



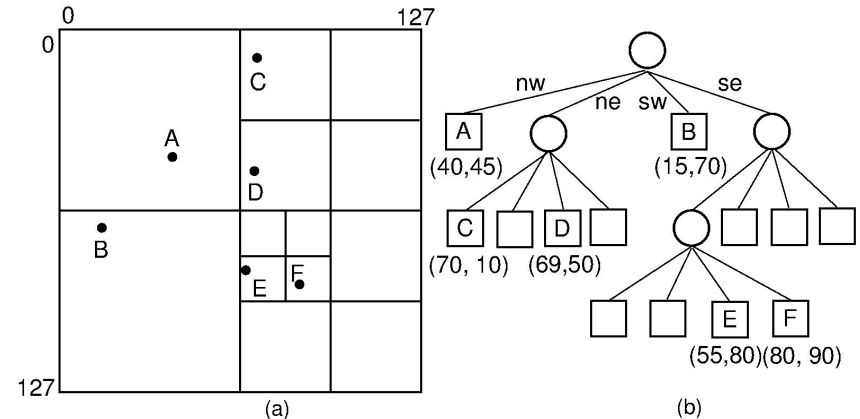
Universidad de Jaén

Mallas regulares y Quadrees

Lección 17:

- EEDD multidimensionales
- Mallas regulares
- Quadrees

Estructuras de Datos



Parte I: Introducción a las EEDD espaciales

- Motivación
- Estructuras de datos multidimensionales/espaciales
- Aplicaciones

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
Ejemplo: las farmacias de Jaén

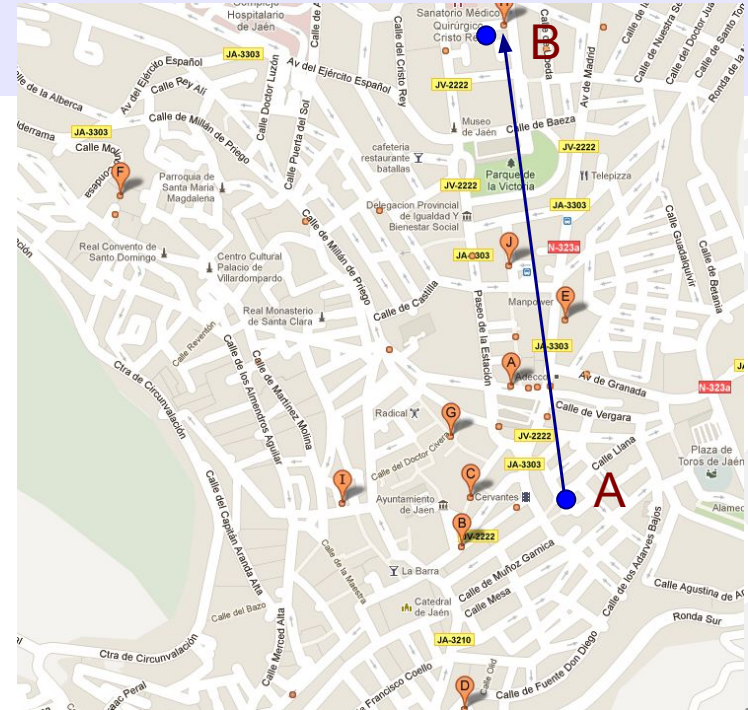
Desea acelerar la búsqueda para atender n peticiones solicitando el servicio para m comercios posibles considerando que los usuarios se pueden mover



Motivación

Ha pensado en utilizar una estructura de datos tipo 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



Estructuras de datos Multidimensionales

- Muchas de las entidades que deben guardar las EEDD son de naturaleza bidimensional o tridimensional
- Hasta el momento las entidades manejadas usaban relaciones de orden en base a una magnitud o atributo de la clase
 - 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

Estructuras de datos 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
- Los datos (coordenadas) tienen valores continuos pero estas EEDD dividen el espacio en regiones disjuntas
- Un dato representado por un punto 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

