MODULUL 9 - Quadtrees - Arbori Quad

Arborii quad sunt structuri de date ierarhice care au la bază un principiu de descompunere recursivă a spaţiului de tip *divide et impera*. Denumirea provine de la faptul că fiecare nod intern al unui arbore quad are 4 fii.

Arborii quad se diferenţiază după:

- Tipul de date pe care îl stochează: point data, regiuni, curbe, suprafeţe, volume.
- Principiul de descompunere: descompunere în părţi egale la fiecare nivel (eg. poligoane regulate) sau descompunere pe baza anumitor condiţii de intrare.
- Rezoluţia descompunerii: poate fi fixată dinainte sau poate depinde de anumite proprietăţi ale datelor de intrare.

1 Quadtrees pentru regiuni

Un arbore quad pentru regiuni se utilizează în general pentru partiţionarea spaţiului bi- sau tridimensional în regiuni, conform unui anumit criteriu de descompunere.

Exemplu: partiţionarea unei imagini digitale în regiuni de culoare uniformă, conform algoritmului *Split and Merge*. Ideea principală a etapei de descompunere - *split* - este aceea de a descompune recursiv suprafaţa pătrată curentă în patru suprafeţe pătrate de dimensiuni egale, până când se ajunge la suprafeţe uniforme. Se consideră pentru aceasta imagini de dimensiune $2^n \times 2^n$.

În figura 1 este prezentat un exemplu de descompunere a unei imagini binare $2^3 \times 2^3$, iar în fig. 2 arborele quad corespunzător. Codurile asociate frunzelor vor fi explicate la sfârşitul secţiunii.

Observaţii:

- Structura pătrată pentru partiţionarea unei regiuni este cea mai simplă, dar există şi alte tipuri de descompuneri: triunghiuri echilaterale, triunghiuri dreptunghice isoscele, hexagoane echilaterale.
- Arborii quad cresc viteza de execuţie a anumitor algoritmi pe imagini, suprafeţe, structuri geometrice, dat fiind că majoritatea acestor algoritmi se reduc la o parcurgere, de obicei în preordine, a quadtree-ului asociat.

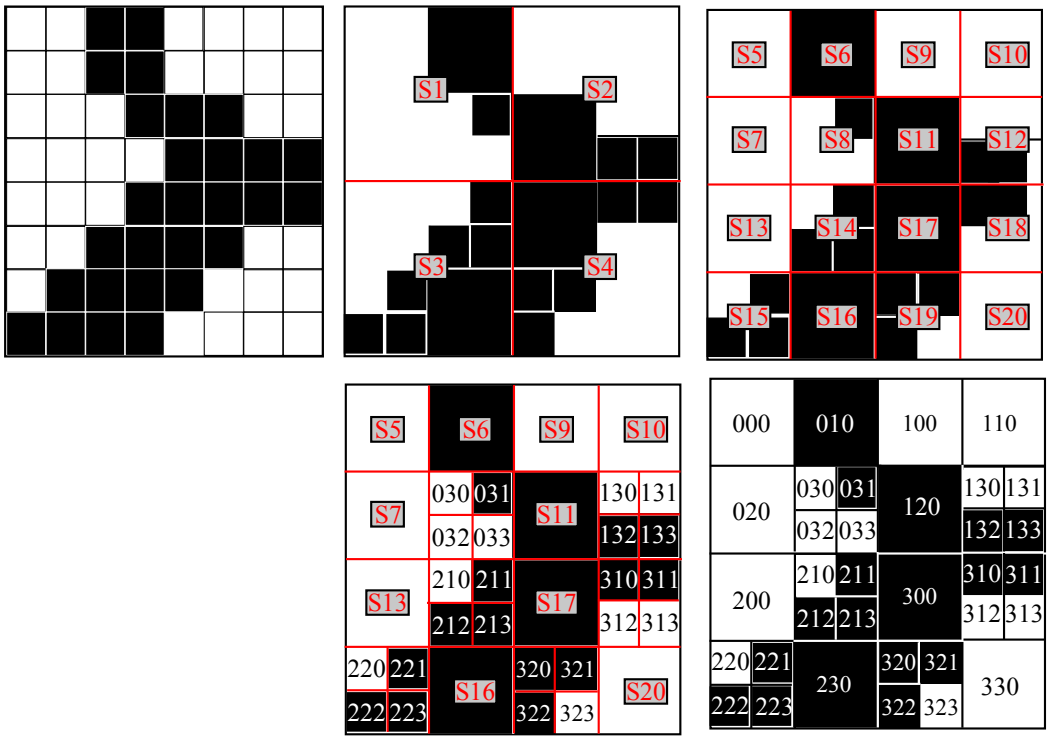

Figure 1: a) Imaginea binară originală; b) Rezultatul descompunerii

- În diferite situaţii - de exemplu etapa de merge a regiunilor vecine uniforme - este necesară identificarea vecinilor unui anumit nod.

1.1 Adiacenţă şi vecinătate

Două regiuni se numesc vecine, dacă sunt adiacente.

Adiacenţa în spaţiu nu înseamnă neapărat o relaţie simplă în cadrul nodurilor corespunzătoare din arbore. De exemplu, în figura 1 - regiunile 211 şi 033 sunt adiacente, dar în cadrul quadtree-ului corespunzător din figura 2 nu sunt nici în relaţia de părinte-fiu, nici în relaţia de frate-frate. Se pune deci problema găsirii în arbore a două noduri, care în spaţiu reprezintă regiuni vecine.

Fiecare nod al unui quadtree corespunde unei regiuni/unui bloc din imagine. Fiecare bloc are 4 laturi şi 4 colţuri.

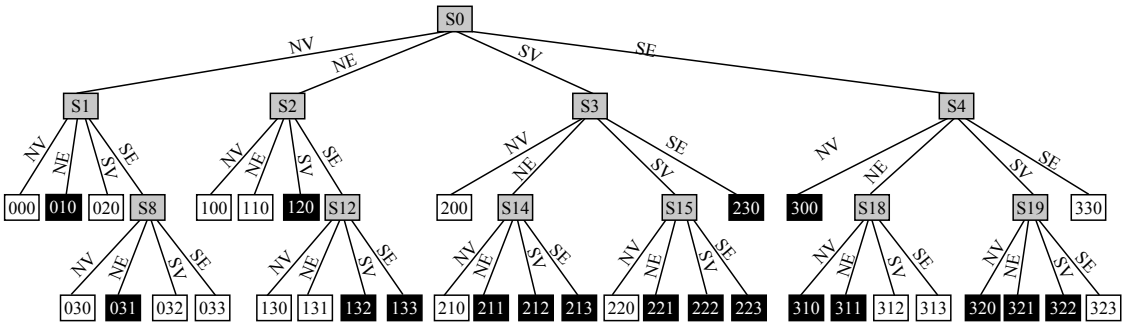

Figure 2: Arborele quad coresounzător imagnii binare din fig. 1

Notaţii:

- Cele patru laturi ale unui bloc: N (nord), S (sud), E (est), V (vest).
- Cele patru colţuri: NV, NE, SV, SE

Adiacenţă: spunem că două blocuri P şi Q disjuncte sunt adiacente după direcţia D ($D = N, S, E, V, NV, NE, SV, SE$), dacă

- P are o parte din latura de pe direcţia D comună cu Q - de exemplu în figura 3 blocul F este vecin la E cu blocul G.
- Colţul din direcţia D al blocului P este adiacent cu colţul opus al blocului Q - în exemplu, blocul 38 este NE adiacent cu blocul H

Probleme:

(1) Determinarea vecinilor care sunt adiacenţi cu îngtreaga latură a unui anumit bloc - de exemplu în figură, blocul I la V cu blocul H, sau blocul 37 la V cu blocul J. În acest caz problema se reduce pentru un anumit bloc P la determinarea celui mai mic bloc cu latura mai mare sau egală cu cea a lui P şi care este adiacent cu P după direcţia D.

