A.

Caz favorabile = caz defavorabile = caz mediu $\Rightarrow$ timpii mediu și defavorabile vor fi egali

$$T(n) = \begin{cases} 1, n = 1 \\ 2T(n/2) + \sum_{i=1}^{n-1} \sum_{j=1}^{i+1} 1 \end{cases}$$

$$\sum_{i=1}^{n-1} \sum_{j=1}^{i+1} 1 = \sum_{i=1}^{n-1} (i+1) = 2 + 3 + \dots + n = \frac{(n+2)(n-1)}{2}$$

Notăm $n = 2^k$

$$T(2^k) = 2T(2^{k-1}) + \frac{(2^k+2)(2^k-1)}{2}$$

$$T(2^{k-1}) = 2T(2^{k-2}) + \frac{(2^{k-1}+2)(2^{k-1}-1)}{2} \quad | \cdot 2$$

$$\vdots$$

$$T(1) = 1$$

$$| \cdot 2^k$$

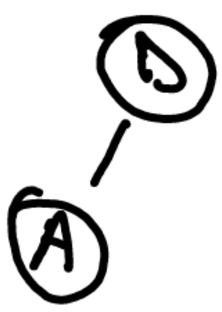
$$T(2^k) \simeq \frac{2^{2k}}{2} + \frac{2^{2k-1}}{2} + \frac{2^{2k-2}}{2} + \dots + 1$$

$$= 2^{2k-1} + 2^{2k-2} + 2^{2k-3} + \dots + 2^k$$

$$= 2^k \cdot \frac{2^k - 1}{2 - 1} = 2^{2k} = 2^{2\log_2 n} = 2^{\log_2 n^2} = n^2$$

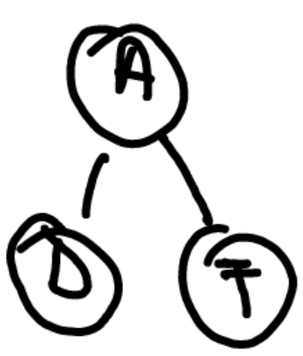
?

B.



