

Quad Trees

Universitatea "Transilvania" din Braşov

May 22, 2017

Quadrees - arbori quad

- Structuri de date ierarhice, au la bază descompunerea recursivă tip *divide et impera* a spațiului.
- Arbori în care fiecare nod intern are 4 descendenți.
- Diferențiați după:
 - Tipul de date pe care îl stochează.
 - Principiul de descompunere.
 - Rezoluția descompunerii.

Arbori quad pentru regiuni

- partiționează spațiului 2D sau 3D în regiuni, conform unui anumit criteriu de descompunere.

Arbori quad pentru regiuni

- partiționează spațiului 2D sau 3D în regiuni, conform unui anumit criteriu de descompunere.
- utilizează cel mai frecvent partiționare în suprafețe pătrate egale.

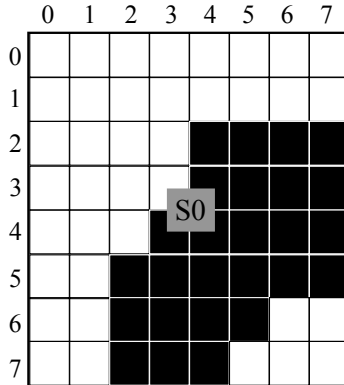
Arbori quad pentru regiuni

- partiționează spațiului 2D sau 3D în regiuni, conform unui anumit criteriu de descompunere.
- utilizează cel mai frecvent partiționare în suprafețe pătrate egale.
- cresc viteza de execuție a anumitor algoritmi pentru imagini, suprafețe, structuri geometrice.

Arbori quad pentru regiuni

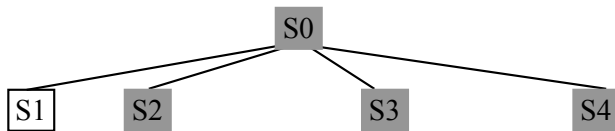
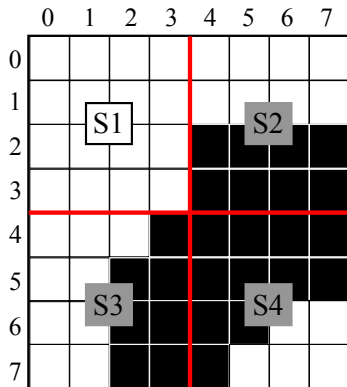
- partiționează spațiului 2D sau 3D în regiuni, conform unui anumit criteriu de descompunere.
- utilizează cel mai frecvent partiționare în suprafețe pătrate egale.
- cresc viteza de execuție a anumitor algoritmi pentru imagini, suprafețe, structuri geometrice.
- exemplu de aplicație: - algoritmul *Split and Merge*

Arbori quad pentru regiuni - Exemplu

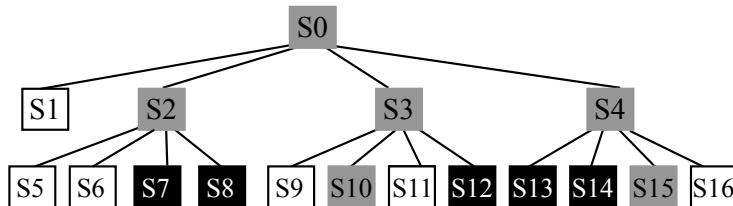
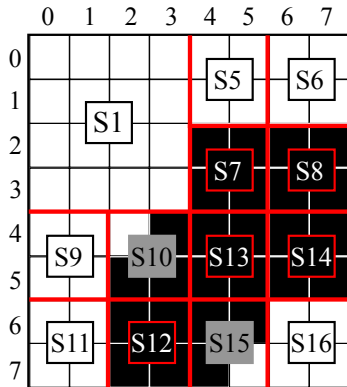


S0

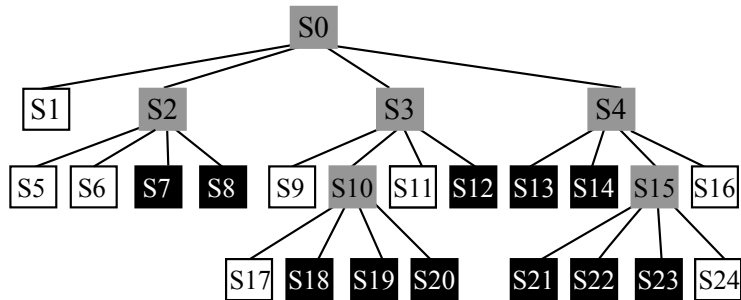
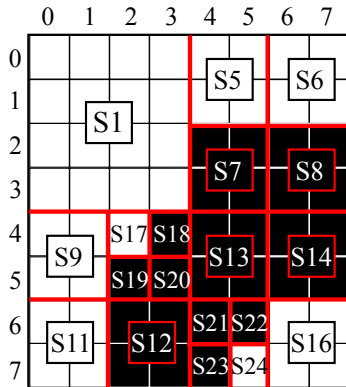
Arbori quad pentru regiuni - Exemplu



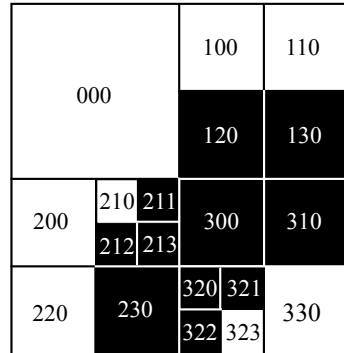
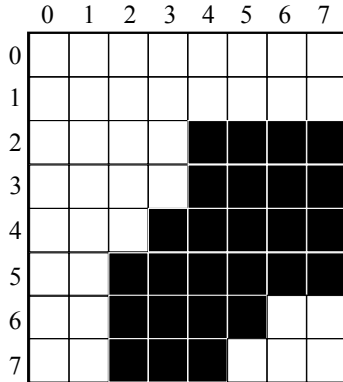
Arbori quad pentru regiuni - Exemplu



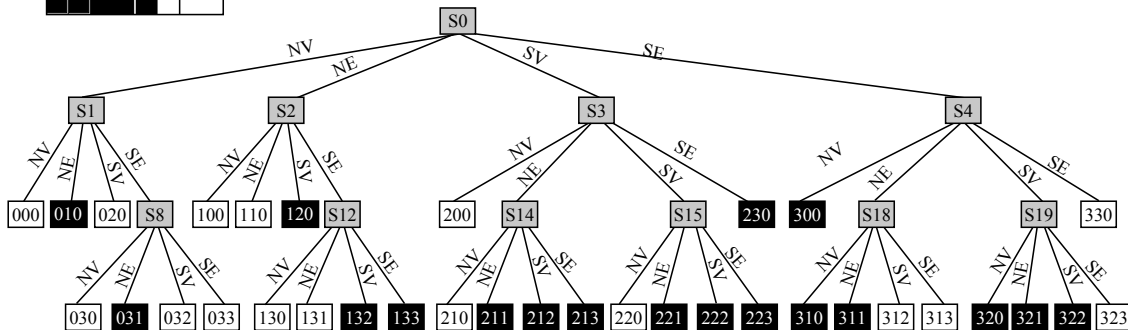
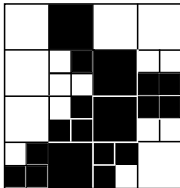
Arbori quad pentru regiuni - Exemplu



