Arbori AVL

Arborii AVL sau arborii echilibrati sunt arbori binari ordonati, care au in plus o proprietate de echilibru .
Proprietatea de echilibru e valabila **pentru orice nod** al arborelui si spune ca: “inaltimea subarborelui stang al nodului difera de inaltimea subarborelui drept al nodului prin cel mult o unitate”.
Structura elementelor unui arbore echilibrat poate fi reprezentata astfel:

```
struct nod
{
   int key;
   int ech;
   nod *left, *right;
};

nod * rad;
```

unde key reprezinta eticheta nodului(un numar intreg), ech- reprezinta factorul de echiligrare, iar left si right reprezinta pointeri catre copilul din stanga, respectiv din drapta.
Def1: Se numeste **inaltime a unui arbore** ca fiind lungimea celui mai lung drum de la nodul radacina la unul din nodurile terminale.
Def2: Se numeste **factor de echilibrare** diferenta dintre inaltimea subarborelui drept si inaltimea subarborelui stang.
Def3: Atasand fiecarui nod un camp care reprezinta factorul de echilibrare al sau, se spune ca **arborele binar este echilibrat** cand toti factorii de echilibrare ai nodurilor sunt -1,0,+1.
Formal, acest lucru se traduce astfel:

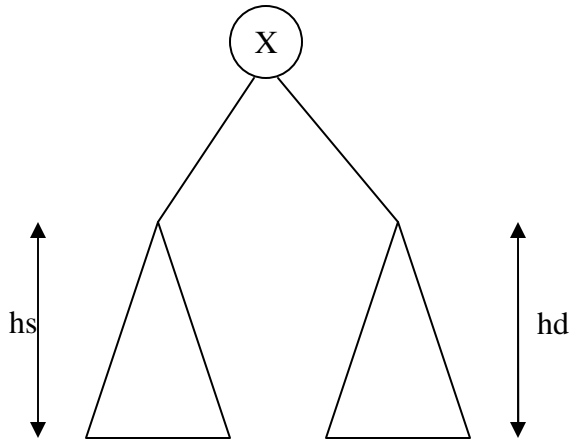

Figura 1.

$| hs - hd | \leq 1$, oricare ar fi nodul X apartinand arborelui, unde hs si hd reprezinta inlatimea subarborelui stang, respective inaltimea subarborelui drept.

De exemplu, fie arborele din firura 2.

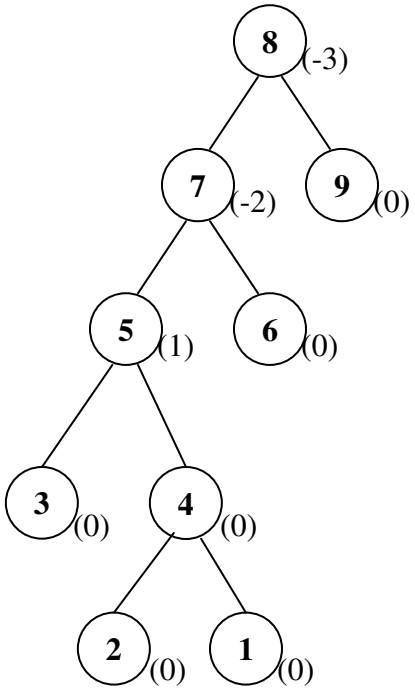

Figura 2.

Exemple de proceduri pentru calculul inaltimii unui subarbore si a factorilor de echilibru

-inaltimea intregului arbore este 5, adica lungimea celui mai lung drum de la nodul radacina la unul din nodurile terminale.
-inaltimea subarborelui stang al radacinii este 4, adica lungimea celui mai lung drum de la nodul cu eticheta 7 la unul din nodurile terminale, cu etchetele 2 respectiv 1.
-pentru a afla factorul de echilibru al radacinii scadem inaltimea subarborelui stang din inaltimea subarborelui drept, adica 1-4=-3.
-factorul de echilibru al nodului cu eticheta 6 este foarte simplu de determinat. Observam ca nodul nu are niciun copil si atunci factorul de echilibru este 0.
-pentru a afla factorul de echilibru al nodului cu eticheta 5 scadem inaltimea subarborelui stang din inaltimea subarborelui drept, adica 2-1=1.

