ACM程序设计实训(二)

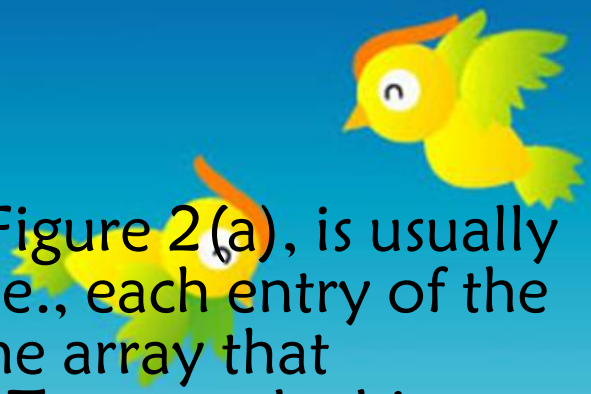
四分树

华南师大讲稿

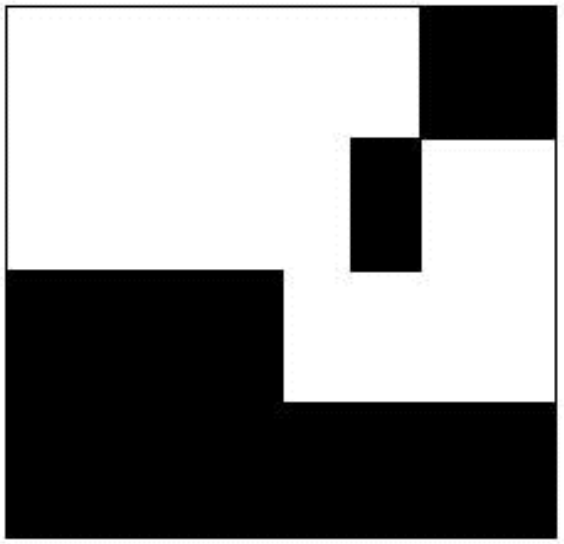
问题引入

- zju 1788

zoj 1788



- A binary image, such as the one shown in Figure 2(a), is usually represented as an array of binary entries, i.e., each entry of the array has value 0 or 1. Figure 2(b) shows the array that represents the binary image in Figure 2(a). To store the binary image of Figure 2(b), the so-called quad tree partition is usually used. For an N X N array, N <= 512 and N = 2^i for some positive integer i, if the entries do not have the same value, then it is partitioned into four N/2 X N/2 arrays, as shown in Figure 2(c). If an N/2 X N/2 array does not have the same binary value, such as the upper right and lower right N/2 X N/2 arrays in Figure 2(c), then we can divide it into four N/4 X N/4 arrays again. These N/4 X N/4 arrays in turn can also, if needed, be divided into four N/8 X N/8 arrays, etc.. The quad tree partition is completed when the whole array is partitioned into arrays of various size in which each array contains only one binary value. Figure 2(c) contains the arrays after the quad tree partition is completed.



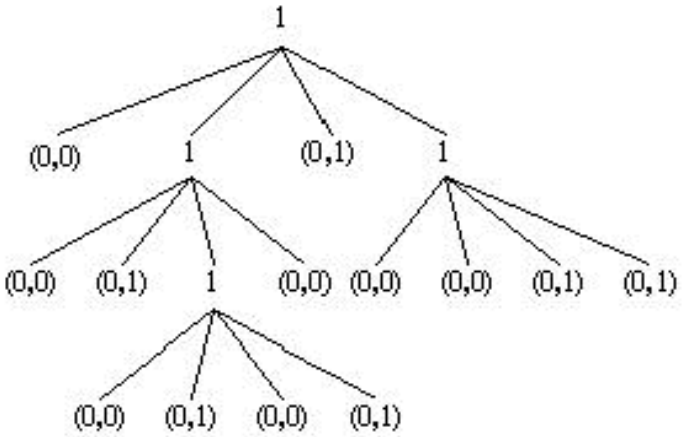
(a)

0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

(b)

0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

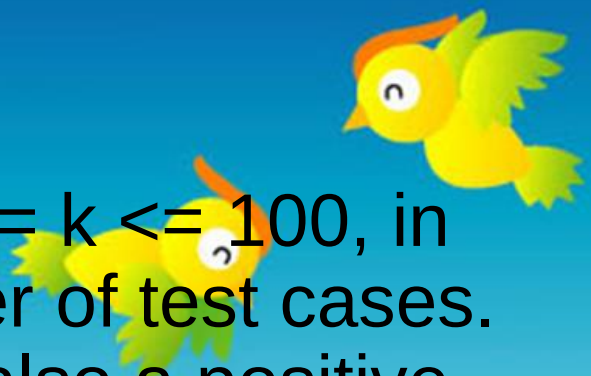
(c)



(d)

Figure 2: A binary image (a), its array representation (b), its quad tree partition (c), and its quad tree representation (d).

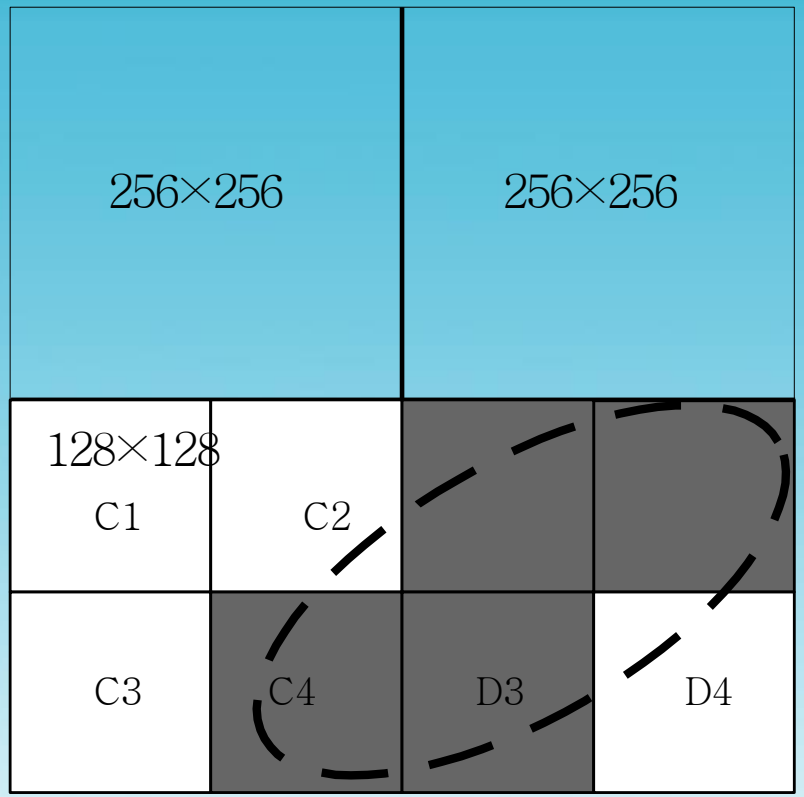
- Instead of storing the binary image of Figure 2(a), we only need to store the quad tree in the form as Figure 2(d) which is encoded from Figure 2(c). In Figure 2(d), each node represents an array of Figure 2(c) in which the root node represents the original array. If the value of a node in the tree is 1, then it means that its corresponding array needs to be decomposed into four smaller arrays. Otherwise, a node will have a pair of values and the first one is 0. It means that its corresponding array is not necessary to decompose any more. In this case, the second value is 0 (respectively, 1) to indicate that all the entries in the array are 0 (respectively, 1). Thus, we only need to store the tree of Figure 2(d) to replace storing the binary image of Figure 2(a). The way to store the tree of Figure 2(d) can be represented by the following code:
- (1)(0,0)(1)(0,1)(1)(0,0)(0,1)(1)(0,0)(0,0)(0,0)(0,1)(0,1)(0,0)(0,1)(0,0)(0,1).
- This code is just to list the values of the nodes from the root to leaves and from left to right in each level. Deleting the parentheses and commas, we can obtain a binary number 10010110001100000010100010001 which is equal to 258C0511 in hexadecimal. You are asked to design a program for finding the resulting hexadecimal value for each given image.



