
Jueves 29 de octubre
Sesión 14
Árboles Multicriterio
Árbol-R. X-tree.

Árboles-R [Guttman 1984]
Acceso a datos espaciales

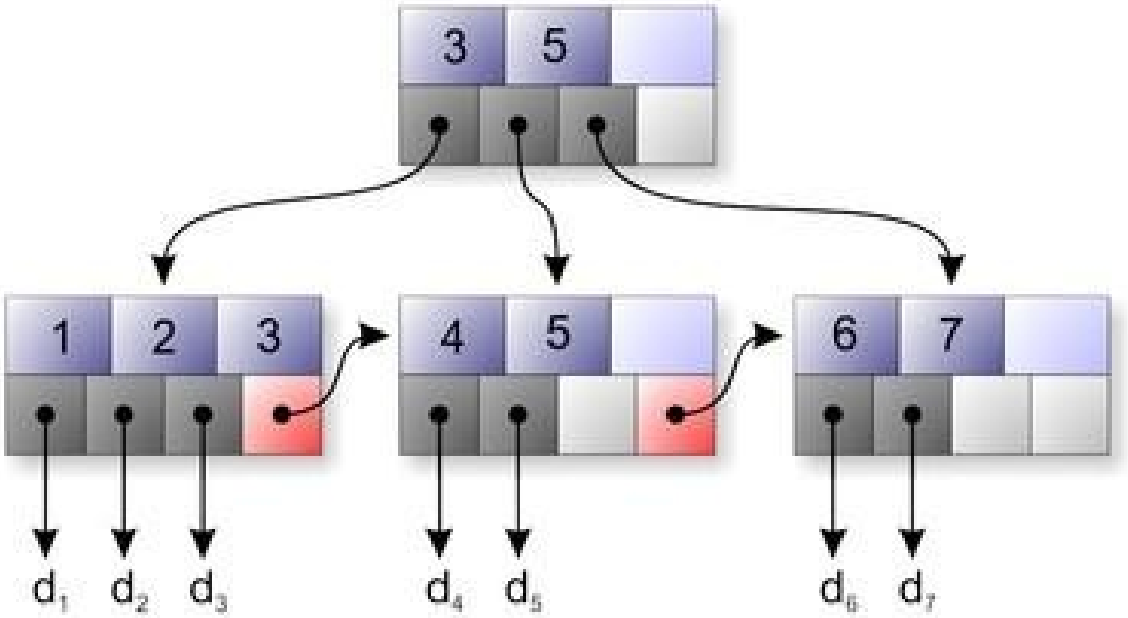
Los árboles-R son estructuras de datos de tipo árbol similares a los árboles-B, con la diferencia de que se utilizan para métodos de acceso espacial, es decir, para indexar información multidimensional; por ejemplo, las coordenadas (x, y) de un lugar geográfico.

Un problema con aplicación práctica en el mundo real podría ser: "Encontrar todos los museos en un radio de dos kilómetros alrededor de la posición actual".

- Los árboles-B sentaron la base de los árboles-R
Entonces ¿Qué es un árbol-B?

- Es una estructura de datos para almacenar datos con tiempos de ejecución amortizados para inserción y borrado.

- Generalmente usado para el almacenamiento de datos de entrada/salida en latencia espaciada (sistemas de archivos y bases de datos ya que los nodos hijo pueden ser accedidos juntos ya que están ordenados)



fuente:Wikipedia

¿Por qué no usar árboles-B?

- Los árboles-B no pueden almacenar nuevos tipos de datos.
- Algunos especialistas tuvieron la necesidad de almacenar datos de contenido geométrico y multidimensional.
- Esta necesidad la suple los árboles-R (Guttman 1984)

árboles-R

- Los árboles-R pueden organizar cualquier información dimensional representando los datos con un Rectángulo Mínimo de Enlace (MBR).

- Cada nodo enlaza a sus nodos hijo. Cada nodo puede contener varios objetos.

- Los nodos hoja apuntan a los objetos actuales (probablemente almacenados en disco).
- Su altura es siempre log n (es un árbol balanceado por la altura).

árboles-R

Los nodos hoja en un árbol-R contienen registros indexados de la forma.