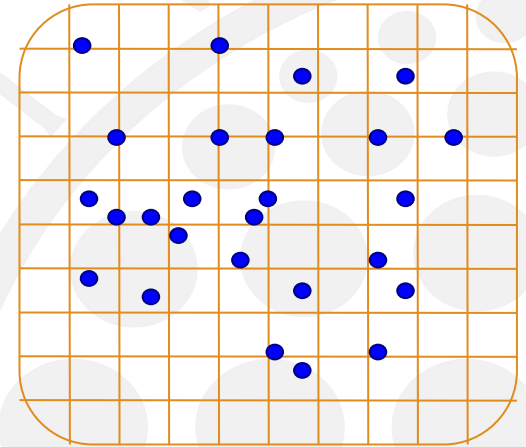
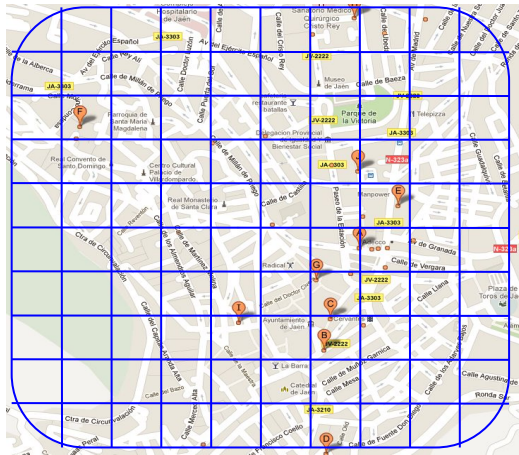
- Se utilizan en muchas disciplinas relacionadas con las ingenierías, por ejemplo la Informática Gráfica
- Encontrar el/los puntos más cercanos a uno dado
 - Encontrar las farmacias más cercanas a un usuario con móvil y GPS
- Conocer si un punto pertenece o no a una región
 - El punto representa por 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?

Parte II: Mallas regulares / grids

- Definición del concepto de malla regular
- Operaciones con mallas regulares
- Implementación
- Eficiencia

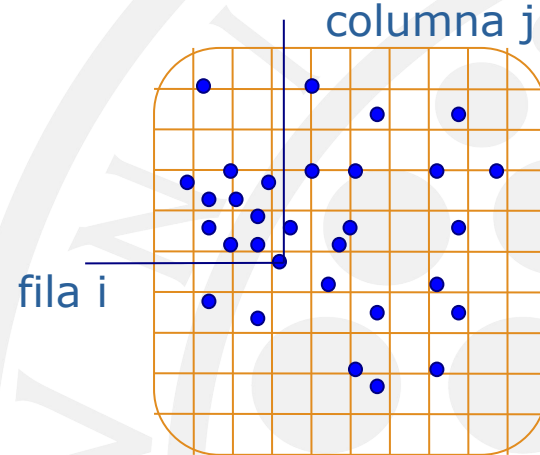
Mallas regulares o grids 2D

- Un grid en 2D, divide el plano en regiones rectangulares todas del mismo tamaño, permitiendo acceso directo al igual que una matriz. Concepto extensible a 3D
- Cada una de estas regiones representa 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 regulares

- **Crear malla:** definir el tamaño del vector es importante de cara a la eficiencia
 - 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