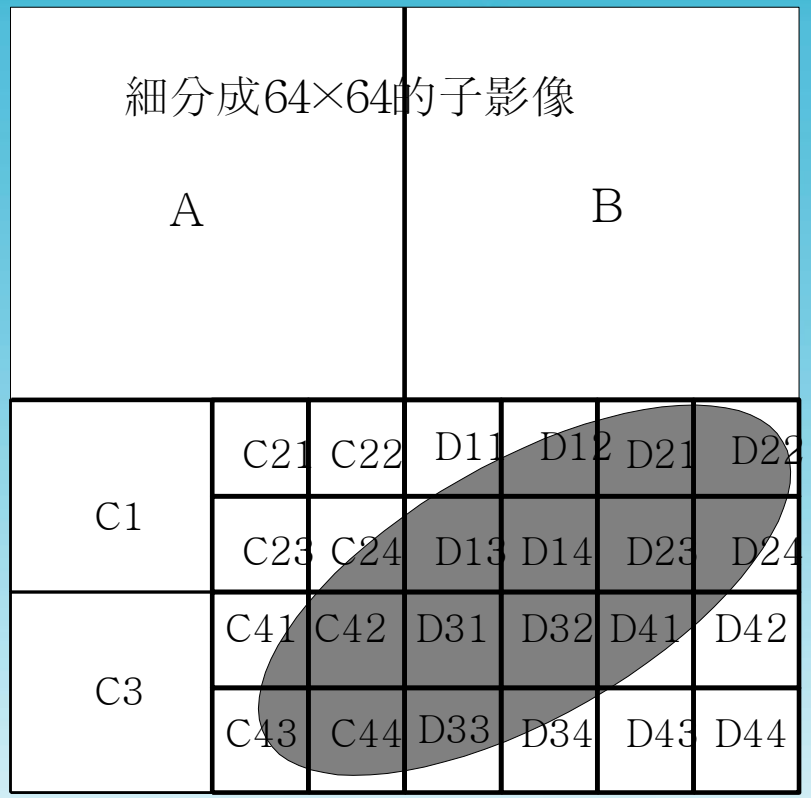
- **Input**
- There is an integer number k, 1 <= k <= 100, in the first line to indicate the number of test cases. In each test case, the first line is also a positive integer N indicating that the binary image is an N X N array, where N <= 512 and N = 2^i for some positive integer i. Then, an N X N binary array is followed in which at least one blank is between any two elements.
- **Output**
- The bit stream (in hexadecimal) used to code each input array.

- **Sample Input**
- 3
  2
  0 0
  0 0
  4
  0 0 1 1
  0 0 1 1
  1 1 0 0
  1 1 0 0
  8
  0 0 0 0 0 0 1 1
  0 0 0 0 0 0 1 1
  0 0 0 0 0 1 0 0
  0 0 0 0 0 1 0 0
  1 1 1 1 0 0 0 0
  1 1 1 1 0 0 0 0
  1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1
- **Sample Output**
- 0
  114
  258C0511

四分树的应用

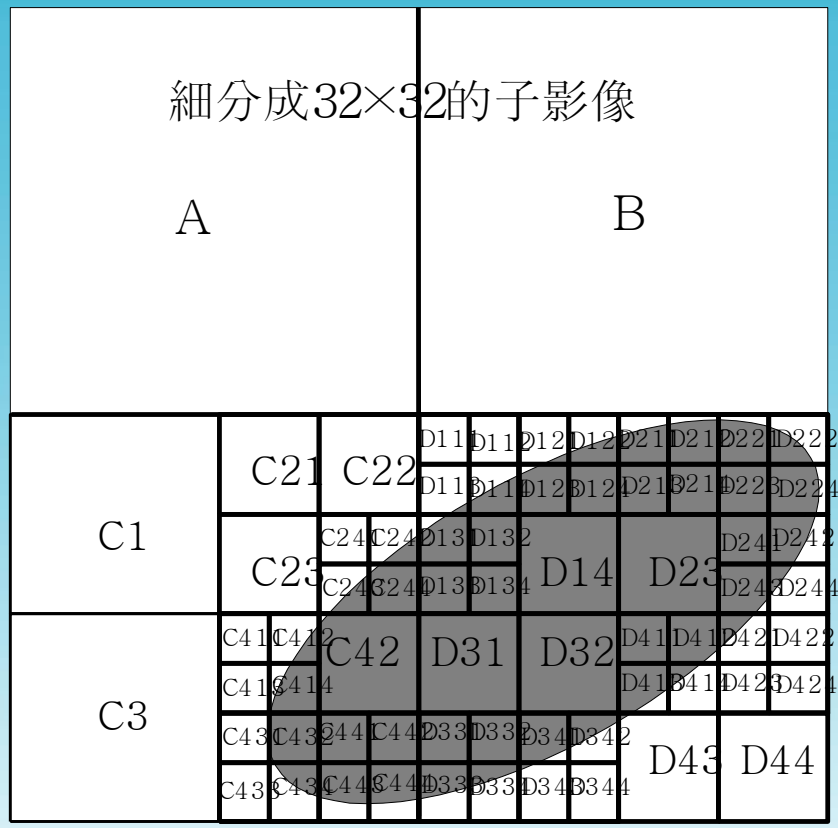

256×256
256×256
128×128
C1
C2
C3
C4
D3
D4

圖（c）

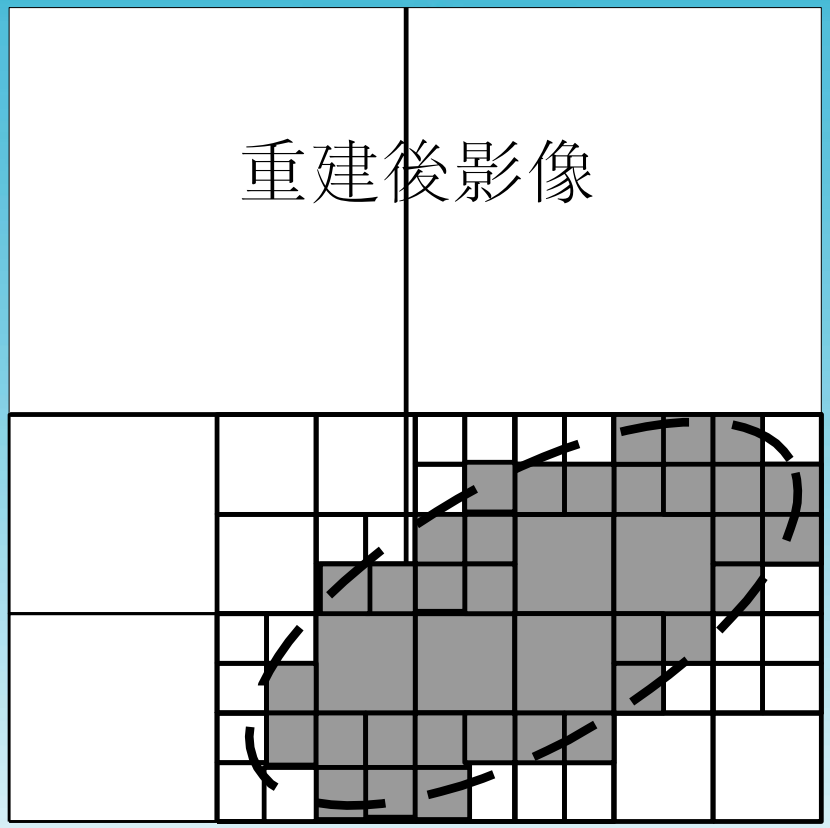

細分成64×64的子影像
A
B
C1
C21
C22
D11
D12
D21
D22
C23
C24
D13
D14
D23
D24
C3
C41
C42
D31
D32
D41
D42
C43
C44
D33
D34
D43
D44

圖（ d）

四分树的应用方法



圖（e)



圖（f)