Vom prezenta in continuare functia drum, care calculeaza cel mai lung drum de la un nod current, adica inaltimea unui subarbore:

```
void drum_maxim(nod* p,int &max,int lung){
   if (p!=NULL){
      drum(p->right,max,lung+1);
      if ((p->left==NULL)&&(p->right==NULL)&&(max<lung))
         max=lung;
      drum(p->left,max,lung+1);
   }
}
```

Folosindu-ne de aceasta functie putem determina indicatorul de echilibru al fiecarui nod al arborelui cu functia fact_ech:

```
void calcul_factor_echilibru(nod *p){
    int hLeft,hRight;
    h_left=1; h_right=1;
    if(p->left!=NULL)
      drum_maxim(p->left,h_left,1);
    else
       h_left=0;
    if(p->right!=NULL)
      drum_maxim(p->right,h_right,1);
    else
       h_right=0;
    p->ech=h_right-h_left;
}
```

Practic, arborii AVL se comporta la fel ca arborii binari ordonati simpli, mai putin in cazul operatiilor de insertie si suprimare de chei

O insertie intr-un arbore binar ordonat poate duce la dezechilibrarea anumitor noduri, dezechilibrare manifestata prin nerespectarea formulei | hs – hd | ≤ 1 pentru respectivele noduri.
In principiu, o cheie se insereaza intr-o prima faza, ca si intr-un arbore binar ordonat obisnuit, adica se porneste de la radacina si se urmeaza fiul stang sau fiul drept, in functie de relatia dintre cheia de inserat si cheia nodurilor prin care se trece, pana se ajunge la un fiu nul, unde se realizeaza insertia propriu-zisa
In acest moment se parcurge drumul invers (care este unic) si se cauta pe acest drum ***primul nod care nu este echilibrat***, adica primul nod ai carui subarbori difera ca inaltime prin 2 unitati
Acest nod trebuie echilibrat si el se va afla intotdeauna intr-unul din cele 4 cazuri prezentate in continuare.
Cazuri de echilibrare

Cazul 1-rotatie simpla dreapta

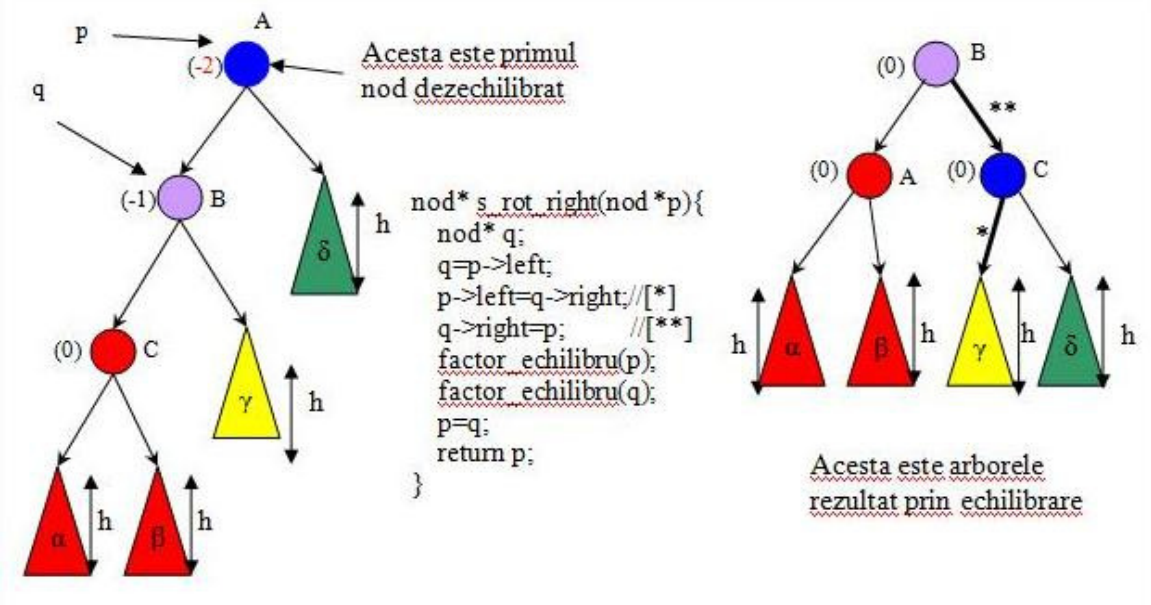

Figura 3.