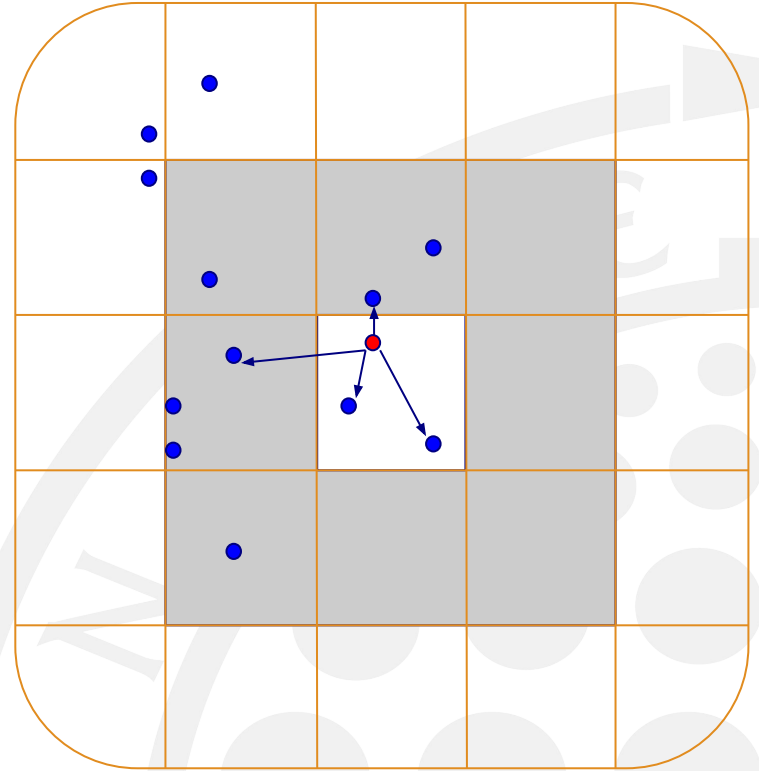


Operaciones con mallas regulares

- **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

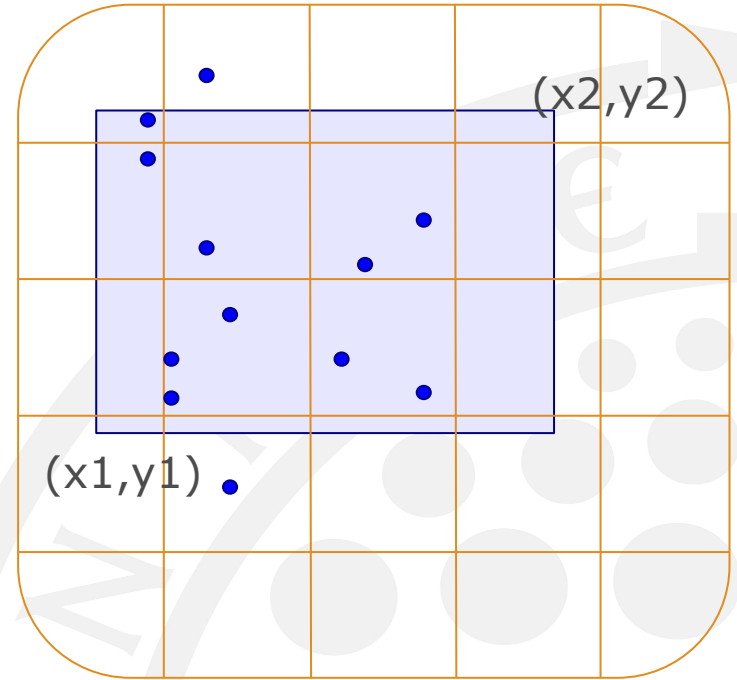
Operaciones con mallas regulares

- **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 punto más cercano de la casilla donde está p y guardarlo
 - Localizar el más cercano de los 8 vecinos
 - Quedarse con el menor de los dos
 - Cuidado: si no hay ningún otro punto en la casilla localizada, extender la búsqueda al siguiente conjunto de vecinos



Operaciones con mallas regulares

- 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: clase Casilla

```
template<typename T>
class Casilla{
    list<T> puntos;
public:
    friend class MallaRegular<T>;
    Casilla(): puntos() {}
    void insertar(const T &dato) { puntos.push_back(dato); }
    T *buscar(const T &dato);
    bool borrar(const T &dato);
};
```

T se instancia a punto o a objetos con coordenadas

```
template<typename T>
T *Casilla<T>::buscar(const T& dato){
    typename list<T>::iterator it;
    it = puntos.begin();
    for (;it != puntos.end(); ++it){
        if (*it == dato)
            return &(*it);
    }
    return 0;
}
```

Búsquedas
secuenciales

```
template<typename T>
bool Casilla<T>::borrar(const T& dato){
    typename list<T>::iterator it;
    it = puntos.begin();
    for (;it != puntos.end(); ++it){
        if (*it == dato) {
            puntos.erase(it);
            return true;
        }
    }
    return false;
}
```

Implementación de mallas regulares

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

Importante: para que no haya problemas de desbordamiento en los puntos extremos, mejor crear la malla unas décimas mayor que los datos albergados

```
template<typename 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(float aXMin, float aYMin, float aXMax, float aYMax, int aNDiv);

    void insertar(float x, float y, const T &dato);
    T *buscar(float x, float y, const T &dato);
    bool borrar(float x, float y, const T &dato);
};
```