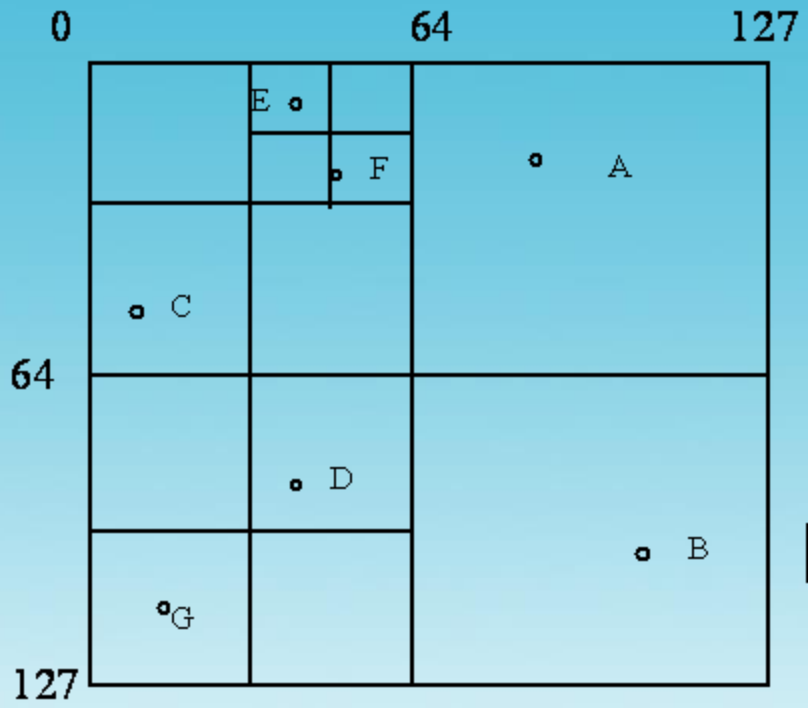
四分树

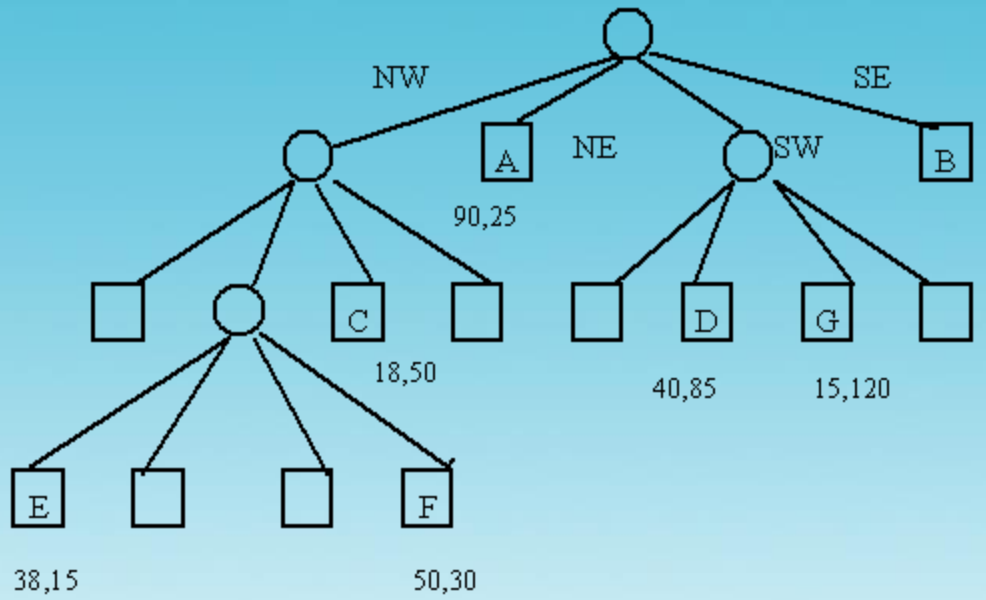
- 四分树数据结构是一种广受关注，被学者进行了大量研究的数据结构。它在20世纪60年代中期就被应用到加拿大地理信息系统中。代表性的研究学者有Klinger等人。
- 将图件覆盖的区域按照四个象限进行递归分割，直到子象限的属性码值变为单一为止或达到预定精度的网格大小为止。(对于达到预定精度的网格其覆盖的属性编码未能单一，则按占据面积大者的属性编码为该网格的属性编码)；凡数值是单一的单元，无论单元大小，均作为最后的存储单元。这就是常规四分树的建立过程。

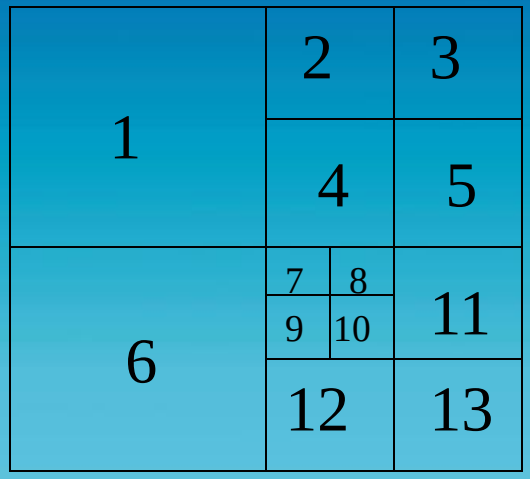
四分树

- PR四分树，即点-区域四分树（Point-Region Quadtree）：
  - 每个内部结点都恰好有四个子结点，是一棵完全四分树
  - 每个内部结点将当前空间均等地划分为四个区域
    - NW（西北）、NE（东北）、SW（西南）和SE（东南）
- 如果某区域包含的数据点数大于1，那么就把该区域继续均等的划分为四个区域；依此类推，直到每个区域所包含的数据点数不超过1为止。

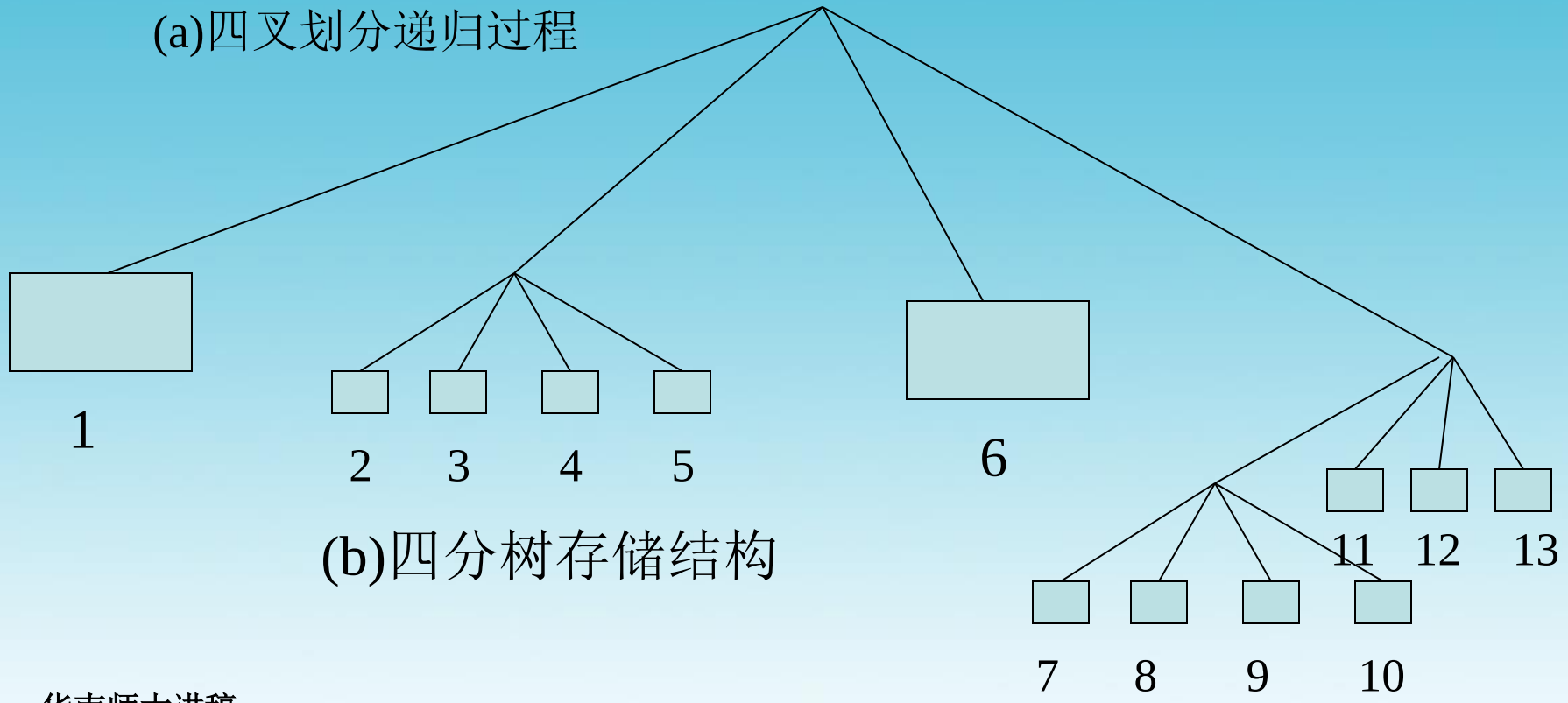
四分树的图示


0
64
127
E
F
A
C
D
B
G
64
127


NW
NE
SW
SE
A
B
90,25
C
D
G
18,50
40,85
15,120
E
F
38,15
50,30



(a)四叉划分递归过程



(b)四分树存储结构

编码过程

7	7	7	6	6	6	6	6
7	7	7	7	7	6	6	6
7	7	7	7	7	7	6	6
4	4	4	4	4	6	6	6
4	4	4	7	4	6	6	6
4	4	4	6	6	6	6	6
0	0	4	4	6	6	6	6
0	0	0	0	6	6	0	0