`(I, tupla-identificador)`

árboles-R

$$I = (I_0, I_1, I_2, \ldots, I_{n-1})$$

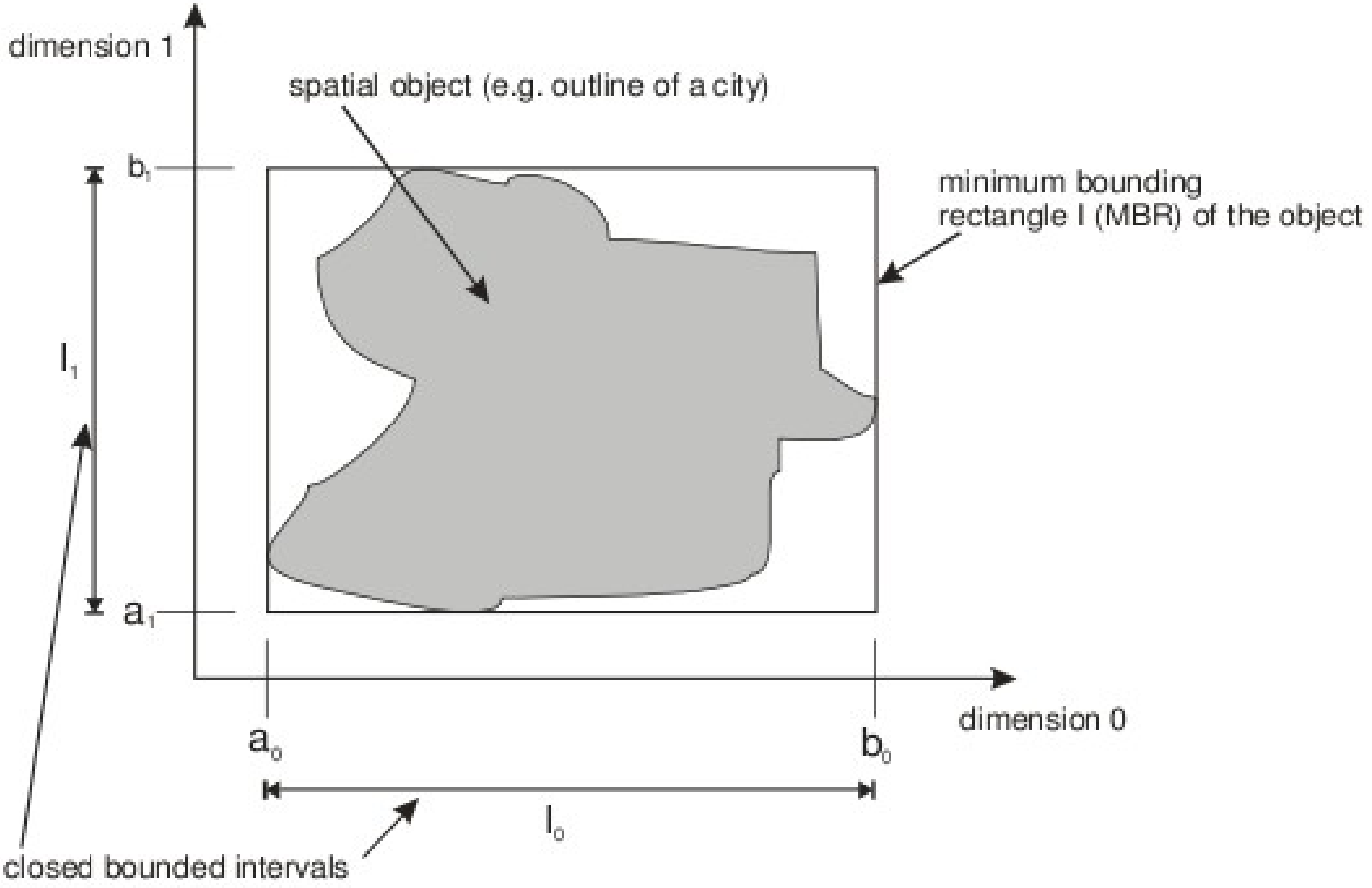
Donde n es el número de dimensiones e I_i es un intervalo cerrado [a,b] el cual describe la dimensión *i*

Los nodo no hoja contienen registros de la forma:

`(I,puntero-hijo)`

Donde puntero-hijo es la dirección de un nodo inferior en el árbol-R e I cubre todos los rectángulos en las entradas al nodo inferior

árboles-R


dimension 1
spatial object (e.g. outline of a city)
minimum bounding
rectangle I (MBR) of the object
b_1
l_1
a_1
a_0
b_0
l_0
dimension 0
closed bounded intervals

Propiedades

Si **M** es el máximo número de entradas que se ajustan a un nodo y **m** ≤ M/2 el mínimo número de entradas en un nodo, entonces el árbol-R debe cumplir las siguientes condiciones:

Propiedades

• Cada nodo hoja, que no sea la raíz contiene entre m y M registros indexados.