(2) Determinarea acelor vecini care sunt adiacenţi numai cu un segment din latura unui anumit bloc - de exemplu în figură blocul 57 la V cu blocul M. În acest caz vecinul căutat trebuie specificat prin informaţii suplimentare

În continuare prezentăm tipurile de cereri care pot fi efectuate asupra unui quad-tree pentru determinarea de vecini.

Notaţii:

- G ="greater then or equal"
- C = "corner"
- S = "side"
- N = "neighbor"

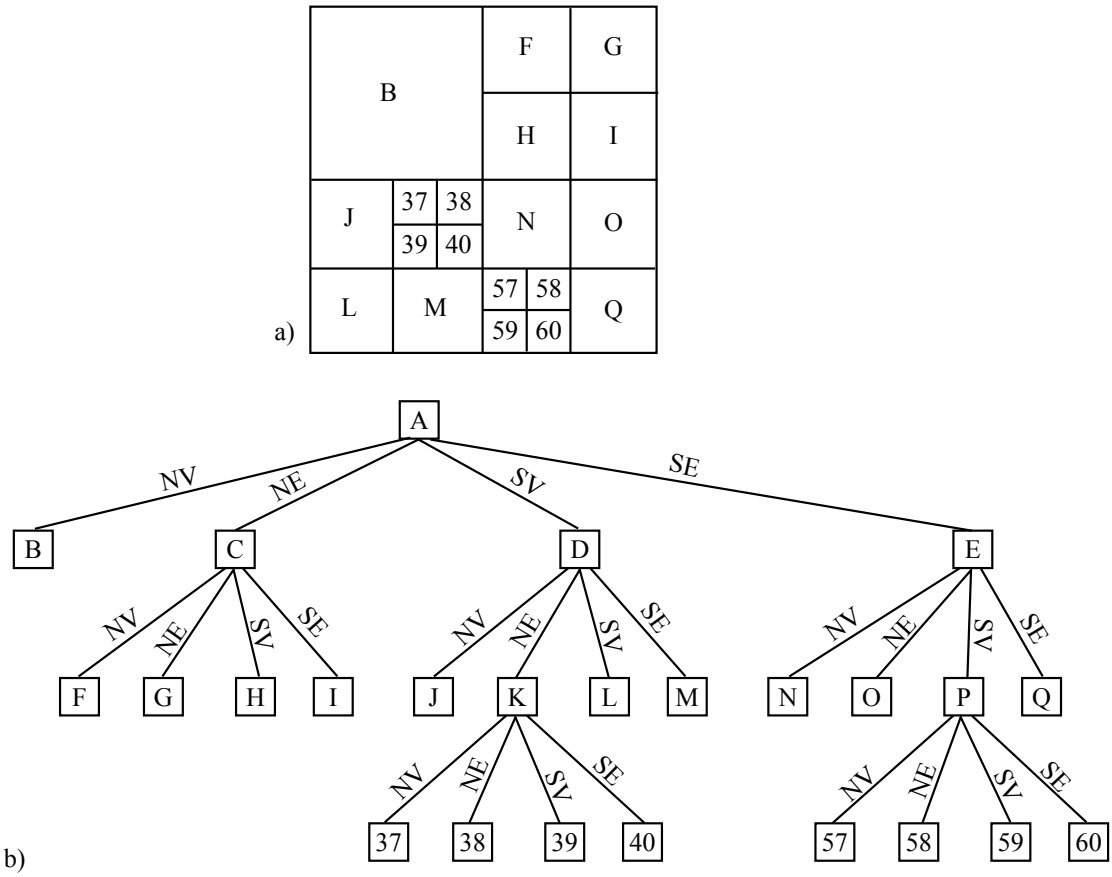

Figure 3: a) Suprafaţă descompusă în suprafeţe pătrate uniforme; b) Arborele quad corespunzător

Cereri

(1) **GSN(P,D)** = cel mai mic bloc adiacent cu latura din direcţia D a blocului P şi care are latura mai mare sau egală cu latura lui P.
Exemplu: $GSN(J, N) = B$; $GSN(N, V) = K$ (care se descompune în blocurile 37,38,39,40).

(2) **CSN(P,D,C)**= cel mai mic bloc care este adiacent cu latura D şi colţul C al nodului P.
Exemplu: $CSN(J, E, SE) = 39$; $CSN(Q, V, NV) = 58$.

(3) **GCN(P,C)** = cel mai mic bloc adiacent cu P şi aflat de partea opusă a colţului C al blocului P şi care are latura mai mare sau egală cu latura lui P. **Atenţie:** acest bloc trebuie să intersecteze suprafaţa determinată de colţul opus al colţului C.
Exemplu: $GCN(J, NE) = B$ NU 37; $GCN(B, SE) = E$; $GCN(58, NV) = N$.

(4) **CCN(P,C)** = cel mai mic bloc aflat de partea opusă a colţului C al blocului P. Şi în acest caz este obligatoriu ca blocul astfel determinat să se intersecteze cu suprafaţa determinată de colţul opus colţului C al lui P.

Exemplu: $CCN(58, NV) = N$; $CCN(59, NE) = 58$.

Observaţii:

- GSN, CSN, GCN, CCN nu definesc relaţii 1 - la - 1. Pot exista mai multe blocuri care au acelaşi vecin: $GSN(H, E) = B$ şi $GSN(F, E) = B$.
- Relaţiile de vecinătate astfel definite nu sunt neapărat simetrice: $GSN(H, E) = B$, dar $GSN(B, V) = C$.
- Un bloc care nu se află pe marginea imaginii are minim 5 vecini. Acest lucru poate fi dedus din următoarele observaţii:
  - Un bloc nu poate fi adiacent cu două blocuri de dimensiuni mai mari aflate pe direcţii opuse (vezi blocul 37, fig. 3).
  - Un bloc poate avea doar doi vecini de latură mai mare, aflaţi pe direcţii care nu sunt opuse (ex: vest şi nord). Pe celelate două laturi şi pe un colţ mai sunt vecini mai mici sau egali ca dimensiune. (vezi fig. 3).
- Un nod are maxim 8 vecini (de dimensiuni mai mari sau egale).

În continuare vor fi considerate doar cererile GSN şi GCN.

1.2 Determinarea vecinilor adiacenţi după o direcţie verticală sau orizontală

Problematică: determinare unui bloc Q vecin al lui P, $Q = GSN(P, D)$.

Idee generală este aceea de a urca de la nodul P către primul strămoş comun cu Q, iar apoi de a coborî de la acest strămoş comun către Q.

Observaţii:

- Dacă direcţia $D = E$, căutăm vecinul de la E, deci P se află pe ramuri SV sau NV faţă de strămoşul comun.
- Dacă direcţia este $D = V$, căutăm vecinul la V deci P se află pe ramuri SE sau NE faţă de strămoşul comun.
- Dacă direcţia $D = N$, căutăm vecinul de la N, deci P se află pe ramuri SV sau SE faţă de strămoşul comun.
- Dacă direcţia $D = S$, căutăm vecinul de la S, deci P se află pe ramuri NV sau NE faţă de strămoşul comun

Stămoşul comun se obţine când se urcă de la nodul curent către părinte pe o ramură ce nu conţine direcţia de căutare!