图 4-19 栅格数据

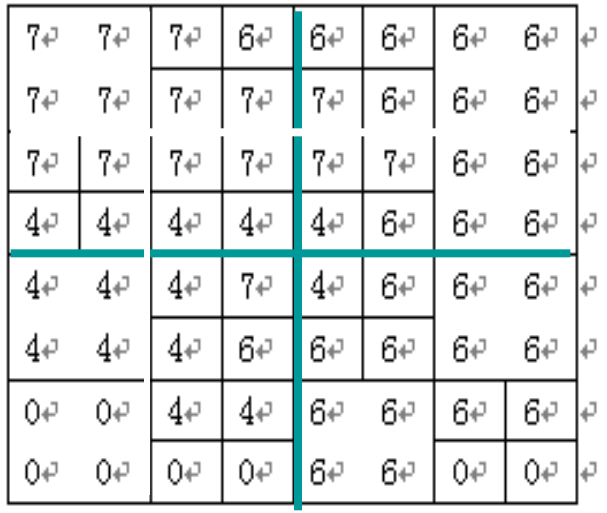
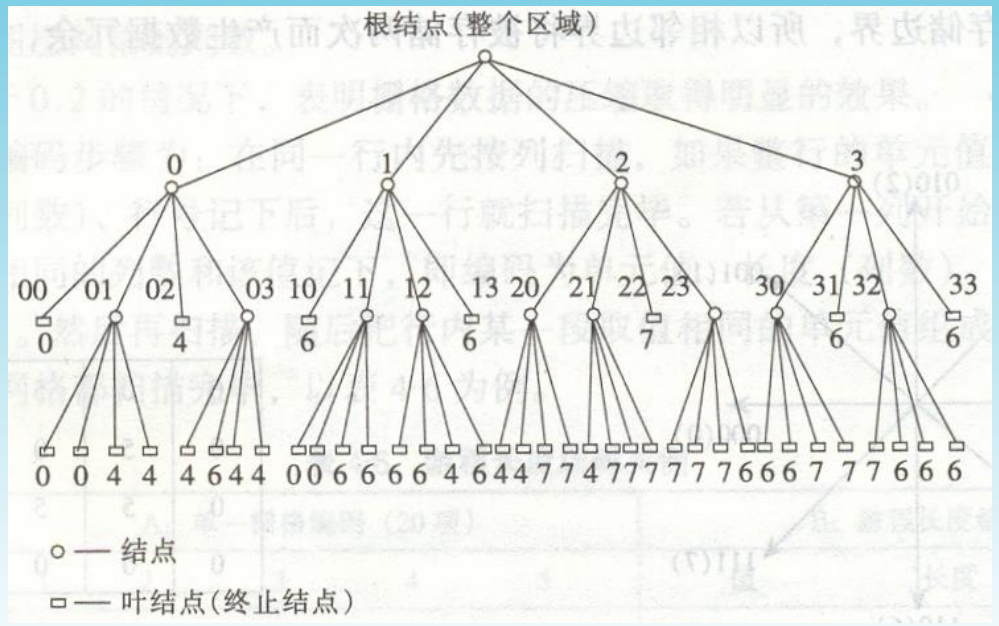


图 4-20 四叉树编码结构象限图

地址编码

图 4-22 四叉树地址编码



四分树的划分

- 上图所表示的PR四分树，其对象空间为128×128，并且其中包含点A、B、C、D、E、F和G
- 根结点的四个子结点把整个空间平分为四份大小为64×64的子空间
  - NW，NE，SW，SE
  - NW（包含三个数据点）和SE(包含两个数据点) 需要进一步分裂

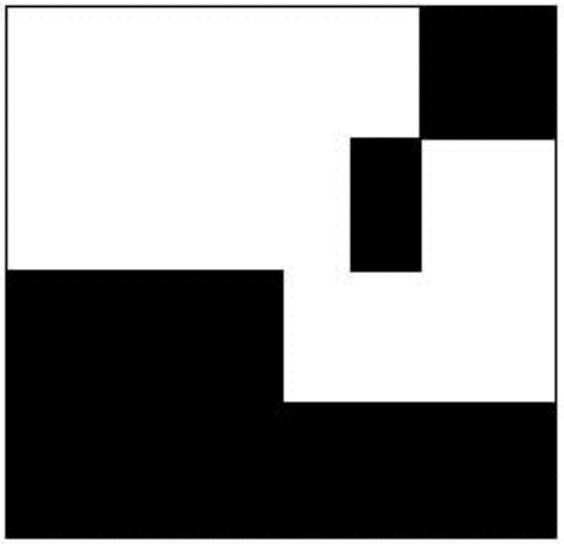
四分树的分割方法

- 从上而下建立（top-down）

  先检查全区域，内容不同的再四分割，往下逐次递归。这种方法需要大量的运算，因为大量的数据需要重复检查才能确定划分。例如E,F等格网需要检查4次。

- 从下而上建立(bottom-up)

  对栅格数据按一定的顺序进行检测，如果相邻的四个格的网值相同，则进行合并，逐次向上递归。



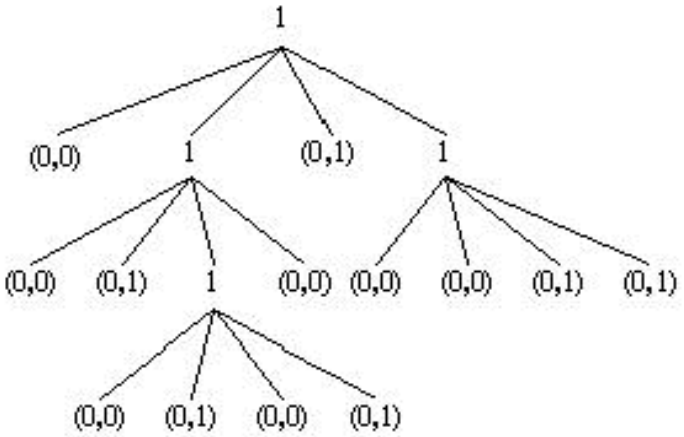
(a)

0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

(b)

0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

(c)



(d)

Figure 2: A binary image (a), its array representation (b), its quad tree partition (c), and its quad tree representation (d).

```
quadtree *DFS(int r,int c,int s)
{   //  char value[3];
    int i;    bool f=1;     quadtree *temp=new quadtree;
    if(s==1) //  value值为  “00”,“01”,“1“  00表示全0，01表示全1，1表示混合点
      {   temp->value[0]='0';  temp->value[1]='0'+MAP[r][c];
          temp->value[2]=0;    return temp;   }
    s/=2;
    temp->q[0]=DFS(r,c,s);
    temp->q[1]=DFS(r,c+s,s);
    temp->q[2]=DFS(r+s,c,s);
    temp->q[3]=DFS(r+s,c+s,s);
    for(i=1;i<4;i++)
       {   if( ! (*temp->q[0]==*temp->q[i])) {   f=0;    break;   }   } //不用合并
    if( f )
     {    strcpy(temp->value,temp->q[0]->value);    // 合并
          for(i=0;i<4;i++)
          {   delete temp->q[i];      temp->q[i]=0;    }
     }
    else {   strcpy(temp->value,“1”);      } // 不合并
    return temp;
}
```

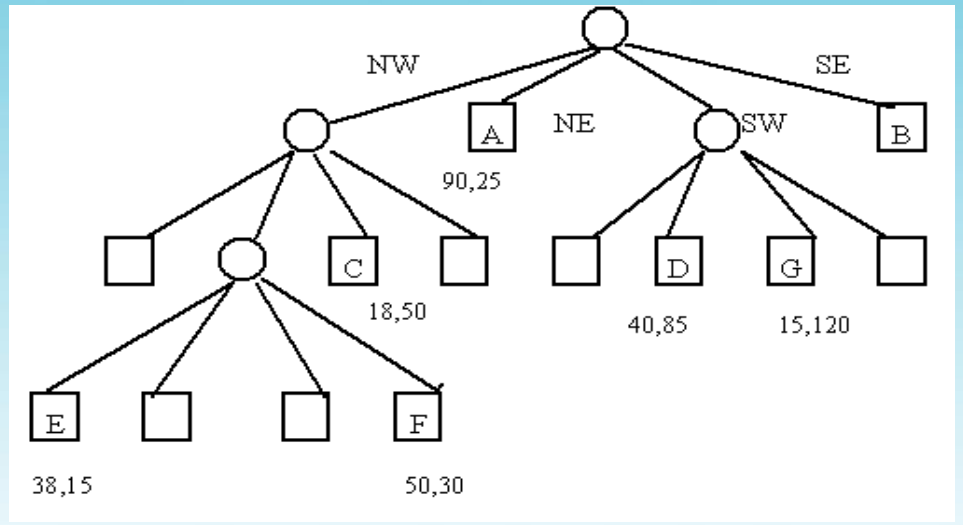
```
class quadtree
{  public:
     char value[3];//"00","01","1"00表示全0，01表示全1，1表示mixed
     quadtree *q[4];
   quadtree()
   {     q[0]=q[1]=q[2]=q[3]=0;       }

bool operator ==(const quadtree& p) const
{
   if(strcmp(value,"1")==0||strcmp(value,p.value)!=0)
        return 0;
   else
       return 1;
}
```