Arbori quad pentru regiuni - Exemplu



Arbori quad pentru regiuni - Exemplu



Adiacență - Vecini

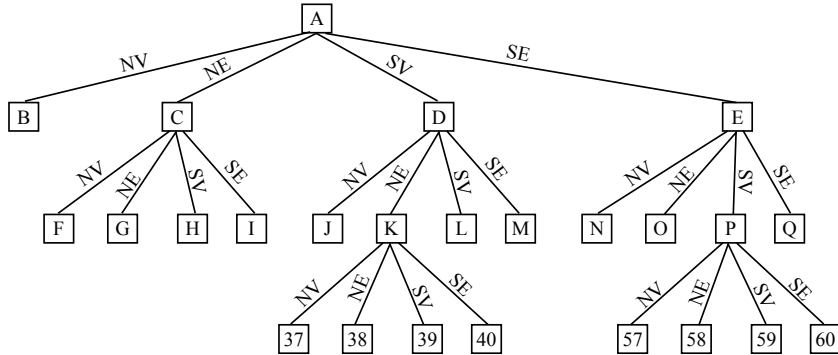
- Nod din arbore \Leftrightarrow regiune/bloc din imagine.
- Fiecare bloc are
 - 4 laturi: N (nord), S (sud), E (est), V (vest).
 - 4 vârfuri: NV, NE, SV, SE
- Două blocuri P și Q disjuncte sunt adiacente după direcția $D \in \{N, S, E, V, NV, NE, SV, SE\}$, dacă
 - P are o parte din latura de pe direcția D comună cu Q
 - Vârful din direcția D al blocului P este adiacent cu vârful opus al blocului Q
- P și Q vecine $\Leftrightarrow P$ și Q adiacente.

Vecini - Exemplu

B			F	G
			H	I
J	37	38	N	O
	39	40		
L	M	57	58	Q
		59	60	

Vecini - Exemplu

B			F	G
			H	I
J	37	38	N	O
	39	40		
L	M		57	58
			59	60
			Q	

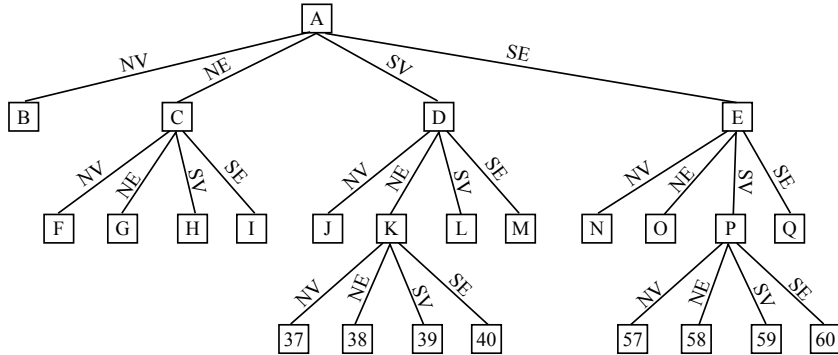


Determinarea vecinilor

- (1) Determinarea vecinilor adiacenți cu întreaga latură a unui anumit bloc
- (2) Determinarea vecinilor adiacenți cu un segment din latura unui anumit bloc

Vecini - Exemplu

B			F	G
			H	I
J	37	38	N	O
	39	40		
L	M		57	58
			59	60
			Q	



Algoritmi pentru determinarea vecinilor

Notatii

- G = "greater then or equal", C = "corner", S = "side", N = "neighbor"
- $GSN(P,D)$ = cel mai mic bloc adiacent cu latura din direcția D a blocului P cu latura mai mare sau egală cu latura lui P .
- $GCN(P,C)$ = cel mai mic bloc adiacent cu P și aflat de partea opusă a colțului C al blocului P cu latura mai mare sau egală cu latura lui P .

Algoritmi pentru determinarea vecinilor

B			F	G
			H	I
J	37	38	N	O
	39	40		
L	M	57	58	Q
		59	60	

Observații:

- GSN și GCN nu definesc relații 1 - 1.

Algoritmi pentru determinarea vecinilor

B			F	G
			H	I
J	37	38	N	O
	39	40		
L	M	57	58	Q
		59	60	

Observații:

- GSN și GCN nu definesc relații 1 - 1.
- GSN și GCN nu sunt simetrice.

Algoritmi pentru determinarea vecinilor

B			F		G
			H		I
J	37	38	N		O
	39	40			
L	M		57	58	Q
			59	60	

Observații:

- GSN și GCN nu definesc relații 1 - 1.
- GSN și GCN nu sunt simetrice.
- Un bloc care nu se află pe marginea imaginii are minim 5 vecini.

B			F		G
			H		I
J	37	38	N		O
	39	40			
L	M		57	58	Q
			59	60	

- GSN și GCN nu definesc relații 1 - 1.
- GSN și GCN nu sunt simetrice.
- Un bloc care nu se află pe marginea imaginii are minim 5 vecini.
- Un nod are maxim 8 vecini.

Determinarea vecinilor adiacenți pe orizontală / verticală

- **Problematică:** determinare $REZ = GSN(NOD, D)$

Determinarea vecinilor adiacenți pe orizontală / verticală

- **Problematică:** determinare $REZ = GSN(NOD, D)$
- **Idee generală:**

Determinarea vecinilor adiacenți pe orizontală / verticală

- **Problematică:** determinare $REZ = GSN(NOD, D)$
- **Idee generală:**
 - 1 se urcă de la nodul NOD către primul strămoș S comun cu REZ ;

Determinarea vecinilor adiacenți pe orizontală / verticală

- **Problematică:** determinare $REZ = GSN(NOD, D)$
- **Idee generală:**
 - 1 se urcă de la nodul NOD către primul strămoș S comun cu REZ ;
 - 2 se coboară de la S către REZ pe ramuri simetrice față de direcția D ale ramurilor de urcare.

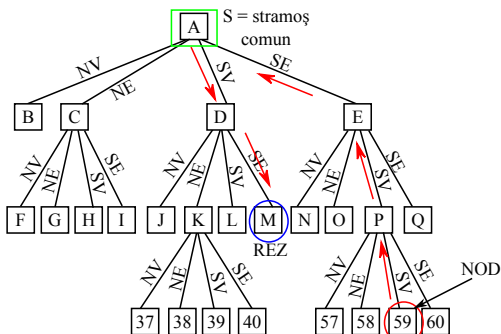
Determinarea vecinilor adiacenți pe orizontală / verticală

- **Problematică:** determinare $REZ = GSN(NOD, D)$
- **Idee generală:**
 - 1 se urcă de la nodul NOD către primul strămoș S comun cu REZ ;
 - 2 se coboară de la S către REZ pe ramuri simetrice față de direcția D ale ramurilor de urcare.
- **Observație:** Strămoșul comun se obține urcând de la nodul curent către părinte pe o ramură ce nu conține direcția de căutare!