Exemplu: $GSN(N, V) = K$. Se observă că se urcă de la N către E, N fiind descendentul NV al lui E. Apoi de la E spre A, E fiind descendentul SE al lui A. În acest moment subarborele care îl conţine pe K se află la **EST** faţă de nodul curent, care este chiar rădăcina. Deci vecinul de la **VEST** va fi pe una dintre ramurile NV sau SV ale lui A. Coborârea în arbore către vecinul căutat se face pe ramuri simetrice faţă de direcţia D relativ la ramurile pe care s-a făcut urcarea către A.

Atenţie: urcarea s-a făcut pe ramurile NV, SE, iar simetria este $D = V$, deci ramurile simetrice vor fi SV (simetricul lui SE faţă de verticală) şi NE (simetricul lui NV faţă de verticală) şi se observă că se ajunge la vecinul căutat K.

Algoritm general pentru GSN

```
GSN(P,D)
      nod=P
      parinte=nod.p
      Stiva=∅
      cat timp (parinte≠NULL si ramura pe care urc de la nod la parinte
               contine D)
               pune pe Stiva simetricul ramurii de urcare de la nod la parinte
               fata de directia D
               nod=parinte
               parinte=nod.p
      sfarsit cat timp
      daca parinte=NULL atunci
           RETURN NULL
      sfarsit daca

      plaseaza pe stiva ramura simetrica fata de D
      a ramurii corespunzatoare a lui nod fata de parinte

      nod=parinte
      cat timp Stiva≠ ∅
           directie ⇐ Stiva
           daca nod≠frunza atunci
                nod=nod.directie
           altfel BREAK
      sfarsit cat timp
RETURN nod
```

Exemple: considerăm din nou figura

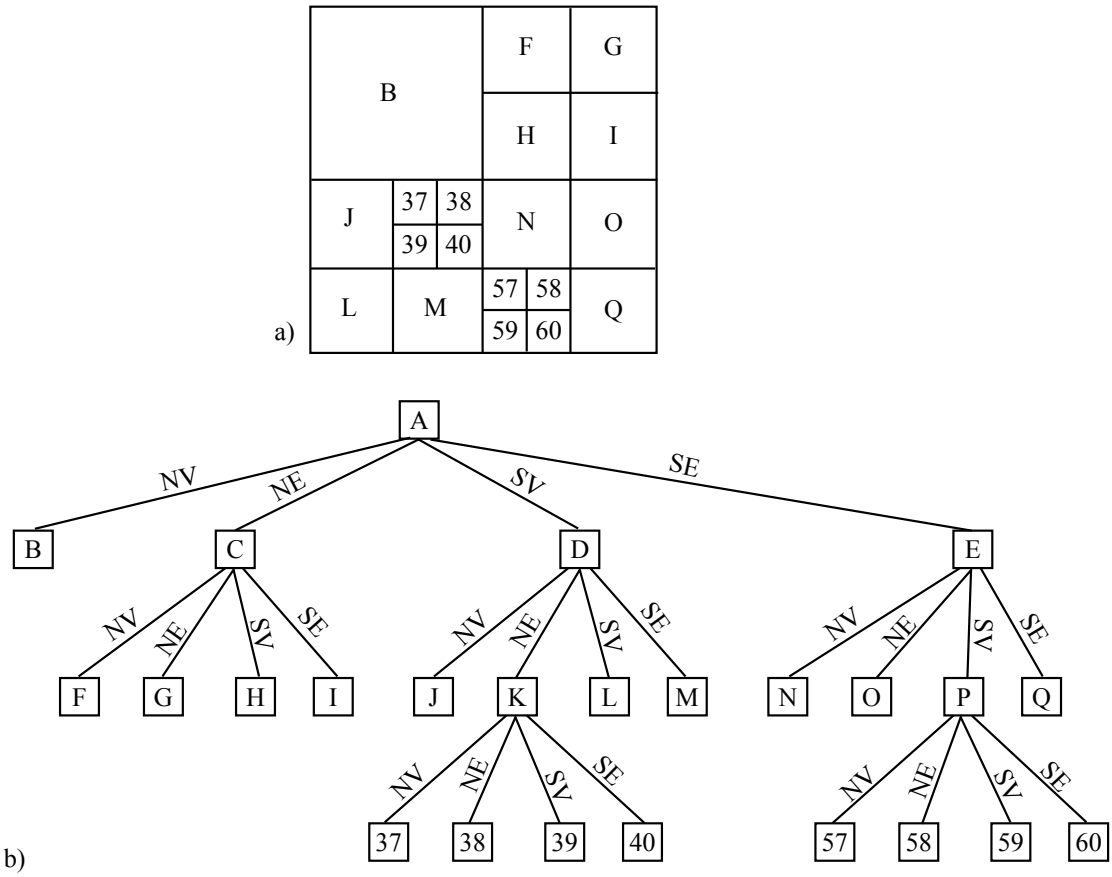

Figure 4: a) Suprafaţă descompusă în suprafeţe pătrate uniforme; b) Arborele quad corespunzător

1. GSN(39, N): Stiva = $\emptyset$, nod = (39), parinte = (K), $39 = K.SV$, $D = N$.

   Deci ramura de urcare nu conţine $N \Rightarrow$ strămoşul comun este chiar $K \Rightarrow$ se opreşte bucla cât timp.

   Se plasează pe stivă simetricul lui SV faţa de orizontală $\Rightarrow Stiva = (NV) \Rightarrow$ cobor din K pe ramura NV şi ajung la nodul 37.

   Din imagine se vede clar că s-a determinat vecinul dorit.

2. GSN(59,V): Stiva = $\emptyset$, nod = (59), parinte = (P), $(59) = (P).SV$, $D = V$

   Deci ramura de urcare conţine direcţia $V \Rightarrow$ plasez pe stivă simetricul lui SV faţă de verticală $\Rightarrow$ Stiva = (SE)

   În plus nod = (P), parinte = (E) $(P) = (E).SV$, deci ramura de urcare conţine $V \Rightarrow$ Stiva =(SE, SE).

   În plus nod = (E), parinte = (A) şi $(E) = (A).SE$. Nu mai conţine direcţia $\Rightarrow$ ne oprim din ciclare.

În plus se pune pe stivă SV, deci Stiva $= (SV, SE, SE)$.

De la rădăcină se coboară pe ramurile din stivă: $A.SV = D$, $D.SE = M$ dar M este frunză deci ne oprim şi vecinul căutat este M (se verifică corectitudinea în fig. 6)

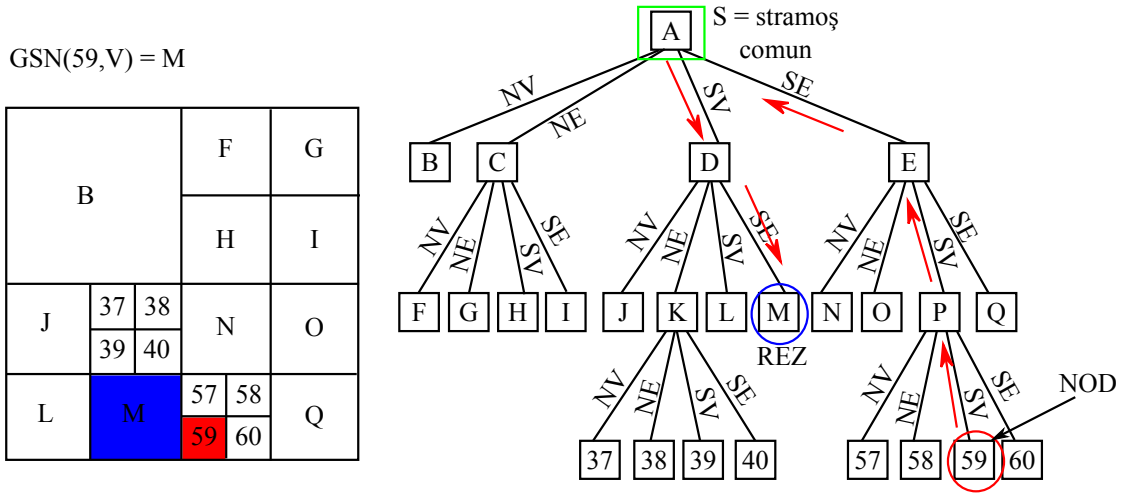

Figure 5: a) Suprafaţă descompusă în suprafeţe pătrate uniforme; b) Drumul parcurs în arbore pentru găsirea vecinului cu $GSN(59, V)$

1.3 Determinarea vecinilor diagonali

Problematică: determinare unui bloc Q vecin al lui P, $Q = GCN(P, D_1D_2), D_1 \in \{N, S\}$, $D_2 \in \{E, V\}$.