四分树的检索

检索的过程与划分过程类似

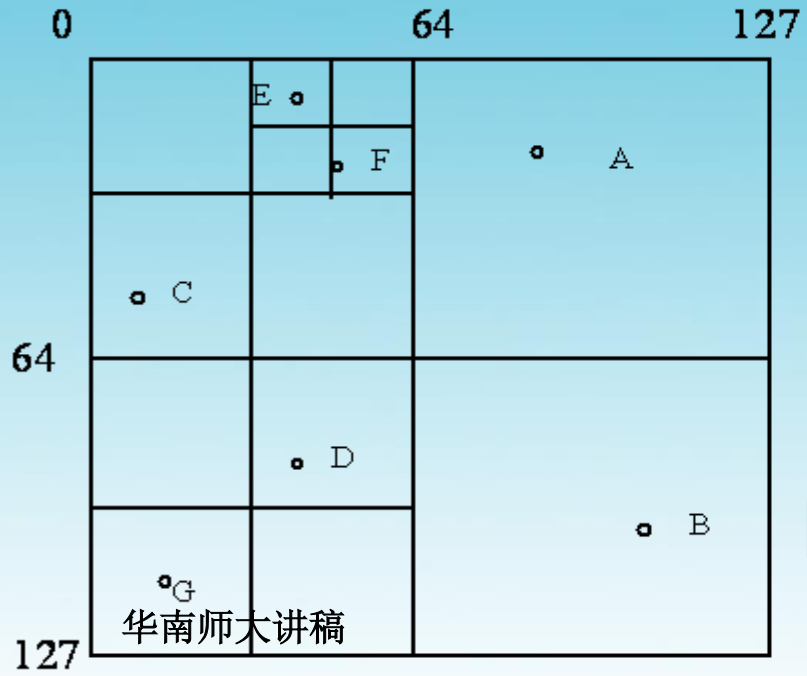
- 例如，检索数据点D，其坐标为（40，85）。
  - 在根结点，它属于SW子结点（范围为(0-64，64-127)）
  - 进入SW子树，其四个子树的范围是（0-32，64-96）、（32-64，64-96）、（0-32，96-127）和（32-64，96-127），D应该位于下一个NE子树中
  - NE子树只有一个叶结点代表数据点D，所以返回D的值。
- 如果已经到达叶结点却没有找到该数据点，那么返回错误

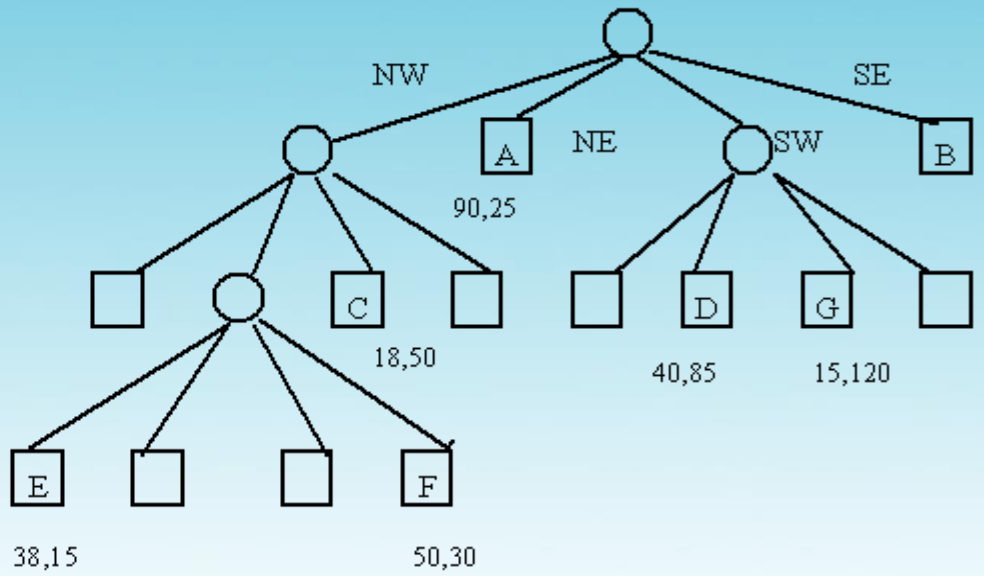


四分树的插入

首先通过检索确定其位置

- 如果这个位置的叶结点没有包含其他的数据点
  - 那么我们就把记录插入这里；
- 如果这个叶结点中已经包含P了(或者一个具有P的坐标的记录)
  - 那么就报告记录重复；
- 如果叶结点已经包含另一条记录X
  - 那么就必须继续分解这个结点，直到已存在的记录X和P分别进入不同的结点为止

