 [Move up to next level.] Set $N=P$ and set $NN=PP$ if a split occurred. Repeat from AT2.

27

3.3.4 Borrado y update en el R-tree

- Borrado
  - Buscar registro a borrar en el árbol y eliminar registro
  - Si se produce underflow, eliminar nodo y reinsertar todos los registros/entradas que quedaban

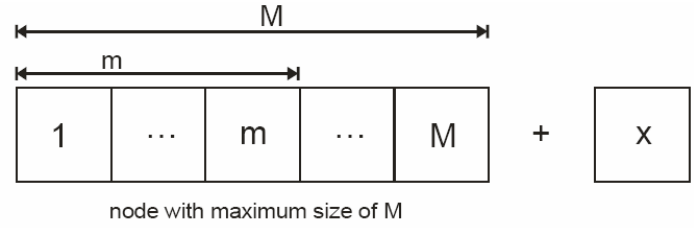


  - Si no se produjo underflow, ajustar MBR respectivo
- Update
  - Borrar registro e insertar nuevo registro

28

3.3.5 Node splitting

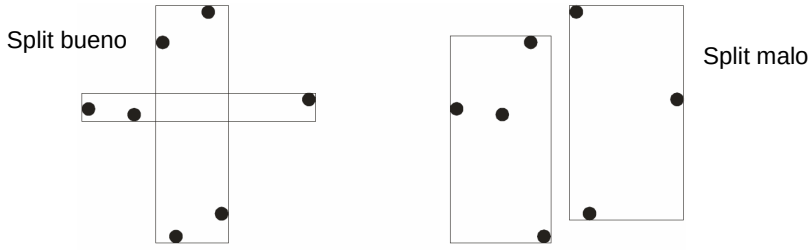
- División de nodos (node splitting)
  - Debe realizarse en caso de overflow



29

3.3.5 Node splitting

- División de nodos (node splitting)
  - Split debe ser realizado de forma que evite tener que visitar ambos nodos en búsquedas futuras
  - Criterio: minimizar área total de los MBRs



30

3.3.5 Node splitting

- División de nodos (node splitting)
  - Algoritmo exhaustivo
    - Encuentra solución óptima
    - Muy lento: $O(2^{M-1})$
  - Algoritmo cuadrático
    - No garantiza encontrar solución óptima
    - Costo cuadrático en *M*, lineal en *n*

3.3.5 Node splitting

- Algoritmo cuadrático

Algorithm **Quadratic Split** Divide a set of $M+1$ index entries into two groups

QS1 [Pick first entry for each group] Apply Algorithm **PickSeeds** to choose two entries to be the first elements of the groups Assign each to a group

QS2 [Check if done] If all entries have been assigned, stop If one group has so few entries that all the rest must be assigned to it in order for it to have the minimum number m, assign them and stop

QS3 [Select entry to assign] Invoke Algorithm **PickNext** to choose the next entry to assign Add it to the group whose covering rectangle will have to be enlarged least to accommodate it Resolve ties by adding the entry to the group with smaller area, then to the one with fewer entries, then to either Repeat from QS2

Algorithm **PickSeeds** Select two entries to be the first elements of the groups

PS1 [Calculate inefficiency of grouping entries together] For each pair of entries E_1 and E_2, compose a rectangle J including $E_1 I$ and $E_2 I$ Calculate $d =$ area(J) - area($E_1 I$) - area($E_2 I$)

PS2 [Choose the most wasteful pair] Choose the pair with the largest d

Algorithm **PickNext** Select one remaining entry for classification in a group.