• Por cada registro indexado (I, identificador-hoja), I es el rectángulo más pequeño que contiene los objetos de datos n-dimensionales representados por su tupla.

Propiedades

• Cada nodo que no es hoja, que tampoco sea la raíz contiene entre m y M nodos hijo.

• Por cada entrada (I, hijo-puntero) en un nodo que no es hoja, I es el rectángulo más pequeño que contiene el nodo hijo.

• El nodo raíz tiene al menos dos hijos a menos que sea nodo hoja.

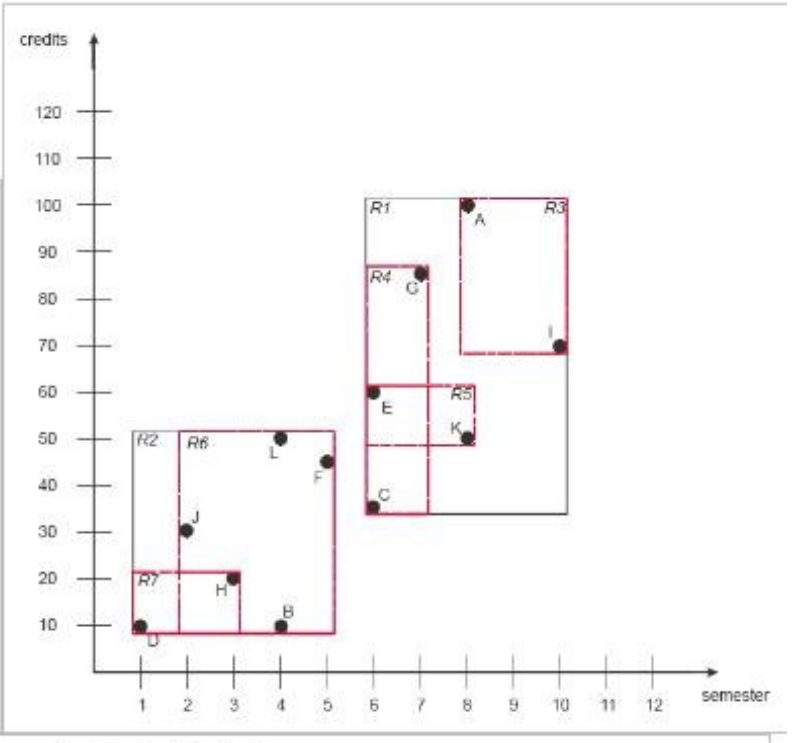
•Todos los nodos hojas aparecen al mismo nivel.

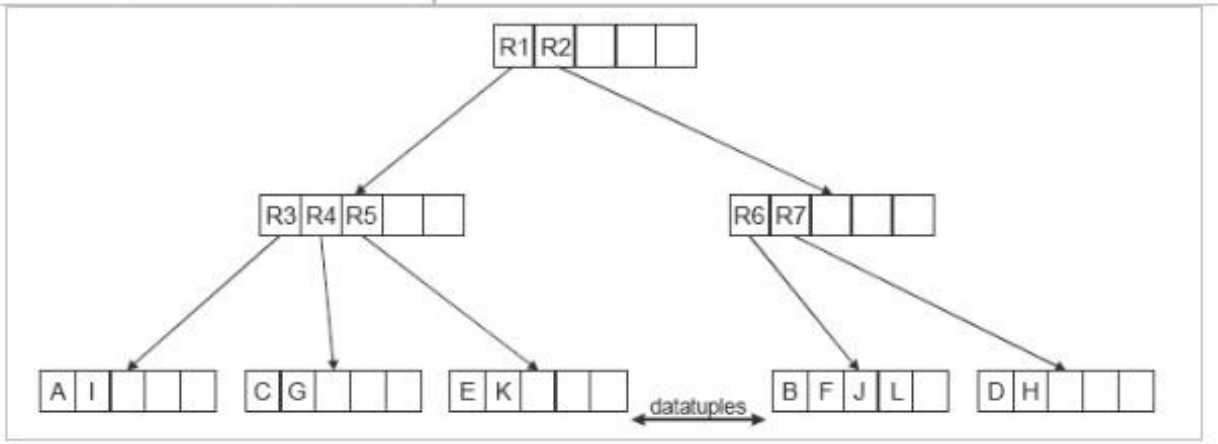
Altura del árbol-R con N registros indexados

$[\log_m N]-1$

Ejemplo de árboles-R

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





Fuente: Neumann 2001

Operaciones

- Búsqueda: Encontrar todos los nodos que se intersectan, luego recorrerlos.

