Lección 17: Mallas regulares y kd-trees

- Motivación
- EEDD multidimensionales y aplicaciones
- Mallas regulares
  - Operaciones con mallas
  - Implementación
- kd-tree
  - Algoritmo de búsqueda
  - Algoritmo de construcción
  - Eficiencia
- Consideraciones finales

Motivación

GPService se ha implantado en Jaén para ofrecer a sus usuarios un sistema para encontrar en cada momento desde el móvil los establecimientos más cercanos, por ejemplo farmacias

Desea acelerar la búsqueda si tiene que atender *n* peticiones solicitando el servicio para *m* comercios posibles porque los usuarios no permanecen en una posición fija

Motivación

Ha pensado en utilizar un mapa por una de las coordenadas, por ejemplo la X

El problema es que el usuario que está en la posición A está muy lejos de la farmacia B aunque tienen valores similares de coordenada X

EEDD multidimensionales

- Muchas de las entidades existentes son de naturaleza bidimensional o tridimensional
- Hasta el momento las entidades manejadas usaban relaciones de orden en base a una magnitud
  - Por ejemplo: los alumnos se ordenan por DNI
- Sin embargo en muchos campos científicos, existen datos bidimensionales:
  - Las coordenadas de las posiciones de las farmacias
  - Las medidas de un dispositivo con sensor de temperatura y presión

EEDD multidimensionales

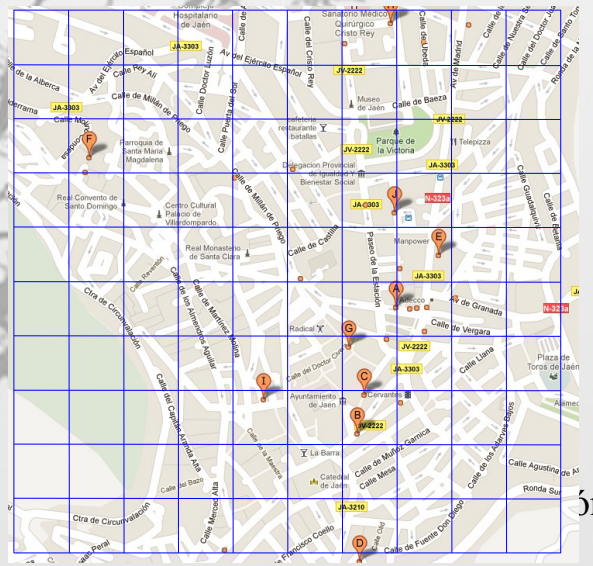
- Las EEDD multidimensionales/espaciales permiten organizar los datos por más de un atributo al mismo tiempo
- Esos atributos se interpretan como coordenadas en el plano o el espacio
- Estas EEDD dividen el espacio en regiones disjuntas
- Un dato representado por un punto (p. e. la posición de la farmacia en el mapa) sólo pertenece a una región
- Según el caso, unas regiones pueden dividirse en otras más pequeñas
- El proceso de búsqueda es eficiente porque en cada etapa se descarta parte del plano o espacio que queda por procesar

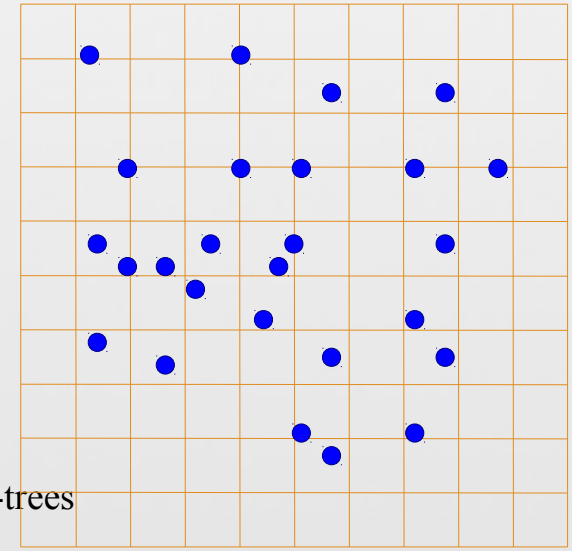
Aplicaciones

- Se utilizan en muchas disciplinas relacionadas con las ingenierías, por ejemplo la Informática Gráfica
- Encontrar el/los puntos más cercanos a uno dado
  - Encontrar las farmacias más cercanas a un usuario con móvil y GPS
- Conocer si un punto pertenece o no a una región
  - El punto representa por ejemplo un valor estadístico y la región un conjunto de valores posibles
- Dada una zona o región del plano obtener los puntos que contiene
  - Telescopios virtuales: dado un recuadro del firmamento, ¿cuales son las estrellas que contiene?

Mallas regulares

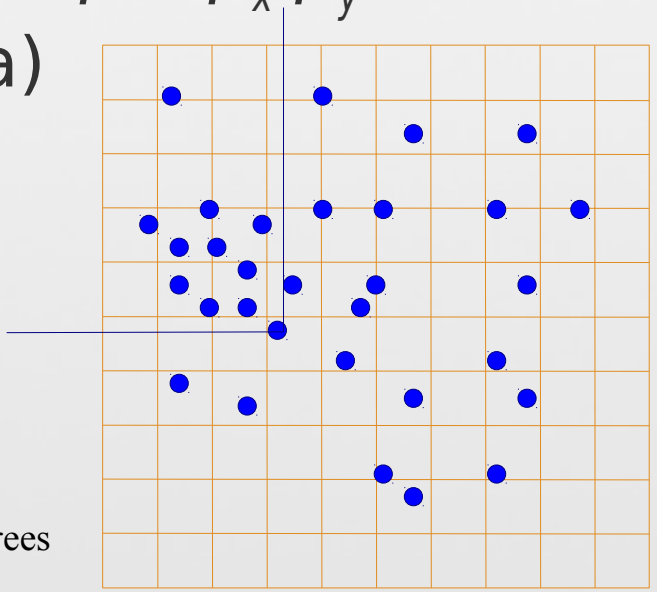
- En 2D, el plano se divide en regiones rectangulares todas del mismo tamaño, permitiendo **acceso tipo array**. Concepto extensible a 3D
- Cada una de estas regiones queda representada mediante una celda de una matriz 2D
- Cada celda mantiene una lista de puntos (punteros a puntos) que contiene el área que representa