Algoritmi pentru determinarea vecinilor

$GSN(59, V) = M$

B			F	G
			H	I
J	37	38	N	O
	39	40		
L	M	57	58	Q
		59	60	



Observații:

- Dacă direcția $D = E$, \Rightarrow *NOD* pe ramuri SV sau NV față relativ la S.
- Dacă direcția $D = V$, \Rightarrow *NOD* pe ramuri SE sau NE față relativ la S.
- Dacă direcția $D = S$, \Rightarrow *NOD* pe ramuri NE sau NV față relativ la S.
- Dacă direcția $D = N$, \Rightarrow *NOD* pe ramuri SE sau SV față relativ la S.

Algoritmul GSN

```

GSN (P, D)
    //urcarea spre stramosul comun
    nod=P
    parinte = nod→p
    Stiva=∅
    cat timp parinte≠ NULL si ramura spre parinte contine D
        pune pe Stiva simetricul fata de D al acestei ramurii
        nod = parinte
        parinte=nod→p
    sfarsit cat timp
    daca parinte=NULL atunci
        RETURN NULL
    sfarsit daca
    pune pe Stiva simetricul fata de D a ramurii de la nod la parinte
    nod = parinte
    
```

Algoritmul GSN

```
// coborare catre vecinul cautat
cat timp Stiva  $\neq \emptyset$ 
    directie  $\leftarrow$  Stiva
    daca nod  $\neq$  frunza atunci
        nod = nod  $\rightarrow$  directie
    altfel
        Stiva =  $\emptyset$ 
    sfarsit daca
sfarsit cat timp
RETURN nod
```

Determinarea vecinilor adiacenți pe diagonală - algoritmul GCN

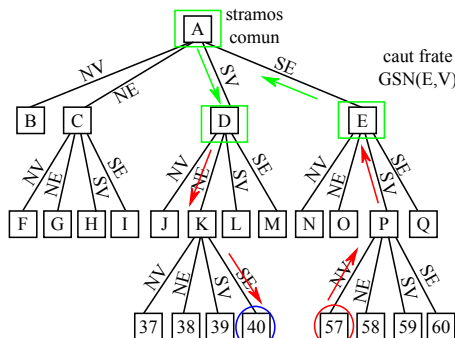
Notăție: $GCN(NOD, D_1 D_2)$ - direcția de adiacență $D_1 D_2$, unde $D_1 \in \{N, S\}$ și $D_2 \in \{E, V\}$.

Idee generală: se caută un strămoș al lui NOD vecin orizontal / vertical cu un strămoș al nodului căutat.

Algoritmul GCN

GCN(57,NV) = 40

B			F	G
			H	I
J	37	38	N	O
	39	40		
L	M	57	58	Q
		59	60	



Algoritm general

- Urcă de la nodul curent Z la părintele Y , până când $Y \rightarrow D_3 D_4 = Z$ și $D_1 D_2 \neq D_3 D_4$.
- Dacă $D_1 \neq D_3$ și $D_2 \neq D_4$ atunci Y este strămoș comun \Rightarrow coboară din Y pe arce complementare cu cele pe care s-a urcat.
- altfel
 - dacă $D_1 \neq D_3$ atunci caută vecinul $X = GSN(Y, D_2)$.
 - altfel dacă $D_2 \neq D_4$ atunci caută vecinul $X = GSN(Y, D_1)$.
- Coboară din X pe arce complementare cu cele pe care s-a ajuns la Y .

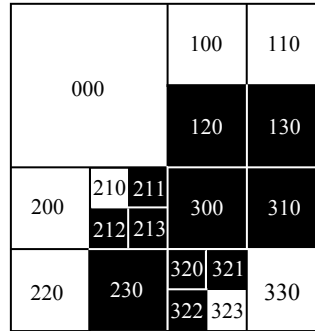
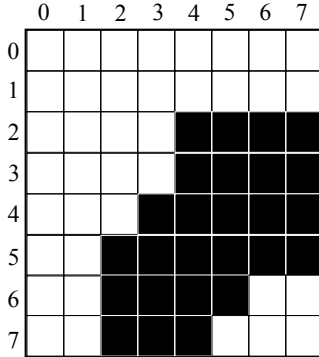
Arce complementare - (NV, SE) (NE, SV).

Implementare - Arbori quad liniari

Problematică: - numărul mare de noduri interne în cazul imaginilor reale.

Soluție: - arbori quad liniari = listă a frunzelor arborelui quad clasic în ordinea parcurgerii de la stânga la dreapta.

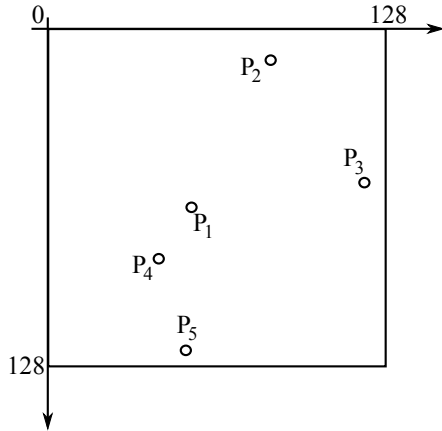
Arbori quad liniari - Exemplu



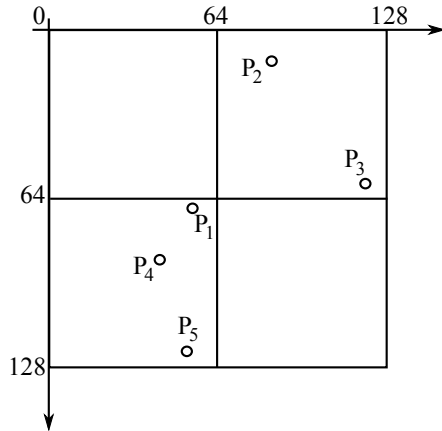
Point Region Quadrees

- **Descriere:** împart o suprafață pătrată pe baza unui set de puncte plasate pe această suprafață.
- **Idee generală:** Fie suprafața definită prin domeniul $R = [0, dim] \times [0, dim]$ și n puncte $P_i = (x_i, y_i)$, $P_i \in R, i = \overline{1, n}$
Suprafața se împarte în patru suprafețe egale în mod recursiv, până când fiecare frunză conține cel mult unul dintre punctele P_i .

