Arbori binari de căutare echilibrați AVL

2015

Procedeul clasic de construire a arborelui binar de căutare ne dă un arbore a cărui formă depinde foarte mult de ordinea în care sunt furnizate valorile nodurilor. În cazul cel mai general nu obținem un arbore de înălțime minimă.

Procedeul clasic de construire a arborelui binar de căutare ne dă un arbore a cărui formă depinde foarte mult de ordinea în care sunt furnizate valorile nodurilor. În cazul cel mai general nu obținem un arbore de înălțime minimă.

Cazul cel mai favorabil

în care obținem înălțime minimă, este cel în care ni se furnizează pe rând mijloacele intervalelor (subintervalelor) vectorului sortat.

Procedeul clasic de construire a arborelui binar de căutare ne dă un arbore a cărui formă depinde foarte mult de ordinea în care sunt furnizate valorile nodurilor. În cazul cel mai general nu obținem un arbore de înălțime minimă.

Cazul cel mai favorabil

în care obținem înălțime minimă, este cel în care ni se furnizează pe rând mijloacele intervalelor (subintervalelor) vectorului sortat.

Cazul cel mai nefavorabil

este cel în care valorile vin în ordine crescătoare (sau descrescătoare), caz în care arborele binar de căutare obținut este degenerat (e chiar o listă înlănțuită, cu legăturile date de fii drepți, cei stângi fiind toȚi nil (crescător)).

Procedeul clasic de construire a arborelui binar de căutare ne dă un arbore a cărui formă depinde foarte mult de ordinea în care sunt furnizate valorile nodurilor. În cazul cel mai general nu obținem un arbore de înălțime minimă.

Cazul cel mai favorabil

în care obținem înălțime minimă, este cel în care ni se furnizează pe rând mijloacele intervalelor (subintervalelor) vectorului sortat.

Cazul cel mai nefavorabil

este cel în care valorile vin în ordine crescătoare (sau descrescătoare), caz în care arborele binar de căutare obținut este degenerat (e chiar o listă înlănțuită, cu legăturile date de fii drepți, cei stângi fiind toŢi nil (crescător)).

Problemă:

Cum **modificăm algoritmul de construcție astfel încât să obținem înălțime minimă** pentru arbore, pentru a îmbunătăți timpul de căutare?
Să **observăm** că **la inserarea unui nou element crește cu 1 înălțimea subarborelui în care s-a făcut inserția**. Ne propunem, pentru noua metodă de construcție, următorul criteriu: diferența dintre înălțimile fiului stâng și cel drept să nu depășească pe 1.

Arbori binari de căutare echilibrați AVL - def

Se numește *arbore binar de căutare echilibrat AVL* (Adelson-Velskii-Landis) un arbore care în fiecare nod are proprietatea că înălțimile subarborilor stâng și drept diferă cu cel mult 1.

Arbori binari de căutare echilibrați AVL - def

Se numește *arbore binar de căutare echilibrat AVL* (Adelson-Velskii-Landis) un arbore care în fiecare nod are proprietatea că înălțimile subarborilor stâng și drept diferă cu cel mult 1.

Pentru un nod dat, fie h_l și h_r înălțimile subarborelui stâng, respectiv drept. Avem trei situații posibile în acest nod, codificate cu valorile variabilei $bal = h_r h_l$, pe care o numim *factor de echilibru*, după cum urmează:

Arbori binari de căutare echilibrați AVL - def

Se numește *arbore binar de căutare echilibrat AVL* (Adelson-Velskii-Landis) un arbore care în fiecare nod are proprietatea că înălțimile subarborilor stâng și drept diferă cu cel mult 1.
Pentru un nod dat, fie h_l și h_r înălțimile subarborelui stâng, respectiv drept. Avem trei situații posibile în acest nod, codificate cu valorile variabilei $bal = h_r h_l$, pe care o numim *factor de echilibru*, după cum urmează:

$$bal = \begin{array}{rl} 1, & h_l = h_r - 1 \\ 0, & h_l = h_r \\ -1, & h_l = h_r + 1 \end{array}$$

Informația despre valoarea factorului de echilibru în fiecare nod p al unui arbore o vom scrie într-un nou câmp al lui p, câmpul $bal : -1..1$.

