UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

QUAD TREE

Curso : Estructuras de Datos Avanzadas

Alumno

HERRERA COOPER MIGUEL ALEXANDER

Docente

MACHACA ARCEDA, VICENTE

Índice

Índice **1**

1. Introduction **2**

2. Quad Tree **3**
2.1. Definición . . . 3
2.2. Construyendo un Quad Tree . . . 4
2.3. Quad Tree no balanceados . . . 5
2.4. Longitudes Laterales . . . 5
2.5. Propiedades . . . 6
2.6. Encontrar un vecino . . . 6
2.7. Quad Tree Balanceado . . . 7

3. Actividad **8**
3.1. A . . . 8
3.2. B . . . 9
3.3. C . . . 9
3.4. D . . . 11
3.5. E . . . 12

4. Pruebas - F - G **12**

5. Conclusión **13**

1. Introduction

Una gran variedad de estructuras jerárquicas existen para representar los datos espaciales. Una técnica normalmente usada es Quadtree. El desarrollo de éstos fue motivado por la necesidad de guardar datos que se insertan con valores idénticos o similares.

Quadtree también se usa para la representación de datos en los espacios tridimensionales o con hasta 'n' dimensiones.

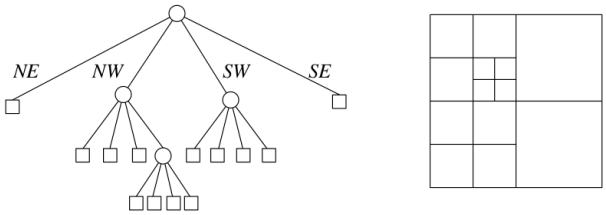
Los Quadtree se usa para representar puntos, áreas, curvas, superficies,mallas y volúmenes. La descomposición puede hacerse en las mismas partes en cada nivelado (la descomposición regular), o puede depender de los datos de la entrada.

La resolución de la descomposición, en otros términos,es el número de tiempos en que el proceso de descomposición es aplicado, puede tratarse de antemano, o puede depender de las propiedades de los datos de la entrada es decir su capacidad maxima.

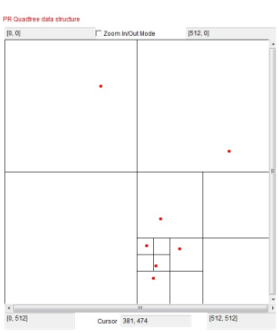
2. Quad Tree

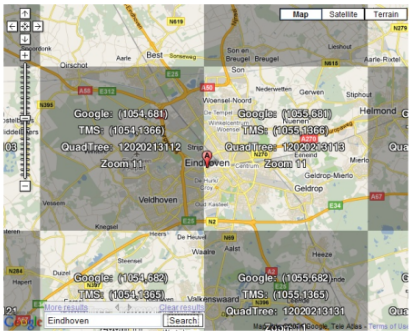
2.1. Definición

- Es una generalización del árboles binarios para el tratamiento de los datos que juegan escencialmente en el espacio bidimensional.
- Cada registro tiene asociado un punto del espacio bidimensional que se almacena en un nodo de orden 4.
- La raíz del árbol divide el espacio en 4 cuadrantes denominados, por analogía con los mapas: NE, NO, SO y SE.
- La raíz de cada subárbol divide a cada uno de estos cuadrantes en 4 subcuadrantes y el proceso se repite, recursivamente, hasta alcanzar un nodo que no tenga hijos.