PN1 [Determine cost of putting each entry in each group] For each entry E not yet in a group, calculate d_1 = the area increase required in the covering rectangle of Group 1 to include EI Calculate d_2 similarly for Group 2

PN2 [Find entry with greatest preference for one group] Choose any entry with the maximum difference between d_1 and d_2

3.3.5 Node splitting

- Ejemplo inserciones en R-tree

Registros (2D):

name	semester	credits
A	8	100
B	4	10
C	6	35
D	1	10
E	6	40
F	5	45
G	7	85
H	3	20
I	10	70
J	2	30
K	8	50
L	4	50

33

3.4 Algoritmos de búsqueda

- En esta sección se presentan algunos algoritmos básicos de búsqueda en índices multidimensionales
- Serán presentados en forma general
  - Sin restricciones a algún índice en particular
  - En el Capítulo 9 se estudiarán más índices multidimensionales específicos

34

3.4.1 Búsqueda exacta

- Características
  - Usuario ingresa objeto de consulta *q*
  - Sistema comprueba si existe algún objeto en la BD con la misma posición
- Definición formal

$$ex_q := \{o \in DB \,|\, o = q\}$$

- Versión no-determinística

$$ex_q := \text{ALGÚN } o \in DB \,|\, o = q$$

35

3.4.1 Búsqueda exacta

- Algoritmo básico sin índice (búsqueda secuencial)

$result = \phi$;
for $i := 1$ to n do
    if $q = DB[i]$ then
        $result := result \cup DB[i]$;

36

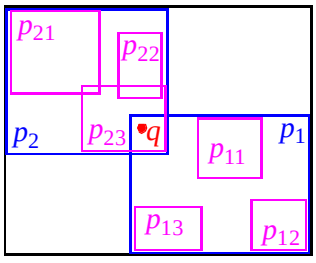
3.4.1 Búsqueda exacta

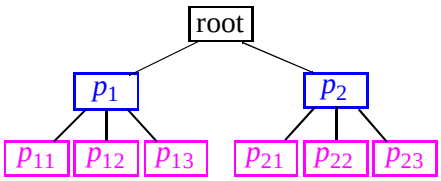
- Algoritmo básico con índice multidimensional (búsqueda en profundidad)

function ExactMatchQuery(q : Point, pa : DiskAddress) : Set of Point
$result = \phi$;
$p :=$ Loadpage(pa)
if IsDataPage(p) then
    for $i := 1$ to $p.num\_objects$ do
        if $q = p.object[i]$ then
            $result := result \cup p.object[i]$;
else
    for $i := 1$ to $p.num\_objects$ do
        if IsPointInRegion($q, p.object[i]$) then
            $result := result \cup$ ExactMatchQuery($q, p.childpage[i]$);
return $result$;

3.4.1 Búsqueda exacta

- Ejemplo

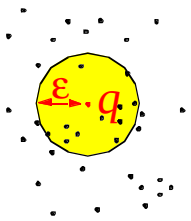




3.4.2 Consulta por rango

- Características
  - Usuario ingresa objeto de consulta *q* y radio de tolerancia ε
  - Sistema retorna todos los objetos que se encuentren a distancia menor o igual que ε de *q*
- Definición formal

$$sim_q(\varepsilon) := \{o \in DB \,|\delta(q,o) \leq \varepsilon\}$$



39

3.4.2 Consulta por rango

- Algoritmo básico sin índice

$result = \phi$;
for $i := 1$ to n do
    if $\delta(q, DB[i]) \leq \varepsilon$ then
        $result := result \cup DB[i]$;

40

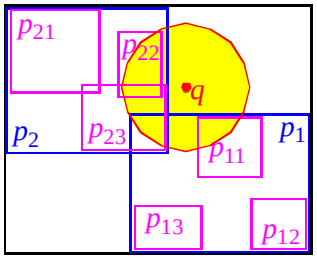
3.4.2 Consulta por rango

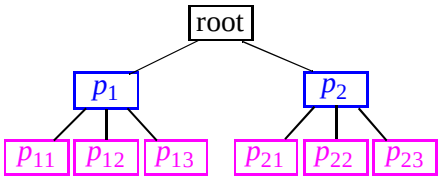
- Algoritmo básico con índice multidimensional (búsqueda recursiva en profundidad)

function RangeQuery(q : Point, ε : Real, pa : DiskAddress) : Set of Point
$result = \phi$;
$p :=$ Loadpage(pa)
if IsDataPage(p) then
    for $i := 1$ to $p.num\_objects$ do
        if $\delta(q, p.object[i]) \leq \varepsilon$ then
            $result := result \cup p.object[i]$;
else
    for $i := 1$ to $p.num\_objects$ do
        if MINDIST$(q, p.region[i]) \leq \varepsilon$ then
            $result :=$ $result \cup$ RangeQuery$(q, p.childpage[i])$;
return $result$;