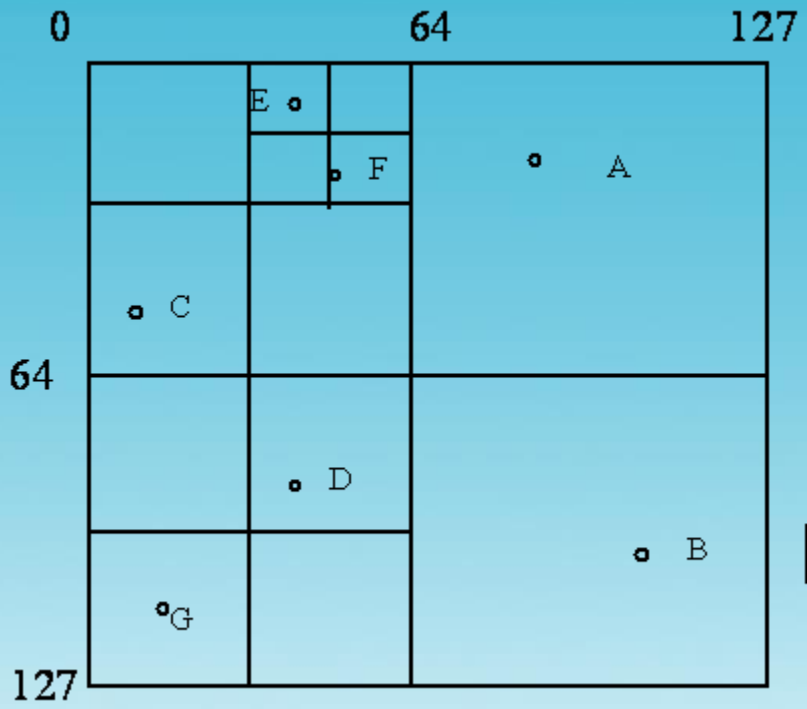


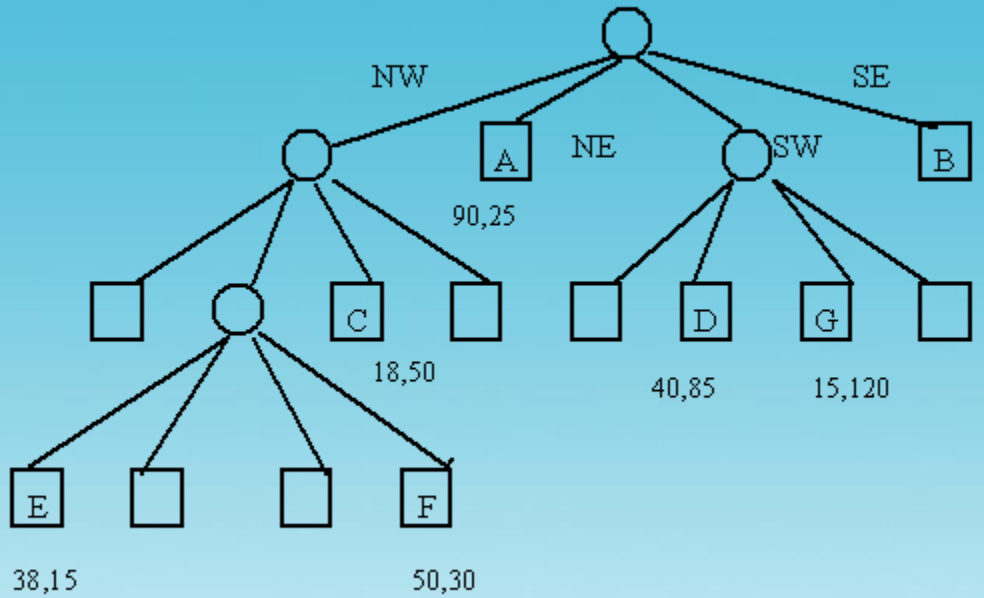


四分树的删除

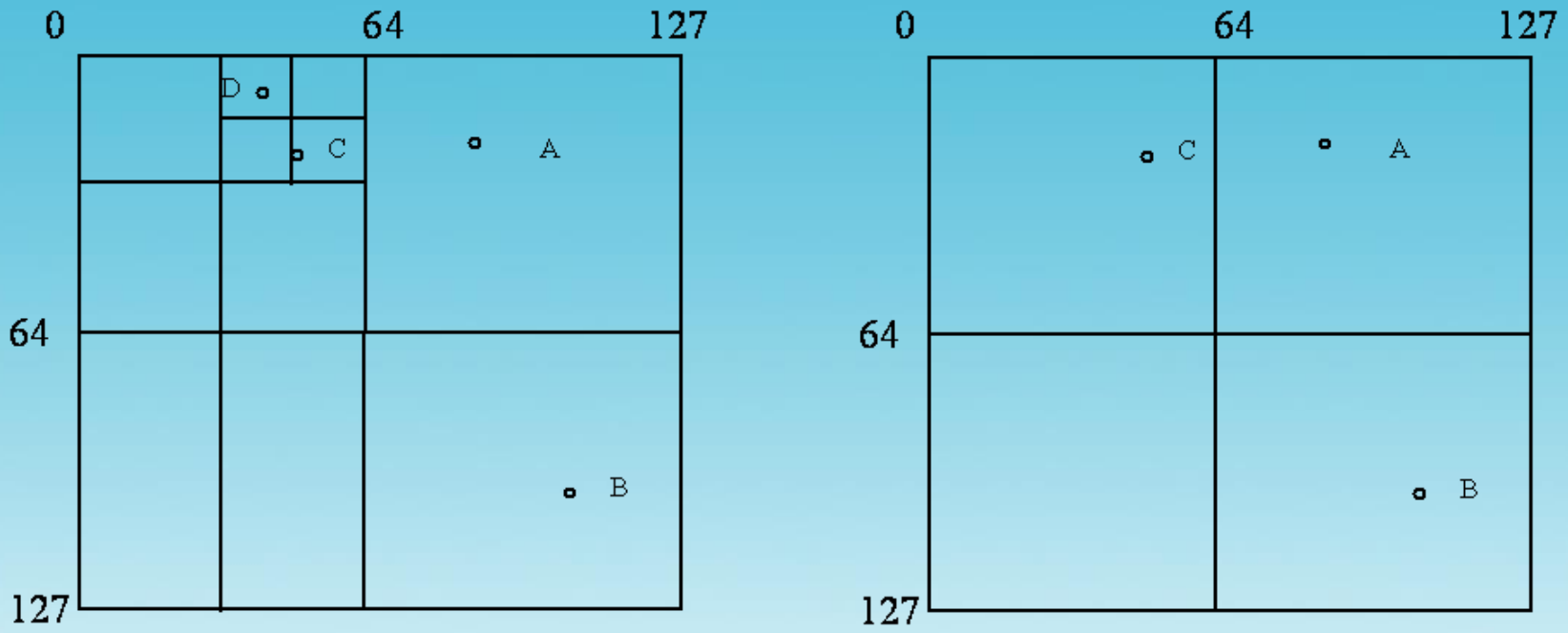
- 删除纪录
  - 首先检索到P所在的结点N
  - 将结点N所包含的记录改为空
- 调整
  - 查看它周围的三个兄弟结点的子树
  - 如果只有一个兄弟记录（不可能少于一个记录，否则就没有必要分裂为四个子结点）
    - 把这四个结点的子树合并为一个结点N'，代替它们的父结点
  - 这种合并会产生连锁反应
    - 在上一层也可能需要相似的合并
    - 直到到达某一层，该层至少有两个记录分别包含在结点N'以及N'的某个兄弟结点子树中，合并结束

删除操作


0
64
127
64
127
E
F
A
C
D
B
G


NW
NE
SW
SE
A
B
90,25
C
D
G
18,50
40,85
15,120
E
F
38,15
50,30

四分树的删除产生的合并



删除结点D导致的区域合并

四分树的区域查询

检索以某个点为中心、半径为r的范围内所有的点

- 根结点为空
  - 查询失败
- 如果根结点是包含一个数据记录的叶结点
  - 检查这个数据点的位置，确定它是否在范围内

- 如果根结点是一个内部结点
  - 在根结点确定包含该范围的子树
  - 然后进入符合这种条件的子树
  - 递归地继续这样的过程
    - 直到找不到这样的子树或者到达叶结点为止
    - 当到达叶结点时
      - 如果不包含任何的记录

        则直接返回；
      - 如果包含一个数据记录

        则检查这个数据记录是否在范围之内，如果在就返回该数据

Zju 1788解题思路

- （1）生成四分树

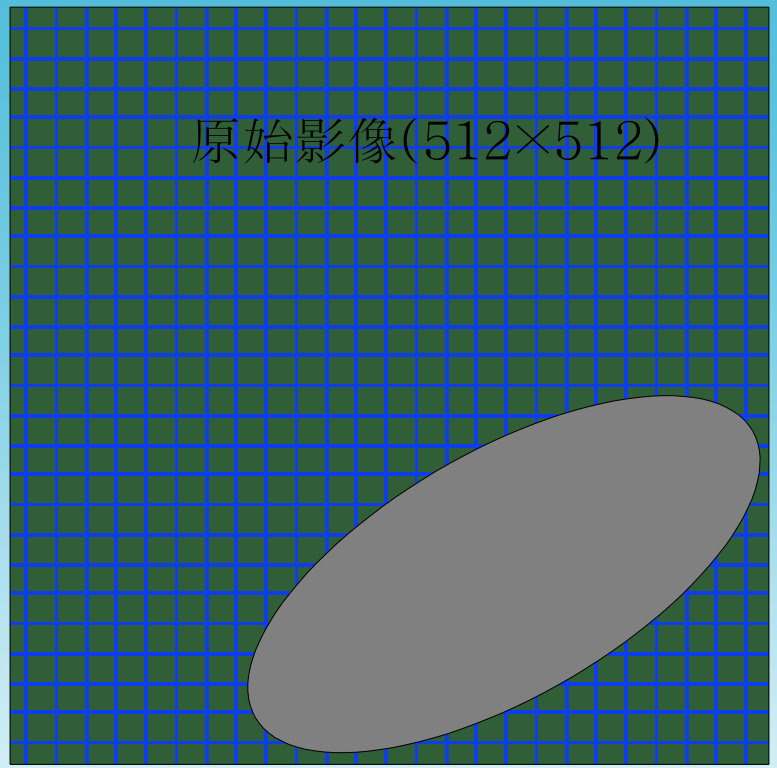
  深度优先的方法

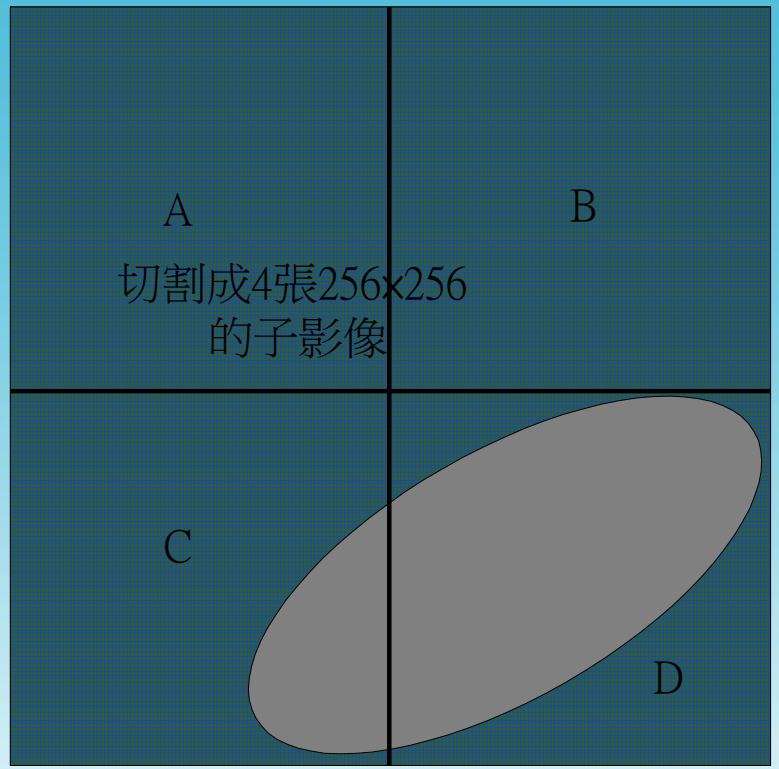
- （2）遍历四分树

  层次遍历得到二进制数列

- （3）反向对数列进行编码

四分树的应用方法





圖(a)