Arbori binari de căutare echilibrați AVL - def

Se numește *arbore binar de căutare echilibrat AVL* (Adelson-Velskii-Landis) un arbore care în fiecare nod are proprietatea că înălțimile subarborilor stâng și drept diferă cu cel mult 1.
Pentru un nod dat, fie h_l și h_r înălțimile subarborelui stâng, respectiv drept. Avem trei situații posibile în acest nod, codificate cu valorile variabilei $bal = h_r h_l$, pe care o numim *factor de echilibru*, după cum urmează:

$$bal = \begin{array}{rl} 1, & h_l = h_r - 1 \\ 0, & h_l = h_r \\ -1, & h_l = h_r + 1 \end{array}$$

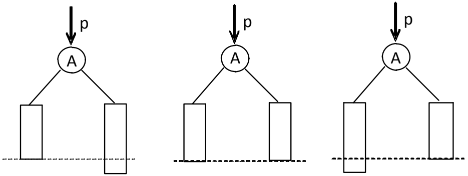
Informația despre valoarea factorului de echilibru în fiecare nod p al unui arbore o vom scrie într-un nou câmp al lui p, câmpul $bal : -1..1$.

Algoritm de inserare

cu operații suplimentare, *re-echilibrări*

Pentru nodurile unui arbore AVL vom folosi următoarele definiții de tip:

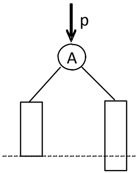
```
struct arbore {
  int info;
  arbore *left, *right;
  int bal;
}
```



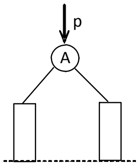
(a)
p↑.bal=1

(b)
p↑.bal=0

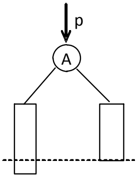
(c) p↑.bal=
-1



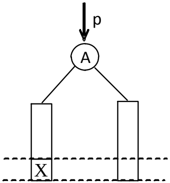
(a)
p↑.bal=1



(b)
p↑.bal=0

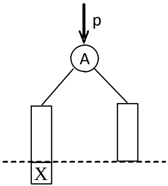


(c) p↑.bal=
-1



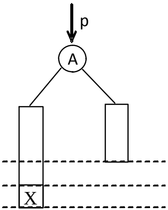
noul p↑.bal=0

(a')



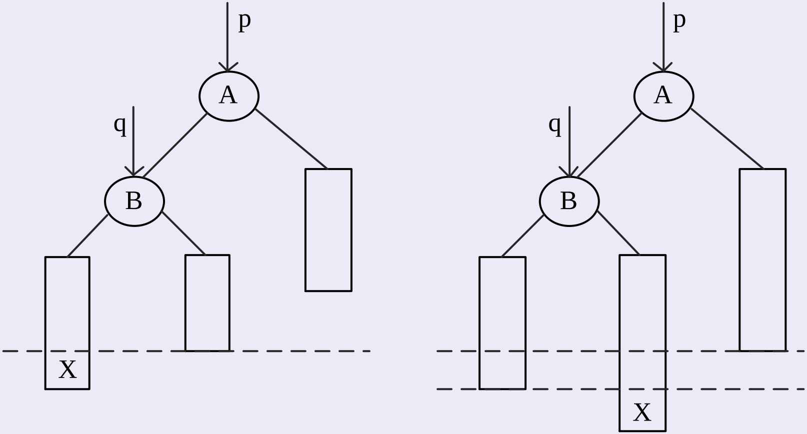
noul p↑.bal=1

(b')



(c')

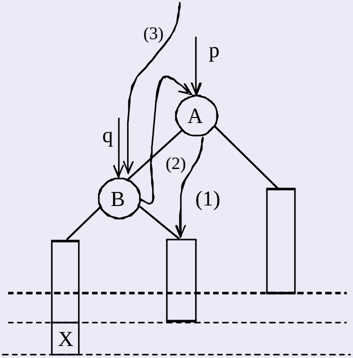
În detaliu cazul (c'): ori (A) ori (A')