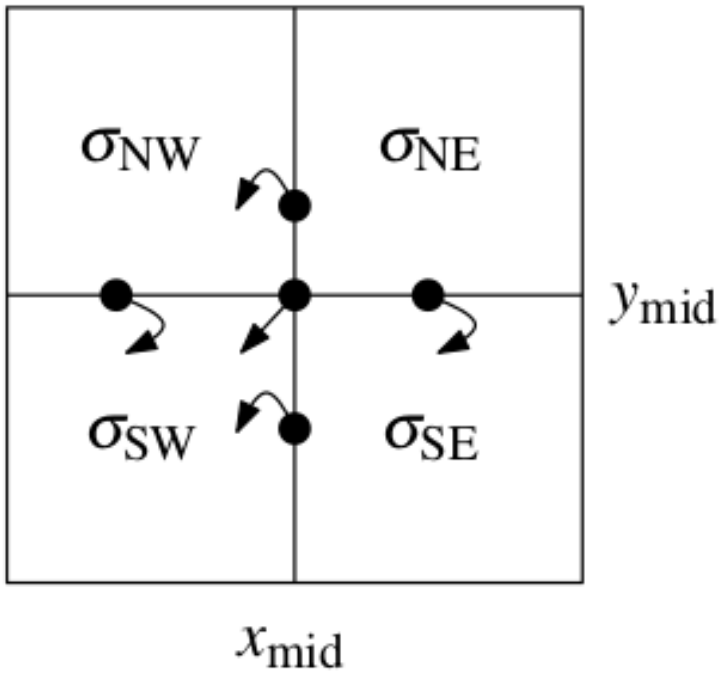
Ejemplo:





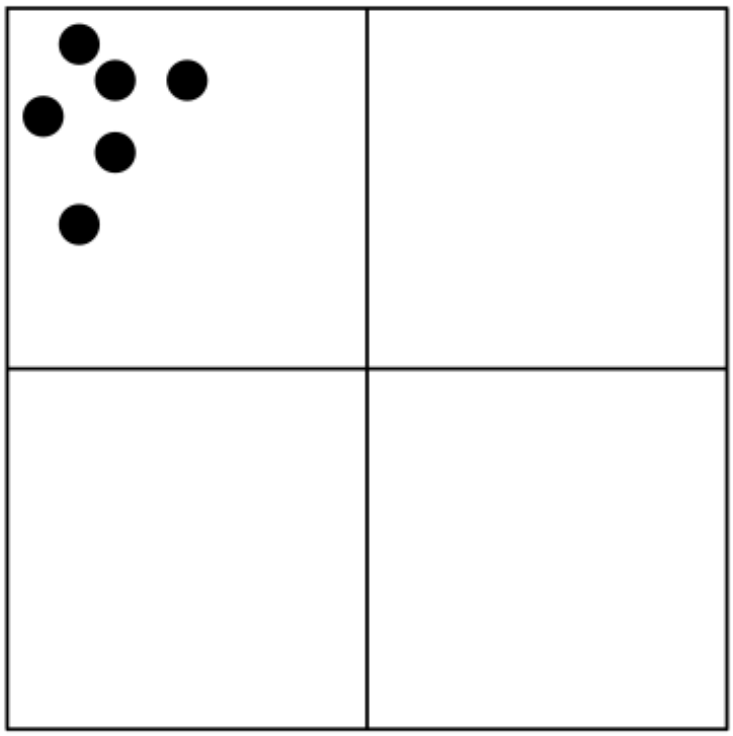
2.2. Construyendo un Quad Tree

- Puede almacenar diferentes tipos de datos; aquí: conjuntos de puntos.
- **Principio:**: dividir cuadrados hasta que cada cuadrado contenga 1 punto.
- **Construcción Recursiva:**
  Comenzar con un cuadrado que contenga todo puntos; si hay más de 1 punto en el cuadrado entonces:
  - Dividir el cuadrado en 4 cuadrantes.
  - Asignar puntos a cuadrados.
  - Recurrir en cada uno de los cuadrantes.