Cazul 1-rotatie simpla stanga – este simetric in oglinda fata de cazul 1-rotatie simpla dreapta

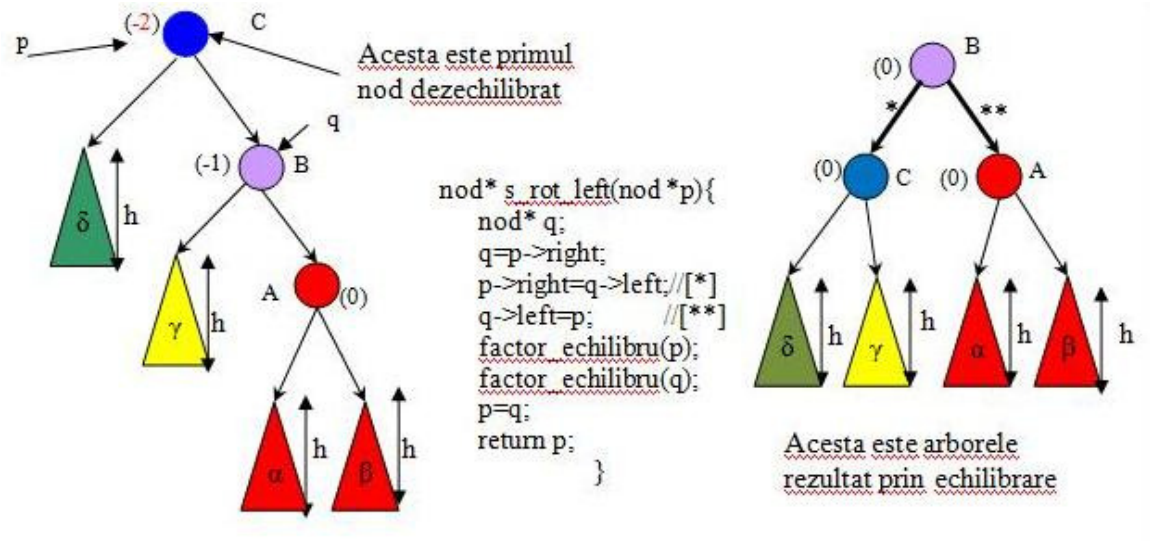

Figura 4.