Operaciones con mallas

- **Crear malla**: definir el tamaño del vector
  - Un tamaño muy grande de casilla ubica muchos puntos y haría la búsqueda poco eficiente
  - Un tamaño muy pequeño subdivide mucho el plano (espacio), generando muchas casillas que pueden estar vacías
- **Búsqueda**: localizar un punto $p=(p_x,p_y)$ en una casilla (hay relación matemática)
  - Determinar la fila usando p_y
  - Determinar la columna con p_x
  - Búsqueda lineal en la casilla

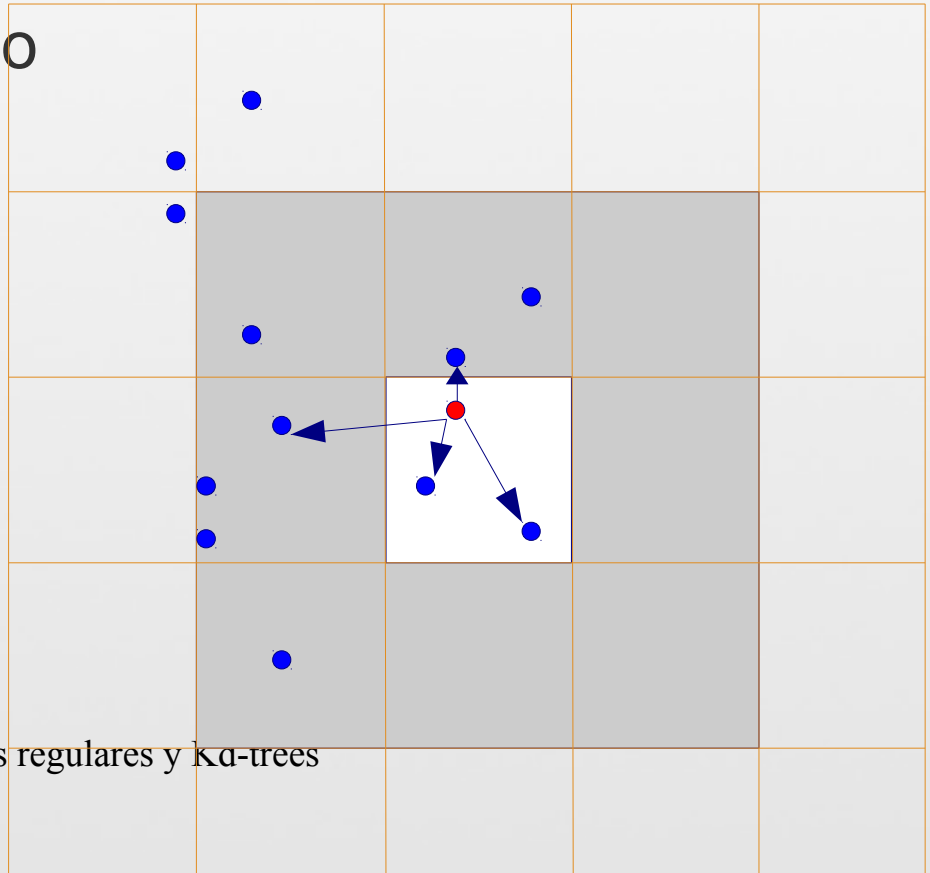


Operaciones con mallas

- **Insertar**: un punto $p=(p_x,p_y)$
  - Localizar la casilla como se hizo en la búsqueda
  - Añadir el nuevo dato a la lista de puntos de dicha casilla
- **Borrar**: un punto $p=(p_x,p_y)$
  - Localizar la casilla como se hizo en la búsqueda
  - Localizar el dato mediante búsqueda lineal y eliminarlo