Implementación de mallas regulares

```
template<typename T>
MallaRegular<T>::MallaRegular(float aXMin, float aYMin, float aXMax, float aYMax, int aNDiv) :
xMin(aXMin), yMin(aYMin), xMax(aXMax), yMax(aYMax){
    tamaCasillaX = (xMax - xMin)/aNDiv;
    tamaCasillaY = (yMax - yMin)/aNDiv;
    mr.insert(mr.begin(), aNDiv, vector<Casilla<T> >(aNDiv));
}

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

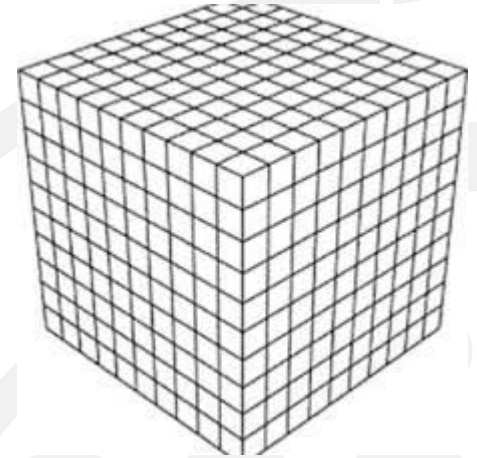
template<typename T>
void MallaRegular<T>::insertar(float x, float y, const T& dato){
    Casilla<T> *c = obtenerCasilla(x,y);
    c->insertar(dato);
}

template<typename T>
bool MallaRegular<T>::borrar(float x, float y, const T& dato){
    Casilla<T> *c = obtenerCasilla(x,y);
    return c->borrar(dato);
}
```

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

Eficiencia de las mallas regulares

- Es extensible a 3D o k-dimensiones
- Las mallas regulares pueden ser muy eficientes, con tiempo cercano al $O(1)$ pero sólo 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

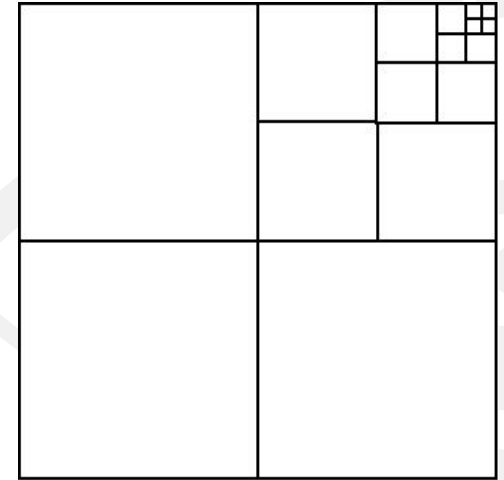


Parte III: Quadrees

- Definición de los quadrees
- Implementación de operaciones
- Aplicaciones
- Consideraciones finales

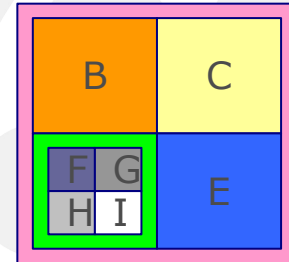
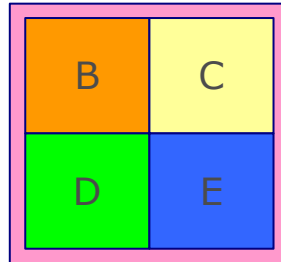
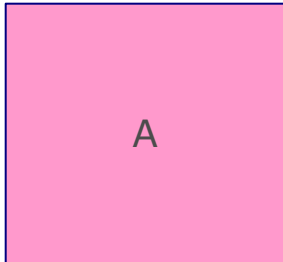
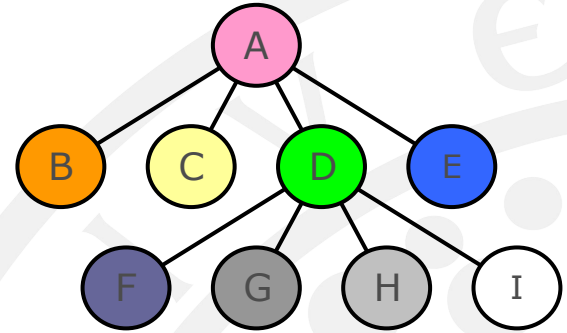
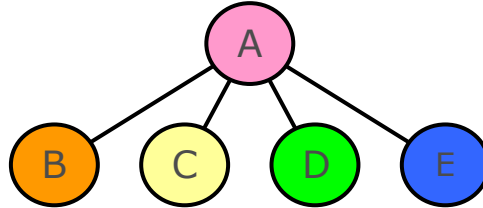
Quadtrees