Idee generală: caut un nod strămoş al lui P vecin orizontal / vertical cu un nod strămoş nodului căutat. Acest lucru se realizează după cum urmează:

- se urcă de la nodul curent spre părinte, până când nodul se află pe partea $D_1D_2 \neq D_3D_4$ a părintelui. În acest moment nodul curent devine acest părinte, pe care îl notăm cu Q.
- **dacă** $D_1 \neq D_3$ şi $D_2 \neq D_4$ atunci Q este strămoş comun şi tot ceea ce trebuie să fac este să cobor din Q pe arce complementare cu cele pe care s-a urcat, adică dacă s-a urcat pe arc SV se coboară pe arc NE, dacă s-a urcat pe arc SE se coboară pe arc NV, etc.
- **altfel**
  - **dacă** $D_1 \neq D_3$ atunci caut vecinul R lui Q după direcţia D_2 cu algoritmul $R = GSN(Q, D_2)$, R va deveni nodul curent.

– **altfel** dacă $D2 \neq D_4$ atunci caut vecinul R lui Q după direcţia D_1 cu algoritmul $R = GSN(Q, D_1)$, R va deveni nodul curent.

– Din nodul R astfel obţinut se coboară pe arce complementare cu cele pe care s-a ajuns la Q.

Perechile de arce complementare: $(NV, SE)(NE, SV)$.

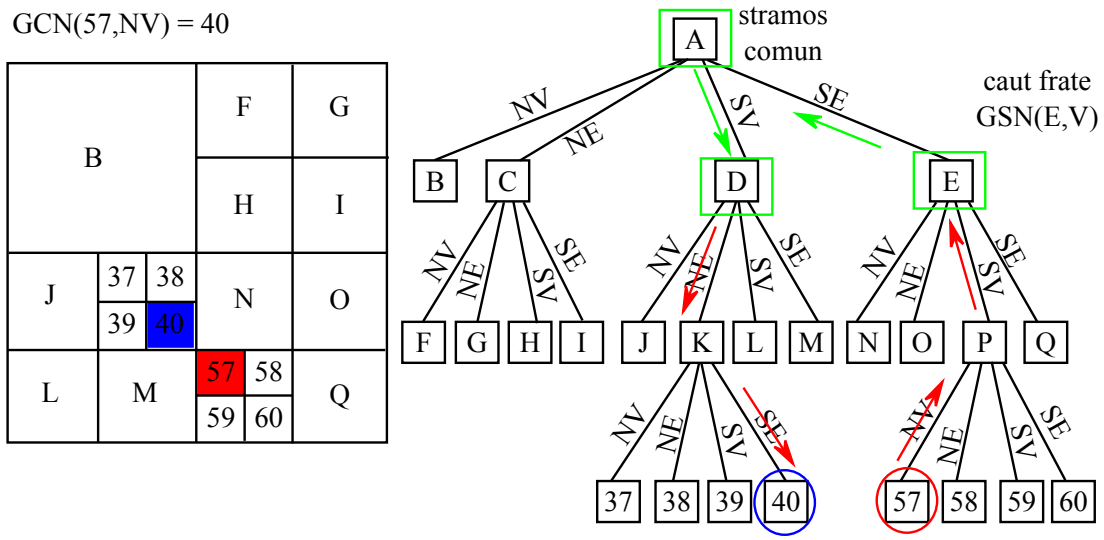

Figure 6: a) Suprafaţă descompusă în suprafeţe pătrate uniforme; b) Drumul parcurs în arbore pentru găsirea vecinului cu $GCN(57, NV)$

Exemple:

1. $GCN(57, NV)$:
   - urcăm la părintele P pe ramura NV egală cu direcţia din apel, deci continuăm la părintele lui P, E, pe ramură $SV \neq NV$
   - Dar $D_1 = N$, $D_2 = V$ şi $D_3 = S$, $D_4 = V$, deci $D_1 \neq D_3$ şi $D_2 = D_4$ $Q = E$, rezultă căutarea vecinului R al lui Q, după algoritmul $GSN(E, V)$, adică urc la părinte, cât timp sunt pe o ramură cu V. Ramura de urcare de la E la A este SE care nu conţine pe V, deci ne oprim.
   - Se mai urcă o dată. Părintele este A = rădăcina. Acum se coboară confrom algoritmului GSN pe ramura SV, adică la D, care este vecinul la vest al lui E.
   - Acum, conform algoritmului GCN, cobor pe ramură opusă a lui SV, adică NE şi ajung la K şi de aici cobor pe ramura SE şi ajung la 40.
   - Din figura 6 se observă că am ajuns exact la vecinul căutat.
2. GCN(38, NE):

- Urcăm la părintele K pe o ramură NE, egală cu direcţia de apel, deci se continuă urcarea la părintele lui K, D, pe o ramură NE, tot egală cu ramura de apel. Se urcă la părintele lui D, care este rădăcina A, pe o ramură $SV \neq NE$, cu $D_1 = N$, $D_2 = E$, $D_3 = S$, $D_4 = V$.
- Deci $D_1 \neq D_3$ şi $D_2 \neq D_4$. Am ajuns la un părinte comun al lui 38 cu vecinul căutat. Se coboară acum pe ramurile NE la C şi SV la H. Nu mai există descendenţi pentru H, deci algoritmul se opreşte.
- S-a obţinut vecinul H, care este cel căutat (vezi fig. 5).

3. GCN(N, NE):
   - Urc de la N la părintele E pe o ramură NV. Observăm $NV \neq NE$ cu $D_1 = N = D_3$, $D_2 = E$, $D_4 = V$.
   - Deci se aplică $GSN(E, N)$. Urcăm de la E la părintele A (rădăcina) pe o ramură SE. De aici se coboară pe o ramură simetrică faşă de orizontală, deci pe ramura NE la nodul C.
   - Acum s-a găsit vecinul la N al lui E, care este C.
   - De aici se continuă coborârea conform algoritmului GCN pe o ramură complementară cu NV, adcă SE şi se ajunge la I.
   - Vecinul căutat este deci I, ceea ce rezută şi din figura 5.

Aplicaţii: algoritmul *Split and Merge* de segmentare pentru imagini digitale.

1.4 Linear Quadtrees

Principala problemă a implementării clasice a unui arbore quad, adică folosind o structură de pointeri, în cazul unei imagini, este numărul mare de noduri interne necesare. De fapt în aplicaţii reale, în cazul imaginilor se utilizează o structură de date numită arbori quad liniari - *Linear Quadtrees* - care reprezintă de fapt lista frunzelor în parcurgerea de la stânga la dreapta. Fiecare frunză este o structură de tip nod, care conţine un câmp pentru culoare, un câmp pentru nivelul la care se află în arbore şi un cod unic în baza 4.

