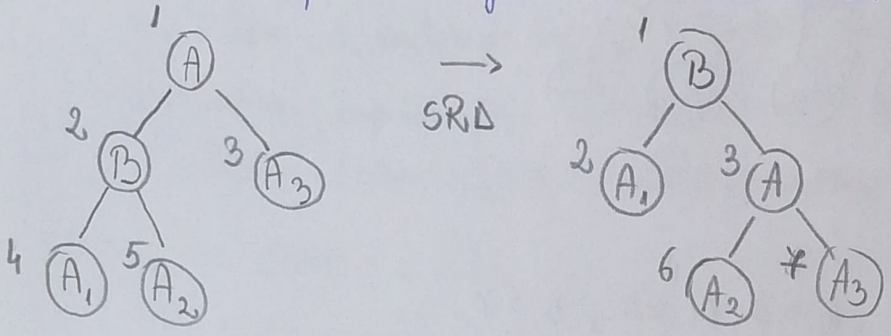
SUBIECT Δ

ROTAȚII

SRD: recursiv pe vector folosind ca schemă de memorare ansamblu

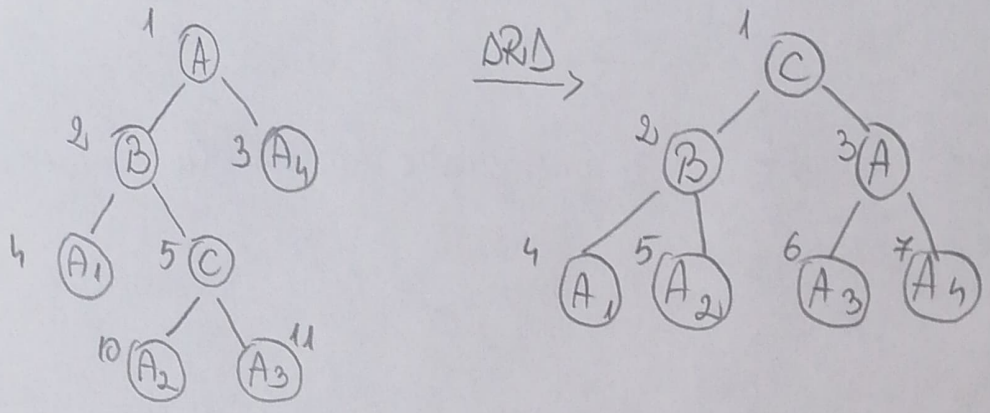


trebuie să refac descendenții

```
-subalg sch_arbore(i/o v, i vechi, i nouă, i a)   {alg. ajutător}
    v[nouă] = a.v[vechi]
    creează(c)
    adaugă(c, {vechi, nouă})
   -cât timp ¬vidă(c) execută
        șterge(c, {i, j})   {i -> vechea poz, j -> noua poz a nodului curent}
       -dacă 2·i ≤ a.cp ∧ a.v[2i] != NIL   {dacă este stânga}
            v[2·j] = a.v[2i]
            adaugă(c, {2i, 2j})
       -sf dacă
       -dacă 2i+1 ≤ a.cp ∧ a.v[2i+1] != NIL   {dacă este dreapta}
            v[2j+1] = a.v[2i+1]
            adaugă(c, {2i+1, 2j+1})
       -sf dacă
   -sf cât timp
-sf subalg

-subalgoritm SRD(i/o a)
  {pre și post}
  -pentru i ← 1, n execută
       v[i] = NIL
   -sf pentru
  v[1] = a.v[2]
  v[3] = a.v[1]
  {am sch. poz nodurilor, refacem descendenții}
   sch_arbore(v, 4, 2, a)
   sch_arbore(v, 5, 6, a)
   sch_arbore(v, 3, 7, a)
  -pentru i ← 1, n execută
       a.v[i] = v[i]
   -sf pentru
-sf subalgoritm
```

DRD: același reprezentare (refolosesc alg. ajutător)



1→3, 2→2, 5→1, 4→4, 10→5, 11→6, 3→7

- subalg DRD ($\overset{i/o}{a}$)
  {pre, post}
  - pentru i ← 1, n execută
    v[i] ← NIL
  - sf pentru
  v[3] ← a.v[1]
  v[1] ← a.v[5]
  v[2] ← a.v[2]
  sch_arbore(v, 4, 4, a)
  sch_arbore(v, 10, 5, a)
  sch_arbore(v, 11, 6, a)
  sch_arbore(v, 3, 7, a)
  - pentru i ← $\overline{1,n}$ execută
    a.v[i] ← v[i]
  - sf pentru
- sf subalg