- Los quadrees son EEDD adaptativas que descomponen el plano de forma recursiva en cuatro cuadrantes disjuntos de igual tamaño
- Se implementan mediante árboles; el nodo raíz representa todo el plano objeto de estudio
- Cada nodo no hoja tiene **4 nodos hijos** que representan una partición ortogonal de la misma forma que el nodo padre pero de menor tamaño
- Una región se vuelve a dividir si no cumple la condición de parada:
 - p.e: contiene muchos puntos



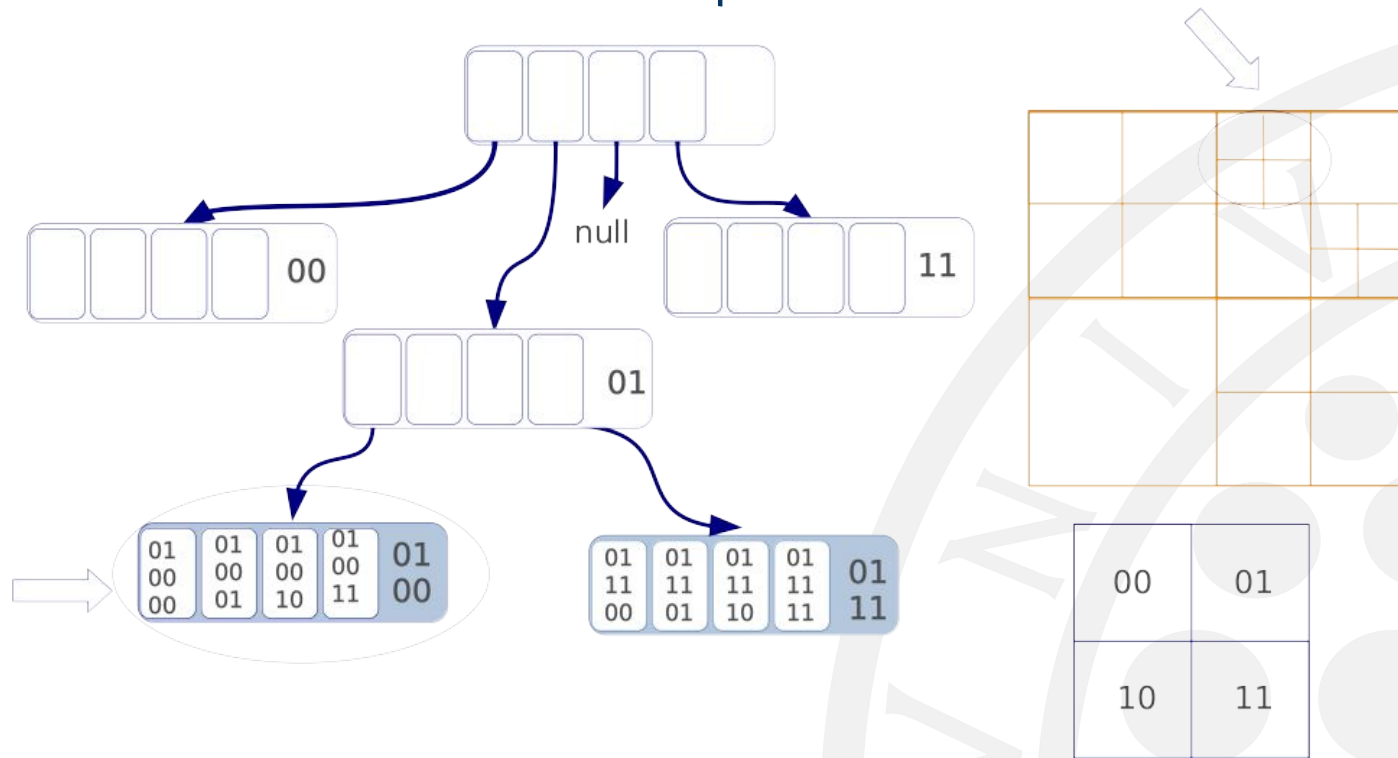
Quadtrees

- El árbol se adapta a la distribución de puntos y no tiene por qué ser equilibrado
- El tiempo esperado de acceso es por tanto $O(\log_4 n)$