Codul fiecărei frunze este alcătuit prin interclasarea codurilor binare ale coordonatelor y şi x ale pixelului din colţul stânga-sus al blocului din imagine reprezentat prin frunză. Codul binar obţinut prin interclasarea celor două coduri este transformat într-un cod în baza 4 prin gruparea cifrelor binare două câte două într-o cifră din baza 4.

Algoritmul presupune imagini de dimensiuni $2^n \times 2^n$.

Exemplu: în imaginea $2^3 \times 2^3$ din fig. 6 coordonatele y, x ale colţului stânga sus:

- ale blocului B: sunt $0, 0$, în binar pe trei poziţii $y = 000$, $x = 000$. după interclasare codul este 000000. Grupând două câte două cifrele binare şi transformând în baza 4 obţinem codul 000
- ale blocului F: sunt $0, 4$, în binar pe trei poziţii $y = 000$, $x = 100$. după interclasare codul este 010000. Grupând două câte două cifrele binare şi transformând în baza 4 obţinem codul 100
- ale blocului G: sunt $0, 6$, în binar pe trei poziţii $y = 000$, $x = 110$. după interclasare codul este 010100. Grupând două câte două cifrele binare şi transformând în baza 4 obţinem codul 110
- ale blocului 40: sunt $5, 3$, în binar pe trei poziţii $y = 101$, $x = 011$. după interclasare codul este 100111. Grupând două câte două cifrele binare şi transformând în baza 4 obţinem codul 213

În figura 7 este prezentat modul de codificare al frunzelor obţinute prin descompunerea unei imagini binare $2^3 \times 2^3$ în blocuri de culoare uniformă.

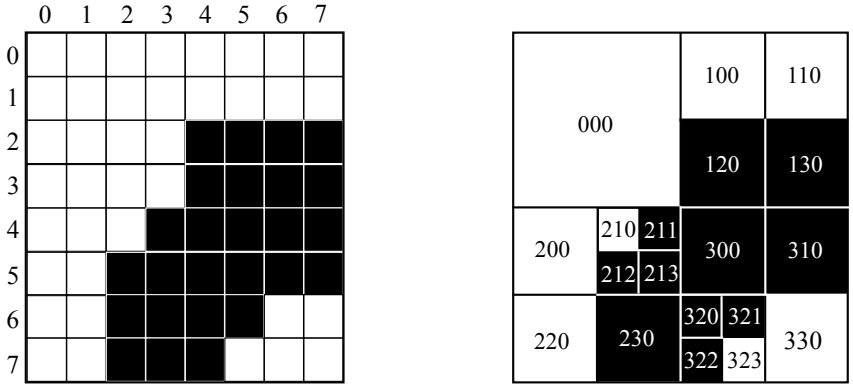

Figure 7: Suprafaţă descompusă în suprafeţe pătrate uniforme şi etichetele corespunzătoare în baza 4

Arborele corespunzător este reprezentat în fig. 8.

Determinarea vecinilor într-un QuadTree liniar

- **calculul codului unui vecin**: este realitv simplă determinarea codului unui vecin de pe acelaşi nivel de descompunere, de exemplu pentru un anumit nod, vecinii de pe acelaşi nivel care au acelaşi părinte au codul în care diferă doar cifra corespunzătoare nivelului respectiv.
- **căutarea în lista de frunze**: nu este obligatoriu ca să existe un vecin pe acelaşi nivel, de exemplu blocul cu codul 210 nu are la nord vecin pe acelaşi nivel de descompunere, ci doar pe un nivel superior. Rezultă că, după calculul codului corespunzător vecinului, acesta trebuie căutat în lista de frunze.

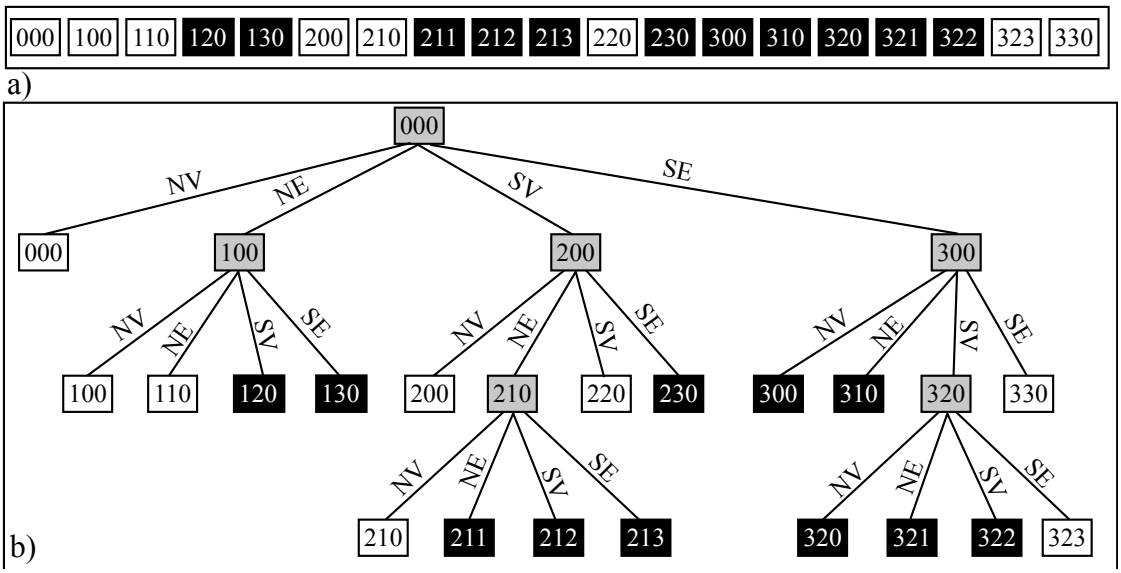

Figure 8: Arborele liniar păstrat sub forma unui vector ce conţine doar etichetele frunzelor considerate de la stânga la dreapta şi arborele quad corespunzător,

2 Point Region Quad Trees - PR-Quadtrees

Arborii quad de tip point-region împart o suprafaţă pătrată în mod similar ca şi arborii quad pentru suprafeţe/imagini, cu deosebirea că împărţirea nu se realizează pe baza culorii suprefeţei, ci pe baza unui set de puncte plasate în suprafaţa dată.

Idee generală: se consideră o suprafaţă bidimensională cu coordonatele (x, y) în domeniul $[0, dim] \times [0, dim]$ şi n puncte $P_i = (x_i, y_i)$, $i = \overline{1n}$ plasate pe această suprafaţă.

Suprafaţa se împarte în patru suprafeţe egale în mod recursiv, până când fiecare frunză conţine cel mult unul dintre punctele P_i.

Un exemplu este prezentat în figura 9.

2.1 Căutarea unui nod

Se doreşte căutarea nodului în care s-ar afla punctul P de coordonate $P.x$ şi $P.y$, dacă acest punct ar exista pe suprafaţa dată. Dacă se consideră nodul curent Z la care s-a ajuns în arbore, notăm cu $Z.top\_left$ colţul stânga sus al suprafeţei reprezentate de Z şi cu $Z.bottom\_right$ colţul dreapta jos al acestei suprafeţe. Pe baza coordonatelor acestor colţuri poate fi stabilit în care cadran ar trebui să se afle punctul P şi deci pe ce ramură trebuie coborât în arbore.