- Insertar: Localizar un espacio para insertar un nodo previa búsqueda. Si el nodo está ocupado es necesario realizar una división (split).

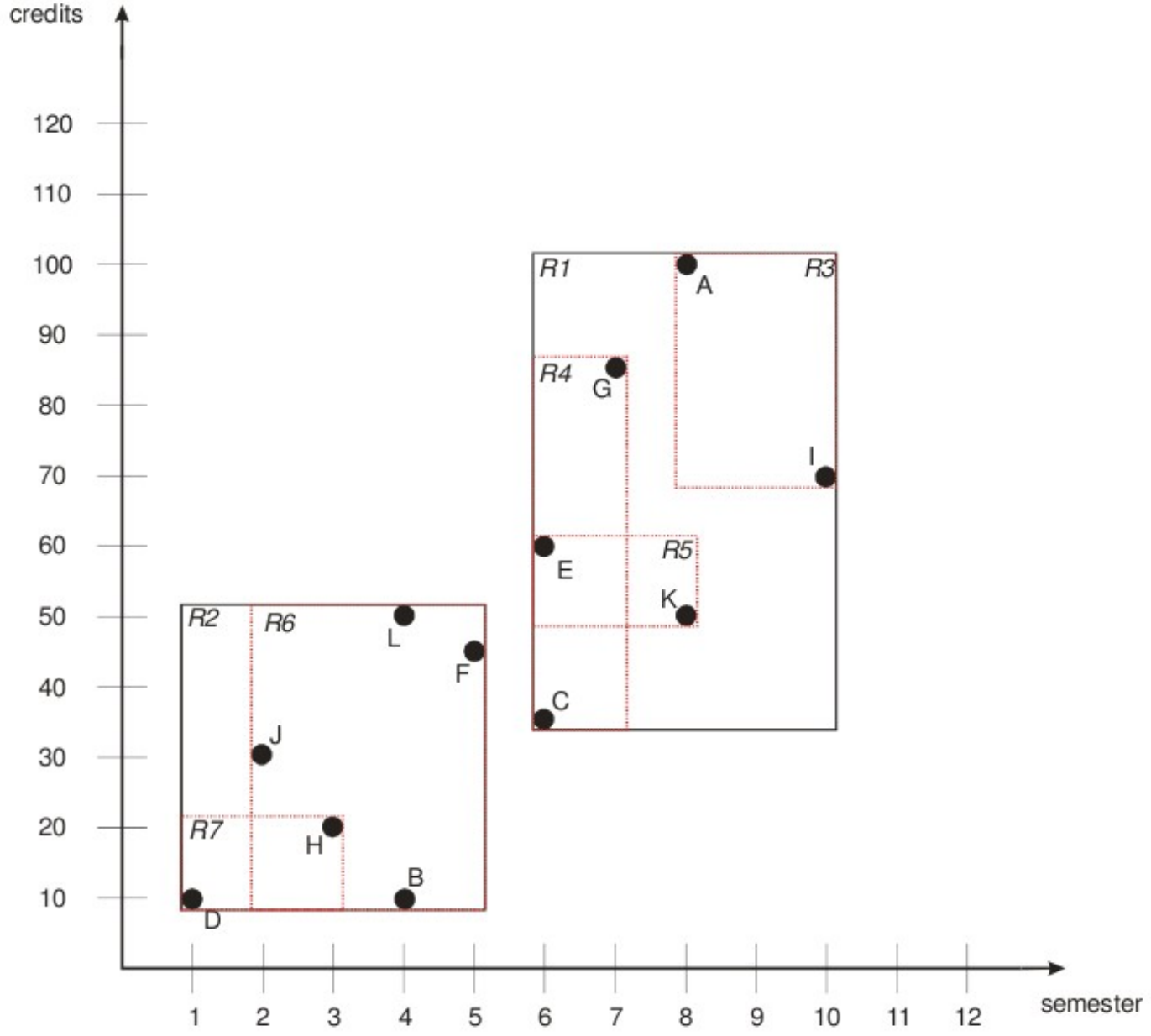
- Borrar: Si un nodo se encuentra vacío, reinsertar otros nodos para mantener el balance.