Quadtrees: creación

- Definición recursiva de los nodos del quadtree



Quadtrees: EEDD

- Cada nodo almacena **4 punteros**
- El puntero al padre es opcional, pero es interesantes para recorrer una región sin tener que comenzar el proceso desde la raíz
- El nodo raíz representa todo el escenario de trabajo **[xMin, yMin][xMax, yMax]**
- Los nodos hoja mantienen una lista con los puntos ubicados en la región que representa
- Puede ser que esa lista esté vacía si no hay puntos
- La clase QuadTree tiene un apuntador al nodo raíz
- Es necesario definir la condición de parada: por ejemplo que no haya más de un **MAX_PUNTOS_CAJA**

Quadtrees:

Rectángulo y Entrada

```
class Rect {  
public:  
    float xi, yi, xs, ys;  
  
    Rect(float aXi = 0.0f, float aYi = 0.0f,  
         float aXs = 1.0f, float aYs = 1.0f) :  
        xi(aXi), yi(aYi), xs(aXs), ys(aYs) {}  
  
    void centro(float& xc, float& yc) const {  
        xc = 0.5f * (xi + xs);  
        yc = 0.5f * (yi + ys);  
    }  
};
```

```
template<typename T>  
class Entrada {  
    float x, y;  
    T dato;  
    friend class QuadTree<T>;  
  
public:  
    Entrada(float aX, float aY, const T& aDato) :  
        x(aX), y(aY), dato(aDato) {}  
};
```

Quadtrees: clase Nodo

```
#define MAX_PUNTOS_CAJA 10

template<typename T>
class Nodo {
    Nodo<T> *hijos[4]; //ne(00), ne(01), so(10), so(11)
    list<Entrada<T> > datos;
    friend class QuadTree<T>;

public:
    Nodo() : datos() {
        hijos[0] = hijos[1] = hijos[2] = hijos[3] = 0;
    }

    bool esHoja() {
        return !(hijos[0] || hijos[1] || hijos[2] || hijos[3]);
    }
};
```


Quadtrees: clase Quadtree

```
template<typename T>
class QuadTree {
    Rect rectRaiz;
    Nodo<T> *raiz;

    Nodo<T>*& localizar(Nodo<T>*& nodo, const Rect& rect, Rect& rectNodo,
        float x, float y, const T& dato);
    void insertar(Nodo<T>*& nodo, const Rect& rectNodo,
        float x, float y, const T& dato);
    void eliminar(Nodo<T> *nodo);
public:
    QuadTree(float xMin, float yMin, float xMax, float yMax) :
        rectRaiz(xMin, yMin, xMax, yMax) {
        raiz = new Nodo<T>();
    }
    ~QuadTree() {
        eliminar(raiz);
    }
    void insertar(float x, float y, const T& dato);
    T *buscar(float x, float y, const T& dato);
    bool borrar(float x, float y, const T& dato);
};
```

Quadtrees: operaciones

- **Buscar:** un punto $p=(px,py)$
 - Comenzando por la raíz realizar una búsqueda recursiva por el cuadrante correspondiente hasta llegar a un nodo hoja
 - Hacer una búsqueda secuencial en la lista de datos del nodo
- **Insertar:** un punto $p=(px,py)$
 - Realizar una búsqueda del nodo hoja que le corresponde
 - Insertar el dato en la lista del nodo
 - Si el número de datos supera MAX_PUNTOS_CAJA, mover los puntos en los hijos correspondientes (creando los hijos)