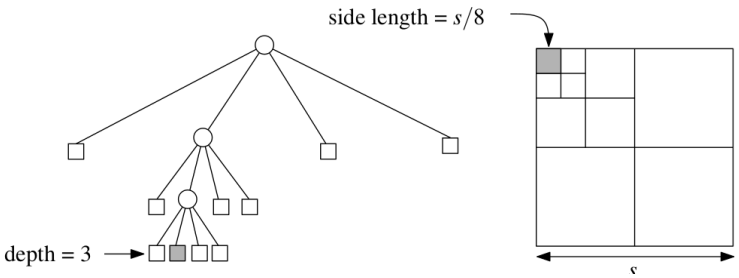
2.3. Quad Tree no balanceados

Los quadtrees se desequilibran cuando muchos puntos se encuentran juntos.



2.4. Longitudes Laterales

Longitudes laterales de cuadrados en un quadtree reducido a la mitad con profundidad creciente.



2.5. Propiedades

- **Lemma.**
  Sea c la distan más pequeña entre dos puntos en un conjunto de puntos P, y sea s la longitud del lado del cuadrado inicial (más grande). Entonces la profundidad de un quadtree para P es como máximo log (s / c) + 3/2.

- **Lemma**
  Un quadtree de profundidad d almacenando n puntos tiene O ((d + 1) n) nodos y se pueden construir en O ((d + 1) n). tiempo.

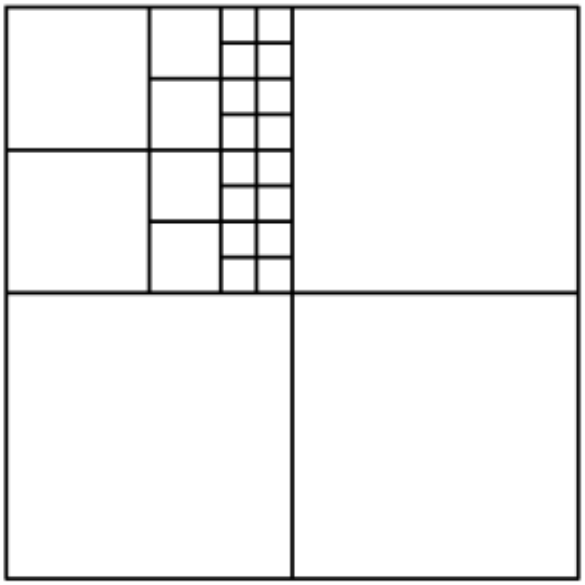
2.6. Encontrar un vecino

- Quad Tree es una subdivisión en regiones
- Operación típica: moverse entre regiones
- Búsqueda de vecinos: dado un nodo v (un cuadrado) y una dirección (norte, sur, oeste u este), qué nodo (cuadrado) es adyacente a v en la dirección dada?
- **Algoritmo para encontrar un vecino**.
  Entrada: un nodo v en un quadtree T .
  Salida: vecino norte de v.

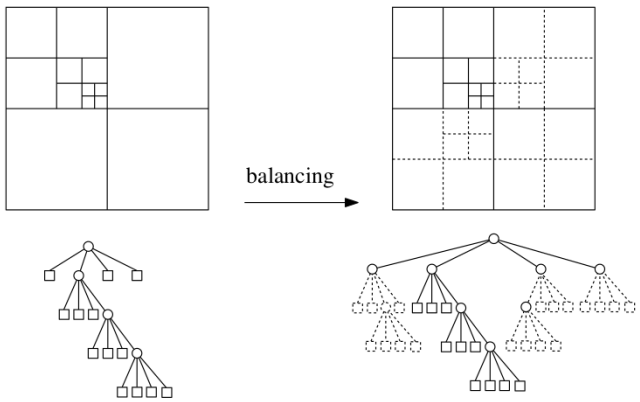
  - Si v es un niño SW o SE, entonces su vecino del norte es un hermano.
  - de lo contrario, suba T hasta alcanzar un nodo w que sea un SW o SE-hijo.
    - Si esto no existe, devuelva nulo
  - y descender hacia el vecino norte de w encontrando el movimiento SW o SE a la misma profundidad que v.
  - esto también funciona para el sur, oeste, este
  - **Tiempo de ejecución:** tiempo O (d + 1) en un quadtree de profundidad d.