Cazul 2-rotatie dubla dreapta

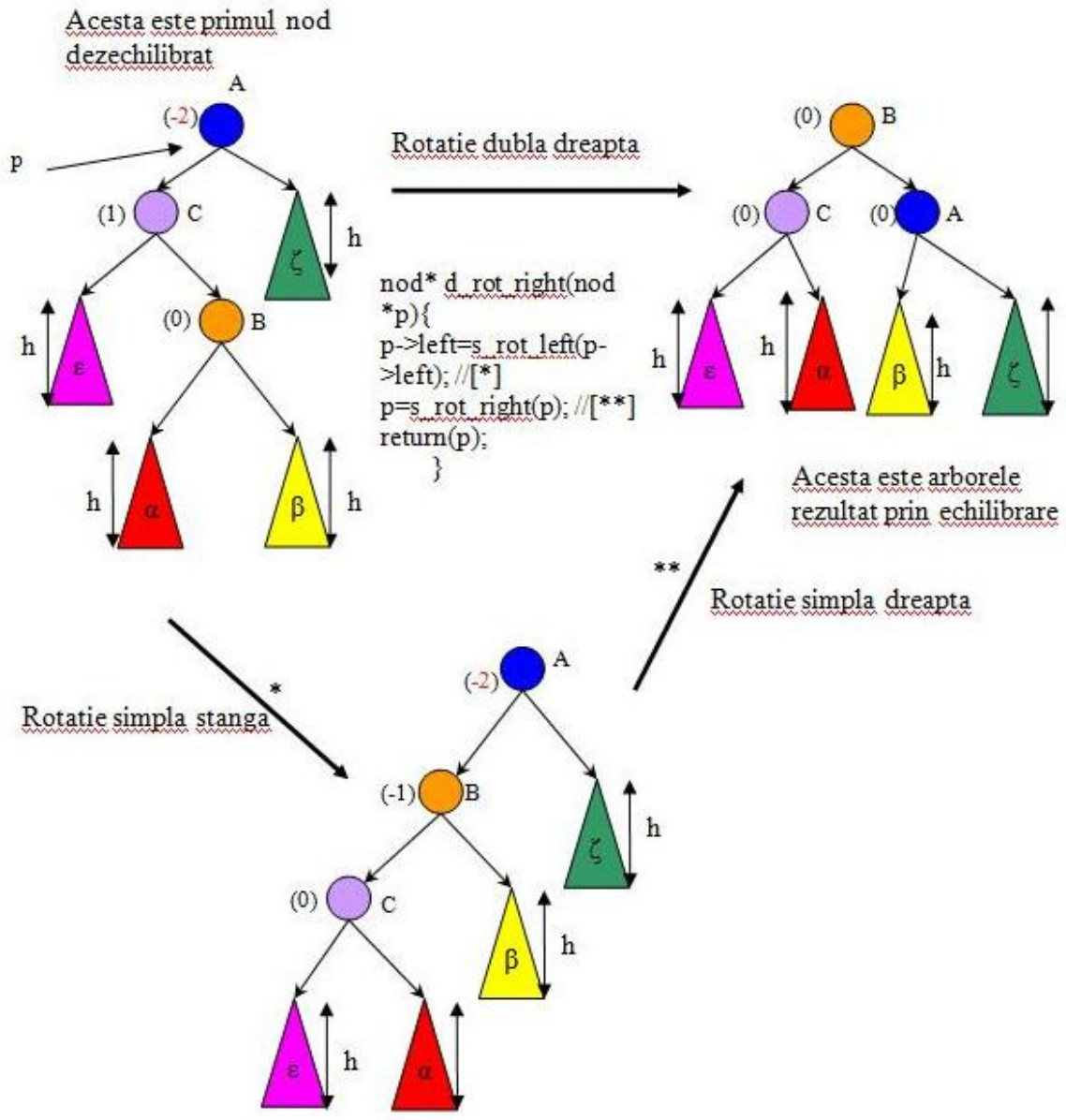

Figura 5.

Cazul 2-rotatie dubla stanga – simetric in oglinda fata de cazul 2-rotatie dubla dreapta

