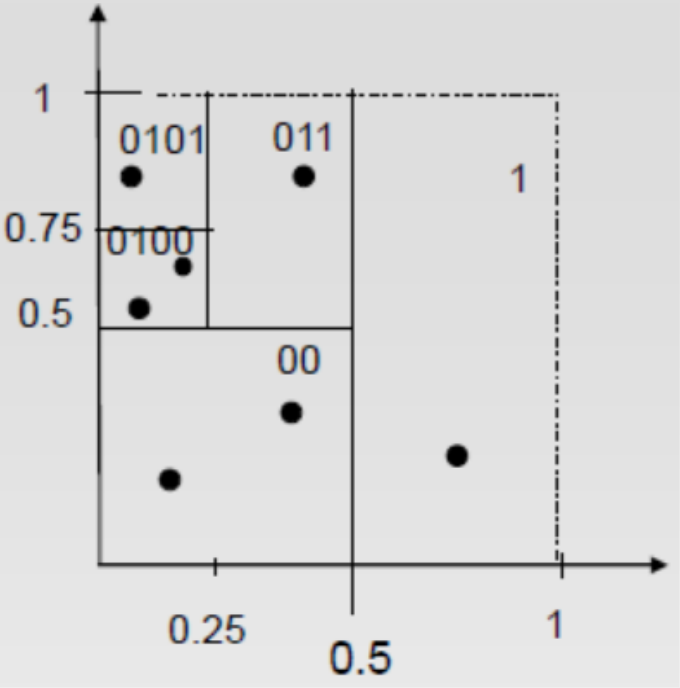
G-trees

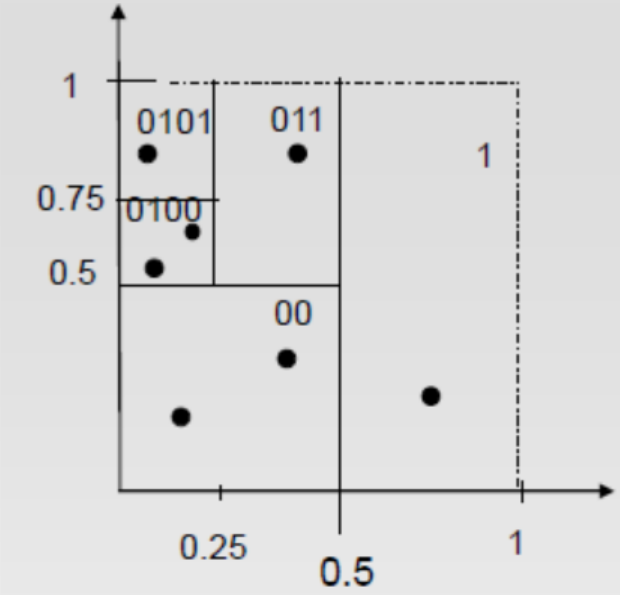


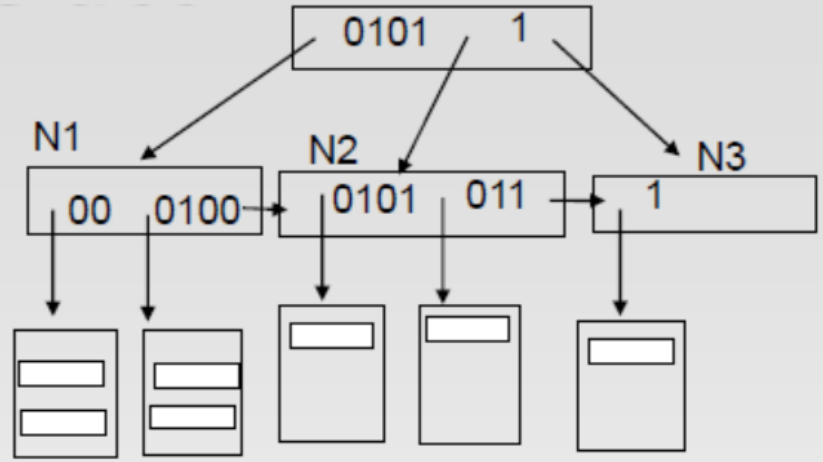
Max number or records in a block: 2

Half partitioning of a region

R0: 0 < x <= 0.5
R1: 0.5 < x <= 1

Region identifier: string of bits built as follows
-Initial region is identified by empty string
-Region R0: 0 < x <= 0.5 identified by 0
- Region R1: 0.5 < x <= 1 identified by 1
- When a region is divided add 0 or 1 to its identifier (add 0 if the region has lower values and 1 if the region has greater values)





RegionOf(S):
S=00 corresponds to the region {(0;0.5), (0;0.5)}
S=011 corresponds to the region {(0,25;0.5), (0,5;1)}

Strings are stored into a B+-tree (G-tree) Use prefix as ordering relation
For each leaf (S,P), S is the code of a region whose objects are store into the block reachable by P