Base de datos ejemplo

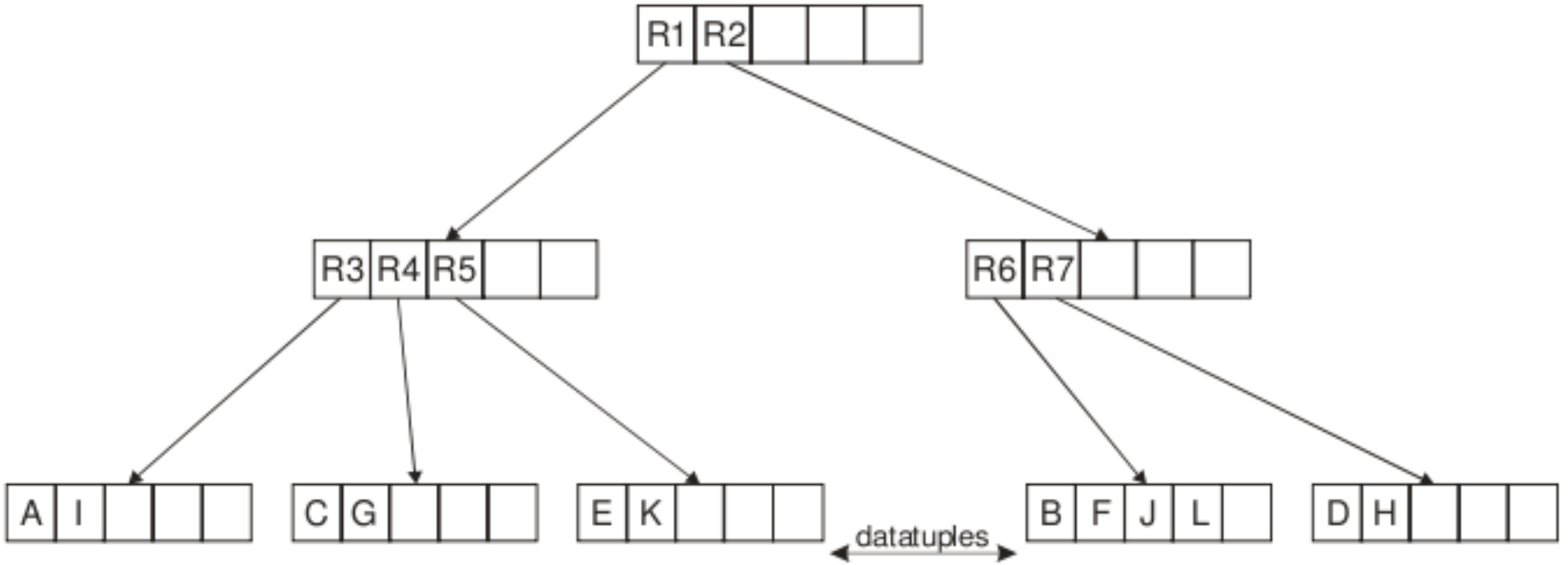
name	semester	credits
A	8	100
B	4	10
C	6	35
D	1	10
E	6	40
F	5	45
G	7	85
H	3	20
I	10	70
J	2	30
K	8	50
L	4	50