Quadtrees: operaciones públicas

```
template<typename T>
void QuadTree<T>::insertar(float x, float y, const T& dato) {
    Rect rectNodo;
    Nodo<T>*& nodo = localizar(raiz, rectRaiz, rectNodo, x, y, dato);
    insertar(nodo, rectNodo, x, y, dato);
}

template<typename T>
T *QuadTree<T>::buscar(float x, float y, const T& dato) {
    Rect rectNodo;
    Nodo<T> *nodo = localizar(raiz, rectRaiz, rectNodo, x, y, dato);

    if (nodo) {
        typename list<Entrada<T> >::iterator i = nodo->datos.begin();
        for (; i != nodo->datos.end(); ++i)
            if (i->dato == dato) return &(i->dato);
    }

    return 0;
}
```

```

template<typename T>
void QuadTree<T>::insertar(Nodo<T>*& nodo, const Rect& rect, float x, float y, const T& dato) {
    if (nodo == 0)
        nodo = new Nodo<T>();

    nodo->datos.push_back(Entrada<T>(x, y, dato));

    if (nodo->datos.size() > MAX_PUNTOS_CAJA) {
        float xCentro, yCentro;
        rect.centro(xCentro, yCentro);

        typename list<Entrada<T> >::iterator i = nodo->datos.begin();
        for (; i != nodo->datos.end(); ++i) {
            if (i->x <= xCentro && i->y <= yCentro)
                insertar(nodo->hijos[2], Rect(rect.xi, rect.yi, xCentro, yCentro),
                    i->x, i->y, i->dato); //10
            else if (i->x <= xCentro && yCentro < i->y)
                insertar(nodo->hijos[0], Rect(rect.xi, yCentro, xCentro, rect.ys),
                    i->x, i->y, i->dato); //00
            else if (xCentro < i->x && i->y <= yCentro)
                insertar(nodo->hijos[3], Rect(xCentro, rect.yi, rect.xs, yCentro),
                    i->x, i->y, i->dato); //11
            else
                insertar(nodo->hijos[1], Rect(xCentro, yCentro, rect.xs, rect.ys),
                    i->x, i->y, i->dato); //01
        }
        nodo->datos.clear();
    }
}

```

Quadtrees: inserción (privada)

```

template<typename T>
Nodo<T>*& QuadTree<T>::localizar(Nodo<T>*& nodo, const Rect& rect, Rect& rectNodo,
    float x, float y, const T& dato) {
    if (!nodo || nodo->esHoja()) {
        rectNodo = rect;
        return nodo;
    }

    float xCentro, yCentro;
    rect.centro(xCentro, yCentro);

    if (x <= xCentro && y <= yCentro)
        return localizar(nodo->hijos[2], Rect(rect.xi, rect.yi, xCentro, yCentro),
            rectNodo, x, y, dato); // 10
    else if (x <= xCentro && yCentro < y)
        return localizar(nodo->hijos[0], Rect(rect.xi, yCentro, xCentro, rect.ys),
            rectNodo, x, y, dato); // 00
    else if (xCentro < x && y <= yCentro)
        return localizar(nodo->hijos[3], Rect(xCentro, rect.yi, rect.xs, yCentro),
            rectNodo, x, y, dato); // 11
    else
        return localizar(nodo->hijos[1], Rect(xCentro, yCentro, rect.xs, rect.ys),
            rectNodo, x, y, dato); // 01
}

```

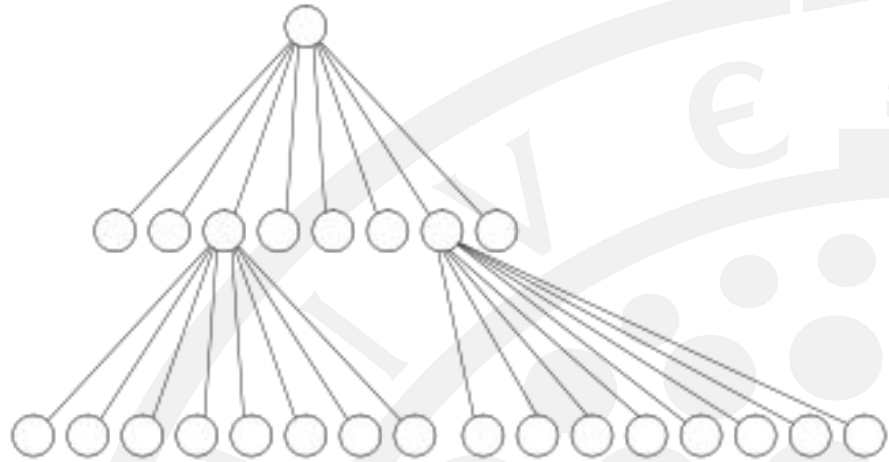
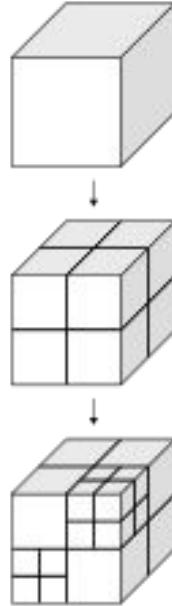
Quadtrees: localizar (privada)