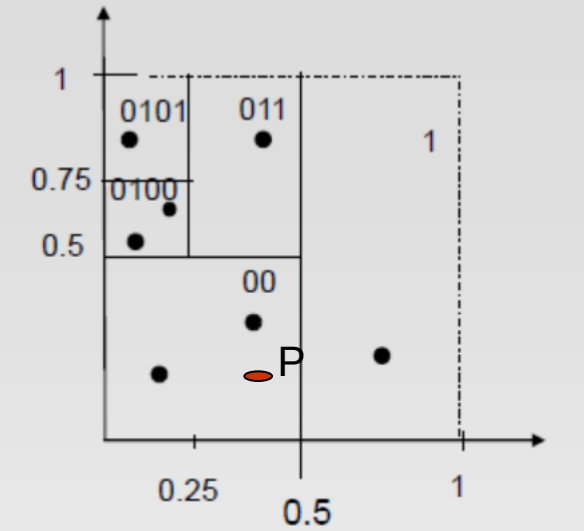
Search of an object P=(x1, y1) .
Let M the maximum length of strings of regions. Find the string Sp of the region that contains P, using M bits. Search Sp in the tree (as usual) from the root to the leaf. Sp or a prefix of Sp is in the leaf . Tranfer the corresponding block

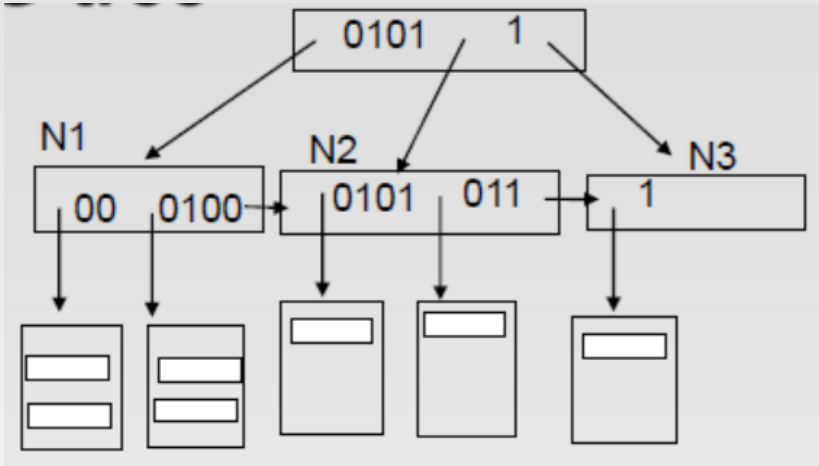
Example:

P=(0.3; 0.6) M=4 Sp=0110

Search Sp : leaf N2 is reached.

Get the block associated to the prefix 011 of Sp.
P does not belong to the block. P is not in the database.

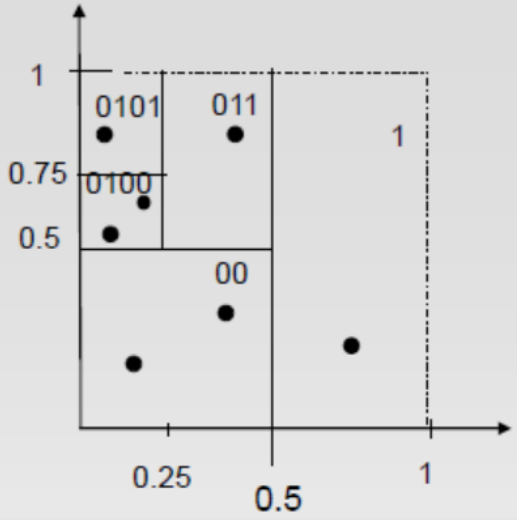




- Properties:
  - Let T be a region whose code is S=s1…sn, T has been obtained by a partition of the region whose code is S'=s1…sn-1 (it is S'<S)
  - The length of a code of a region T denotes the number of partitions needed to obtain T

Search of points in a Region R

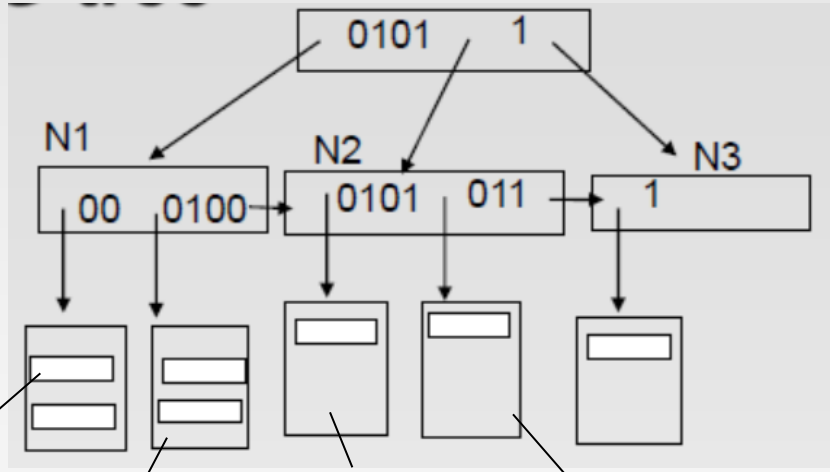
Let P1 the leftmost lowest point of R
Let P2 the rightmost highest point of R
Search P1 in the tree. Assume leaf N1 is returned
Search P2 in the tree. Assume leaf N2 is returned
For each leaf between N1 and N2, for each string, generate its region R', for regions which overlap with R
Find point in the corresponding blocks