3.4.2 Consulta por rango

- Ejemplo

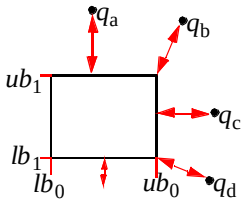




3.4.2 Consulta por rango

- Test de traslape entre la región definida por la consulta y la región de una página
  - Distancia entre consulta y región es la *mínima* distancia entre consulta y algún punto de la región (MINDIST). Para la distancia euclidiana:

$$MINDIST(p,q) = \sqrt{\sum_{1 \le i \le d} \begin{cases} (lb_i - q_i)^2 & \text{si} \quad q_i \le lb_i \\ 0 & \text{si} \quad lb_i \le q_i \le ub_i \\ (q_i - ub_i)^2 & \text{si} \quad ub_i \le q_i \end{cases}}$$



43

3.4.3 Consulta por vecino más cercano

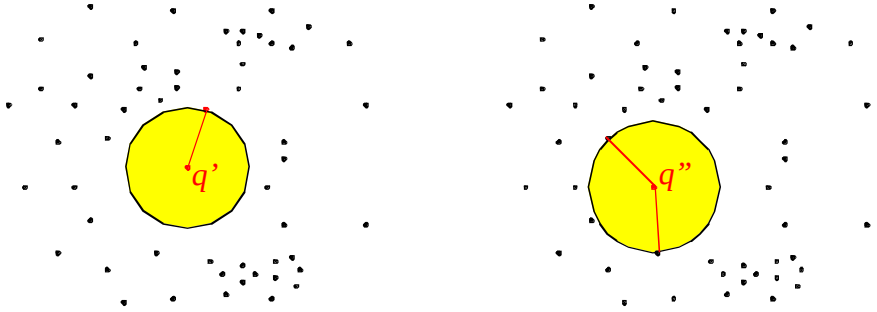
- Características
  - Usuario ingresa objeto de consulta *q*
  - Sistema retorna el objetos de la BD que se encuentren a menor distancia de *q*
- El resultado puede ser ambiguo
  - Existe más de una respuesta, o
  - El resultado es no-determinístico
- Definición formal

$$NN_q := \left\{ o \in DB \,\middle|\, \forall o' \in DB,\ \delta(o,q) \le \delta(o',q) \right\}$$

44

3.4.3 Consulta por vecino más cercano

- Ejemplo de respuesta ambigua



a) *q*' tiene un sólo NN

b) *q*'' tiene 2 NN

3.4.3 Consulta por vecino más cercano

- Variante no-determinística

$$NN_q := \text{SOME } o \in DB \,\big|\, \forall o' \in DB,\ \delta(o,q) \leq \delta(o',q)$$

- Algoritmo básico sin índice (no-determinístico)

```
result = φ;
resultdist = +∞;
for i := 1 to n do
    if δ(q, DB[i]) ≤ resultdist then
        result := DB[i];
        resultdist := δ(q, DB[i]);
```

3.4.3 Consulta por vecino más cercano

- Algoritmo simple (búsqueda en profundidad)
  - Diferencias con algoritmos previos
    - En un principio la región de consulta no es conocida; el vecino más cercano podría estar lejos
    - No se puede decidir a partir de una página dada si ésta debe ser visitada o no, también depende del contenido de las otras páginas