2.7. Quad Tree Balanceado

- Un QuadTree se equilibra si dos nodos vecinos difieren en más 1 en profundidad. una subdivisión desequilibrada de QuadTree.



- **Balanceando un QuadTree**
  agregar nodos!!



- **Algoritmo para Balancear un Quad Tree**
  - Entrada : QuadTree T.
  - Salida : QuadTree balanceado versión T.

```
1. insert all leaves of T into a linear list L
2. while L is not empty
3.  do remove a leave    from L
4.   if   (  ) has to be split
5.    then make    an internal node with 4 children;
        if    stores a point, store it in the correct leave
6.      insert the 4 new leaves into L
7.      check if   (  ) had neighbors that now need to be split,and
if so, insert them into L.
8. return T
```

- **Complejidad de un Árbol Balanceado Teorema**
  Sea T un quadtree con m nodos. Entonces la versión equilibrada de T tiene nodos O (m) y puede ser construido en O ((d + 1) m) tiempo.

3. Actividad

3.1. A

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>quadtree</title>
        <script type="text/javascript" src="js/p5.min.js"></script>
        <script type="text/javascript" src="quadtree.js"></script>
  <script type="text/javascript" src="sketch.js"></script>
</head>
<body>
</body>
</html>
```

3.2. B

```
contains(point) {
    return (point.x >= this.x - this.w &&
      point.x < this.x + this.w &&
      point.y >= this.y - this.h &&
      point.y < this.y + this.h);
  }

  intersects(range) {
    return !(range.x - range.w > this.x + this.w ||
      range.x + range.w < this.x - this.w ||
      range.y - range.h > this.y + this.h ||
      range.y + range.h < this.y - this.h);
  }
```

3.3. C

```
subdivide() {
    let x = this.boundary.x;
    let y = this.boundary.y;
    let w = this.boundary.w;
    let h = this.boundary.h;

    let ne = new Rectangle(x + w / 2, y + h / 2, w / 2, h / 2);
    this.NE = new QuadTree(ne, this.capacity);
    let nw = new Rectangle(x - w / 2, y + h / 2, w / 2, h / 2);
    this.NO = new QuadTree(nw, this.capacity);
    let se = new Rectangle(x + w / 2, y - h / 2, w / 2, h / 2);
    this.SE = new QuadTree(se, this.capacity);
    let sw = new Rectangle(x - w / 2, y - h / 2, w / 2, h / 2);
    this.SO = new QuadTree(sw, this.capacity);


    this.divided = true;
  }
```

```
insert(point) {

  if (!this.boundary.contains(point)) {
    return;
  }

  if (this.points.length < this.capacity) {
    this.points.push(point);

  } else {
    if (!this.divided) {
      this.subdivide();
    }
    if (this.NE.insert(point)) {
      return;
    } else if (this.NO.insert(point)) {
      return;
    } else if (this.SE.insert(point)) {
      return;
    } else if (this.SO.insert(point)) {
      return;
    }
  }
}
```

3.4. D

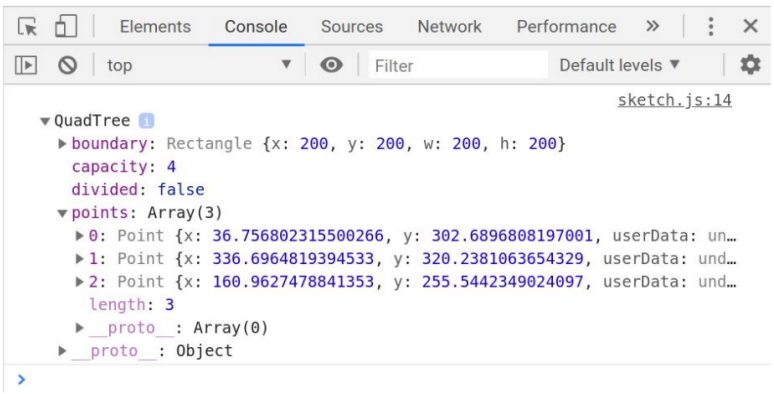
```
function setup() {
        createCanvas(400,400);