R= { (0.3; 0.4) (0.2; 0.6)}
M=4

P1= (0.3, 0.2) SP1=0010
P2 = (0.4, 0.6) SP2=0110

P1: Leaf N1
P2: Leaf N2



R1= { (0; 0.5) (0; 0.5)}
R2= { (0; 0.25) (0.5; 0.75)}
R3= { (0; 0.25) (0.75; 1)}
R4= { (0.25; 0.5) (0.5; 1)}

Intersection betewen R and R1 not empty
Intersection betewen R and R4 not empty ⇨ Block tranfer

Insert P

Let Sp the code of P

Search Sp in the tree from the root to the leaf.

If P is not in the tree and there is space in the block of the region,

insert P

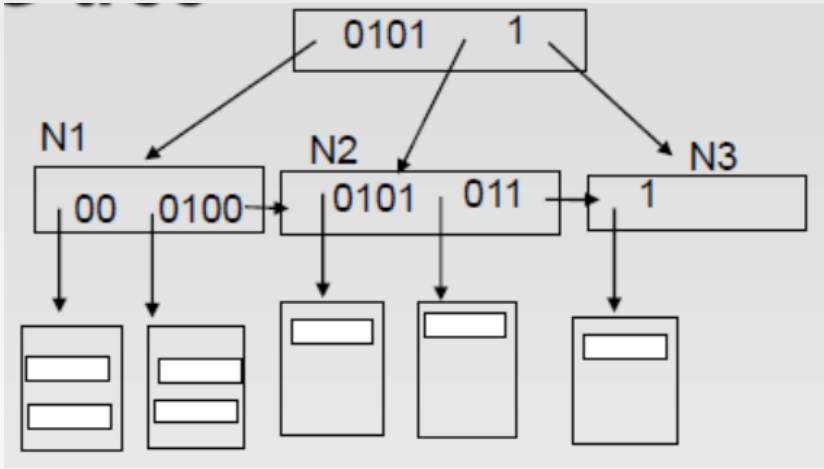
otherwise split R into two subregions:

R1: with SR1 = SR 0 and SR2=SR 1

Distribute the points of R +P in the two sub-regions according to codes.

Substitute SR by SR1 and SR2.

Propagate updates upwards (if needed), similar to B+-tree



Insert P

Let Sp the code of P

Search Sp in the tree from the root to the leaf.

If P is not in the tree and there is space in the block of the region,

insert P

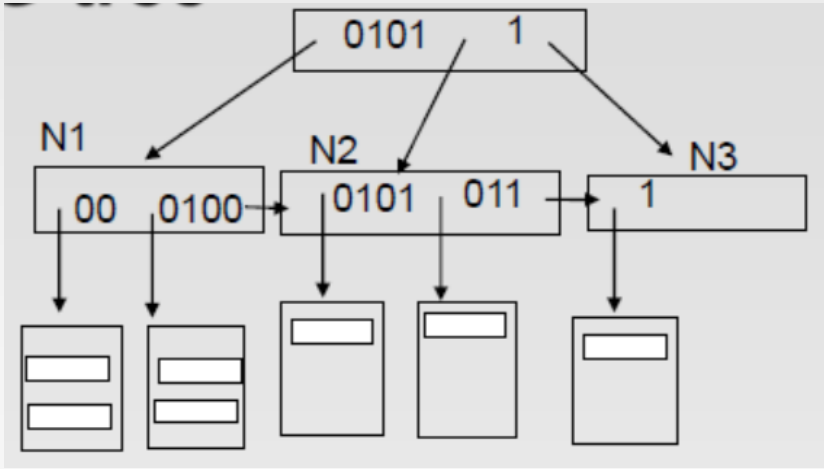
otherwise split R into two subregions:

R1: with SR1 = SR 0 and SR2=SR 1

Distribute the points of R +P in the two sub-regions according to codes.