47

3.4.3 Consulta por vecino más cercano

- Algoritmo simple (búsqueda en profundidad)
  - Bastaría una consulta por rango si se conociera la distancia al vecino más cercano
    - Algún punto de la BD conocido: utilizarlo como cota superior de la distancia al vecino más cercano
    - Algunos puntos conocidos: utilizar la distancia más pequeña como cota superior
  - Adaptación del algoritmo de consulta por rango
    - ε se reemplaza por la distancia al mejor candidato hasta ese momento
    - Inicialmente no hay candidato y *resultdist* = +infinito

48

3.4.3 Consulta por vecino más cercano

- Algoritmo simple (búsqueda en profundidad)

```
procedure SimpleNNQuery(q : Point, pa : DiskAddress)
p := Loadpage(pa)
if IsDataPage(p) then
    for i := 1 to p.num_objects do
        if δ(q, p.object[i]) ≤ resultdist then
            result := p.object[i];
            resultdist := δ(q, p.object[i]);
else
    for i := 1 to p.num_objects do
        if MINDIST(q, p.region[i]) ≤ resultdist then
            SimpleNNQuery(q, p.childpage[i]);
```

49

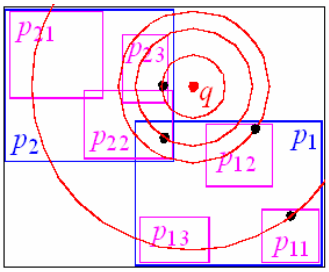
3.4.3 Consulta por vecino más cercano

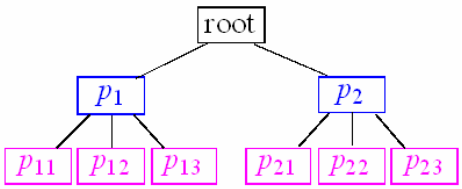
- Desventajas del algoritmo simple (búsqueda en profundidad)
  - Empieza con *resultdist*= +infinito
  - Por esto, empieza con cualquier camino, no con el camino que está más cerca de la consulta
  - Por esto, los primeros puntos que encuentre están (probablemente) lejos de la consulta
  - Esto implica que el método reduce el espacio de búsqueda lentamente. Muchos caminos deben visitarse inútilmente

50

3.4.3 Consulta por vecino más cercano

- Ejemplo





51

3.4.3 Consulta por vecino más cercano

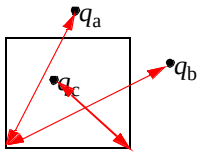
- Algoritmo de búsqueda en profundidad de Roussopoulos, Kelley y Vincent [RKV95]
  - Este algoritmo evita la lenta reducción del espacio de búsqueda a través de
    - Estimación de la distancia al NN a partir de las regiones
    - Priorizar la búsqueda en profundidad según la distancia de *q* a las regiones

52

3.4.3 Consulta por vecino más cercano

- Para estimar la distancia al NN se añadirán a MINDIST dos distancias a regiones
  - MAXDIST: máxima distancia entre q y algún punto en la región

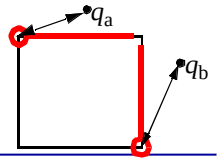
$$MAXDIST := \sqrt{\sum_{1 \le i \le d} \text{máx}\{(q_i - ub_i)^2, (q_i - lb_i)^2\}}$$



53

3.4.3 Consulta por vecino más cercano

- Para estimar la distancia al NN se añadirán a MINDIST dos distancias a regiones
  - MINMAXDIST: si el índice utiliza MBRs como regiones
    - Se puede utilizar la propiedad que en cada arista (para *d*>2 cada superficie (*d*-1)-dimensional) del rectángulo se encuentra al menos un punto (sino no sería minimal)
    - "Arista más cercana, punto más lejano"



54

3.4.3 Consulta por vecino más cercano

- Cálculo de MINMAXDIST

$$MINMAXDIST^2 := \min_{1 \le k \le d} \left\{ |q_k - rm_k| + \sum_{1 \le i \le d,\ i \ne k} |q_i - rM_i|^2 \right\}$$

donde

$$rm_k = \begin{cases} lb_k & \text{si } q_k \le \frac{lb_k + ub_k}{2} \\ ub_k & \text{si no} \end{cases}, \quad rM_i = \begin{cases} lb_i & \text{si } q_i \ge \frac{lb_i + ub_i}{2} \\ ub_i & \text{si no} \end{cases}$$