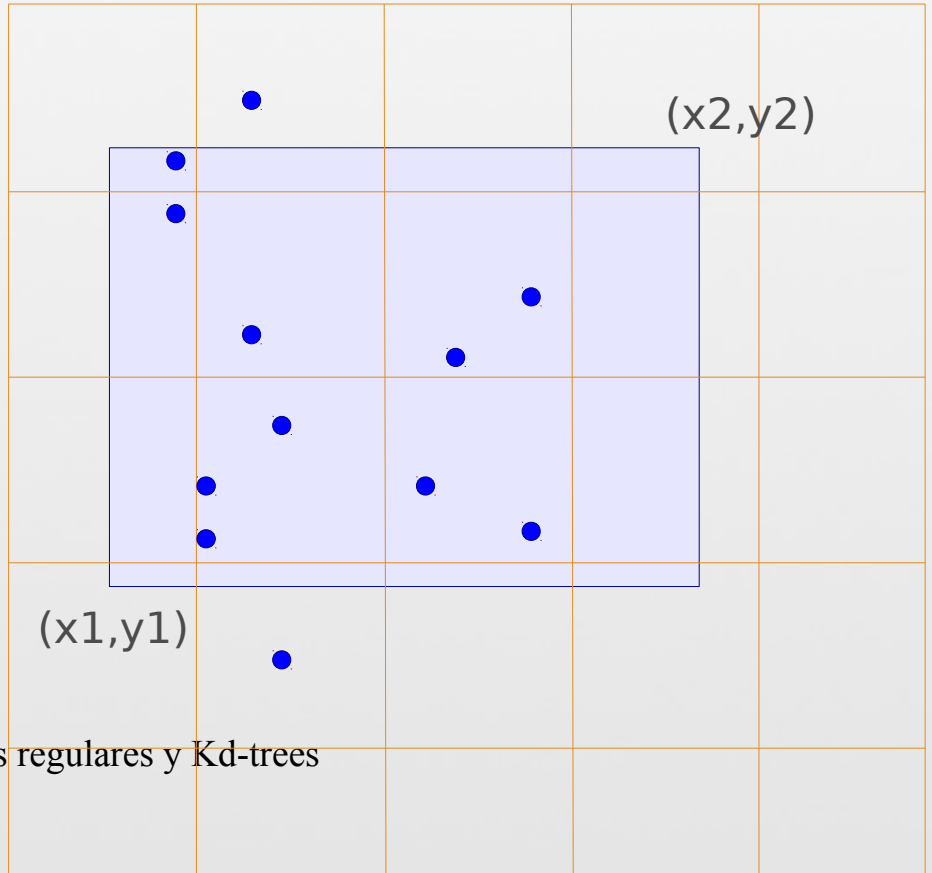
Operaciones con mallas

- **Más cercano**: de un punto $p=(p_x,p_y)$
  - Localizar la casilla, el punto más cercano puede estar en la misma casilla o en alguna anexa
  - Localizar el más cercano de la casilla donde está p
  - Localizar el más cercano de los 8 vecinos
  - Quedarse con el menor
  - Cuidado: si no hay ningún otro punto en la casilla, extender la búsqueda al siguiente conjunto de vecinos



Operaciones con mallas

- **Búsqueda por rangos:** [x1,x2][y1,y2]
  - Consiste en obtener todos los valores comprendidos en ese rango
  - Buscar las casillas correspondientes a (x1,y1) y a (x2,y2)
  - Visitar las casillas comprendidas en ese rango y devolver los puntos que contienen con coordenada (x,y) si x1<x<x2 y y1<y<y2



Implementación de mallas regulares: casilla

```
template <class U>
class Casilla{
    list<U> puntos;
    public:
        friend class MallaRegular<U>;
        Casilla(): puntos(){}
        void inserta (U &dato) { puntos.push_back(dato); }
        bool busca (U &dato);
        bool borra (U &dato);
};
```

U se instancia a punto o a objetos con coordenadas

```
template <class U>
bool Casilla<U>::busca(U& dato){
    typename list<U>::iterator it;
    it = puntos.begin();
    while (it != puntos.end()){
        if (*it == dato)
            return true;
    }
    return false;
}
```

```
template <class U>
bool Casilla<U>::borra(U& dato){
    typename list<U>::iterator it;
    it = puntos.begin();
    while (it != puntos.end()){
        if (*it == dato)
            puntos.erase(it);
            return true;
    }
    return false;
}
```

búsquedas secuenciales

Implementación: M.R.

Se introduce el tamaño de la superficie $[x_{min},y_{min}]$ $[x_{max},y_{max}]$ y el número de divisiones *n*

```
tetemplate <class T>
class MallaRegular {
    float xMin, yMin, xMax, yMax; //tamaño real global
    float tamaCasillax, tamaCasillaY; //tamaño real de cada casilla

    vector<vector<Casilla<T> > > mr; //vector 2D de casillas

    Casilla<T> *obtenerCasilla(float x, float y);

public:
    MallaRegular(int aXMin, int aYMin, int aXMax, int aYMax, int
aNDiv);
    void insertarDato(float x, float y, T &dato);
    Casilla<T> *buscarDato(float x, float y, T& dato);
    Casilla<T> *borrarDato(float x, float y, T& dato);
};
```

Implementación: M.R.

```
template <class T>
MallaRegular<T>::MallaRegular(int aXMin, int aYMin, int aXMax, int
aYMax, int aNDiv) : xMin(aXMin), yMin(aYMin), xMax(aXMax), yMax(aYMax){
    tamaCasillaX = (xMax-xMin)/aNDiv;
    tamaCasillaY = (yMax-yMin)/aNDiv;
}

template <class T>
Casilla<T> *MallaRegular<T>::obtenerCasilla (float x, float y){
    int i = (x - xMin) / tamaCasillaX;
    int j = (y – yMin) / tamaCasillaY;
    return &mr[i][j];
}

template <class T>
void MallaRegular<T>::insertarDato(float x, float y, T& dato){
    Casilla<T> *c = obtenerCasilla(x,y);
    c->inserta(dato);
}

template <class T>
Casilla<T> *MallaRegular<T>::borrarDato(float x, float y, T& dato){
    Casilla<T> *c = obtenerCasilla(x,y);
    if (c->borra(dato))
        return c;
    return 0;
}
```

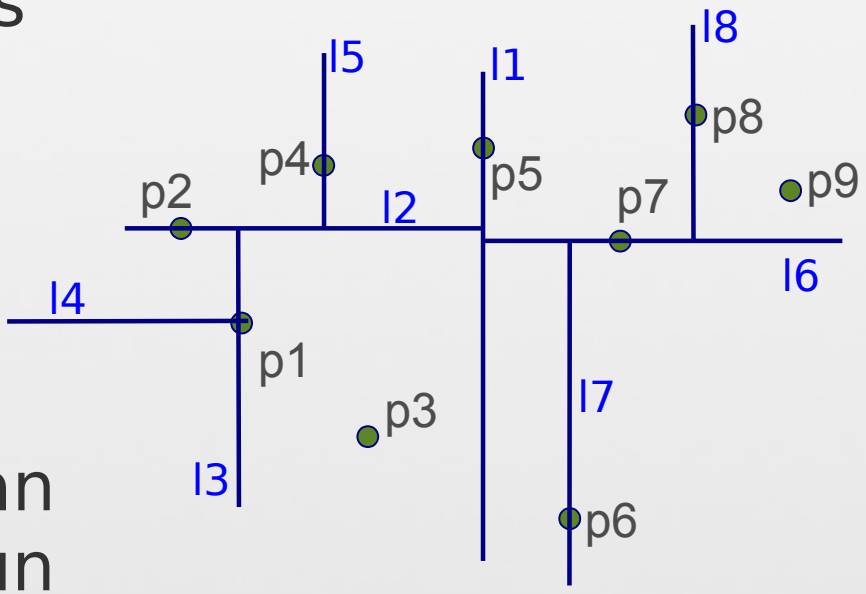
la búsqueda hace lo mismo, pero llamando a Casilla::busca()

Eficiencia de las M.R.