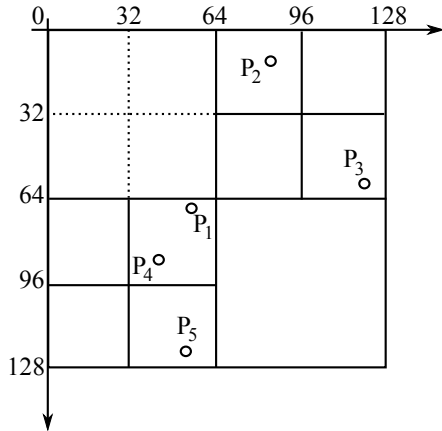
Point Region Quadtrees - Exemplu



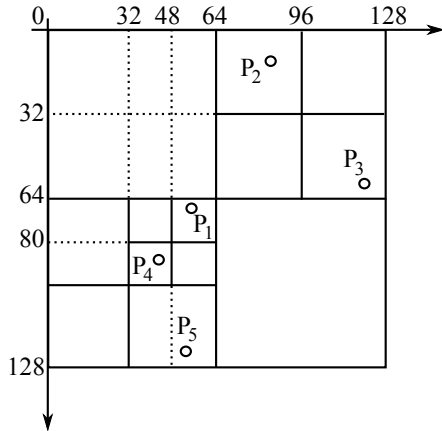
Point Region Quadtrees - Exemplu



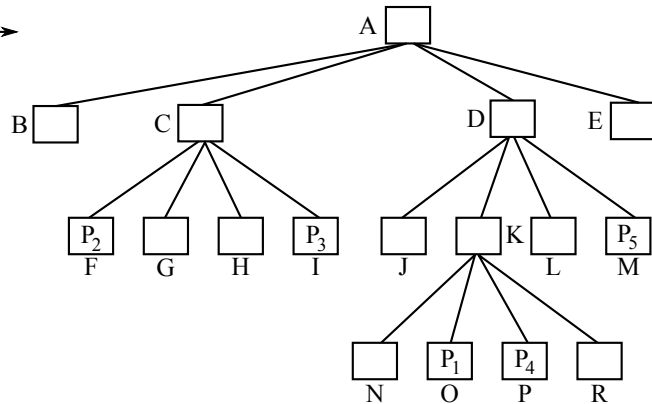
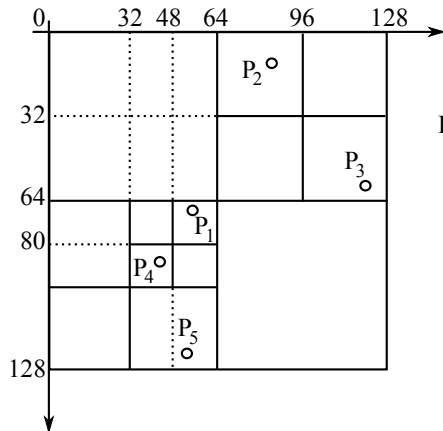
Point Region Quadtrees - Exemplu



Point Region Quadtrees - Exemplu



Point Region Quadtrees - Exemplu



Point Region Quadrees

Operații:

- PR_SEARCH(T, P)
- PR_INSERT(T, P)
- PR_CONSTRUCT($PList, dim$)
- PR_DELETE(T, P)

Câmpurile unui nod Z :

- $Z.info$ - punctul conținut de nod sau NULL
- $Z.TL$ - perechea de coordonate corespunzătoare colțului top-left al regiuni reprezentate de Z
- $Z.BR$ - perechea de coordonate corespunzătoare colțului bottom-right al regiuni din Z
- $Z.NV, Z.NE, Z.SV, Z.SE$ - legăturile către cei patru fii.

Point Region Quadrees - PR_SEARCH

```

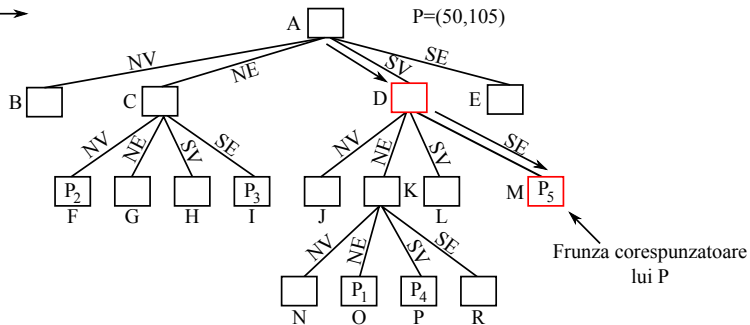
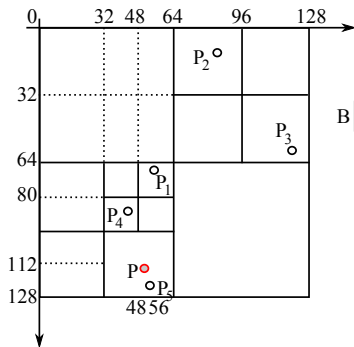
PR_ SEARCH(Z,P)
    daca P.x<0 sau P.x>Z.BR.x sau P.y<0 sau P.y>Z.BR.y atunci
        RETURN NULL
    sfarsit daca
    cat timp Z nu e frunza
        xm=(Z.TL.x+Z.BR.x)/2 si ym = (Z.TL.y+Z.BR.y)/2
        daca P.x ≤ xm si P.y ≤ ym atunci Z=Z.NV
        altfel daca P.x>xm si P.y > ym atunci
            Z=Z.SE
            altfel daca P.x ≤ xm
                Z=Z.SV
                altfel Z=Z.NE
            sfarsit daca
        sfarsit daca
    sfarsit cat timp
    RETURN Z
    
```

Point Region Quadrees - PR_INSERT

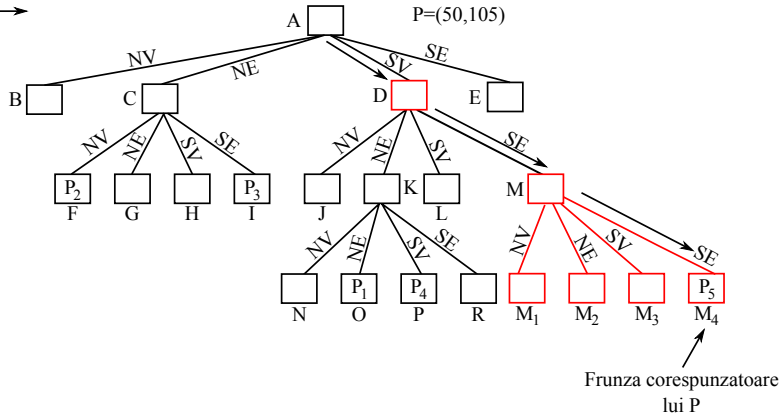
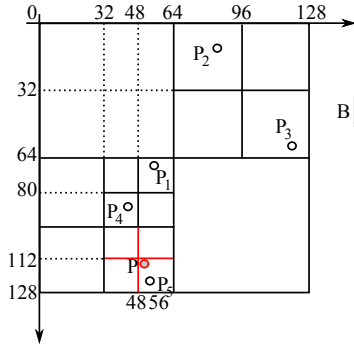
Etape:

- Se caută frunza Z corespunzătoare punctului P
- Dacă Z nu conține nici un punct atunci se inserează P
- Altfel se sparge Z în patru frunze noi, se plasează punctul din Z în frunza corespunzătoare și se reia algoritmul de la Z

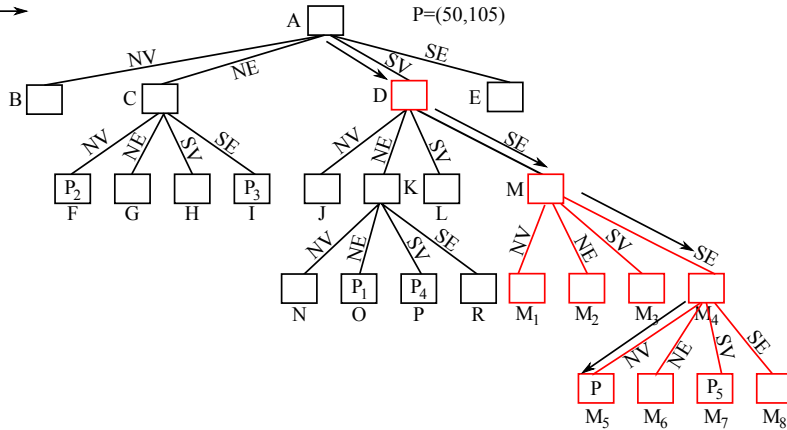
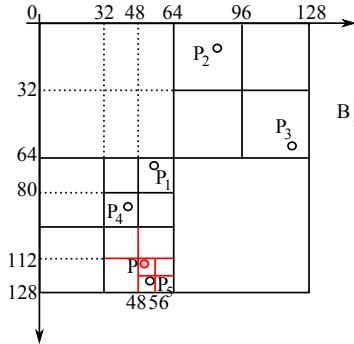
Point Region Quadtrees - PR_INSERT - Exemplu