3.4.3 Consulta por vecino más cercano

- Para regiones con otra geometrías (e.g., círculos) o rectángulos generales (no minimum bounding)
  - MINDIST y MAXDIST se definen análogamente
  - MINMAXDIST no está definida
- Criterio de exclusión: mínimo entre
  - *resultdist*
  - MINMAXDIST de todas las regiones conocidas (MAXDIST para otras geometrías)

3.4.3 Consulta por vecino más cercano

- Para el descenso recursivo
  - Ordenar los hijos por prioridad
  - Experimentalmente se muestra que MINDIST es un buen criterio de prioridad
- La variable global *pruningdist* se inicializa como *resultdist* con + infinito

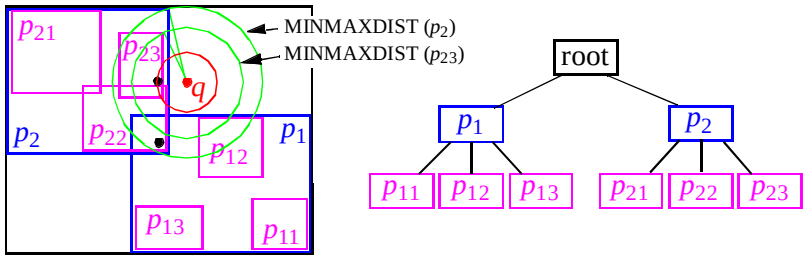
3.4.3 Consulta por vecino más cercano

- Algoritmo RKV para vecino más cercano

```
procedure RKV(q : Point, pa : DiskAddress)
p := Loadpage(pa)
if IsDataPage(p) then
    for i := 1 to p.num_objects do
        if δ(q, p.object[i]) ≤ resultdist then
            result := p.object[i];
            resultdist := δ(q, p.object[i]);
            if resultdist < pruningdist then
                pruningdist = resultdist;
else
    for i := 1 to p.num_objects do
        if MINMAXDIST(q, p.region[i]) < pruningdist then
            pruningdist = MINMAXDIST(q, p.region[i]);
    sort(p.object, MINDIST);
    for i := 1 to p.num_objects do
        if MINDIST(q, p.region[i]) ≤ pruningdist then
            RKV(q, p.childpage[i]);
```

3.4.3 Consulta por vecino más cercano

- Ejemplo

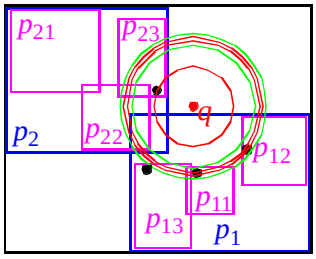


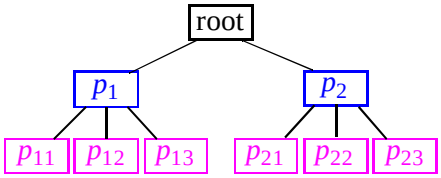
3.4.3 Consulta por vecino más cercano

- El criterio de exclusión mejorado con MINMAXDIST obtiene de vez en cuando la "distancia de poda"
- A pesar de ordenar por prioridad, podría suceder que la búsqueda en profundidad comienza mal
  - Una región en el primer nivel está muy cerca de la consulta
  - Sus hijos están relativamente lejos

3.4.3 Consulta por vecino más cercano

- Este es un problema fundamental con la búsqueda en profundidad





61

3.4.3 Consulta por vecino más cercano

- Búsqueda en amplitud
  - Además de MINMAXDIST, uno puede utilizar distancias a puntos cuando se llega al nivel de las hojas
  - La primera distancia a un punto podría evitar el acceso a muchas páginas de directorio
  - Búsqueda por amplitud: en el peor caso todas las páginas de directorio del último nivel deben ser almacenadas, si es que no se pueden excluir con el criterio de MINMAXDIST