- Es extensible a 3D o k-dimensiones de forma sencilla
- Las mallas regulares pueden ser muy eficientes, con tiempo cercano al O(1)
- Pero esto sólo ocurre cuando los datos se distribuyen uniformemente por la malla
- El problema se presenta cuando los datos no se reparten de modo homogéneo:
  - Unas celdas tienen muchos datos sobre los que se realizan búsquedas secuenciales
  - Otras muchas celdas quedan vacías, malgastándose espacio en memoria
- Este problema se mejora con EEDD adaptativas

Kd-tree

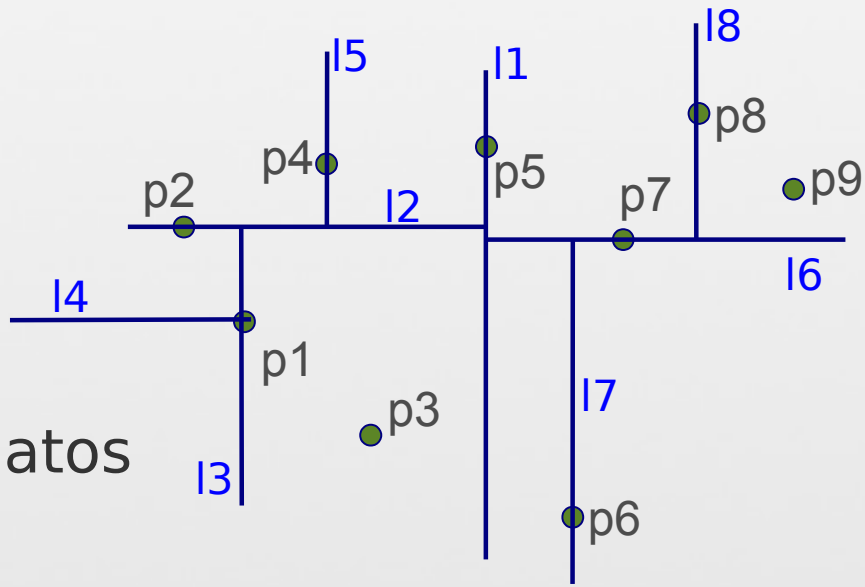
- Un Kd-tree es un árbol binario que representa un espacio de k dimensiones
- Trabajaremos en el plano con 2d-trees
- Cada nodo no hoja del árbol representa un eje ortogonal que parte la región del plano representada en 2 sectores
  - En el ejemplo, l1, l2, l3...
- Estos sectores no tienen por qué ser del mismo tamaño
- Los nodos hoja representan las regiones divididas un solo dato



Kd-tree

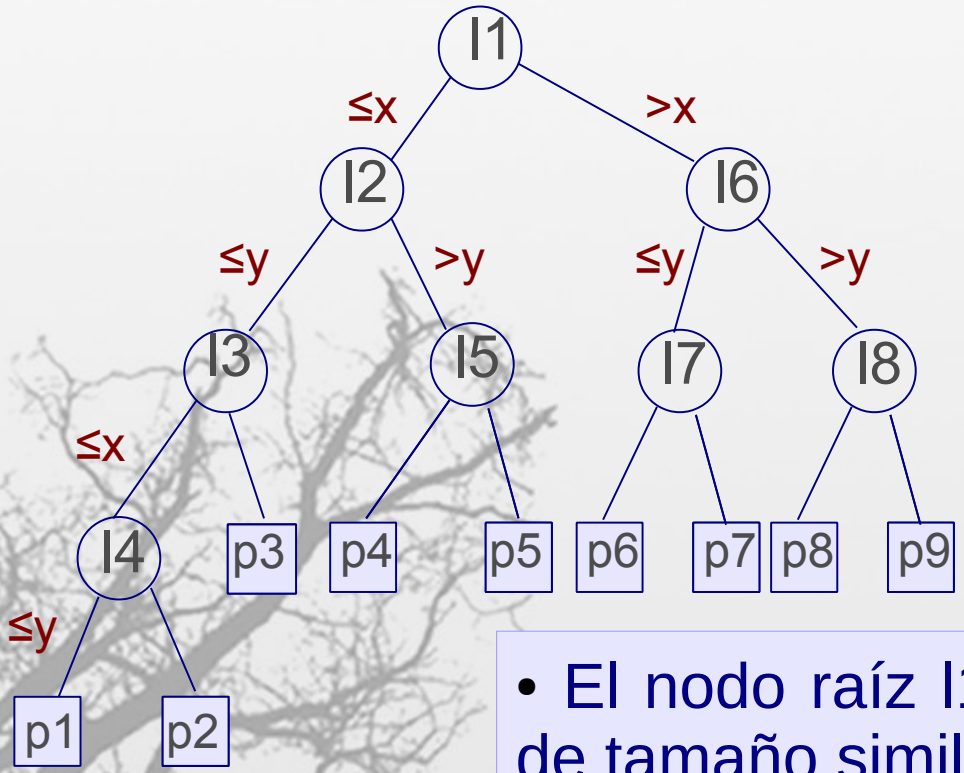
Existen dos tipos de nodos interiores (no hoja):