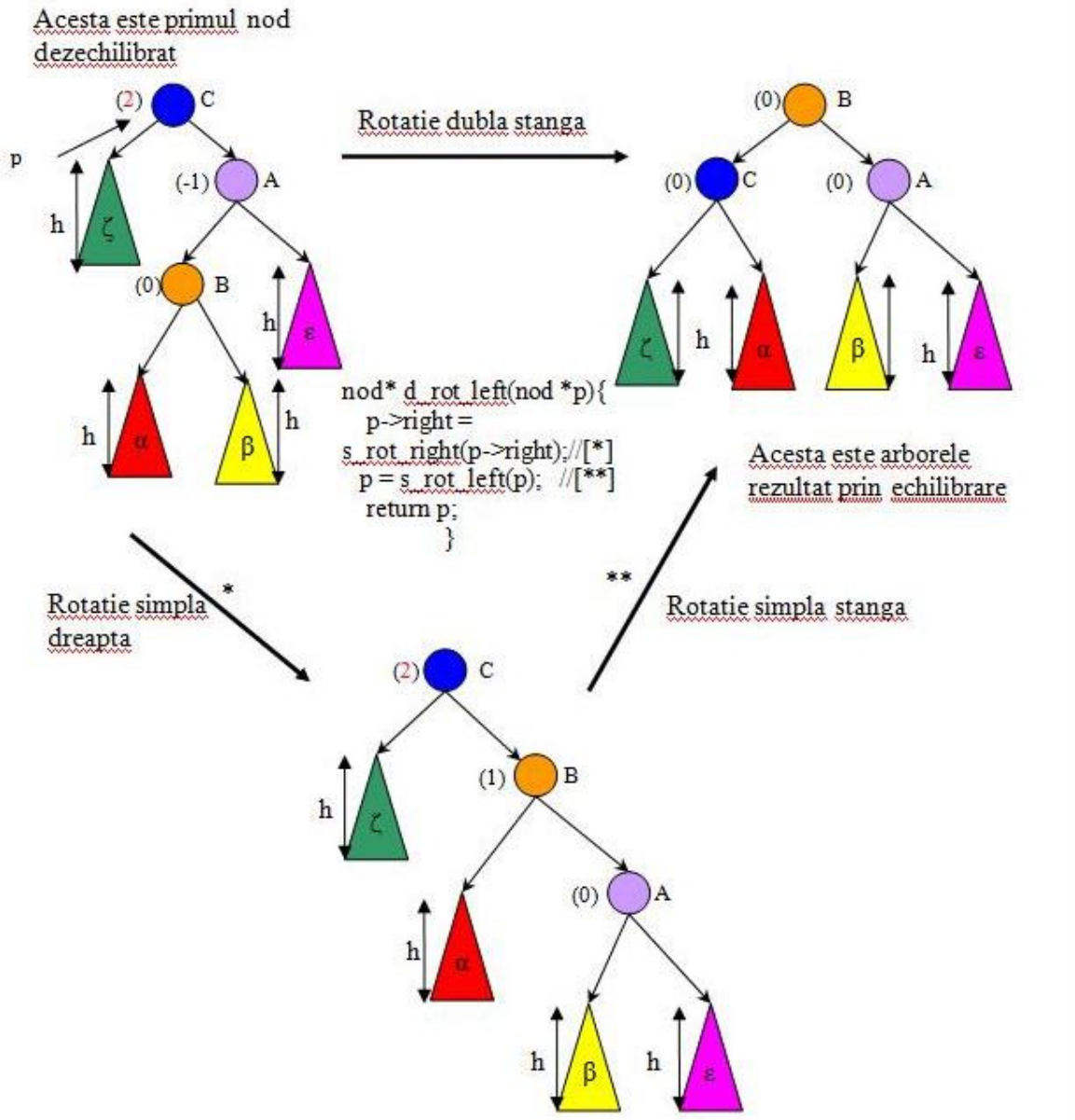

Figura 6.

.

Exemplu: Se va considera secventa de chei 4,5,7,2,1,3,6 care se insereaza intr-un arbore AVL initial vid. Evolutia arborelui si echilibrarile sunt:
-nodurile cu cheile 4,5 se vor insera ca si la arborii binari de cautare
-in mod normal nodul cu cheia 7 ar fi inserat in dreapta nodului cu eticheta 5. In acest caz arborele este dezechilibrat.

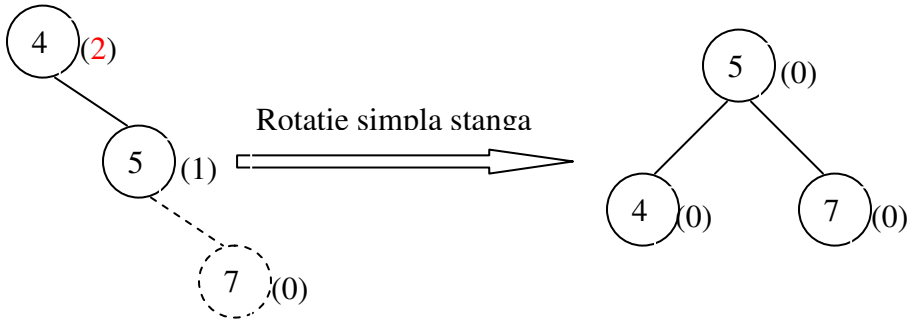

Figura 7.

-nodul cu cheia 2 se va insera in stanga nodului cu cheia 4 fara ca arborele sa se dezechilibreze
-nodul cu cheia 1 se va insera in stanga nodului cu cheia 2, dar in acest caz arborele este dezechilibrat:

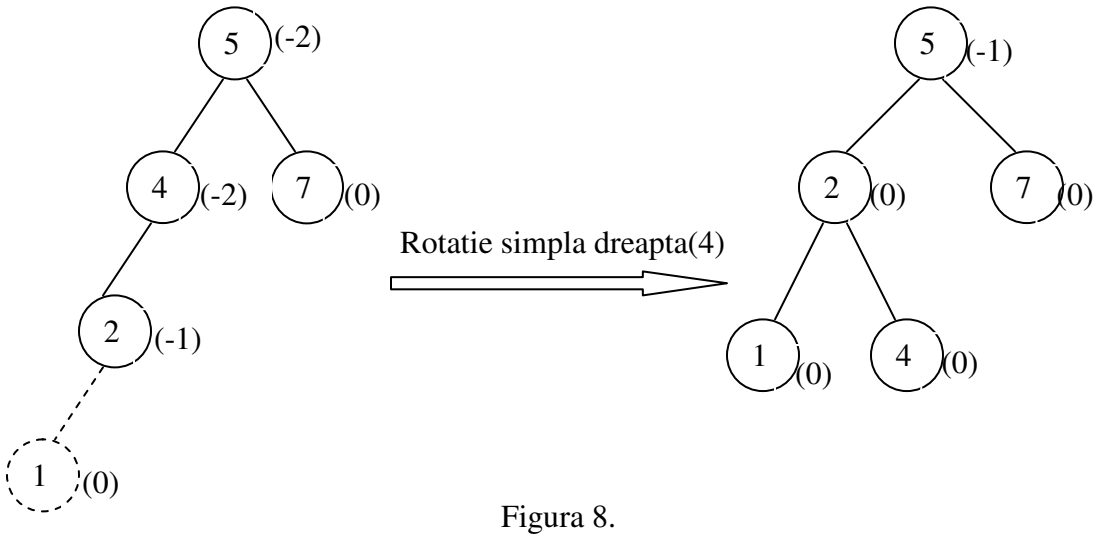

Figura 8.

-dupa inserarea nodului 3 arborele se va dezechilibra din nou:

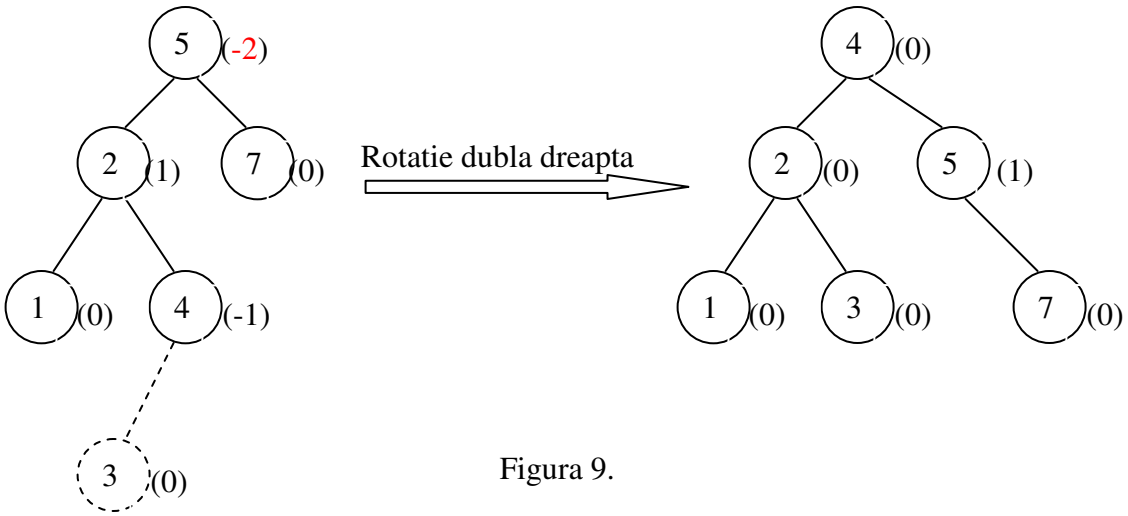

Figura 9.

-nodul cu cheia 6 s-ar insera in stanga nodului cu cheia 7, dar in acest caz arborele ar fi dezechilibrat:

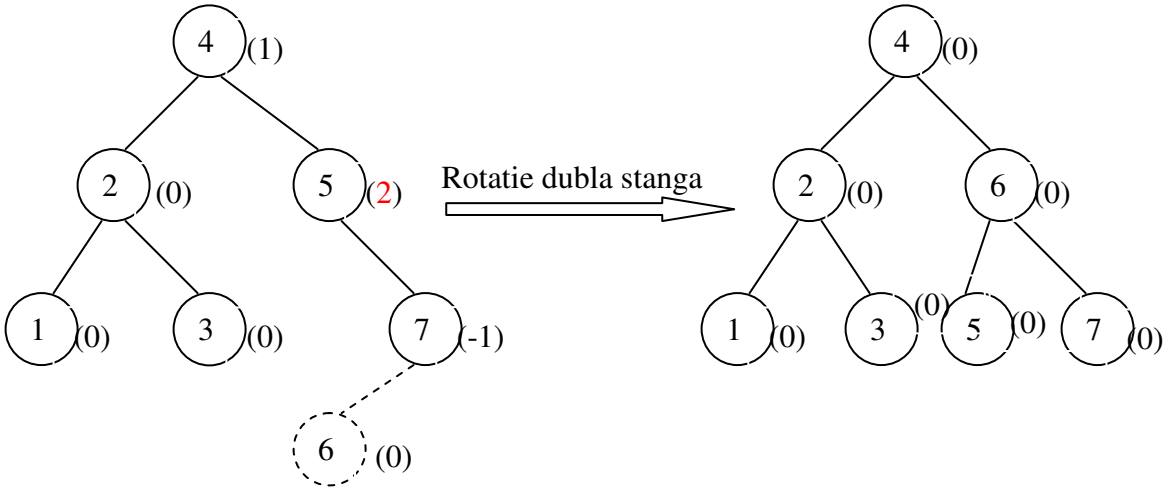

Figura 10.

Functia de inserare este urmatoarea:

```
nod* insereaza(nod *p,int x){
      if (p==NULL){    // daca nodul curent este NULL atunci
      p=new nod; //se aloca spatiu pentru el in zona heap
      p->key=x;  //informatia devine x
      p->ech=0;  // factorul de echilibru este 0 - nodul este echilibrat
```

```
      p->right=NULL;// nodul se insereaza  ca nod frunza
      p->left=NULL; //deci referintele catre copii sai sunt NULL
      return p;
   }
   else {
      if (x<p->key) //daca cheia care se doreste inserata este
                  //mai mica ca valoare decat informatia din nodul curent
         p->left = insereaza(p->left,x);// atunci se insereaza
                  //in subarborele stang al nodului curent
      else
         if (x>p->key) //altfel daca cheia care se va insera
                  //e mai mare decat informatia din nodul curent
            p ->right = insereaza(p->right,x); // atunci se va insera
                  //in subarborele drept
         else
            cout<<"Nodul exista deja"<<endl;
   }
   p = echilibrare(p);// daca intervin cazuri de dezechilibru
               //se va echilibra subarborele sau chiar arborele

}
```

Se observa ca in interiorul functiei de inserare se apeleaza o functie de echilibrare care, in cazul in care intervin cazuri de dezechilibrare, va echilibra subarborele sau chiar arborele. Functia este urmaroarea:

```
nod* echilibrare(nod *p){
   nod *w;
   fact_ech(p);//se calculeaza factorul de echilibru a nodului curent p
   if(p->ech==-2){// daca p nod este critic
     w=p->left; // atunci w este copilul stanga al lui p
     if (w->ech==1)// si daca acesta are factorul de echilibru 1
          p = d_rot_right(p);// atunci se face dubla rotatie dreapta
     else//altfel se face o simpla rotatie dreapta
         p = s_rot_right(p);
   }
   else
   if(p->ech==2){//daca p nod este critic
      w=p->right;//w este copilul dreapta al nodului curent p
      if (w->ech==-1)// si acesta are factorul de ech -1
          p = d_rot_left(p);//se face o dubla rotatie stanga
      else//altfel se face o simpla rotatie stanga
          p = s_rot_left(p);
   }
   return p;
}
```

Inserarea se face recursiv.

Stergerea nodurilor in arborii AVL

Cel mai simplu se pot sterge nodurile terminale si cele care au un singur fiu. Daca nodul care trebuie sa fie sters are doi fii, il inlocuim prin nodul aflat in extrema dreapta a subarborelui sau stang. Echilibrarea este necesara numai daca in urma stergerii s-a redus inaltimea subarborelui.
In cazul cel mai defavorabil stergerea unui nod se realizeaza in O(log n) operatii.