62

3.4.3 Consulta por vecino más cercano

- Búsqueda por prioridad de Hjaltason y Samet [HS95]
  - En vez de un recorrido recursivo del índice, el algoritmo mantiene explícitamente una lista de las páginas activas (*Active Page List* o APL)
  - Definición: una página *p* es activa ssi
    - *p* no ha sido accesada
    - El padre de *p* fue accesado
  - La APL se inicializa con la raíz del índice

63

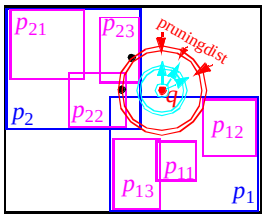
3.4.3 Consulta por vecino más cercano

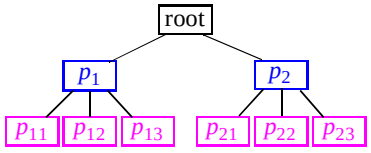
- Búsqueda por prioridad de Hjaltason y Samet
  - El algoritmo saca de la APL la página con mayor prioridad (i.e., con MINDIST mínimo)
  - Dicha página se procesa como sigue
    - Páginas de datos se procesan como siempre
    - Páginas de directorio: hijos con MINDIST menor o igual que *pruningdist* son insertados en la APL
  - Implementación de APL como *cola de prioridad*

64

3.4.3 Consulta por vecino más cercano

- Ejemplo





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}

65

3.4.3 Consulta por vecino más cercano

- Observaciones
  - Las páginas se acceden en orden creciente de distancia (círculos azules)
  - *pruningdist* (círculos rojos) decrece al encontrarse puntos más cercanos
  - El algoritmo se detiene cuando ambos círculos se encuentran

66

3.4.3 Consulta por vecino más cercano

- Algoritmo de Hjaltason y Samet

```
apl : list of (dist : Real, pa : DiskAddress) ordered by dist ascending =< (0,0, root) >
p := Loadpage(pa)
while NotEmpty(apl) and apl[0].dist ≤ resultdist do
    p = LoadPage(apl[0].pa);
    delete_first(apl);
    if (IsDataPage(p)) then
        (* como siempre *)
    else
        for i := 1 to p.num_objects do
            h = MINDIST(q, p.region[i]);
            if h ≤ resultdist then
                insert((h, p.childpage[i]), apl);
```


3.4.3 Consulta por vecino más cercano

- Requerimientos de espacio
  - Al igual que en la búsqueda por amplitud, puede suceder que todas las páginas del último nivel del directorio se inserten en la APL
  - La probabilidad que ocurra esto es baja, al contrario que en la búsqueda por amplitud
  - La complejidad en espacio (peor caso) es *O*(*n*), mucho peor que la búsqueda en profundidad (*O*(log *n*))

3.4.3 Consulta por vecino más cercano

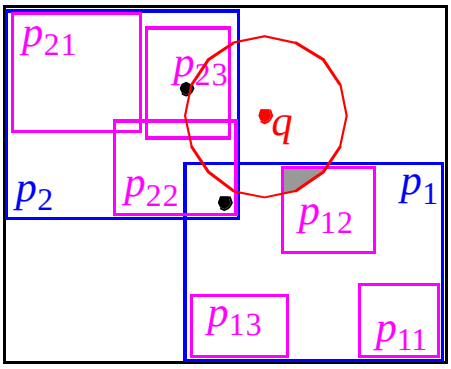
- Optimalidad del algoritmo de Hjaltason y Samet [BBK01]
  - El algoritmo es óptimo con respecto al número de páginas accesadas para un índice dado
  - Demostración en tres partes
    - Lema 1: acceso por lo menos a todas las páginas en la región definida por la consulta y su NN
    - Lema 2: acceso a las páginas en orden de distancia creciente a la consulta
    - Lema 3: ningún acceso a una página con MINDIST mayor que la distancia al NN
    - Corolario: el algoritmo accesa un número óptimo de páginas