Point Region Quadtrees - PR_INSERT - Exemplu



Point Region Quadtrees - PR_INSERT - Exemplu



Point Region Quadrees - PR_INSERT

```
PR_INSERT(T,P, dim)
  daca P.x<0 sau P.x>dim.x sau P.y<0 sau P.y>dim atunci
    RETURN NULL
  sfarsit daca
  daca T.rad=NULL atunci
    aloca T.rad
    T.rad.info = P
    T.rad.TL=(0,0)
    T.rad.BR=(dim,dim)
    T.rad.SV=T.rad.SE=T.rad.NV=T.rad.NE=NULL
    RETURN
  sfarsit daca
```

Point Region Quadtrees - PR_INSERT

```
Z=T.rad
repetă
    Z=PR_SEARCH(Z,P)
    dacă Z.info=NULL atunci
        Z.info=P
    altfel
        sparge Z în 4 noduri Z1,Z2,Z3,Z4
        plasează Z.info în nodul corespunzător
        Z.NV=Z1, Z.NE=Z2, Z.SV=Z3, Z.SE=Z4
        Z.info = NULL
    sfarsit dacă
pană când Z = frunză
RETURN
```


Point Region Quadrees - PR_DELETE

Etape:

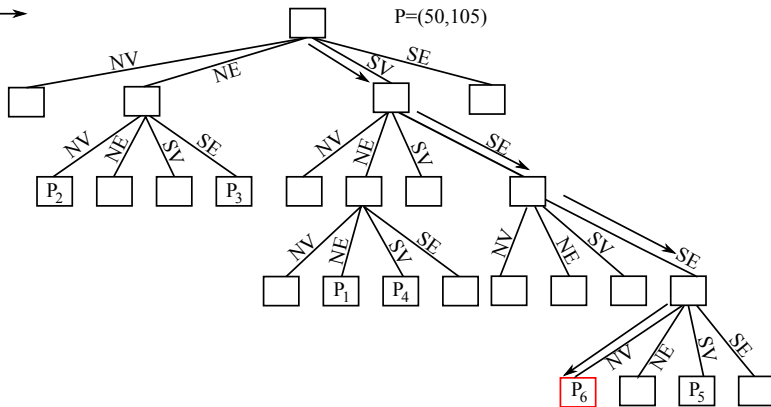
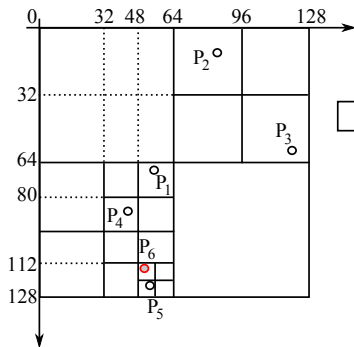
- Se caută frunza Z corespunzătoare punctului P
- Dacă Z nu conține P nu există P în arbore
- Altfel se șterge P din Z și se verifică dacă se pot contopi noduri vecine cu Z

Point Region Quadrees - PR_DELETE

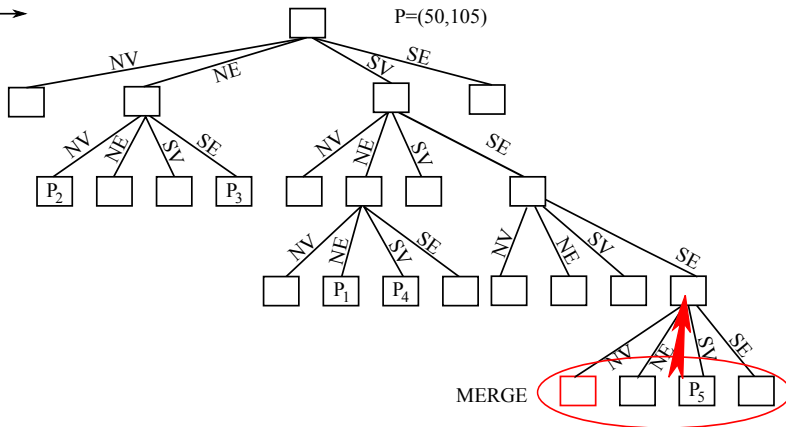
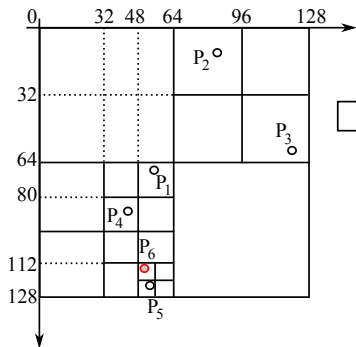
```

PR_DELETE(T,P)
    Z=PR_SEARCH(T,P)
    daca Z.info = P atunci
        Z.info = NULL
        Y=Z.p
        ok = true
        cat timp Z≠NULL si ok=TRUE
            Info = IS_MERGEABLE(Y)
            daca Info = NULL atunci ok=false
            altfel
                Y.info=Info
                Y.NV=Y.NE=Y.SV=Y.SE=NULL
                Z=Y, Y=Y.p
            sfarsit daca
        sfarsit cat timp
        altfel scrie("nu exista P")
        sfarsit daca
    RETURN
    
```

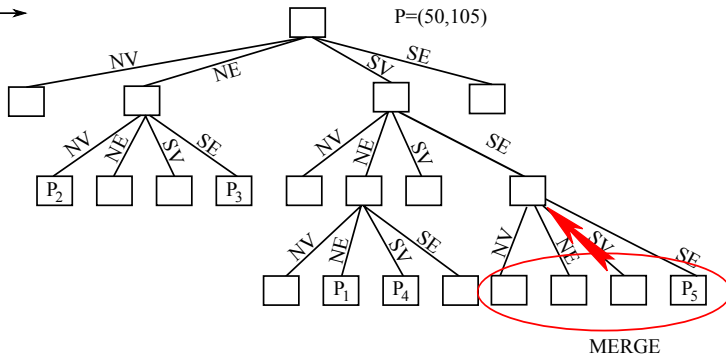
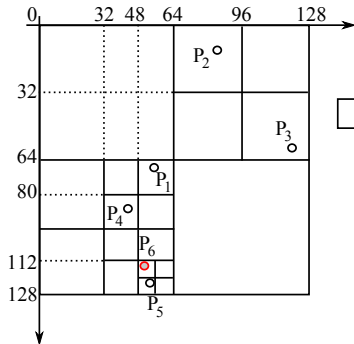
Point Region Quadtrees - PR_DELETE - Exemplu