        let boundary =  new Rectangle(200,200,200,200);
        let qt = new QuadTree(boundary , 4);

        console.log(qt);
        //
        for (let i = 0; i<90; i++) {
                let p = new Point(Math.random() *400 , Math.random()*400);
                qt.insert(p);
        }


        background(100);
        qt.show();
}
```

3.5. E

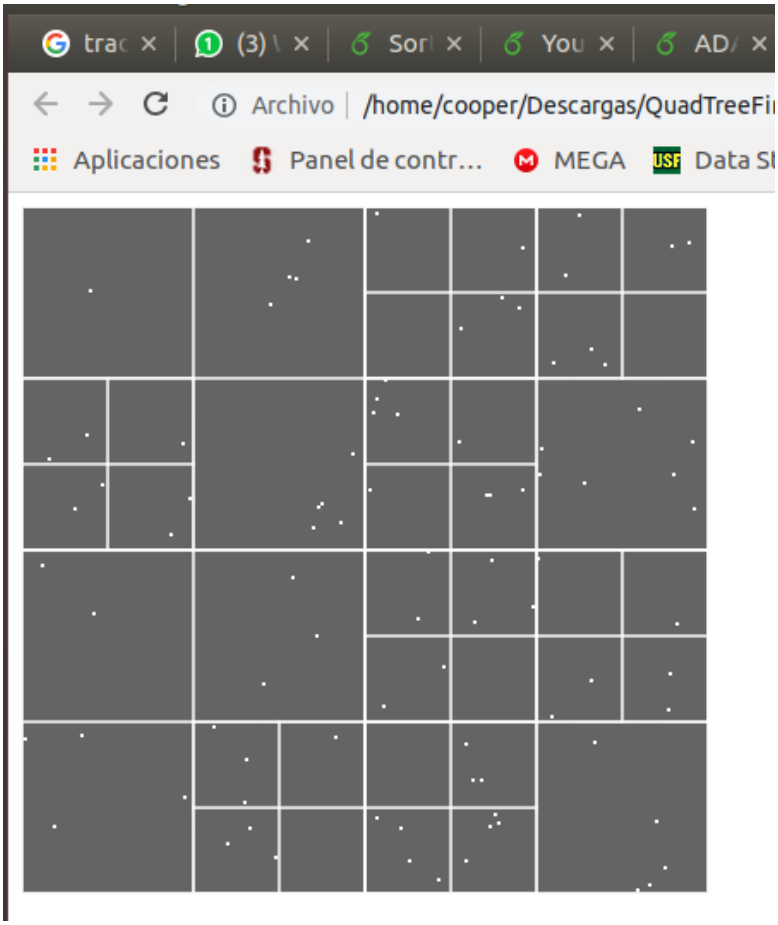


-

4. Pruebas - F - G

- Prueba con 3 pntos

- Insertando más Puntos



5. Conclusión

Las funcion de **Inserción** se utiliza para insertar un nodo en un Quad Tree existente. Esta función primero verifica si el nodo dado está dentro de los límites del quad actual. Si no es así, entonces cesamos inmediatamente la inserción. Si está dentro de los límites, seleccionamos el elemento secundario apropiado para contener este nodo en función de su ubicación. Esta función es O (Log N) donde N es el tamaño de la distancia.

Finalmente se puede apreciar que en al insertar 3 puntos estamos probando el correcto funcionamiento de nuestro QuadTree, seguidamente insertamos más puntos y de esta forma se comprueba la creación de las subdivisiones.