3.4.3 Consulta por vecino más cercano

- Lema 1: un algoritmo correcto de búsqueda del NN debe accesar al menos aquellas páginas que intersectan la esfera definida por la consulta y la distancia al NN
  - Demostración:
    - Supongamos que una página cuyo MINDIST es menor que la distancia al NN no fue accesada
    - Esta página podría contener puntos que estén más cercanos que el NN
    - Por lo tanto, el NN aún no esta validado. La ubicación de los puntos en un subárbol no es conocida, sólo que se encuentran en dicha región espacial

3.4.3 Consulta por vecino más cercano

- Ejemplo:



71

3.4.3 Consulta por vecino más cercano

- Lema 2: El algoritmo accesa las páginas en orden de distancias crecientes a la consulta
  - Demostración:
    - Las páginas son sacadas por orden de distancia (MINDIST) de la APL
    - La MINDIST entre la consulta y una región es siempre mayor que o igual que la distancia entre la consulta y el padre de dicha región
    - Por lo tanto, la distancia entre la consulta y cualquier página en la APL sólo puede crecer o mantenerse
    - Como en la APL se accesa siempre la página activa con mínima distancia, éstas deben ser accesadas en orden de distancia ascendente a la consulta

72

3.4.3 Consulta por vecino más cercano

- Lema 3: El algoritmo no accesa ninguna página cuya distancia a la consulta sea mayor que la distancia al NN
  - Demostración:
    - Al procesar cualquier página *p* en la APL, sólo se pueden encontrar puntos cuya distancia a la consulta es mayor que *m*=MINDIST(*p*,*q*) (por Lema 2)
    - La distancia entre el candidato a NN y *q* no puede ser menor que *m* durante la búsqueda
    - Por lo tanto, aquellas páginas cuyo MINDIST es menor o igual que la distancia al NN serán accesadas. Por Lema 1, un algoritmo correcto debe accesar todas esas páginas

73

3.4.4 Consulta por *k* vecinos más cercano

- Características
  - Usuario ingresa objeto de consulta *q* y un valor *k*
  - Sistema retorna los *k* objetos de la BD que se encuentren a menor distancia de *q*
- Definición formal

$$NN_q(k) = \left\{ o \,\middle|\, \forall o' \in DB - NN_q,\ \delta(q,o) < \delta(q,o') \right\}$$

- Versión no-determinística

$$NN_q(k) = \left\{ o \,\middle|\, \forall o' \in DB - NN_q,\ \delta(q,o) < \delta(q,o') \right\} \wedge |NN_q(k)| = k$$

74

3.4.4 Consulta por *k* vecinos más cercano

- Algoritmo básico sin índice (no-determinístico)

```
result : list of (dist : Real, P : Point) ordered by dist descending := ();
for i := 1 to k do
    insert((δ(q, DB[i]), DB[i]), result);
for i := k + 1 to n do
    if δ(q, DB[i]) ≤ result[0].dist then
        delete_first(result);
        insert((δ(q, DB[i]), DB[i]), result);
```

  - Observación: la lista *result* está ordenada ascendentemente por distancia
    - Cola de prioridad

3.4.4 Consulta por *k* vecinos más cercano

- Algoritmo con índice
  - Adaptación del algoritmo de Hjaltason y Samet
    - 2 colas de prioridad:
      - APL
      - *k* candidatos más cercanos
    - Algoritmo RKV no es directamente adaptable
      - MINMAXDIST da una cota de distancia para un solo punto
      - No sirve en general para *k*-NN

3.5 Referencias

- [BBK01] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322—373, 2001
- [Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM International Conference on Management of Data (SIGMOD'84)*, pages 47—57. ACM Press, 1984
- [HS95] G. Hjaltason and H. Samet. Ranking in Spatial Databases. In *Proc. International Symposium on Large Spatial Databases (SSD'95), LNCS 951*, pages 83—95. Springer-Verlag, 1995
- [RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. ACM International Conference on Management of Data (SIGMOD'95)*, pages 71—79. ACM Press, 1995