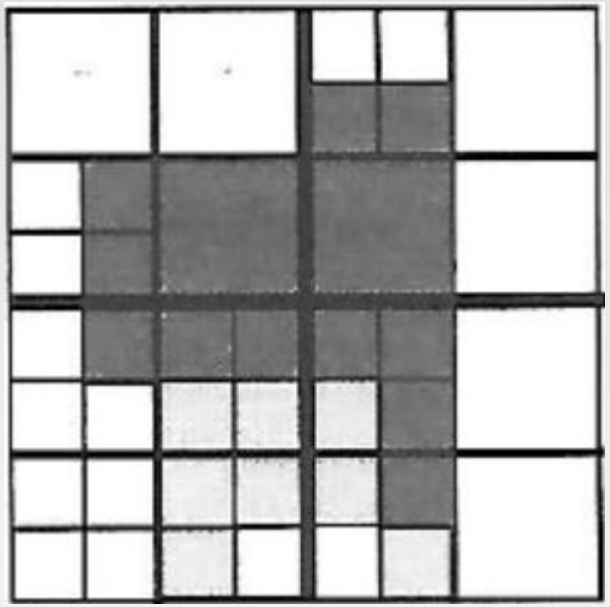
Substitute SR by SR1 and SR2.

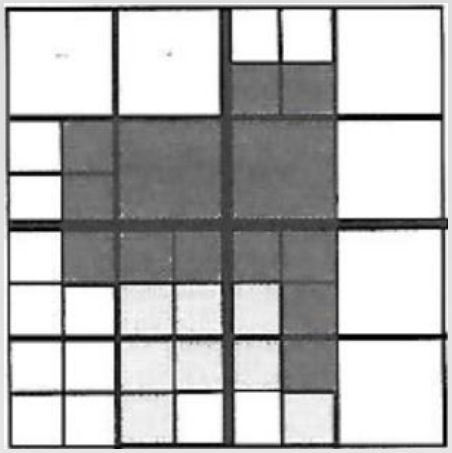
Propagate updates upwards (if needed), similar to B+-tree

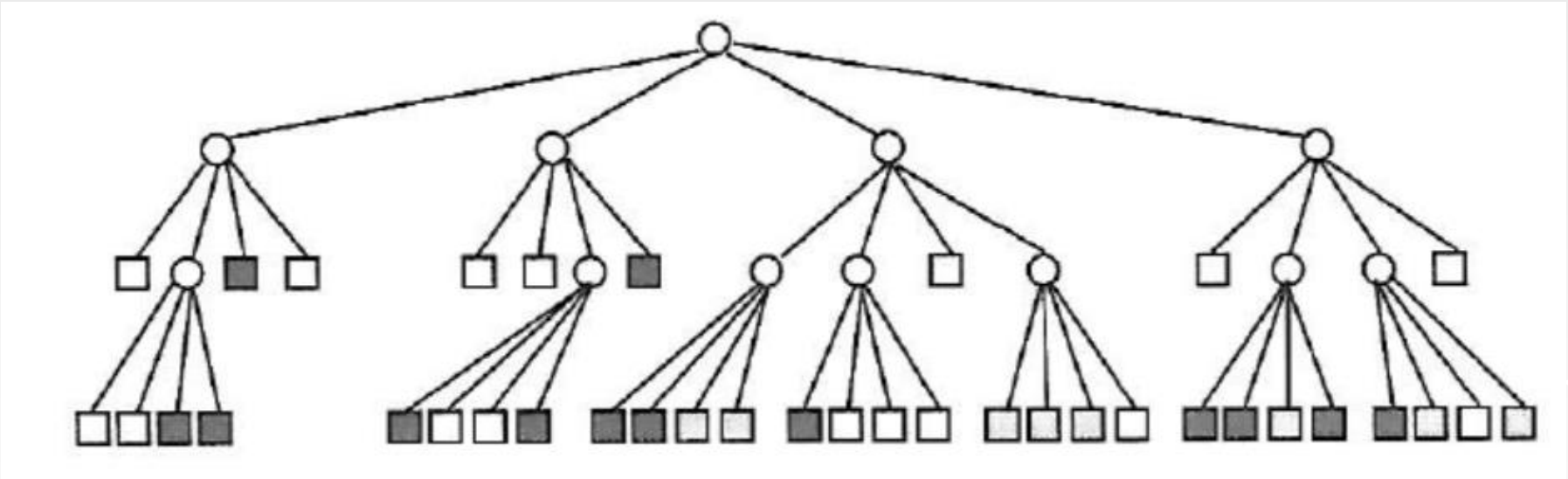


Esercizio

Build the **Region Quadtree** of the following raster data.