Fuente: Neumann 2001

Base de datos como coordenadas espaciales



Base de datos como árbol-R

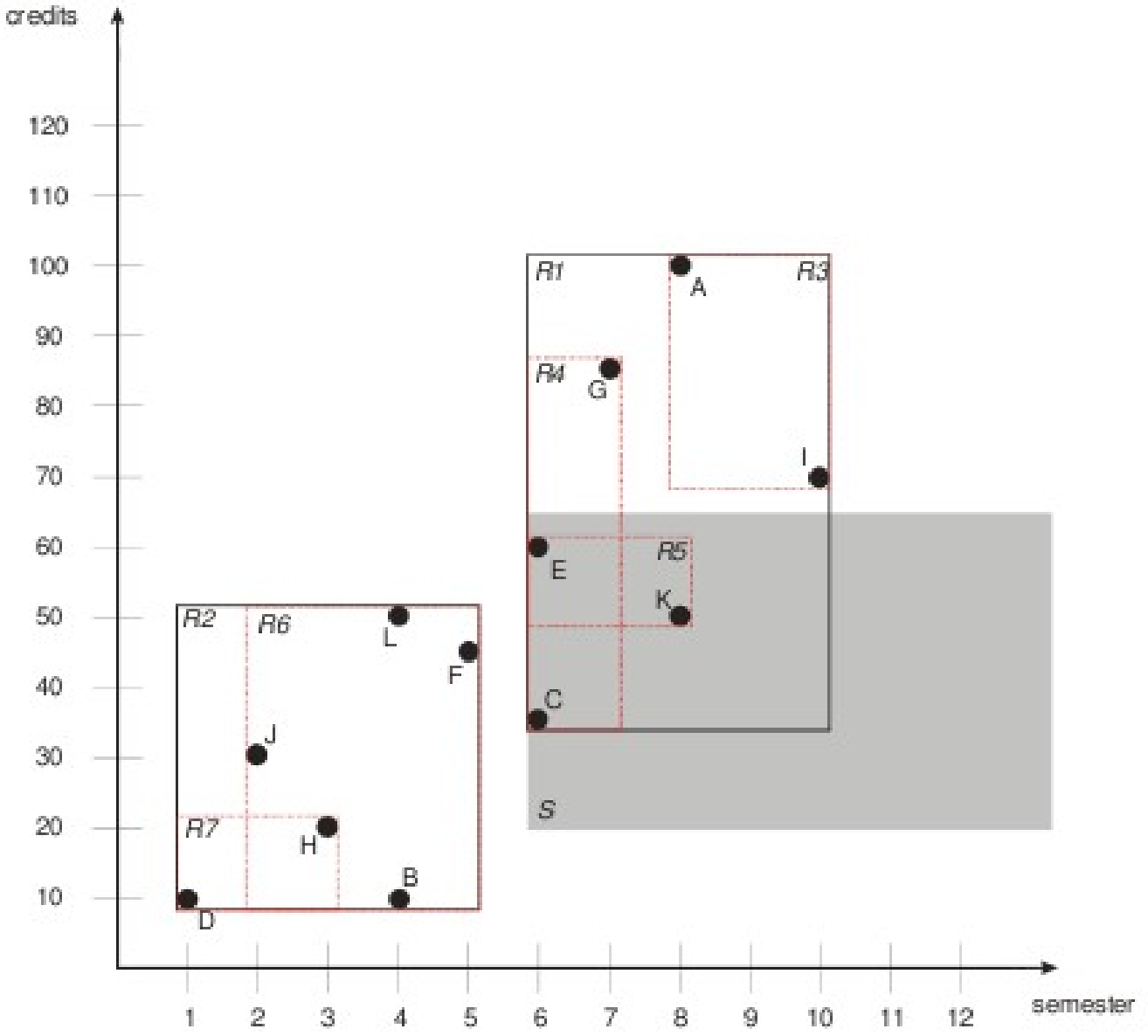


Algoritmos: Búsqueda

Si T es la raíz de un árbol-R. Se busca todos los registros indexados cuyos rectángulos se solapan con un rectángulo específico S.

- Si T no es un nodo hoja entonces se aplica la búsqueda en cada nodo hijo cuya raíz esté apuntada por el puntero *puntero-hijo* y se solape con S.
- Si T es un nodo hoja, retornar todaslas entradas que se solapan con S como conjunto resultado.

Ejemplo: Búsqueda



Algoritmos: Insertar

- Si E es una nueva entrada.
- Use ***ChooseLeaf*** para encontrar el nodo hoja L donde se colocará E.
- Si existe espacio en L (sin desborde) añada E.
  - De lo contrario aplique ***SplitNode*** a L lo cual retornará L y L' conteniendo E y los viejos elementos de L.
- Aplique ***AdjustTree*** a L si hay una división antes realizarlo en L'.
- Si la división alcanza la raíz y debe ser dividido, se crea una nueva raíz cuyos hijos son los dos nodos que retornó la división de la raíz.

Ejemplo: Insertar

Se quiere insertar una tripleta (Q, 10, 65). ***ChooseLeaf*** retorna R1 como su primer nuevo nodo. Luego se elige R3 como el rectángulo en donde se inserta la entrada y no se desborda. Entonces Q se inserta en R3. Luego de eso ***AdjustTree*** actualiza el rectángulo de R3 y R1.

Algoritmos: Borrar

- Aplicar ***FindLeaf*** para encontrar la hoja de L que contiene E. Finalizar si se encuentra E.

- Borrar E de L.

- Usar ***CondenseTree*** sobre L para condensar bajo los nodos llenos.

- Si la raíz tiene solamente un hijo luego del ajuste, entonce este hijo será la raíz. La altura del árbol se decrementa

Ejemplo: Borrar

Se quiere borrar el estudiante K de la base de datos. Se aplica ***FindLeaf*** retornando R5 como el rectángulo correspondiente, se borra la entrada K de R5 notando que hay un *underflow.* Se aplica ***CondenseTree*** a R5, el cual borra R5 del árbol-R se añade E a R4 usando Insertar y Actualizar el rectángulo R1.