四分树的应用方法

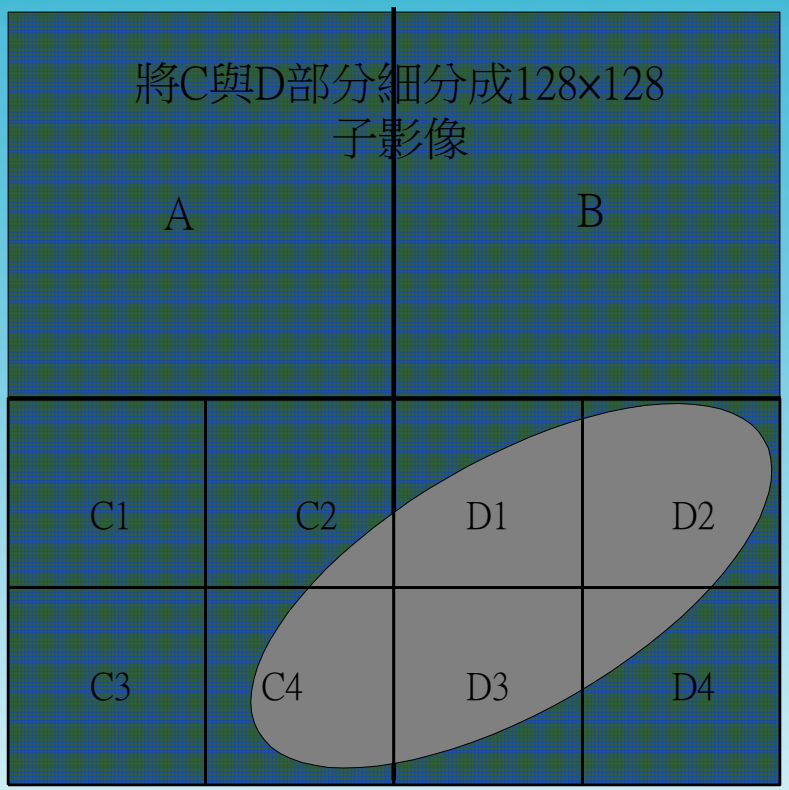
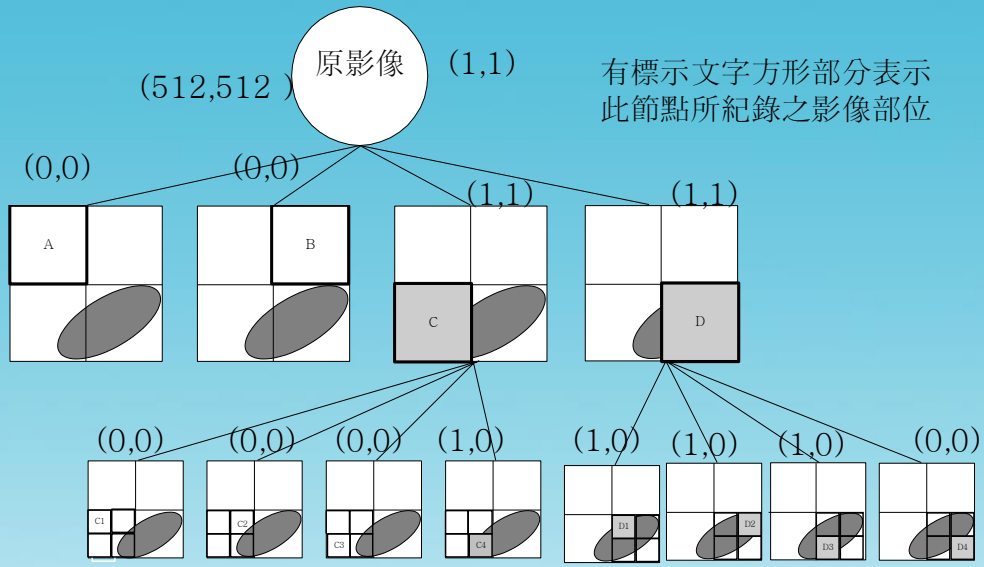
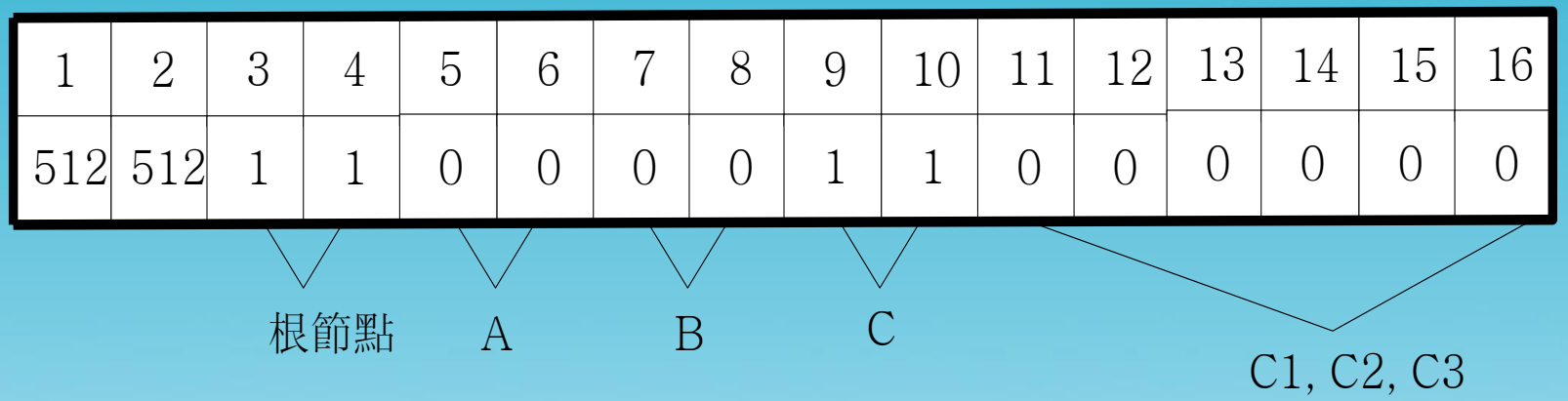
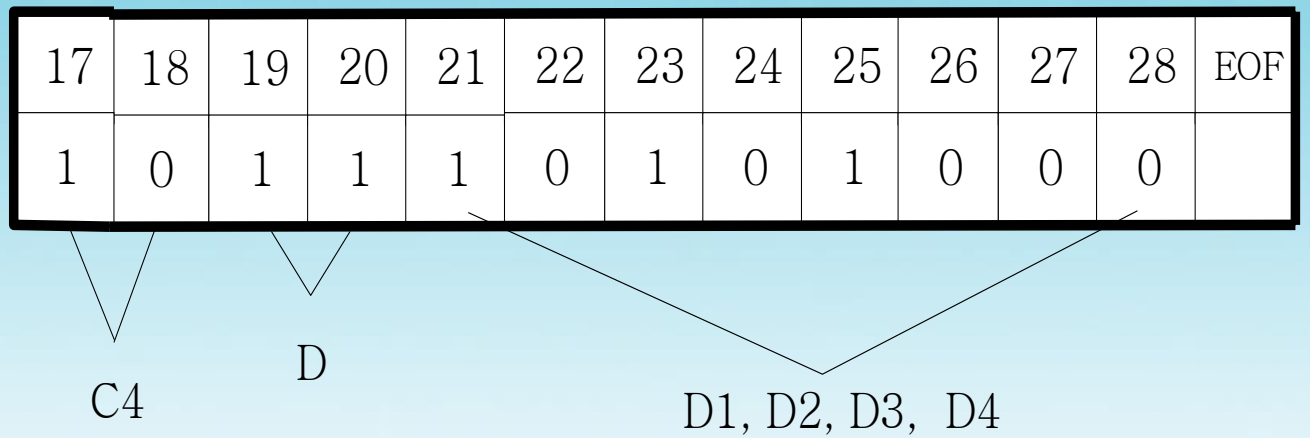

將C與D部分細分成128×128
子影像
A
B
C1
C2
D1
D2
C3
C4
D3
D4

圖 (b)