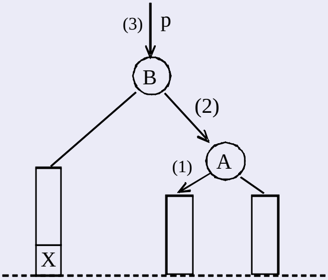
(A) (A')

Cazul (A)

Putem reechilibra arborele resetând legăturile (1),(2),(3):



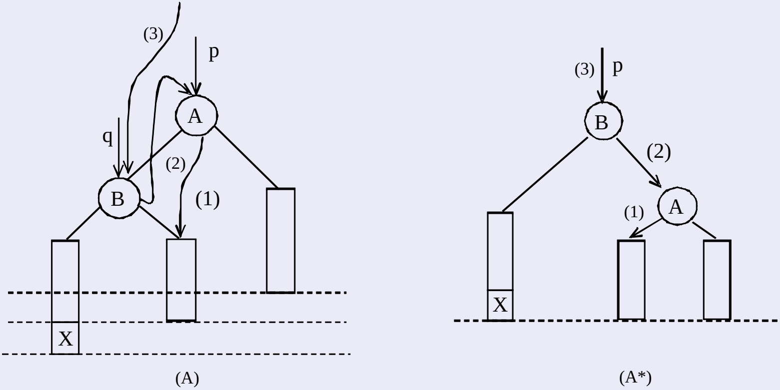
(A)



(A*)

Cazul (A)

Putem reechilibra arborele resetând legăturile (1),(2),(3):



Trecerea de la (A) la (A*) se numește *rotație SS* (Stânga-Stânga):

Rotație SS:

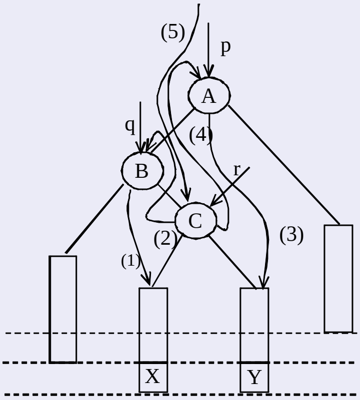
(1) `p->left = q->right;`

(2) `q->right = p;` // înainte de reasignarea lui p calculăm noul factor de echilibru în A
`p->bal = 0;`

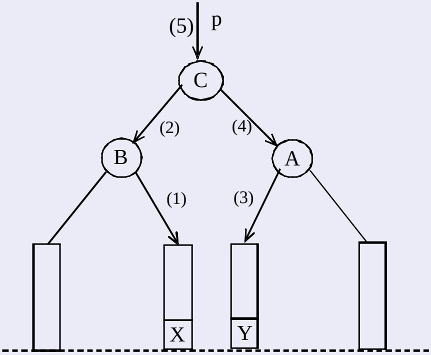
(3) `p = q;` // urmată de calcularea factorului de echilibru pentru noul arbore:
`p->bal = 0;`

Nu putem folosi același procedeu ca la (A), deoarece am obține un arbore neechilibrat. Desfășurând subarborele drept al lui `q`, obținem situația din figura (B), unde noul nod inserat este fie `Y`, fie `X`, iar cazul (A') devine cazul (B): reechilibrăm resetând legăturile (1), (2), (3), (4), (5).

Nu putem folosi același procedeu ca la (A), deoarece am obține un arbore neechilibrat. Desfășurând subarborele drept al lui q, obținem situația din figura (B), unde noul nod inserat este fie `Y`, fie `X`, iar cazul (A') devine cazul (B): reechilibrăm resetând legăturile (1), (2), (3), (4), (5).