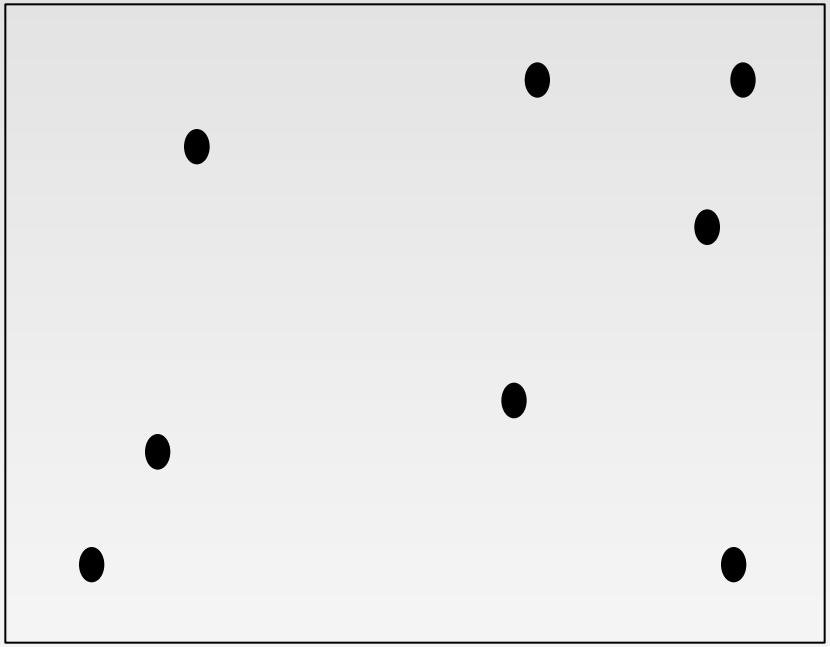






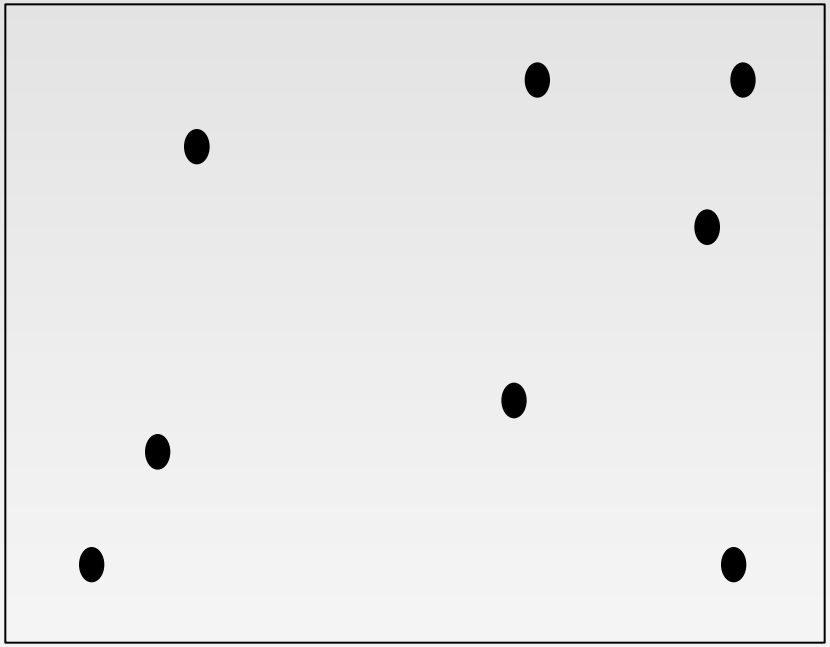
Esercizio

Build the **PointRegion quadtree** (PR quadtree) of the region below.
Assume maximum number of points set to 2