Algoritm

```
PR_SEARCH(T,P)
```

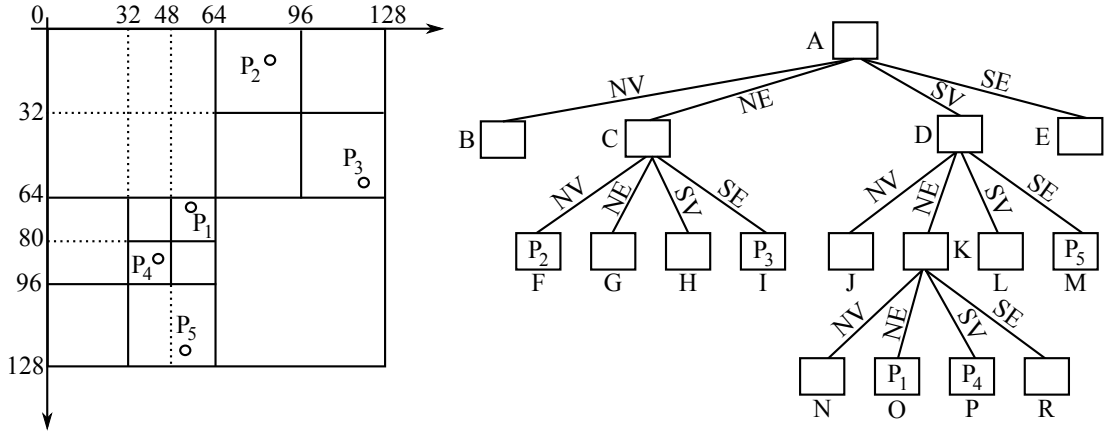

Figure 9: a) Suprafaţa pentru care se realizează împărţirea în regiuni împreună cu 5 puncte. b) PR-tree-ul corespunzător.

```
          daca P.x<0 sau P.x>T.bottom_right.x
               sau P.y<0 sau P.y>T.bottom_right.y atunci
                     RETURN NULL
          sfarsit daca
          cat timp Z nu e frunza
                    daca P.x ≤ (Z.bottom_right.x)/2
                         si P.y ≤(Z.bottom_right.y)/2 atunci
                                  Z=Z.NV
                    altfel
                         daca P.x>(Z.bottom_right.x)/2 si
                              P.y>(Z.bottom_right)/2 atunci
                                   Z=Z.SE
                         altfel
                              daca P.x ≤ (Z.bottom_right.x)/2 atunci
                                   Z=Z.SV
                              altfel Z=Z.NE
                              sfarsit daca
                         sfarsit daca
                    sfarsit daca
          sfarsit cat timp
RETURN Z
```

Exemplu: considerăm arborele din fig. 9, construit pe baza setului de puncte $\{(55, 67), (80, 15), (119, 58), (41, 85), (53, 117)\}$.

SEARCH(T, (50,105)):

$Z = A$ nu e frunză $\Rightarrow$ verific în care dintre cele 4 blocuri descendente poate fi găsit

$P = (50, 105)$.

$$\left.\begin{array}{l} 0 < P.x = 50 < 64 \\ 64 < P.y = 105 < 128 \end{array}\right\} \Rightarrow Z = Z.SV = D$$

D nu este frunză $\Rightarrow$ verific în care dintre cele 4 blocuri descendente se plasează P.

$$\left.\begin{array}{l} 32 < P.x = 50 < 64 \\ 96 < P.y = 105 < 128 \end{array}\right\} \Rightarrow Z = Z.SE = M$$

M este frunza corespunzătoare puctului (50,105).

2.2 Inserarea într-un PR-arbore

Dacă se doreşte inserarea punctului P într-un PR-arbore quad, atunci, după identificarea frunzei M care corespunde punctului P, se verifică dacă frunza respectivă conţine deja un punct, în exemplul anterior P_5:

- Daca nu, atunci se inserează P în câmpul informaţie.
- Altfel (cazul exemplului) se sparge blocul corespunzător frunzei M în 4 blocuri noi, frunza se înlocuieşte cu un nod interior, care are 4 descendenţi, corespunzători noilor blocuri obţinute, se plasează punctul aflat în frunza M în nodul nou construit corespunzător şi se continuă procesul de căutare a unei frunze corespunzătoare lui P, cu evetuale noi spargeri, dacă este necesar.

Exemplul inserţiei nodului $P = (50, 105)$ în arborele din fig. 9 este reprezentat în fig. 10.

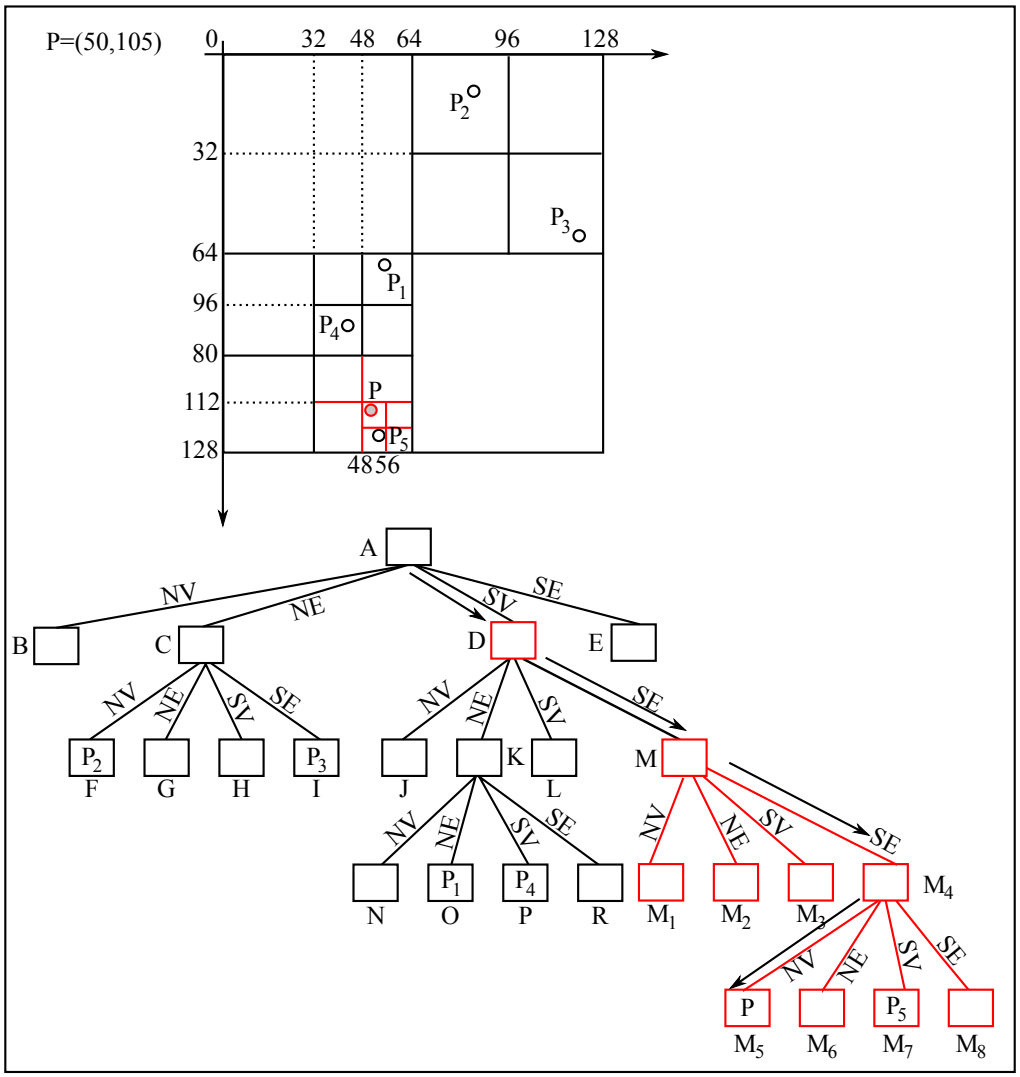

Figure 10: Inserţia punctului $P = (50, 105)$ în arborele din figură.