(B)



(B*)

(1) `q->right = r->left;`

(2) `r->left = q;`

(3) `p->left = r->right;`

(4) `r->right = p;`

Înainte de reasignarea lui `p` către noua rădăcină trebuie să calculăm noii factori de echilibru în nodurile `A` (`p`) și `B` (`q`) în funcție de ce anume s-a inserat: nodul `X` sau nodul `Y`, cu secvența:

Înainte de reasignarea lui p către noua rădăcină trebuie să calculăm noii factori de echilibru în nodurile A (p) și B (q) în funcție de ce anume s-a inserat: nodul X sau nodul Y, cu secvența:

```
if (r->bal == -1)                    // s-a inserat X
{
  q->bal = 0;
  p->bal = 1;
}
else                                 // r->bal == 1, s-a inserat Y
{
  q->bal = -1;
  p->bal = 0;
}
```

Înainte de reasignarea lui `p` către noua rădăcină trebuie să calculăm noii factori de echilibru în nodurile `A` (`p`) și `B` (`q`) în funcție de ce anume s-a inserat: nodul `X` sau nodul `Y`, cu secvența:

```
if (r->bal == -1)                          // s-a inserat X
{
  q->bal = 0;
  p->bal = 1;
}
else                                       // r->bal == 1, s-a inserat Y
{
  q->bal = -1;
  p->bal = 0;
}
```

(5) `p = r;` // reasignarea pointerului către rădăcină

Înainte de reasignarea lui `p` către noua rădăcină trebuie să calculăm noii factori de echilibru în nodurile `A` (`p`) și `B` (`q`) în funcție de ce anume s-a inserat: nodul `X` sau nodul `Y`, cu secvența:

```
if (r->bal == -1)                          // s-a inserat X
{
  q->bal = 0;
  p->bal = 1;
}
else                                       // r->bal == 1, s-a inserat Y
{
  q->bal = -1;
  p->bal = 0;
}
```

(5) `p = r;` // reasignarea pointerului către rădăcină

Urmează apoi calculul factorului de echilibru pentru arborele din figura (B*) reechilibrat

```
p->bal = 0;
```

Înainte de reasignarea lui p către noua rădăcină trebuie să calculăm noii factori de echilibru în nodurile A (p) și B (q) în funcție de ce anume s-a inserat: nodul X sau nodul Y, cu secvența:

```
if (r->bal == -1)                    // s-a inserat X
{
  q->bal = 0;
  p->bal = 1;
}
else                                 // r->bal == 1, s-a inserat Y
{
  q->bal = -1;
  p->bal = 0;
}
```

(5) `p = r;` // reasignarea pointerului către rădăcină

Urmează apoi calculul factorului de echilibru pentru arborele din figura (B*) reechilibrat

```
p->bal = 0;
```

Trecerea de la arborele de tip (B) la (B*) poartă numele de *rotație SD* (Stânga-Dreapta).

Cele spuse mai sus se aplică și pentru cazul în care se inserează un nod nou pe subarborele drept al unui arbore de rădăcină p. Vom avea atunci încă doua cazuri în care trebuie sa facem reechilibrarea, simetricele cazurilor (A) și (B), care se tratează analog. Simetricul cazului (A) va conduce la *rotație DD*, iar al cazului (B) la *rotație DS*.

Procedura recursivă `Search(x, p)` care caută și eventual inserează un nod nou `x` în arborele de rădăcină `p` se va modifica. În lista de parametri mai apare o variabilă booleană, `h`, ce se transmite procedurii apelatoare, cu semnificația:

`h = true`, dacă s-a modificat înălțimea arborelui de rădăcină `p`
`false`, în caz contrar.

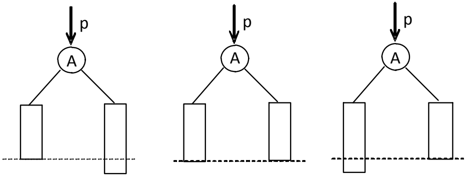
Observăm că în ambele cazuri de reechilibrare, înălțimea arborelui reechilibrat este egală cu înălțimea arborelui dinainte de inserția care a stricat echilibrul, deci dupa reechilibrări h trebuie să ia valoarea `false`. Modificarea procedurii `Search` se face în felul următor: după fiecare apel al ei, se testează `h`, iar daca `h = true`, avem de tratat separat cazurile pentru cele trei valori ale lui `p->bal`.