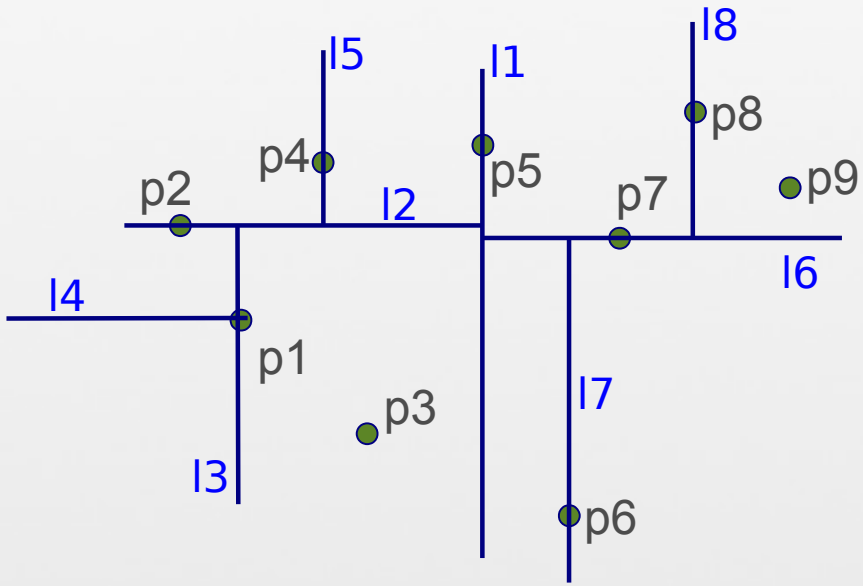
- Nodos x-discriminador en niveles impares
  - Representados por ejes verticales que dividen el plano en izquierda y derecha
  - El subárbol izquierdo del nodo tienen los puntos con menor o igual X
  - En el derecho los de mayor X
- Nodos y-discriminador en niveles pares
  - Dividen arriba y abajo
  - En el subárbol izquierdo los datos con X igual o menor
  - En el derecho los de mayor X



Kd-tree

- Los ejes están asociados a un punto
- Este punto se escoge como el punto más central posible para equilibrar el árbol

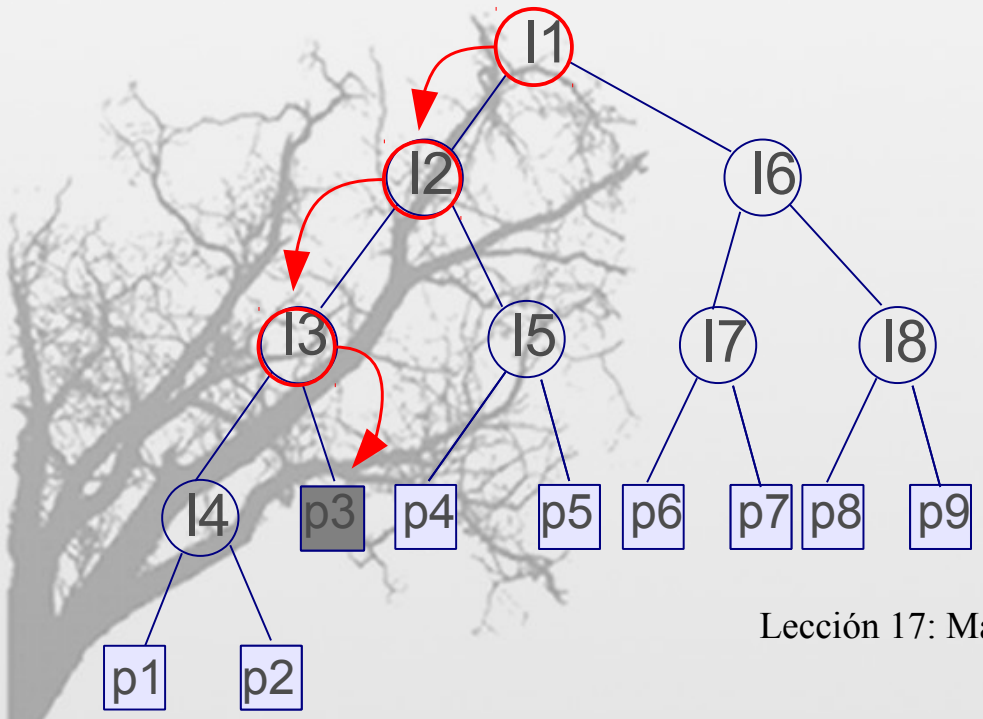


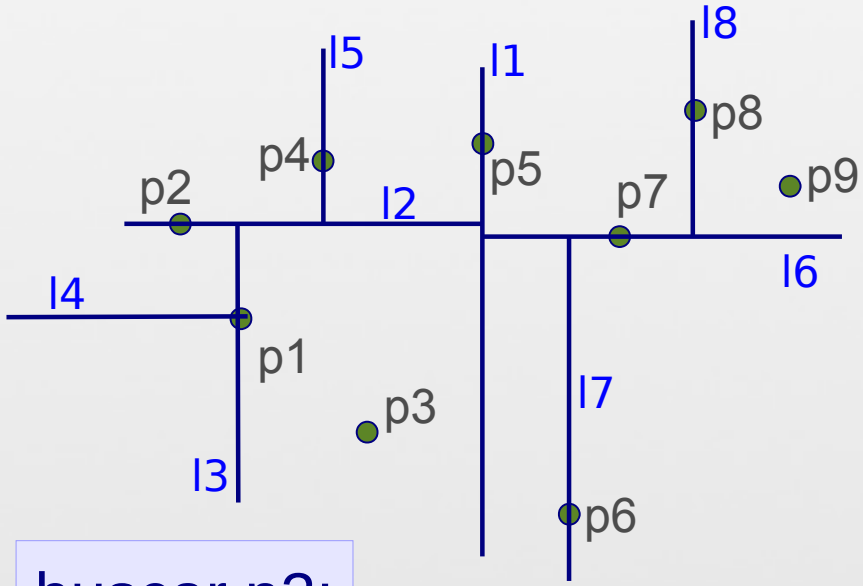


- El nodo raíz l1, divide el plano en dos mitades de tamaño similar
- A la izquierda de l1 están los puntos con X <= l1
- A la derecha los puntos que cumplen l1 < X

Kd-tree: búsqueda

- Buscar punto p=(px,py)
  - si el nodo es interno de nivel impar y dato q=(qx,qy); si (px<=qx) ir al hijo izquierdo, sino al derecho
  - si el nodo es interno de nivel par y dato q=(qx,qy); si (py<=qy) ir al hijo izquierdo, sino al derecho
  - Paro al llegar a una hoja





buscar p3:

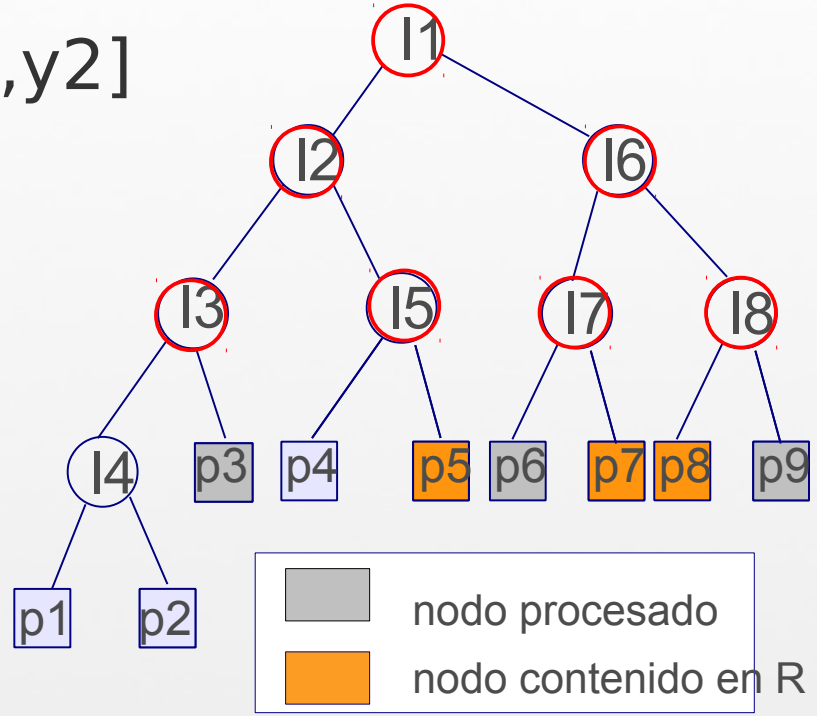
Kd-tree: búsqueda

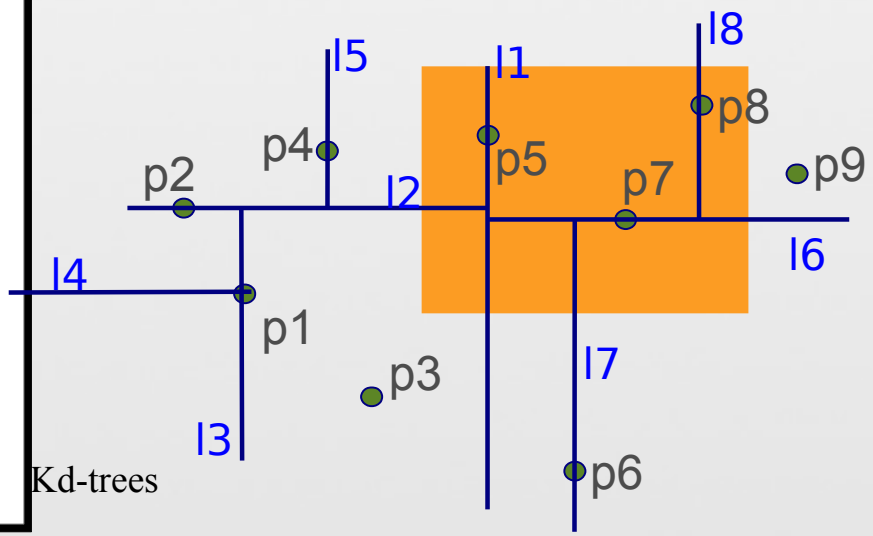
- Buscar en un rango [x1,y1][x2,y2]

```
Algoritmo BuscarKdTree(v,R,Q)
Entrada:
    el kd-tree v, el rango R=[x1,y1][x2,y2]
Salida: Q, los puntos dentro de R

INICIO
SI v es nodo hoja conteniendo a q=(qx,qy)
ENTONCES   SI q está dentro de R
           ENTONCES Q+=q
SINO
    SI Izq(v) está todo contenido en R
    ENTONCES ObtenerSubárbol(Izq(v))
    SINO
        SI region(Izq(v)) intersecta con R
        ENTONCES BuscarKdTree(Izq(v),R,Q)

    SI Der(v) está todo contenido en R
    ENTONCES ObtenerSubárbol(Der(v))
    SINO
        SI region(Der(v)) intersecta R
        ENTONCES BuscarKdTree(Der(v),R,Q)
FIN
```

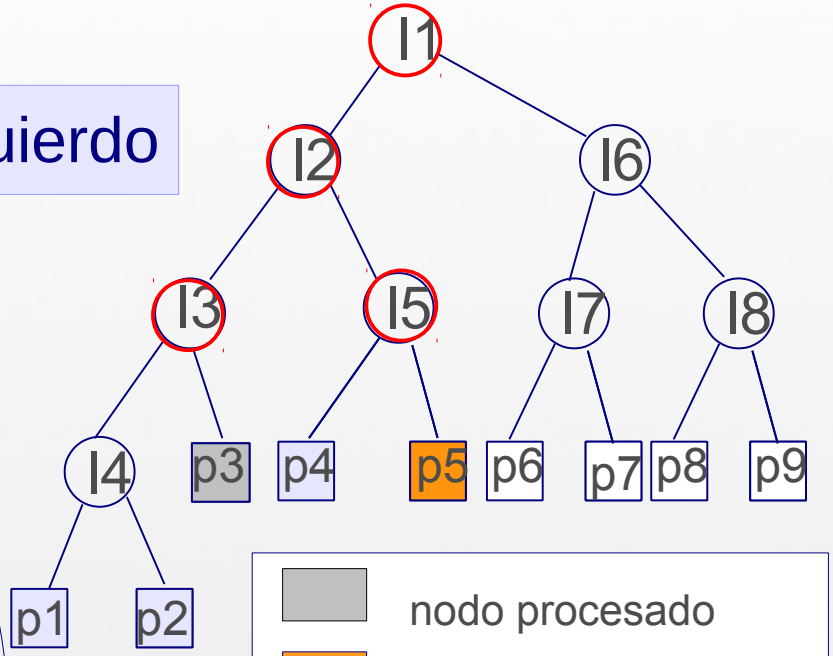




Kd-tree: búsqueda

BuscaKdTree (l1): izda
• intersecta con R:
• BuscaKdTree (l2)

subárbol izquierdo



BuscaKdTree (l2):izda
• intersecta con R
• BuscaKdTree (l3)