Quadtrees: borrado

- **Borrado:** un punto $p=(p_x, p_y)$
 - Realizar la búsqueda del dato correspondiente
 - Eliminar la entrada de la lista del nodo hoja
 - Si la suma del número de datos guardados en el nodo y sus hermanos cae por debajo de $2/3 * \text{MAX_PUNTOS_CAJA}$ entonces pasar todos los datos del nodo y sus hermanos al padre y eliminar los nodos

Quadtrees/Octrees

- Los **quadtrees** particionan el plano, este concepto es fácilmente extensible al espacio con los octrees
- Un **octree** divide recursivamente la zona del espacio que representa en 8 subdivisiones
- La definición y los métodos vistos son extensibles a 3D

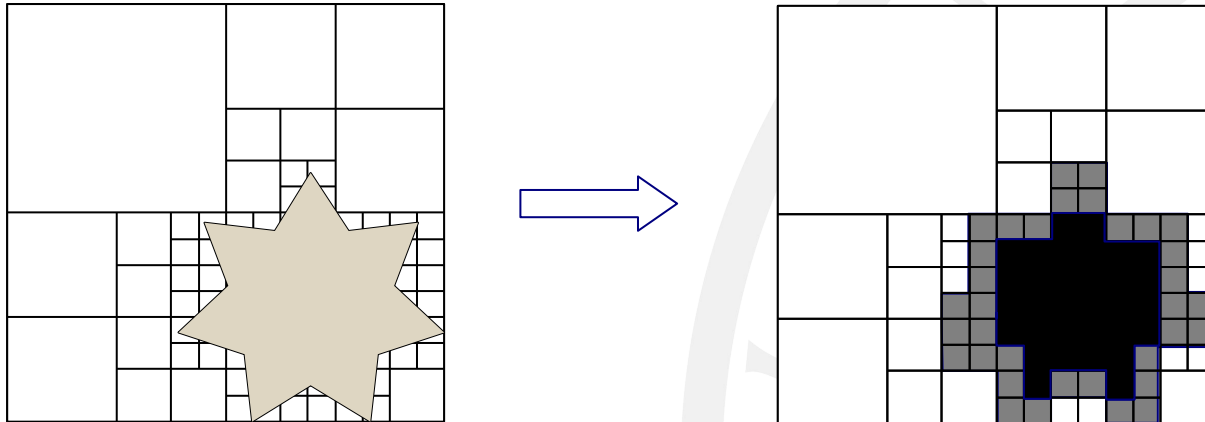


Aplicaciones

- Los quatrees/octrees tienen numerosas aplicaciones en Sistemas de Información Geográficos, Informática Gráfica, Videojuegos, etc.

Representación de una escena con obstáculos para detección de colisiones:

- Nodos blancos: no hay colisión
- Nodos grises/negros: hay (negros)o puede haber colisión (grises)



Aplicaciones

- Un quadtree también sirve para comprimir imágenes
- El proceso de descomposición detecta áreas monocolor que son representadas por un nodo hoja con el color correspondiente
- El proceso de descomposición continúa hasta que todas las hojas tengan un único color

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 quadtrees son extensibles a tres o más dimensiones
- El problema de las mallas regulares es que no son adaptativas
- Los quadtrees sí son adaptativos pero los algoritmos de creación y borrado son más complejos e ineficientes

De ahora en adelante...

- Hasta ahora hemos asumido que toda la información se queda en memoria principal
- Sin embargo, cualquier aplicación real tomará datos de entrada de ficheros, pero también puede guardar información en disco durante la ejecución del programa
- Para esto veremos EEDD en ficheros

Lección 18: Ficheros