Ștergere dintr-o CP repr. folosind un ansamblu cvaternar

containerul:

n: întreg, nr. elems

v: TElem[], vector de biți, indexat de la 1

max: întreg, cap maximă de memorare

r: TElem × TElem → {A, F}, relația de ordine

subalgoritm șterge (a, e)

pre: a → ansamblu

e → elem cel mai prioritar, șters

post: @ aruncă excepție dacă ansamblul e vid

- dacă a.n = 0 atunci

  @ ansamblu vid

- sf dacă

e ← a.v[1]; aux ← a.v[a.n]

a.v[1] ← a.v[a.n] {punem în rad. elem. de pe ultima poz}

a.n ← a.n - 1

{coborâm rădăcina, verificăm care din fii ei e cel mai prioritar}

curent ← 1; ok ← true

- cât timp 4*curent ≤ a.n și ok = true exec {fiul cel mai dim st}

  i ← 4*curent; j ← i

  - dacă i+1 ≤ a.n și a.r(a.v[i+1], a.v[j]) at {fiul urm}

    j ← i+1

  - sf dacă

  - dacă i+2 ≤ a.n și a.r(a.v[i+2], a.v[j]) at {fiul urm}

    j ← i+2

  - sf dacă

  - dacă i+3 ≤ a.n și a.r(a.v[i+3], a.v[j]) at

    j ← i+3

  - sf dacă

  - dacă nu a.r(aux, a.v[j]) exec.

    ok ← true

    a.v[curent] ← a.v[j] {mută în rad fiul cel mai prioritar}

    curent ← j

  altfel ok ← fals {nu mai are fii mai prioritari decât el}

- sf dacă

- sf cât timp
  a.v[curent] ← aux {punem pe poz bună modul a se mută}
- sf subalg

găsește un nod e prin parcurgere în inordine

container: rād: ↑nod, rădăcină arbore

nod: e: TElem, inf. utilă
  st: ↑nod, nodul stâng
  dr: ↑nod, nodul drept

- subalg cnt (a, e, nr)
  {pre: a → arbore binar
    e → TElem, elem. de căutat
  post: nr → int, indicele în inordine
    @ aruncă exp. dacă nr ∉ }
  creează (stv)
  crt ← a.rād
  nr ← 0; ok ← fals
  - cât timp ¬ vidă (stv) ∨ crt ≠ NIL exec
    - cât timp crt ≠ NIL exec
      adaugă (stv, crt)
      crt ← [crt].st
    - sf cât timp
    șterge (stv, crt)
    - dacă [crt].c = e atunci
      nr ← cnt
      ok ← adevărat
      @ STOP
    - sf dacă
    crt ← [crt].dr {am parcurs toată st}
    cnt ← cnt + 1
  - sf cât timp
  - dacă ok = fals atunci
    @ nu există
  - sf dacă
- sf subalgoritm

det dacă e și e' sunt pe același nivel : înlănțuit cu alocare dinamică

container: rād: înod, rădăcina
nod : st : înod, vecin stâng
dr: înod, vecin drept
e : TElem, inf utilă
h : întreg, înălțimea

- subalg acc_niv(a, e, e')
  {pre, post}
  creează (c)
  adaugă (c, a.rād); $h_1 \leftarrow -1$; $h_2 \leftarrow -1$
  - cât timp ¬vidă (c) execută
    șterge (c, ort)
    - dacă [ort].e = e at.
      $h_1 \leftarrow$ [ort].h
    - sf dacă
    - dacă [ort].e = e' at
      $h_2 \leftarrow$ [ort].h
    - sf dacă
    - dacă [ort].st != NIL at
      [[ort].st].h = [ort].h + 1
      adaugă (c, [ort].st)
    - sf dacă
    - dacă [ort].dr != NIL at
      [[ort].dr].h = [ort].h + 1
      adaugă (c, [ort].dr)
    - sf dacă
  - sf cât timp
  - dacă $h_1 = h_2 \wedge h_1 \neq -1$ at.
    acc_niv ← adevărat
  altfel
    acc_niv ← fals
  - sf dacă
- sf subalgoritm