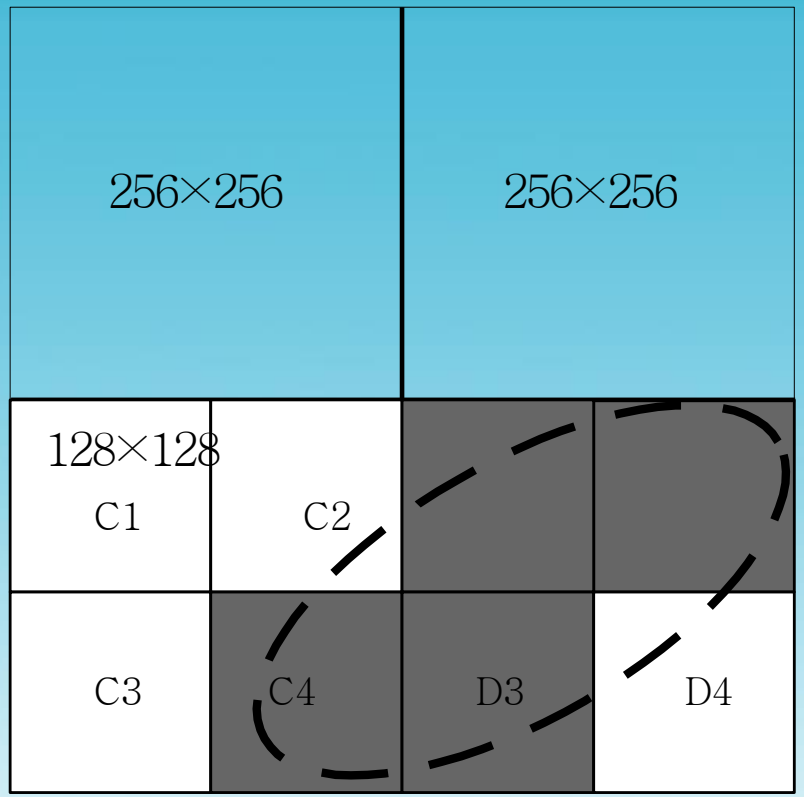

原影像
(512,512)
(1,1)
有標示文字方形部分表示
此節點所紀錄之影像部位
(0,0)
(0,0)
(1,1)
(1,1)
A
B
C
D
(0,0)
(0,0)
(0,0)
(1,0)
(1,0)
(1,0)
(1,0)
(0,0)

四分树的应用方法

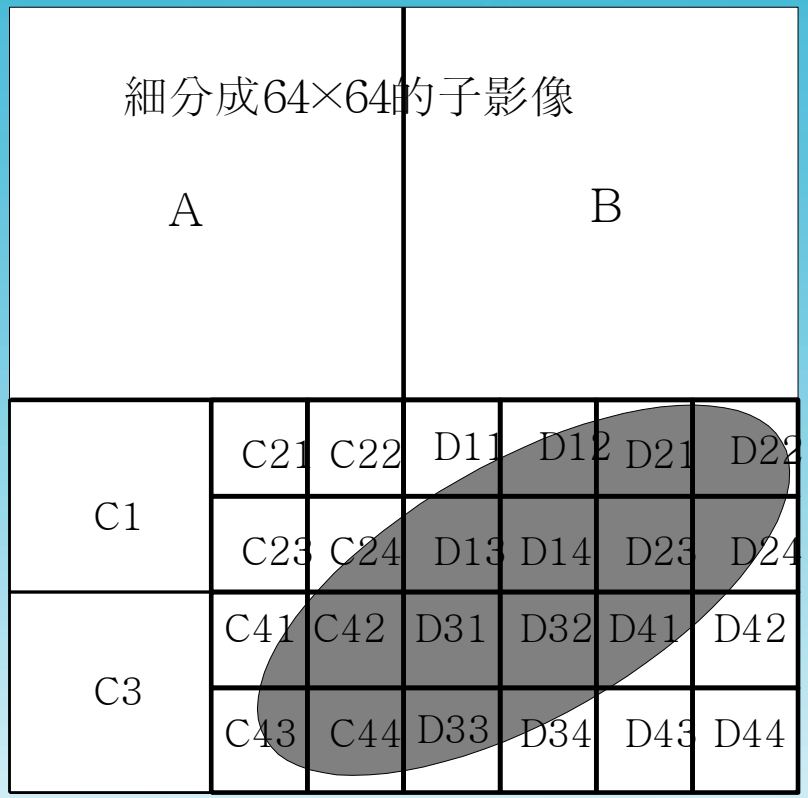

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
512 512 1 1 0 0 0 0 1 1 0 0 0 0 0 0
根節點
A
B
C
C1, C2, C3


17 18 19 20 21 22 23 24 25 26 27 28 EOF
1 0 1 1 1 0 1 0 1 0 0 0
C4
D
D1, D2, D3, D4

四分树的应用方法

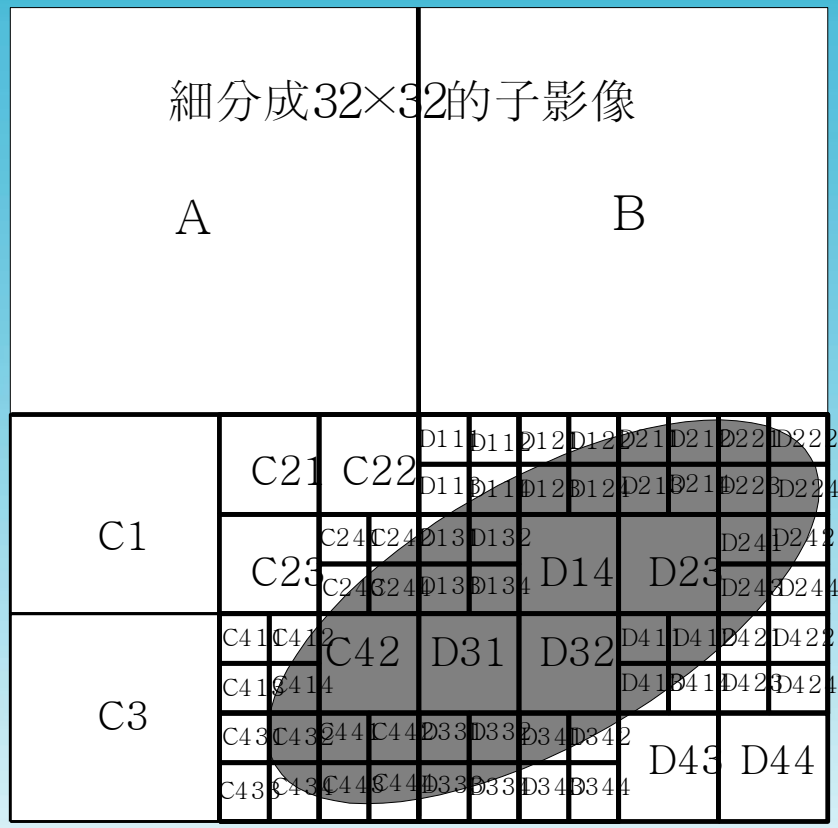


圖（c）

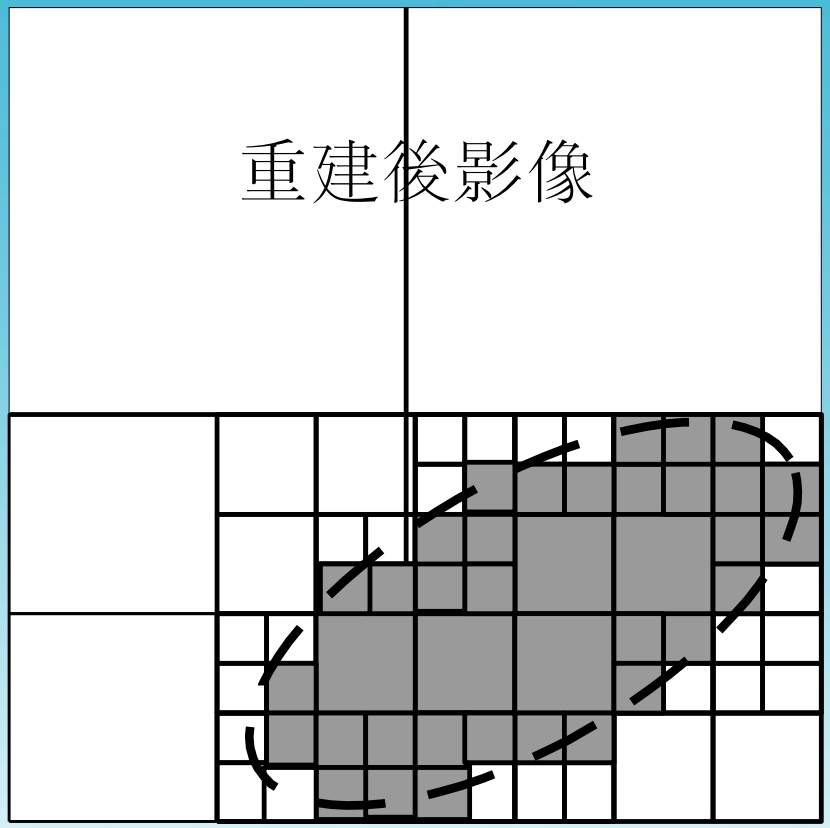


圖（ d）

四分树的应用方法



圖（e)



圖（f)

练习题

- zju 1788
- zju 1805
- zju 1955
- zju 1959