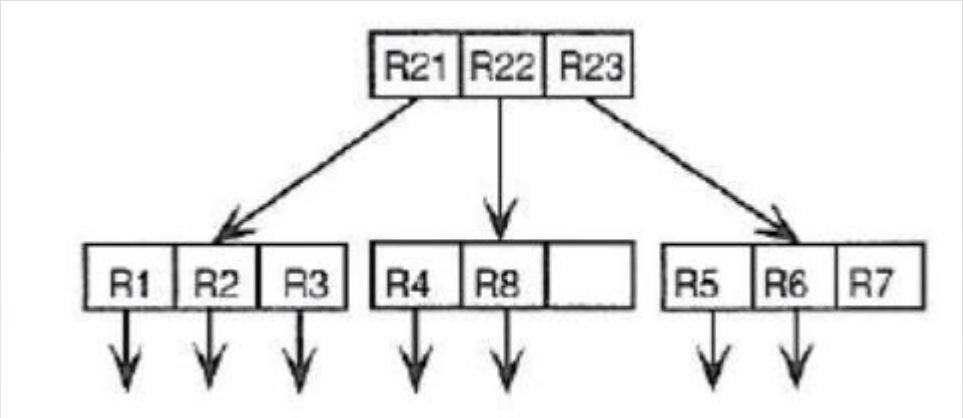
Esercizio

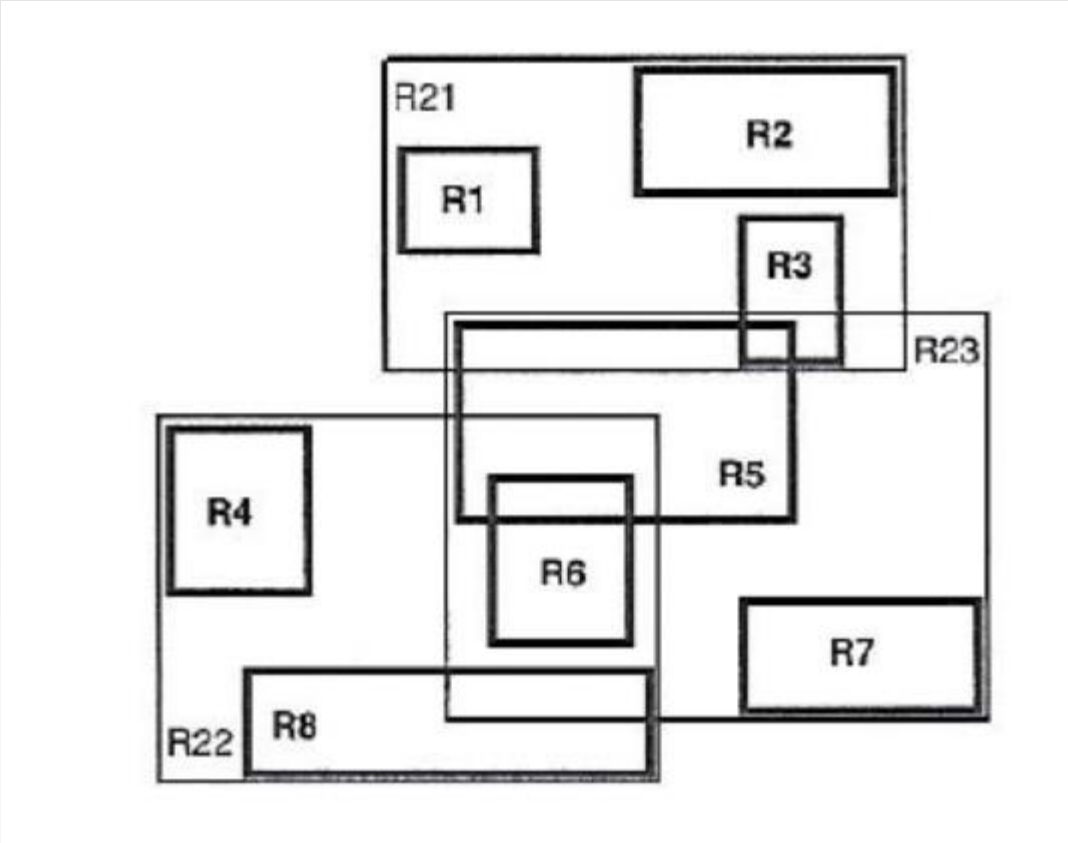
Build the **k-d tree** of the region below. Assume maximum number of points set to 2