Exemplu: Se considera arborele din figura 11. In urma stergerii nodului cu cheia 6 apare necesitatea echilibrarii. Arborele echilibrat este prezentat in figura:

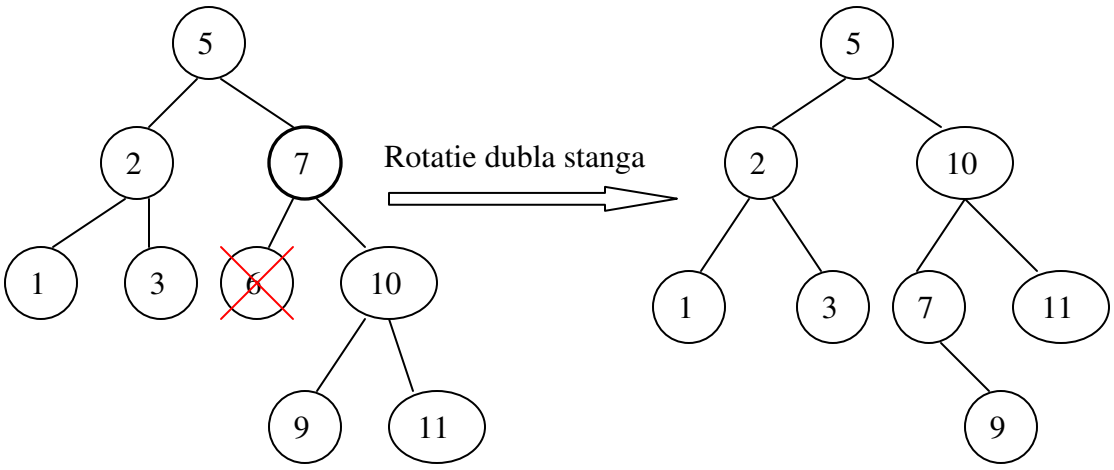

Figura 11.

Nodul cu eticheta 7 se numeste **nod critic**, intrucat dupa stergerea nodului cu etichta 6 factorul lui de echilibru este 2.

Functia de sterge a unei chei din arbore implementata in C++ este urmatoarea:

```
nod* stergere(nod *p,int x){
   nod *q,*r,*w;
   if (p!=NULL)// daca nodul curent este diferit de NULL
      if (x<p->key) //cheia care se doreste stearsa este mai mica decat informatia din nod
         p->left = stergere(p->left,x); // se cauta cheia de sters in subarborele stang al
nodului curent
      else
         if (x>p->key) // daca cheia este mai mare
            p->right = stergere(p->right,x);// se cauta in subarborele drept
         else{
```

```
        //daca cheia este egala cu informatia din nodul curent
        q=p;//un nod q devine p
        if (q->right==NULL) // daca copilul drept al lui q eate NULL
           p=q->left;// atunci p devine q->stanga
        else
           if (q->left==NULL) //altfel daca copilul stang al lui q este NULL
              p=q->right;// p devine q->dreapta
           else{
              w=q->left;//altfel w este copilul stanga al lui q
              r=q;// r devine q
              if (w->right!=NULL)// daca copilul drept al lui w nun este NULL
              {
                while (w->right!=NULL){
                   r=w;
                   w=w->right;
                }
                p->key=w->key;
                q=w;
                r->right=w->left;
                r=p->left;
                w=w->left;
                if (r!=NULL)
                  while ((r!=w)&&(r!=NULL)){
                     r = echilibrare(r);
                     r=r->right;
                  }
              }
              else{
                 p->key=w->key;
                 p->left=w->left;
                 q=w;
              }
           }
           delete(q);// se sterge q
        }
        if (p!=NULL)
           p = echilibrare(p);// se echilibreaza p daca nu este NULL
        return p;
}
```

Arborii AVL reprezinta o alternativa putin costisitoare la arborii binari obisnuiti. Cu pretul unor reechilibrari suplimentare si fara a modifica semnificativ performanta insertiei si suprimarii cheilor (celelalte operatii ramanand nemodificate), proprietatea de echilibru AVL a unui arbore binar ordonat duce la cautari mult mai rapide decat in cazul unui arbore binar ordonat obisnuit, datorita inaltimii mai mici

S-a demonstrat ca un arbore echilibrat AVL va avea intotdeauna inaltimea cuprinsa intre **[log2N+1]** si **[1,43·log2N+1]**, unde N reprezinta numarul de chei din arbore si [x] este partea intreaga a lui x
Spre comparatie, un arbore binar ordonat perfect echilibrat va avea intotdeauna inaltimea egala cu **[log2N+1]** dar a-l mentine perfect echilibrat este mult mai costisitor (ca timp) decat in cazul arborilor AVL
De asemenea, un arbore binar ordonat obisnuit va avea inaltimea cuprinsa intre **[log2N+1]** si **N**, deci poate ajunge la inaltimi mult mai mari decat un arbore AVL cu aceleasi chei