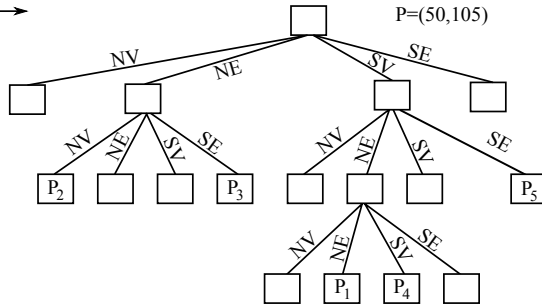
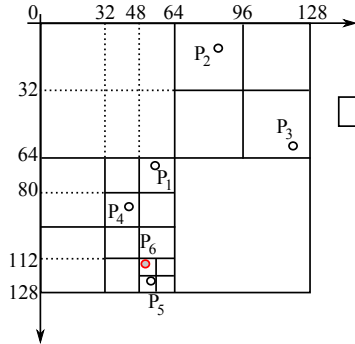
Point Region Quadtrees - PR_DELETE - Exemplu



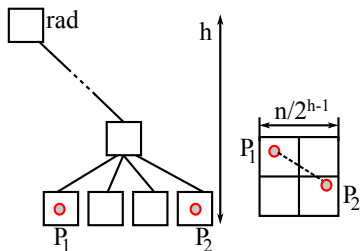
Point Region Quadtrees - PR_DELETE - Exemplu



Point Region Quadtrees - PR_DELETE - Exemplu



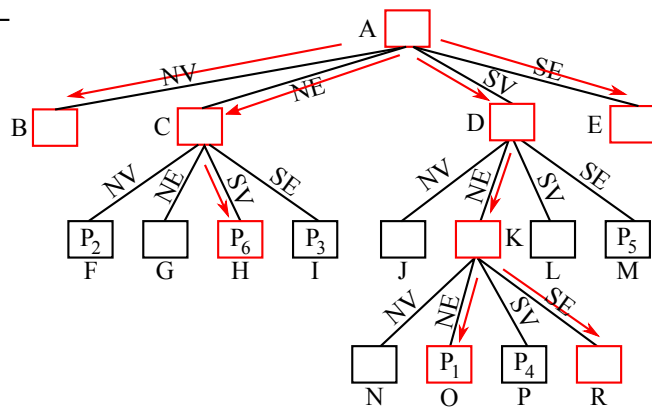
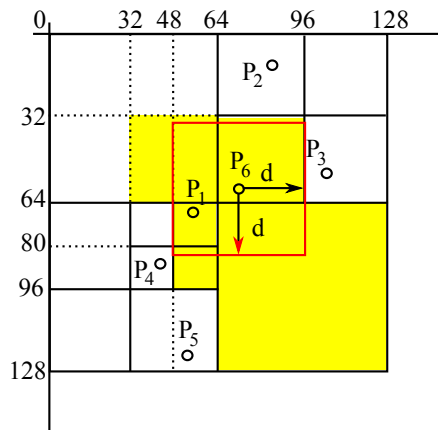
Point Region Quadtrees - Înălțime



d - distanța minimă între două puncte din blocul original
 n - dimensiunea laturii blocului original

$$h \leq \log_2 \left(\sqrt{2}n/d \right) + 1$$

Aplicație - Căutarea celor mai apropiate puncte de un punct dat



Rezumat

- Quadrees - introducere
- Quadrees pentru regiuni
 - Descompunerea recursivă a unei regiuni în 4 regiuni pătrate egale, până la obținerea de suprafețe egale și construcția quadtree-ului asociat
 - Vecini
 - Algoritmul GSN pentru găsirea vecinilor pe orizontală - verticală
 - Algoritmul GCN pentru găsirea vecinilor diagonali
 - Construirea codului pentru frunze și quadtree liniari

Rezumat

- PR-trees
 - Descriere - partiționare a spațiului în funcție de o mulțime de puncte din spațiu
 - Exemplu de partiționare a spațiului în funcție de o mulțime de puncte
 - Operații: Căutare, inserție, construcție, ștergere
 - Înălțimea unui PR-tree

kd-trees

kd-trees - arbori binari pentru stocarea și manipularea de date k-dimensionale

- fiecare nod împarte spațiul k-dimensional printr-un hiperplan perpendicular pe direcția uneia dintre axele de coordonate.
- punctele aflate de-o parte a hiperplanului din nodul x vor fi plasate în descendentul stâng iar celelate în descendentul drept al lui x .
- direcția după care se alege hiperplanul depinde de adâncimea nodului curent.

kd-trees

kd-trees - separarea datelor prin hiperplane.

Se consideră setul de puncte $M = \{P_1, P_2, \dots, P_n\}$ cu $P_i = (x_0^{(i)}, x_1^{(i)}, \dots, x_k^{(i)})$.

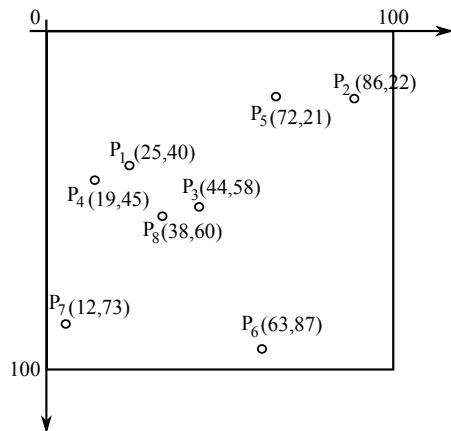
Atunci:

- În rădăcină se plasează punctul P_1 ;
- Se separarea spațiul după prima componentă: toate punctele P_i din M cu $x_0^{(i)} < x_0^{(1)}$ se plasează în subarborele stâng al rădăcinii și punctele cu $x_0^{(i)} > x_0^{(1)}$ se plasează în subarborele drept;
- Inserția punctelor în arbore se face ca la arborele binar de căutare, dar comparația punctelor din noduri la adâncimea j se face pe baza componentei $j \% k$, adică: dacă nodul curent cu care compar $Y = (y_0, y_1, \dots, y_k)$ se află pe nivelul j atunci punctul $P_i = (x_0^{(i)}, x_1^{(i)}, \dots, x_k^{(i)})$ va fi inserat la stânga lui X , dacă $x_{j \% k}^{(i)} < y_{j \% k}$ și la dreapta altfel.

kd-trees - Exemplu

Se consideră mulțimea

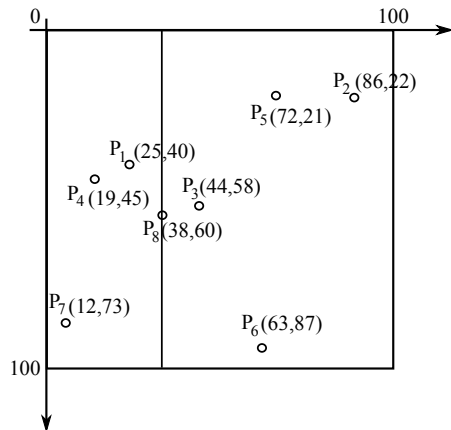
$$A = \{(25, 40), (86, 22), (44, 58), (19, 45), (72, 21), (63, 87), (12, 73), (38, 60)\}$$



kd-trees - Exemplu

Se consideră mulțimea

$$A = \{(25, 40), (86, 22), (44, 58), (19, 45), (72, 21), (63, 87), (12, 73), (38, 60)\}$$

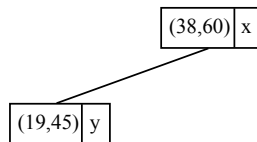
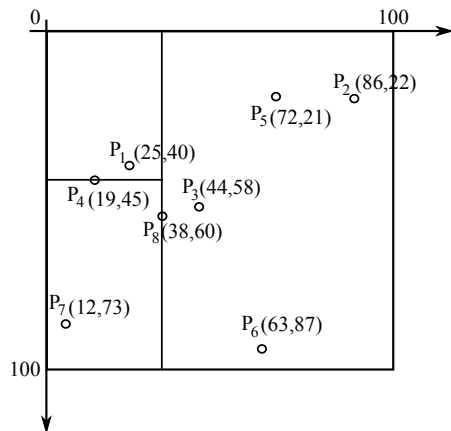