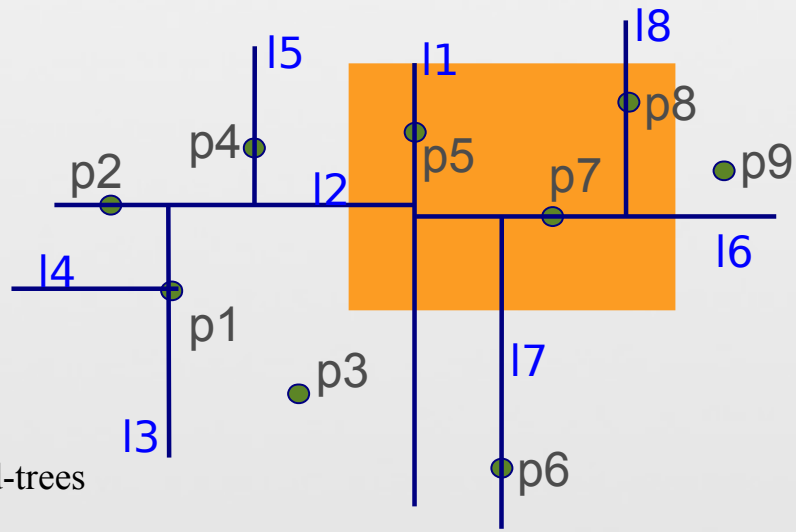
BuscaKdTree (l2):decha
• intersecta con R
• BuscaKdTree (l5)

BuscaKdTree (l3):izda
• no intersecta ni está
• contenido en R

BuscaKdTree (l5):izda
• es hoja
• p4 no está en R

BuscaKdTree (l3):decha
• es hoja
• p3 no está en R

BuscaKdTree (l5):decha
• es hoja
• p5 está en R



Kd-tree: búsqueda

BuscaKdTree (l1): decha
• intersecta con R:
• BuscaKdTree (l6)

subárbol derecho

BuscaKdTree (l6):izda
• intersecta con R
• BuscaKdTree (l7)

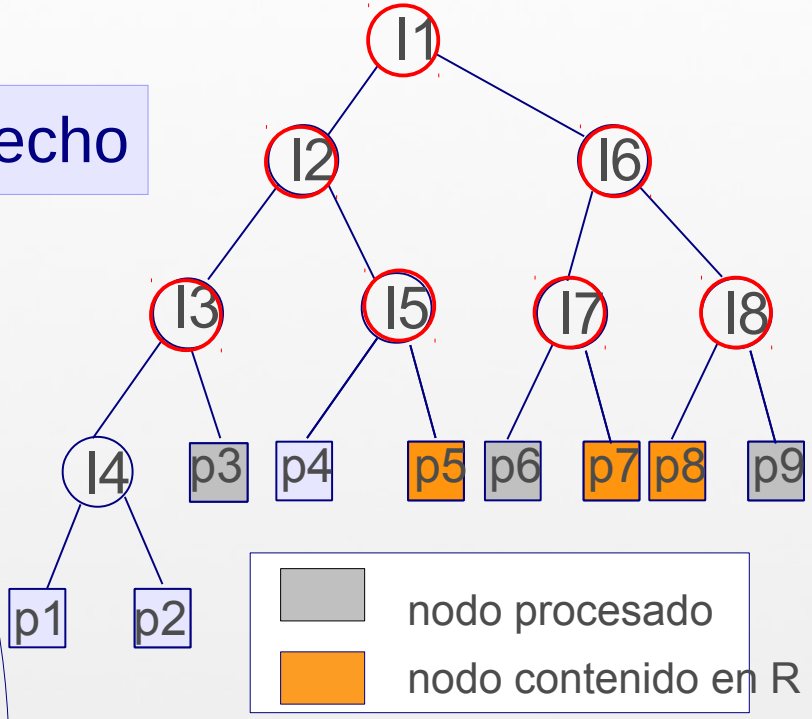
BuscaKdTree (l6):decha
• intersecta con R
• BuscaKdTree (l8)

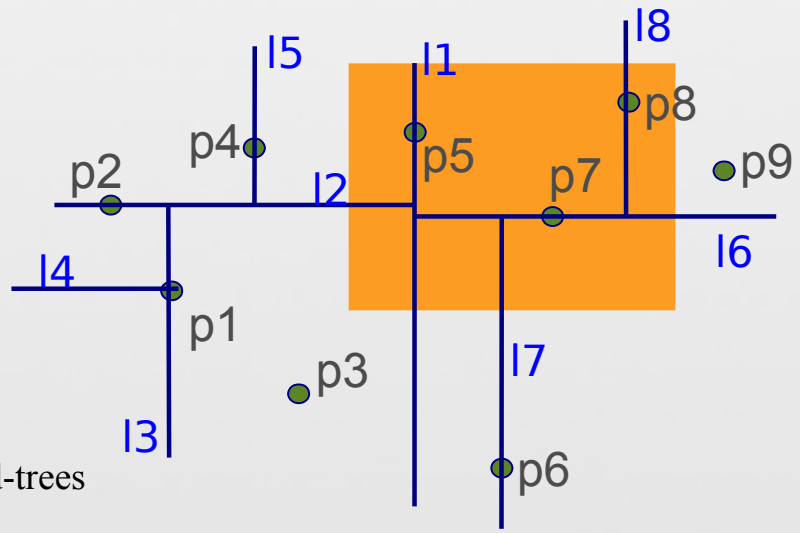
BuscaKdTree (l7):izda
• es hoja
• p6 no está en R