División de nodos

Cuando se añade un elemento a un nodo lleno es necesario dividir esta entrada M+1 en dos nodos nuevos

División de nodos

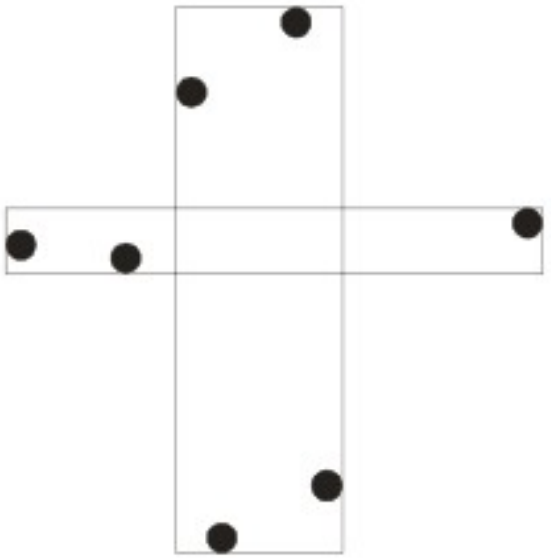


Figure 8a: A good split

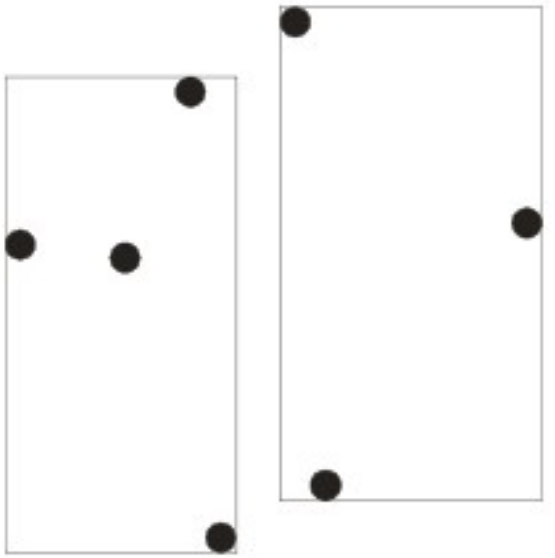


Figure 8b: A bad split

División de nodos completos

Distintos algoritmos pueden ser usados para dividir los nodos cuando se encuentran llenos. Obteniendo como resultado sub-clasificación de árboles-R y distintos costos de ejecución: cuadráticos, lineales y exponenciales.

División de nodos completos

- Lineal: Elegir nodos apartados como finales. Aleatoriamente elegir nodos y asignarlos para que así requieran la menor ampliación del rectángulo mínimo de enlace.

División de nodos completos

- Cuadrático: Elegir dos nodos para que el espacio muerto entre ellos se maximice. Insertar nodos para que el aumento del área sea mínimo.

División de nodos completos

- Exponencial: Algoritmo exhaustivo, busca todos los posibles grupos.

Problemas del árbol-R

- Dado que el único criterio es minimizar el área:
  - Algunos tipos de datos pueden crear pequeñas áreas pero grandes distancias, los cual inicia una mala división (*split*).

Problemas del árbol-R

- Dado que el único criterio es minimizar el área:
  - Si un grupo alcanza el máximo número de entradas, los demás serán asignados sin considerar su geometría.

Green trató de resolver esto, pero usando sólo la técnica *split axis,* para resolver esto es necesario hacer uso de mayores criterios.

X-Tree

eXtended node tree

Una estructura de datos indexada para datos de altas dimensiones [Berchtol et al 1996]

Un árbol X-tree es un estructura de datos indexada basada en árboles-R.

¿Cuál es el problema con los árboles-R?

¿Si maneja tanto coordenadas como datos espaciales?

Su rendimiento se deteriora a medida que aumentan la cantidad de datos

Luego de evaluaciones[Berchtold et al 1996] se encontró que la superposición (overlap) en directorios aumenta rápidamente a medida que crecen las dimensiones de los datos.

Overlap ↑ Rendimiento en búsquedas ↓

Definiendo superposición “overlap”

Es el porcentaje de espacio cubierto por más de un directorio hiper-rectángulo.

Un nodo de árbol-R contiene n hiper-rectángulos

Definiendo “overlap”

$$Overlap = \frac{\left\| \bigcup_{i,j \in \{1 \ldots n\}, i \neq j} (R_i \cap R_j) \right\|}{\left\| \bigcup_{i \in \{1 \ldots n\}} R_i \right\|} . 1$$

Donde:

$\| A \|$ Denota el volumen cubierto por A

$| A |$ Denota el número de elementos contenidos en A

El “overlap” está directamente relacionado con el rendimiento de las búsquedas.