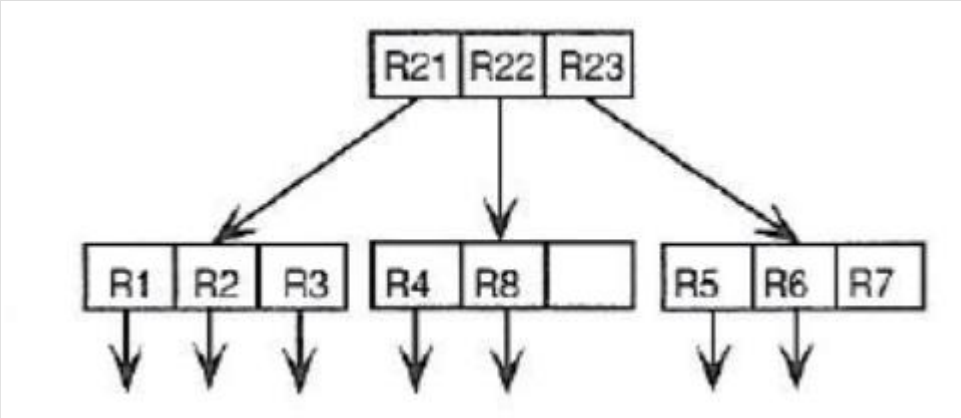
Esercizio

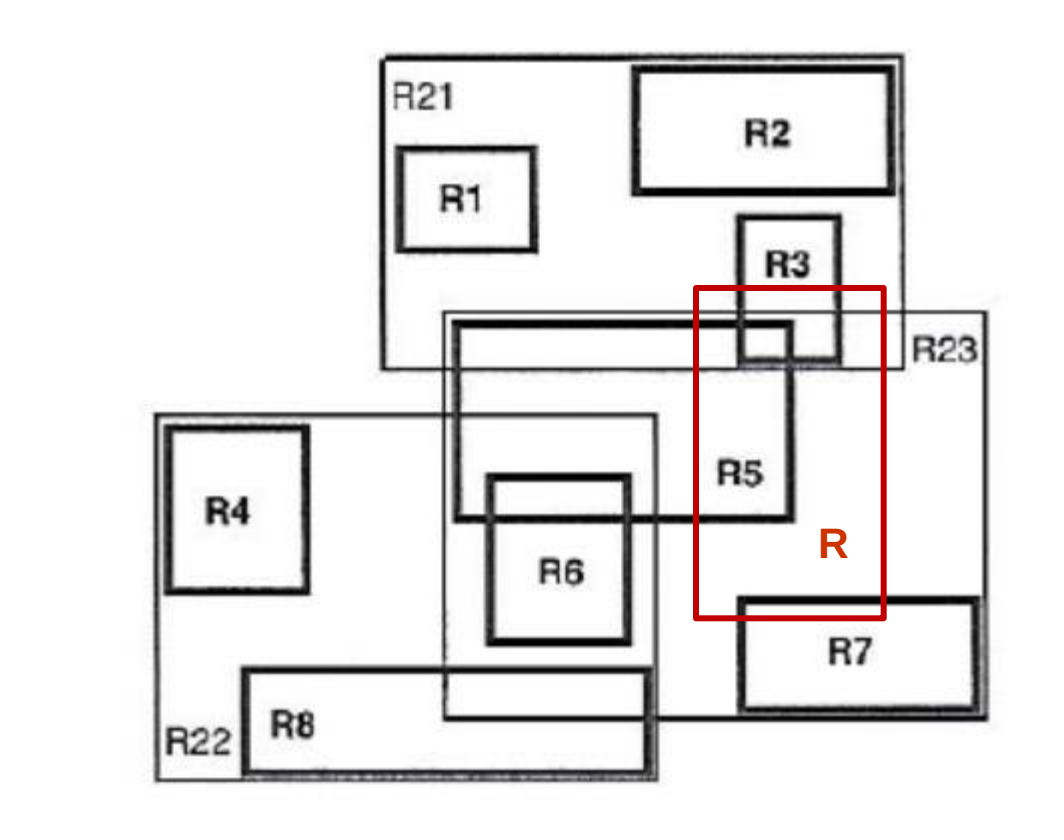
Rectangular-trees (R-trees)
m=4



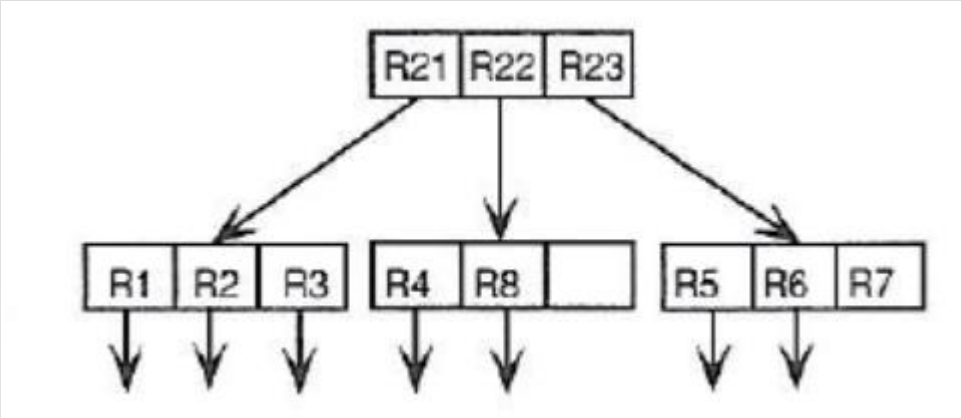


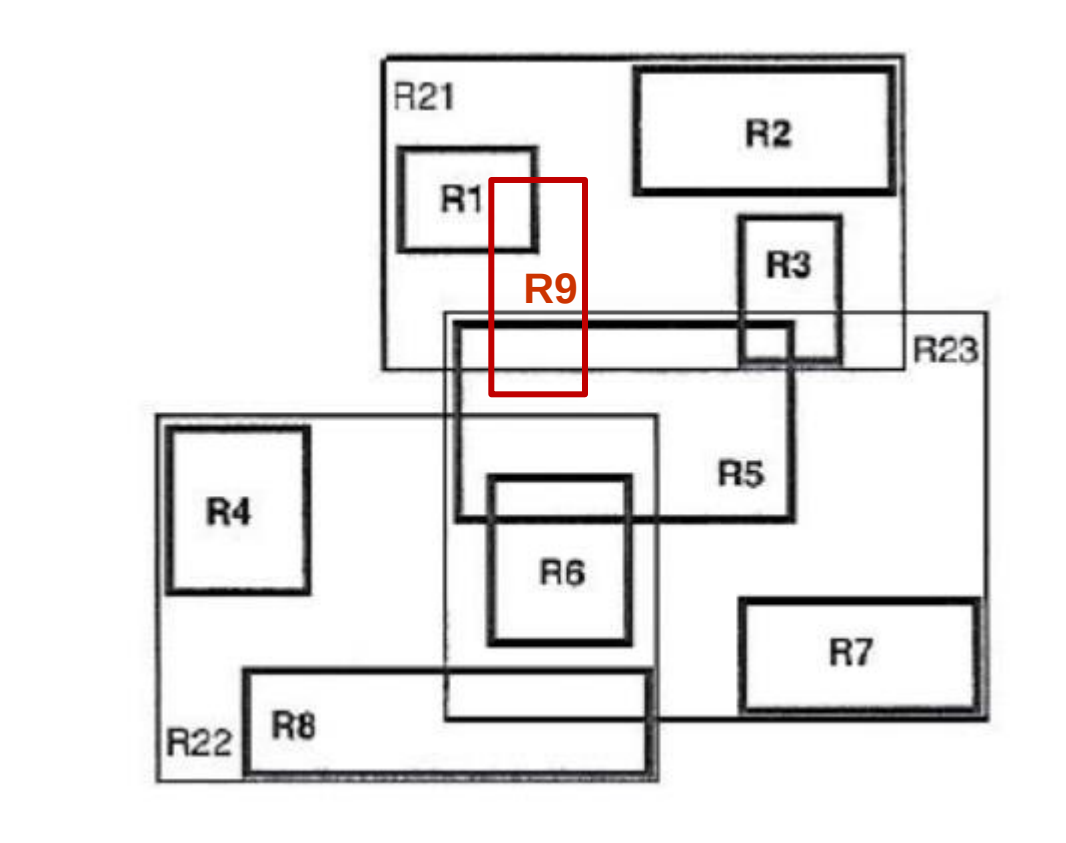
Search rectangles that overlap R (the red rectangle in the figure)