nivelul elementului e: secvențial pe vector, folosind ca schemă de mem. ansamblu

```
container: v: TElem[]  indexat de la 1
           m: intreg
           cp: intreg
- subalg niv (a, e, n)
    {pre, post}
    m ← -1
    creează (c)                    {niv rădăcină}
    adaugă (c, {1, 0})
    - cât timp ¬vidă (c) exec
        șterge (c, {i, nivel})
        - dacă a.v[i] = e at
            m ← nivel
            @ STOP
        - sf dacă
        - dacă 2·i ≤ a.cp ∧ a.v[2·i] != NIL at   {dacă ∃ fiu stâng}
            adaugă (c, {2i, nivel+1})
        - sf dacă
        - dacă 2i+1 ≤ a.cp ∧ a.v[2i+1] != NIL at   {∃ fiu drept}
            adaugă (c, {2i+1, nivel+1})
        - sf dacă
    - sf cât timp
    - dacă m = -1 at
        @ nu există elementul
    - sf dacă
- sf subalgoritm
```

CODARE

container: v: TElem[] indexat de la 1
m: intreg
cp: intreg

{facem o parcurgere în lățime}

- subalg get_cod(a, e, cod_e)
  ok ← fals
  creează(c)
  adaugă(c, {1, "1"}) ← aici se formează codul
  - cât timp ¬vidă(c) exec
    șterge(c, {i, cod})
    - dacă a.v[i] = e at.
      cod_e = cod
      @STOP
    - sf dacă
    - dacă 2·i ≤ a.cp ∧ a.v[2·i] != NIL at {dacă ↓ stânga}
      adaugă(c, {2i, cd + "2"})
    - sf dacă
    - dacă 2·i+1 ≤ a.cp ∧ a.v[2i+1] != NIL at {dacă ↓ dreapta}
      adaugă(c, {2i+1, cd + "0"})
    - sf dacă
  - sf cât timp
- sf subalg

predecesor : înlănțuit cu alocare dinamică

container: rād : ↑nod nod: st: ↑nod
n: intreg dr: ↑nod
c: TCheie

{ dacă are fiu stâng, merg odată în st, apoi tot în dreapta → caut maximul }

- subalg predecesor(arb, ch, rez)
{pre, post}
rez ← NULL_TCheie; găsit ← fals
crt ← arb.rād
- cât timp crt ≠ NIL execută
  - dacă găsit = true atunci
    rez ← max(rez, [crt].e)
    crt ← [crt].dr { deja am mers în st, merg în dr }
  - altfel
    - dacă [crt].e < ch at.
      crt ← [crt].st
    - sf dacă
    - dacă [crt].e > ch at. ← dacă găsesc ch căutată în dreapta unui nod actualizez rez (pt. cazul în care ch nu are st)
      - dacă [crt].dr ≠ NIL ∧ [[crt].dr].e = ch atunci
        rez ← [crt].e
      - sf dacă
      crt ← [crt].dr
    - sf dacă
    - dacă [crt].e = ch at.
      găsit ← true
      crt ← [crt].st { prima dată după ce l-am găsit merg în st }
    - sf dacă
  - sf dacă
- sf cât timp
sf subalg

înălțimea unui nod

container: m: intreg
u: TElem → Inf utilă
cp: intreg

--subalg înălțime(a, e)
{pre, post}
index ← -1; ok ← fals
creează(c)
adaugă(c, 1)
- cât timp ¬vidă(c) și ok = fals exec {caut intervalul lui e - parcurgere latime}
  șterge(c, i)
  - dacă 2*i ≤ a.cp și a.v[2i] ≠ NIL at.
    adaugă(c, 2i)
  - sf dacă
  - dacă 2i+1 ≤ a.cp și a.v[2i+1] ≠ NIL at
    adaugă(c, 2i+1)
  - sf dacă
  - dacă a.v[i] = e at
    index ← i
    ok ← true
  - sf dacă
- sf cât timp
- dacă ok = fals at
  @ nu există
- sf dacă
creează(c2)
adaugă(c2, {i, 0}) ← nivelul rădăcinii, aka nodul nostru
m ←
- cât timp ¬vidă(c2) execută {mai facem o parcurgere în latime pt acest subarbore}
  șterge(c2, {i, nivel})
  m ← max{m, nivel}
  - dacă 2i ≤ a.cp și a.v[2i] ≠ NIL at
    adaugă(c2, {2i, nivel+1})
  - sf dacă
  - dacă 2i+1 ≤ a.cp și a.v[2i+1] ≠ NIL at
    adaugă(c2, {2i+1, nivel+1})
  - sf dacă
- sf cât timp
înălțime ← m
sf subalg