Para resolver estos conflictos se cuenta con el X-tree

Usado para almacenar datos en diferentes dimensiones.

Se diferencia de los árboles R-tree porque hace énfasis en la prevención de solapamiento de cajas entrelazadas o mínimo rectángulo de enlace

En los casos en los que los nodos no puedan ser separados sin prevenir el solapamiento, el mismo será colocado en tramos, teniendo como resultados super-nodos.

En casos extremos, el árbol será linealizado, lo cual previene el peor de los casos observado en otras estructuras de datos.

Algoritmos: Inserción

- Determinar una combinación de estructura jerárquica y lineal.
- Tratar de encontrar un topológico mínimo o solapamiento mínimo.
- Dividir usando heurísticas de árbol-R*.
- Si no se obtienen divisiones el directorio del nodo actual se extiende para convertirse en un supernodo del doble de tamaño estándar.
- Si el supernodo actual está lleno, se añade un bloque adicional al supernodo.

```
int X_DirectoryNode::insert(DataObject obj, X_Node **new_node)
{
    SET_OF_MBR *s1, *s2;
    X_Node *follow, *new_son;
    int return_value;

    follow = choose_subtree(obj);        // choose a son node to insert obj into
    return_value = follow->insert(obj, &new_son); // insert obj into subtree
    update_mbr(follow->calc_mbr());      // update MBR of old son node

    if (return_value == SPLIT){
        add_mbr(new_son->calc_mbr());    // insert mbr of new son node into current node

        if (num_of_mbrs() > CAPACITY){  // overflow occurs
            if (split(mbrs, s1, s2) == TRUE){
                // topological or overlap-minimal split was successfull
                set_mbrs(s1);
                *new_node = new X_DirectoryNode(s2);

                return SPLIT;
            }
            else // there is no good split
            {
                *new_node = new X_SuperNode();
                (*new_node)->set_mbrs(mbrs);

                return SUPERNODE;
            }
        }
    } else if (return_value == SUPERNODE){ // node 'follow' becomes a supernode
        remove_son(follow);
        insert_son(new_son);
    }

    return NO_SPLIT;
}
```

Figure 7: X-tree Insertion Algorithm for Directory Nodes

MBR: Minimun Bounding Rectangles

Algoritmos: Split

- Añadir un MBR a un nodo puede convertirse en un desborde de memoria y causar una división.
- **Criterios para dividir un nodo:**
- Encontrar una división basada en propiedades geométricas y topológicas del MBR.
- Si el paso anterior resulta en un solapamiento mayor use la técnica de división con mínimo solapamiento.
- Si lo anterior resulta en nodos semi-vacios no haga la división y retorne falso.

*Minimum Bounding Rectangles

```
bool X_DirectoryNode::split(SET_OF_MBR *in, SET_OF_MBR *out1, SET_OF_MBR *out2)
{
    SET_OF_MBR t1, t2;
    MBR r1, r2;

    // first try topological split, resulting in two sets of MBRs t1 and t2
    topological_split(in, t1, t2);
    r1 = t1->calc_mbr(); r2 = t2->calc_mbr();

    // test for overlap
    if (overlap(r1, r2) > MAX_OVERLAP)
    {
        // topological split fails -> try overlap minimal split
        overlap_minimal_split(in, t1, t2);

        // test for unbalanced nodes
        if (t1->num_of_mbrs() < MIN_FANOUT || t2->num_of_mbrs() < MIN_FANOUT)
            // overlap-minimal split also fails (-> caller has to create supernode)
            return FALSE;
    }

    *out1 = t1; *out2 = t2;
    return TRUE;
}
```

Figure 8: X-tree Split Algorithm for Directory Nodes

Algoritmos: Borrar y actualizar

- Actualizar es una combinación de borrar e insertar.

- Si existe un *underflow* (cuando el valor es menor que los límites inferiores de precisión del computador) en los supernodos durante el borrado, se mezclan para conformar un solo nodo directorio.

- Por consiguiente la estructura es dinámica.

Conclusiones

Los árboles-R no se comportan positivamente ante espacios de altas dimensiones.

Los x-Tree con el concepto de supernodos y división de mínimo solapamiento provee una mayor velocidad en búsquedas de coordenadas y vecinos mas cercanos.

Sin embargo dado que el tiempo total de búsquedas del x-Tree crece logaritmicamente con el tamaño de la base de datos, se comporta bien con bases de datos de gran tamaño.


¿Preguntas?