*void SearchIns (int x, arbore *p)*

{ este o procedură recursivă care caută valoarea x în arborele binar de căutare de rădăcină p și o inserează dacă nu o găsește, iar dacă o găsește incrementează câmpul contor al nodului respectiv.

```
if (p == NULL)                              // x nu a fost găsit și va fi inserat
   {
   p = new nod;
   p->info = x;
   p->contor = 1;
   p->left = NULL;
   p->right = NULL;
   }
else                                        // p != NULL
   if (x < p->info)
      SearchIns--Recursiv(x, p->left);
      else
         if (x > p->info)
            SearchIns--Recursiv(x, p->right);
         else                 // x a fost găsit și se incrementează contorul
            p->contor = p->contor + 1;
```

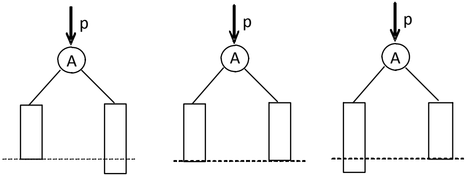
}



(a)
p↑.bal=1

(b)
p↑.bal=0

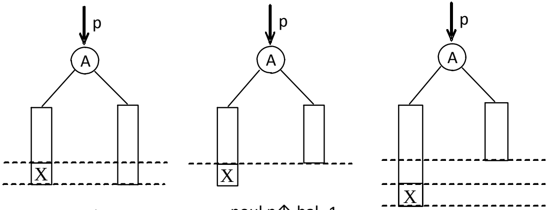
(c) p↑.bal=
-1



(a) p↑.bal=1

(b) p↑.bal=0

(c) p↑.bal= -1



noul p↑.bal=0

(a')

noul p↑.bal=1

(b')

(c')

void Search (int x, arbore *p, bool h)

```
{
  if (p == NULL)                              // inserare nod
    {
    p = new nod;
    p->info = x;
    p->bal = 0;                                 // apare acum
    p->left = NULL;
    p->right = NULL;
    h = true;
    }
  else
    if (x < p->info)
      Search(x, p->left,h);
// de aici începe modificarea față de SearchIns
```

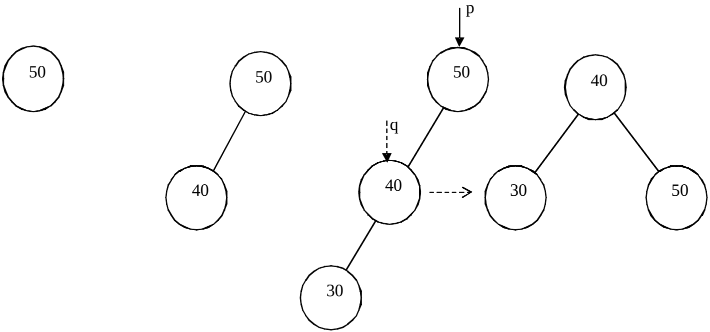
```
if (h == true)                    // a crescut înălțimea ramurii stângi
{
  if (p->bal == 1)      // cele două ramuri ale lui p sunt egale
  {
    p->bal = 0;
    h = false;
  }
  if (p->bal == 0)             // ramura din stânga e mai lungă
    p->bal = -1;                                // h rămâne true
  if (p->bal == -1)                                // reechilibrăm
```

```
// reechilibrăm
        q = p->left;
        if (q->bal == -1)                    // cazul (A) rotație SS
          {
          p->left = q->right;                // (1)
          q->right = p;                      // (2)
          p->bal = 0;
          p = q;                             // (3)
          }
        else                                 // cazul (B) rotație SD
          {
          r = q->right;
          q->right = r->left;                // (1)
          r->left = q;                       // (2)
          p->left = r->right;                // (3)
          r->right = p;                      // (4)
```

```
        if (r->bal == -1)                  // s-a inserat X pe r->left
          {
          q->bal = 0;
          p->bal = 1;
          }
        else                               // s-a inserat Y pe r->right
          {
          q->bal = -1;
          p->bal = 0;
          }
        p = r;                                                  // (5)
        }                                  // linia 26 – caz (B) rotație SD
      p->bal = 0;
      h = false;
    }                                      // linia – if (p->bal == -1)
  }
// Am încheiat inserarea cu reechilibrare pe ramura stângă.
```

```
    else                                              // x >= p->info
      if (x > p->info)
        Search(x, p->right,h);
        if (h == true)               // a crescut înălțimea ramurii drepte
          if (p->bal == -1)           // cele două ramuri ale lui p - egale
            {
            p->bal = 0;
            h = false;
            }
          if (p->bal == 0)                 // ramura dreaptă e mai lungă
            p->bal = -1;                              // h rămâne true
          if (p->bal == 1)                              // reechilibrăm
  ...
}
```