BuscaKdTree (l8):izda
• es hoja
• p8 está en R

BuscaKdTree (l7):decha
• es hoja
• p7 está en R

BuscaKdTree (l8):decha
• es hoja
• p9 no está en R





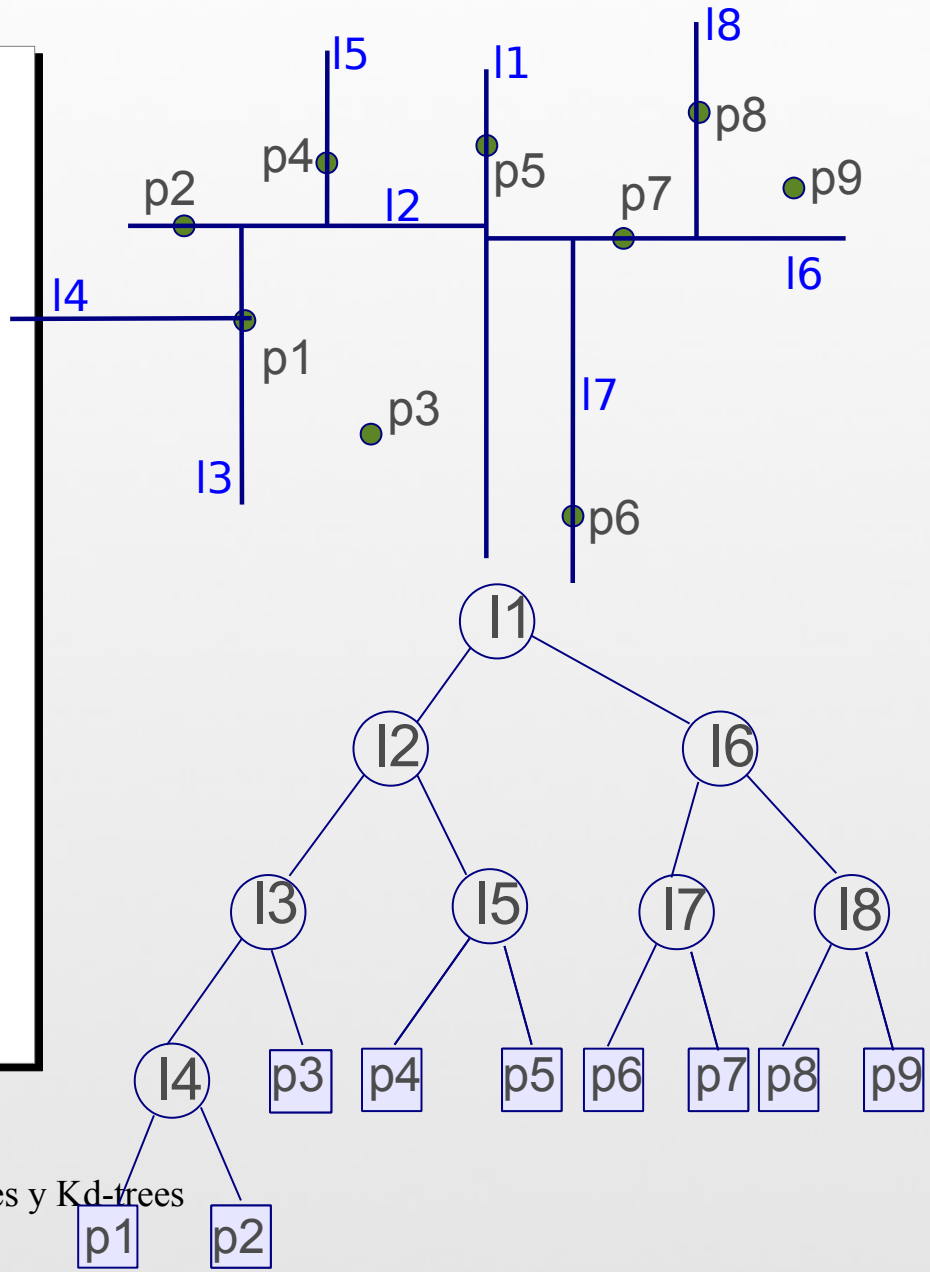
Kd-tree: construcción

```
Algoritmo ConstruirKdTree(P, nivel)
Entrada: P de tamaño n
Salida: El árbol v resultado (su raíz)

INICIO
SI Tamaño(P) > 1
ENTONCES
    SI nivel es impar
    ENTONCES encontrar l y partir P en P1
    (con puntos con menor o igual abscisa
    que l) y en P2 (con mayor abscisa)

    SINO encontrar l y partir P en P1
    (con puntos con menor o igual ordenada
    que l) y en P2 (con mayor ordenada)

    v_izda <- ConstruirKdTree(P1, nivel+1)
    v_dech <-ConstruirKdTree(P2, nivel+1)
    Crear v con hijos v_izda y v_dech
    Devolver (v)
SINO Devolver (v <-P)
FIN
```



Eficiencia del kd-tree

- Construcción del árbol: O(n*log*n)
  - Coste de encontrar la línea divisoria: O(n) (aunque puede mejorarse utilizando cierto preprocesamiento)
  - La función de complejidad es:

$$T(n)= \begin{cases} O(1) \text{ si } n<=1 \\ 2T(n/2) + cn \text{ si } n>2 \end{cases} \Rightarrow O(n \log n)$$

- Búsqueda de un rango:
  - $O(n^{1/2}+k)$ para k datos en el rectángulo

Consideraciones finales

- Las estructuras de datos multidimensionales o espaciales permiten trabajar eficientemente con dos atributos de un dato al mismo tiempo
- Los ejemplos del tema son 2D, pero tanto las mallas regulares como los 2d-tree son extensibles a tres o más dimensiones
- El problema de las mallas regulares es que no son adaptativas
- Los kd-tree sí son adaptativos pero tienen el inconveniente de que la búsqueda no es logarítmica
  - Se visitan muchos nodos fuera del rectángulo R

Consideraciones finales

- Las estructuras de datos multidimensionales o espaciales permiten trabajar eficientemente con dos atributos de un dato al mismo tiempo
- Los ejemplos del tema son 2D, pero tanto las mallas regulares como los 2d-tree son extensibles a tres o más dimensiones
- El problema de las mallas regulares es que no son adaptativas
- Los kd-tree sí son adaptativos pero tienen el inconveniente de que la búsqueda no es logarítmica
  - Se visitan muchos nodos fuera del rectángulo R