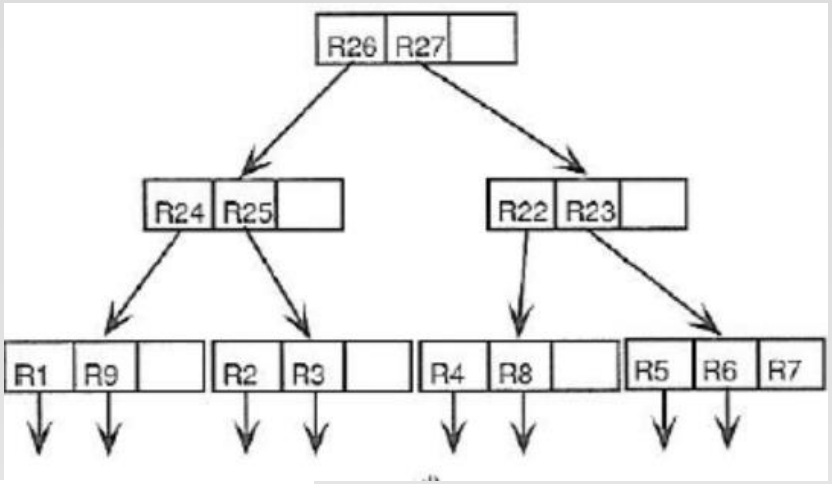


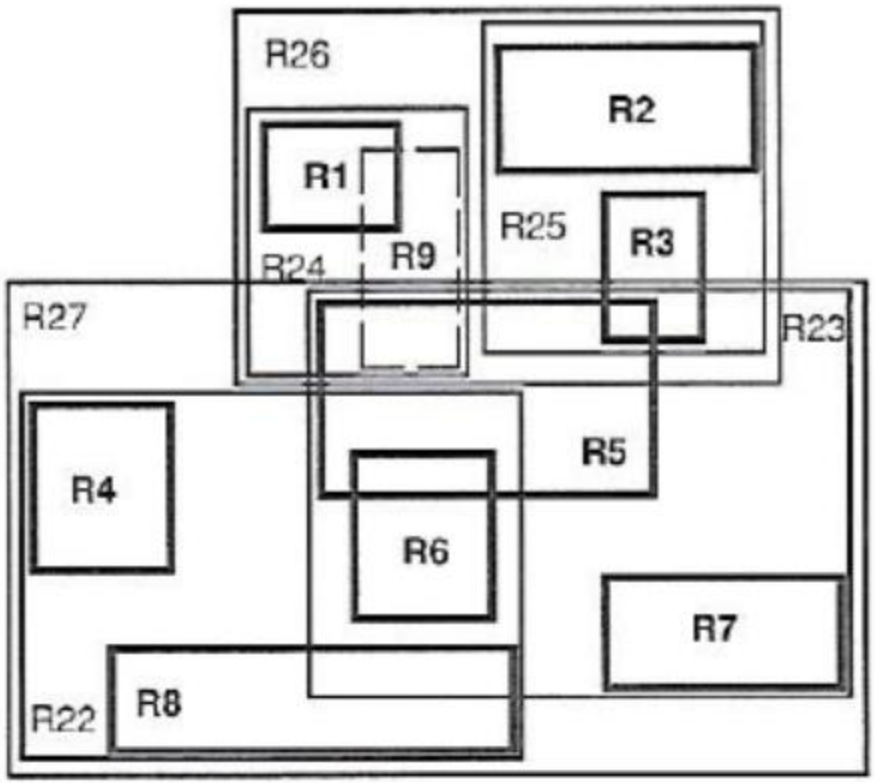
Insert R9





R1,R9 / R2,R3


R26
R27
R24
R25
R22
R23
R1
R9
R2
R3
R4
R8
R5
R6
R7


R26
R2
R1
R25
R3
R24
R9
R27
R23
R5
R4
R6
R7
R22
R8