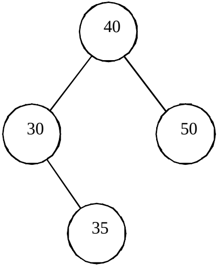
Exemplu de inserări și reechilibrări



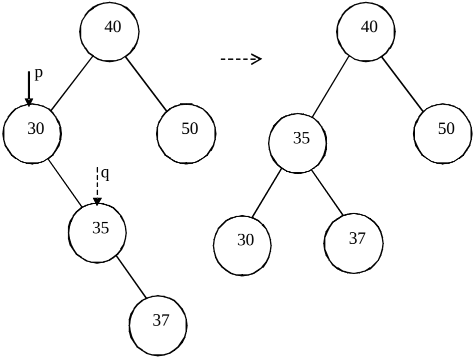
Se inserează 50

Se inserează 40

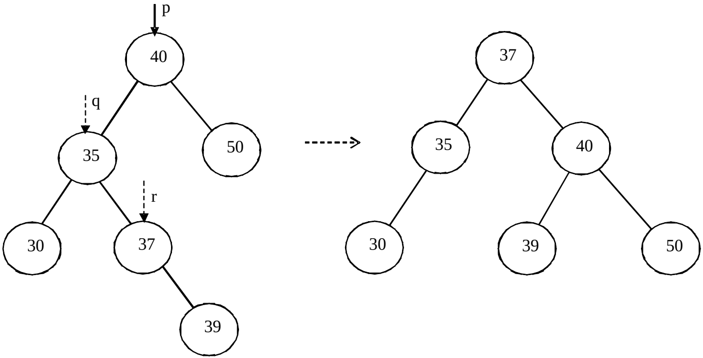
Se inserează 30. E nevoie de reechilibrare în 50.
Rotaţie SS.



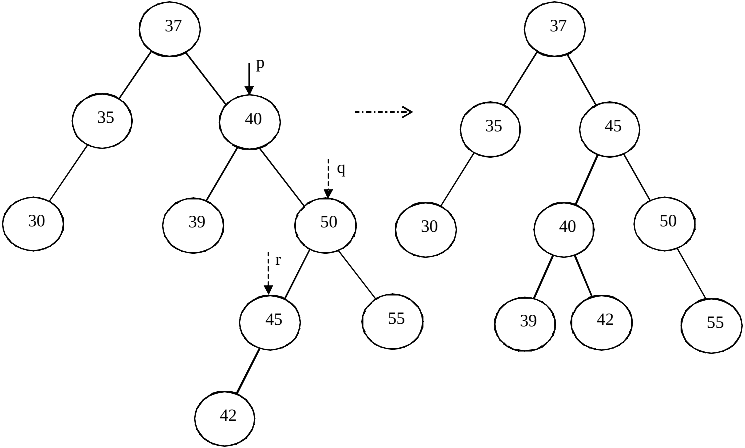
Se inserează 35



Se inserează 37. E nevoie de re echilibrare în 30. Rotaţie DD.



Se inserează 39. E nevoie de re echilibrare în 40. Rotaţie SD.



Se inserează 45 şi 55. Apoi se inserează 42. E nevoie de reechilibrare în 40. Rotaţie DS.