Algoritm

```
PR_INSERT(T, point, dim)
   daca point.x<0 sau point.x>dim
        sau point.y<0 sau point.y>dim atunci
                   Scrie("nu poate fi inserat")
                   RETURN
   sfarsit daca
   daca T = NULL atunci
        Aloca T
        T.info = point
        T.top_left = (0,0)
        T.bottom_right = (dim,dim)
        T.SV = T.SE = T.NV = T.NE = T.p = NULL
```

```
        RETURN
    sfarsit daca
    X = T
    repeta
        //cauta frunza in care se plaseaza punctul point
        Z = PR_SEARCH(X,point)
        daca Z.info = NULL atunci
            Z.info = point
        altfel
            1.  sparge suprafata din Z in patru suprafete
            plasate in noduri noi Z1, Z2, Z3 si Z4
            2.  plaseaza Z.info in nodul corespunzator Z1,Z2,Z3 sau Z4
            3.  Z.NV = Z1, Z.NE = Z2, Z.SV = Z3, Z.SE = Z4
            4.  Z.info = NULL
            5.  Z1.p = Z, Z2.p = Z, Z3.p = Z, Z4.p = Z
        sfarsit daca
        X = Z
        //acum Z nu mai este frunza si se reia cautarea
    pana cand X = frunza
RETURN
```

2.3 Construcţia unui PR-arbore quad

Construcţia unui PR-arbore quad se realizează prin inserţii succesive într-un arbore iniţial vid.

Algoritm

```
PR_CONSTRUCT(P,n,dim)
    Initializare T
    pentru i=1, n
           PR_INSERT(T, P[i])
    sfarsit pentru
RETURN T
```

2.4 Eliminarea unui punct P

Se caută frunza M corespunzătoare nodului P în arbore. Dacă M nu conţine P, atunci P nu există în arbore şi deci nu poate fi şters.

Altfel: se şterge P din frunza M.

Dacă fraţii lui M sunt frunze şi în blocul reprezentat prin aceste 4 frunze se află cel mult un punct Q din mulţimea de puncte ale PR-arborelui, atunci cele patru frunze se şterg, iar punctul Q se mută în părintele lui M, care acum devine frunză.

Această etapă se repetă, până când se ajunge la un nod ai cărui descendenţi nu sunt cu toţii frunze sau sunt 4 frunze care conţin împreună cel puţin 2 puncte.

Utilizăm funcţia IS_MERGEABLE(Y) care verifică dacă nodul Y conţine 4 descendenţi care pot fi eliminaţi şi în caz afirmativ returnează informaţia (dacă există) a unicului descendent frunză care nu este vid.

Algoritm

```
IS_MERGEABLE(Y)
      nr = 0
      info = NULL
      daca Y.NV≠ frunza sau Y.NE ≠ frunza sau Y.SV≠
           frunza sau Y.SE≠ frunza atunci
                      RETURN NULL
      sfarsit daca
      daca Y.NV.info ≠ NULL atunci
           info=Y.NV.info
           nr=nr+1
      sfarsit daca
      daca Y.NE.info ≠ NULL atunci
           info=Y.NE.info
           nr=nr+1
      sfarsit daca
      daca Y.SE.info ≠ NULL atunci
           info=Y.SE.info
           nr=nr+1
      sfarsit daca
      daca Y.SV.info ≠ NULL atunci
           info=Y.SV.info
           nr=nr+1
      sfarsit daca
      daca nr>1 atunci info=NULL
RETURN info

PR_DELETE(T,P)
      Z=PR_SEARCH(T,P)
      daca Z.info = P atunci
           Z.info = NULL
           Y = Z.p
           ok = true
           cat timp Y≠NULL si ok
```

```
                    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
```

Exemplu ştergem punctul (50,105) din arborele care conţine punctele: $\{P_1 = (55, 67)$, $P_2 = (80, 15)$, $P_3 = (119, 58)$, $P_4 = (41, 85)$, $P_5 = (53, 117)$, $P_6 = (50, 105)\}$ (fig. 11 şi 12)

Observaţii

1. Forma unui PR-QuadTree nu depinde de ordinea în care sunt inserate punctele în arbore.
2. Dacă se inserează două puncte foarte apropiate este posibil să trebuiască efectuate numeroase spargeri ale blocurilor, până se ajunge la separarea celor două puncte, ceea ce conduce la creşterea în adâncime a arborelui şi la introducerea unui număr semnificativ de noduri ce nu conţin nici un punct.

2.5 Înălţimea unui PR-QuadTree

Considerând d = distanţa minimă între două puncte din blocul original şi n dimensiunea laturii acestui bloc, atunci adâncimea h a PR-Quadtree-ului corespunzător este:

$$\log_4 n \leq h \leq \log_2 \left(\sqrt{2}n/d\right) + 1$$

Demonstraţie: evident, dacă toate frunzele sunt la aceeaşi adâncime, $h = \log_4 n$.

Altfel: cele mai mici blocuri, adică cele mai adânci frunze, se obţin pentru separarea celor mai apropiate două puncte dintr-un cadran (fig. 13). Presupunem că aceste două puncte P_1 şi P_2 se află la adâncimea h, părintele din care au provenit se afla la dâncimea $h-1$ şi avea dimensiunea laturii $L = n/2^{h-1}$. Distanţa între P_1 şi P_2 este d şi