(38,60)	x
---------	---

kd-trees - Exemplu

Se consideră mulțimea

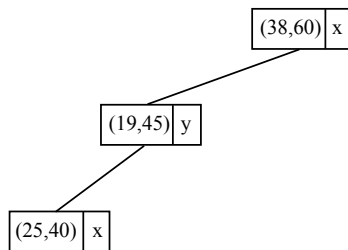
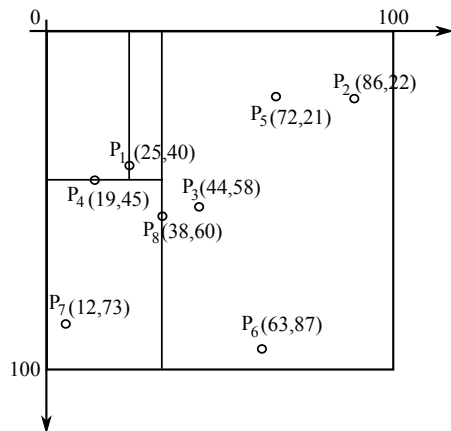
$$A = \{(25, 40), (86, 22), (44, 58), (19, 45), (72, 21), (63, 87), (12, 73), (38, 60)\}$$



kd-trees - Exemplu

Se consideră mulțimea

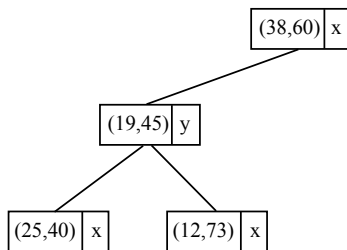
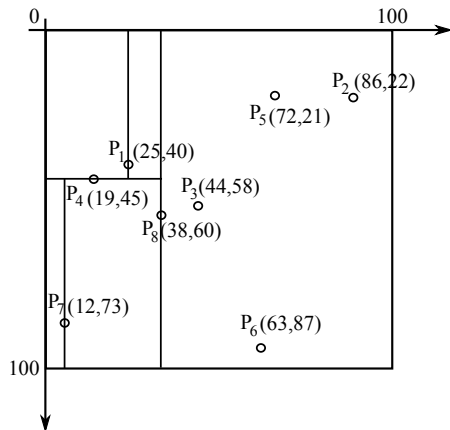
$$A = \{(25, 40), (86, 22), (44, 58), (19, 45), (72, 21), (63, 87), (12, 73), (38, 60)\}$$



kd-trees - Exemplu

Se consideră mulțimea

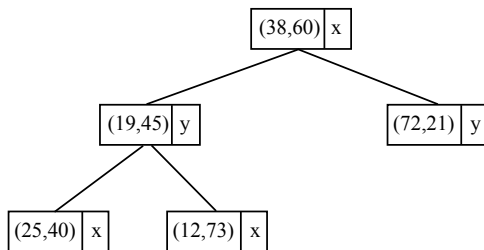
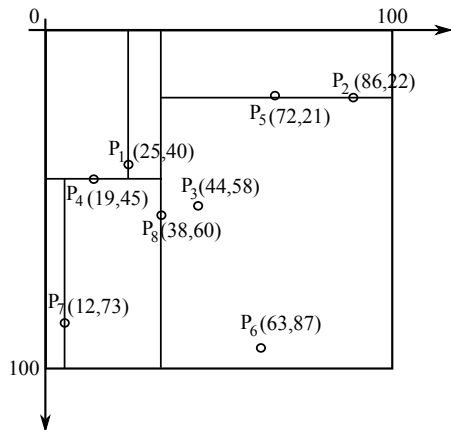
$$A = \{(25, 40), (86, 22), (44, 58), (19, 45), (72, 21), (63, 87), (12, 73), (38, 60)\}$$



kd-trees - Exemplu

Se consideră mulțimea

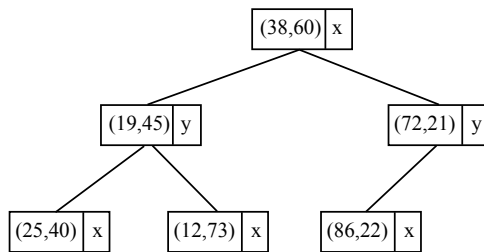
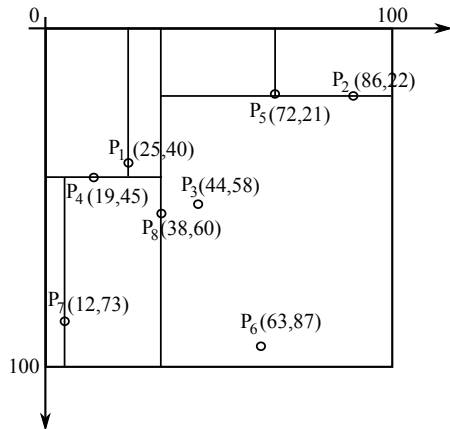
$$A = \{(25, 40), (86, 22), (44, 58), (19, 45), (72, 21), (63, 87), (12, 73), (38, 60)\}$$



kd-trees - Exemplu

Se consideră mulțimea

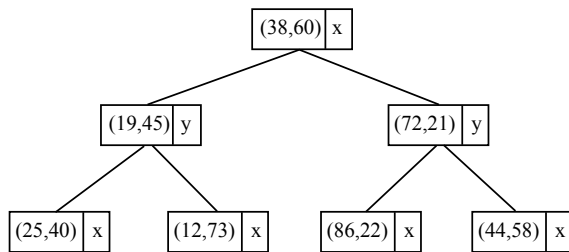
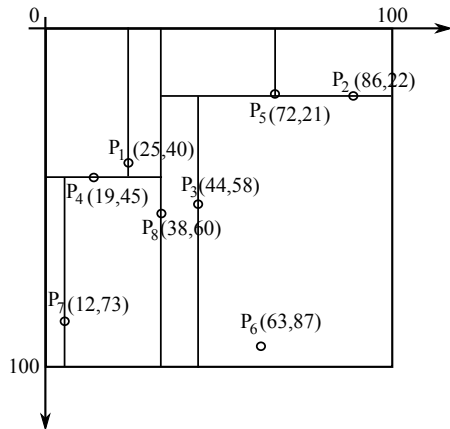
$$A = \{(25, 40), (86, 22), (44, 58), (19, 45), (72, 21), (63, 87), (12, 73), (38, 60)\}$$