$$d \leq \sqrt{2}n/2^{h-1}$$

De aici rezultă

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

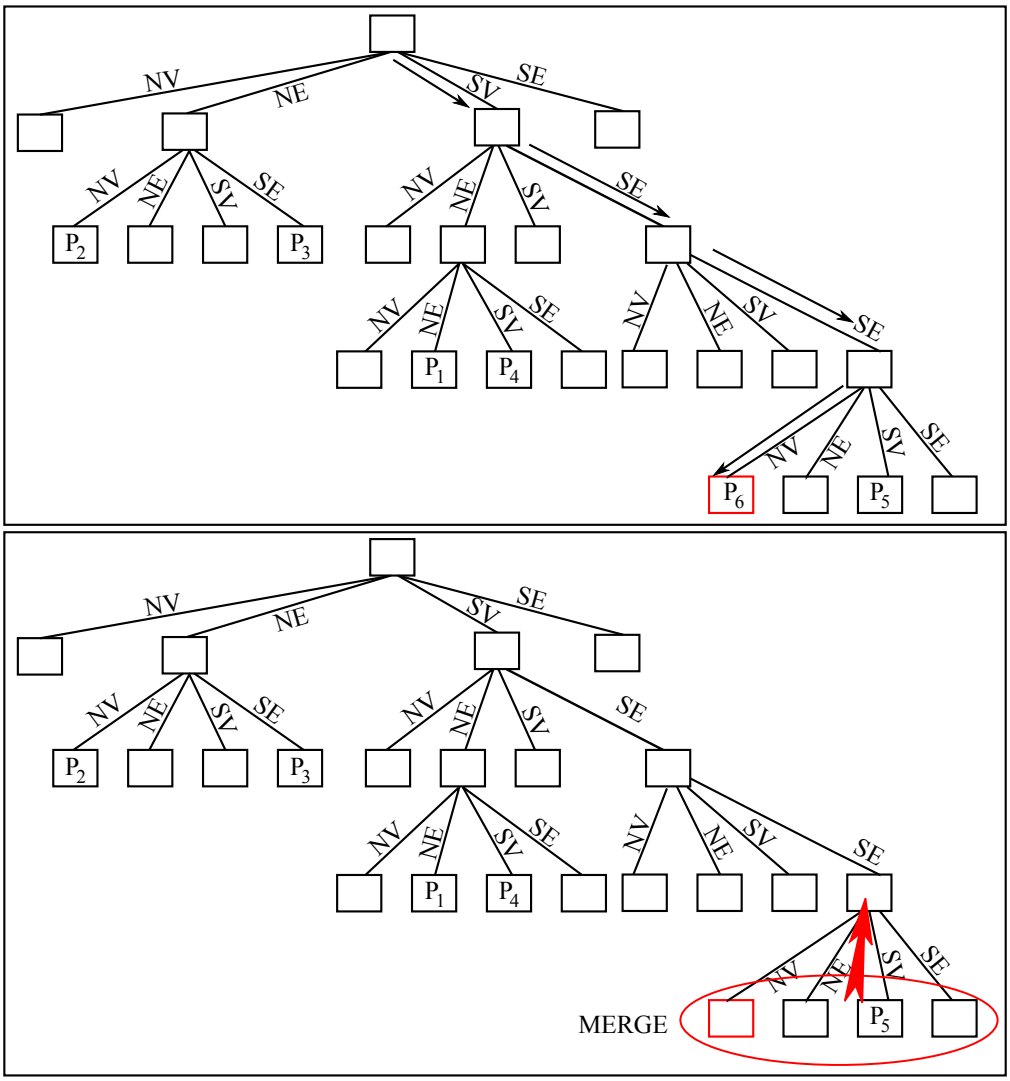

Figure 11: Ştergerea punctului $P = (50, 105)$ din arbore.

Determinarea vecinilor: acelaşi algoritm ca şi pentru Quadtrees.

Exemplu de aplicaţie

Se consideră un bloc pătrat de dimensiune $2^n \times 2^n$ ce reprezintă o hartă, pe care se află amplasate m oraşe. Acestea pot fi reprezentate cu ajutorul unui PR-arbore quad. Să se determine toate oraşele aflate la distanţa maximă d de un oraş dat P.

Idee de rezolvare: Se parcurge în preordine PR-arborele. Pentru fiecare nod curent X, se verifică dacă poate conţine oraşe la distanţa cel mult d de P. Acest lucru poate fi determinat pe baza coordonatelor colţurilor blocului $X.top\_left$ şi $X.bottom\_right$:

$$P.x - d \geq X.top\_left.x \text{ şi } P.y - d \geq X.top\_left.y$$

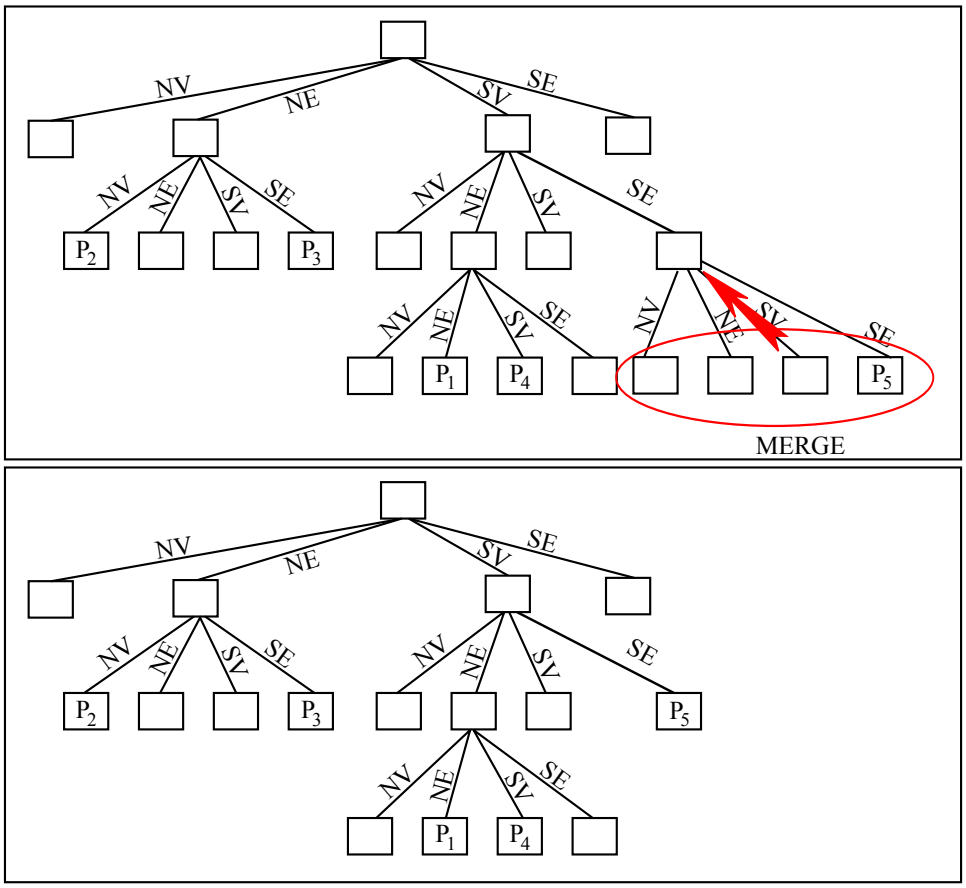

Figure 12: Ştergerea punctului $P = (50, 105)$ (continuare).

şi $P.x + d \leq X.bottom\_right.x$ şi $P.y + d \leq X.bottom\_right.y$.

În caz afirmativ se continuă coborârea pe subarborele de rădăcină X, altfel se opreşte căutarea în acel subarbore.

Un exemplu de astfel de căutarea este prezentat în figura 14.

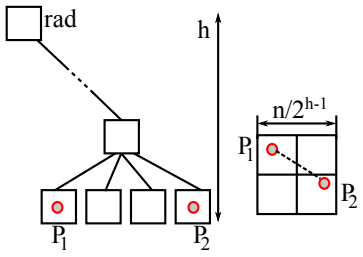

Figure 13: Cele mai apropiate două puncte de la cel mai de jos nivel.

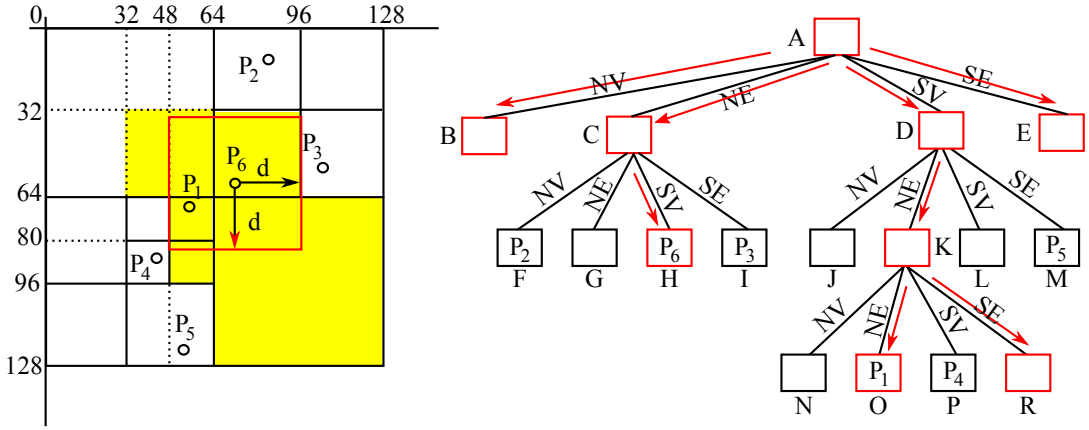

Figure 14: Căutarea punctelor aflate la distanţa cel mult d faţă de un punct dat.