kd-trees - Exemplu

Se consideră mulțimea

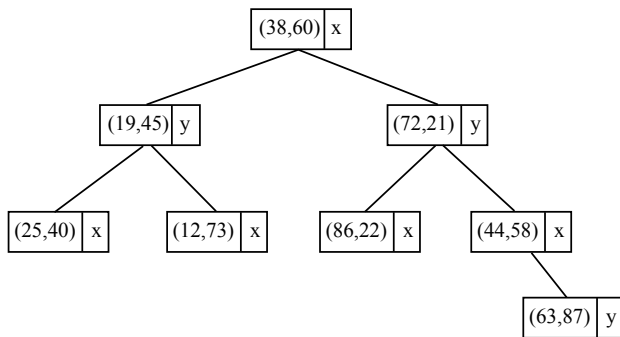
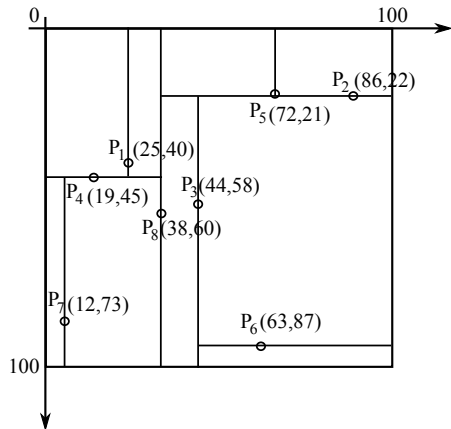
$$A = \{(25, 40), (86, 22), (44, 58), (19, 45), (72, 21), (63, 87), (12, 73), (38, 60)\}$$



kd-trees - Exemplu

Se consideră mulțimea

$$A = \{(25, 40), (86, 22), (44, 58), (19, 45), (72, 21), (63, 87), (12, 73), (38, 60)\}$$



kd-trees

Observații: - la construcția unui kd-tree dintr-o mulțime de puncte M

- la fiecare moment se selectează un punct din M față de care se realizează descompunerea în hiperplane
- se poate alege la fiecare moment primul punct din mulțime care nu a fost încă introdus \Rightarrow arbore dezechilibrat
- Recomandabil: la fiecare moment se alege punctul cu valoarea mediană a componentei corespunzătoare iterației curente \Rightarrow arbore echilibrat

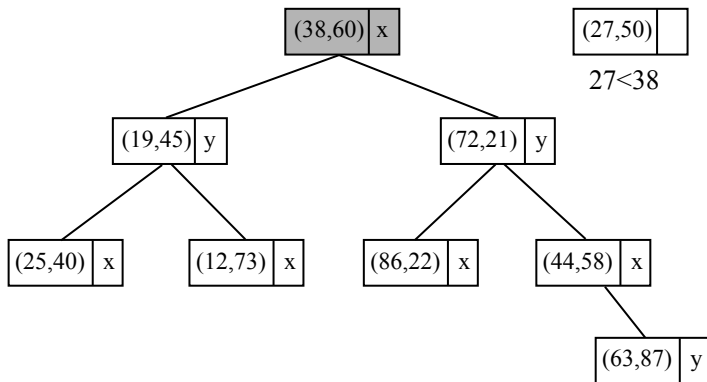
kd-trees - BUILD_KD_TREE

```

BUILD_KD_TREE(A,j)
  daca A=∅ atunci
    RETURN NULL
  P = mediana(A,j)
  A1=stanga(A,P,j)
  A2=dreapta(A,P,j)
  Z.info=P
  Z.plan=j
  Z.st=BUILD_KD_TREE(A1,(j+1) MOD k)
  Z.dr=BUILD_KD_TREE(A2,(j+1) MOD k)
  sfarsit daca
RETURN Z
  
```

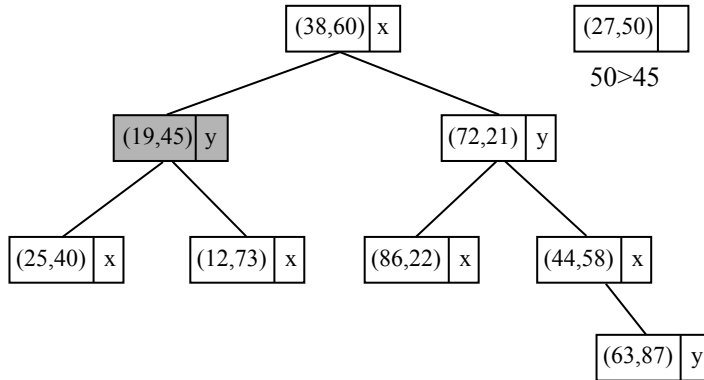
kd-trees - Inserția

Inserția - la fel ca la arborii binari de căutare, doar că la fiecare nivel se ține cont de componenta potrivită pentru comparare.



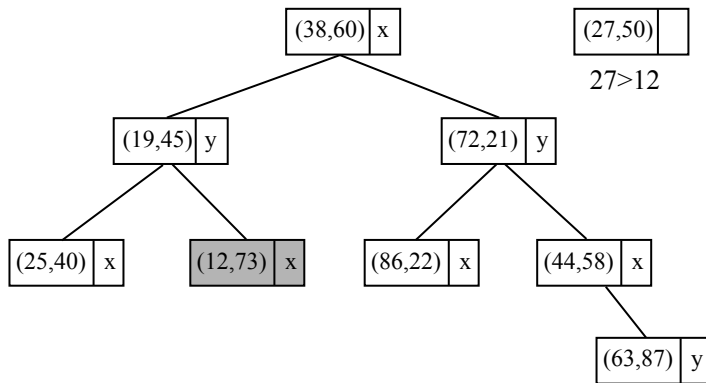
kd-trees - Inserția

Inserția - la fel ca la arborii binari de căutare, doar că la fiecare nivel se ține cont de componenta potrivită pentru comparare.



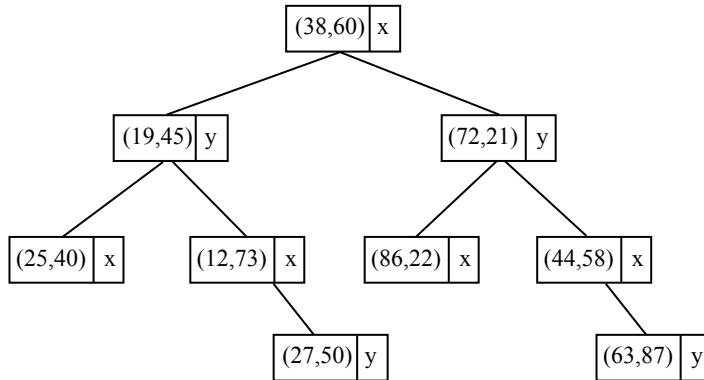
kd-trees - Inserția

Inserția - la fel ca la arborii binari de căutare, doar că la fiecare nivel se ține cont de componenta potrivită pentru comparare.



kd-trees - Inserția

Inserția - la fel ca la arborii binari de căutare, doar că la fiecare nivel se ține cont de componenta potrivită pentru comparare.



kd-trees - Aplicații

Aplicații - căutarea rapidă de puncte sau regiuni chiar cu specificații parțiale:

- valori precizate doar pentru unele dintre câmpuri
- valori ce satisfac inegalități (ex: toate punctele aflate la o anumită distanță de un punct dat)