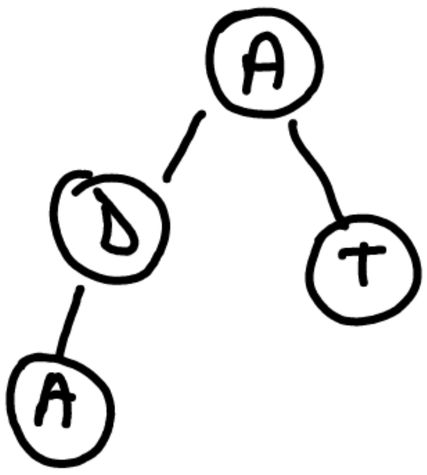
se adaugă A

A < D => înlocuire



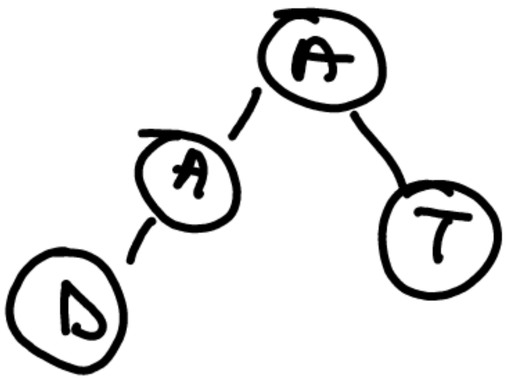
se adaugă T

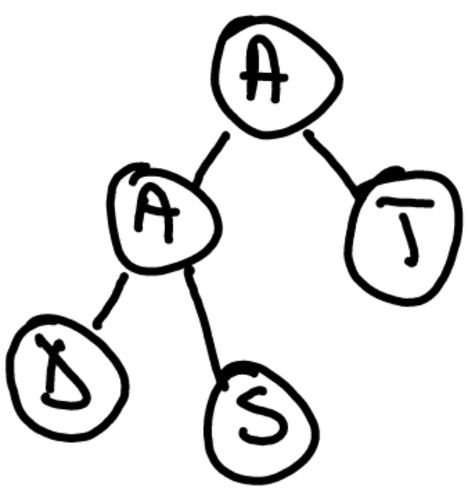
T > A ✓



se adaugă A

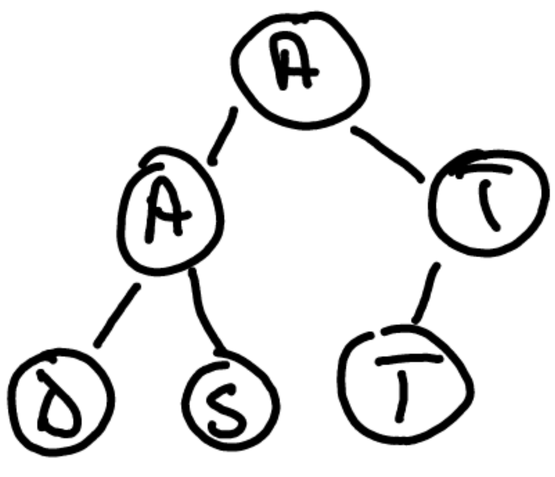
A < D => înlocuire





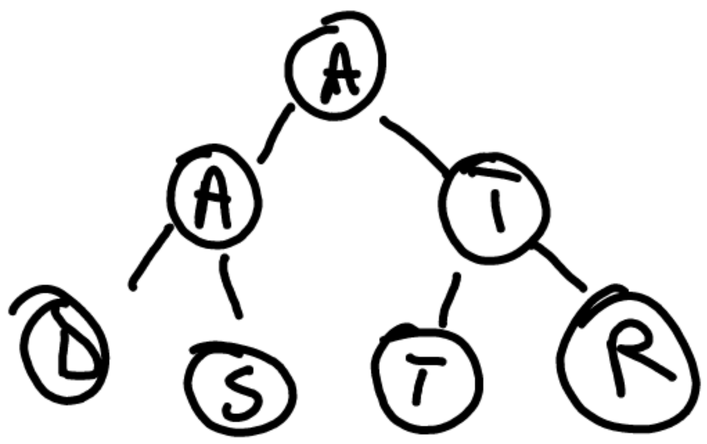
se adaugă S

S > A ✓



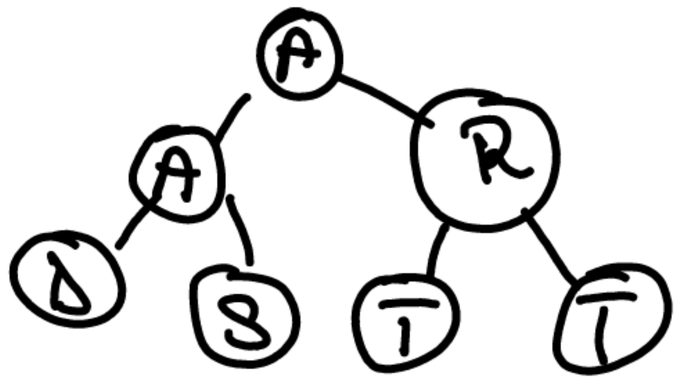
se adaugă T

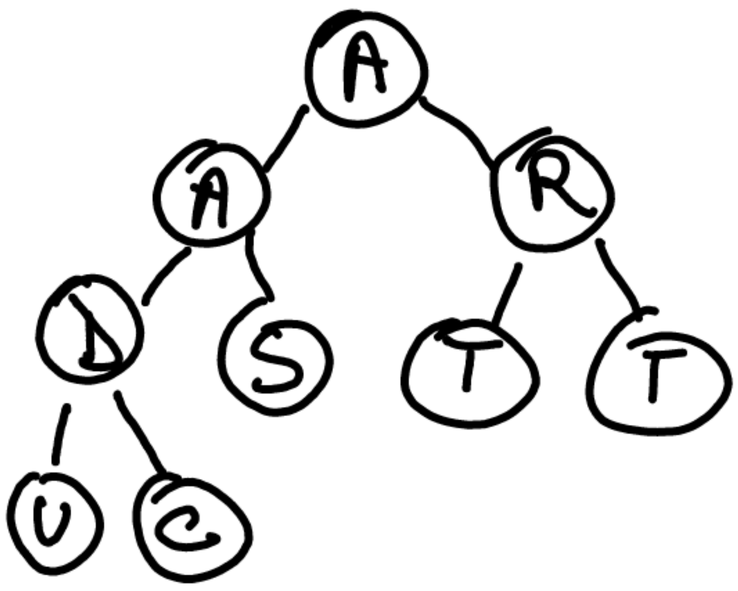
T >= T ✓



se adaugă R

R < T => înălțare

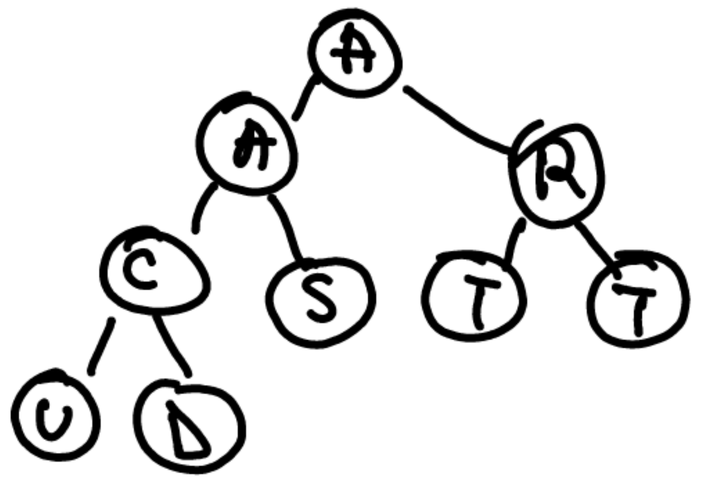


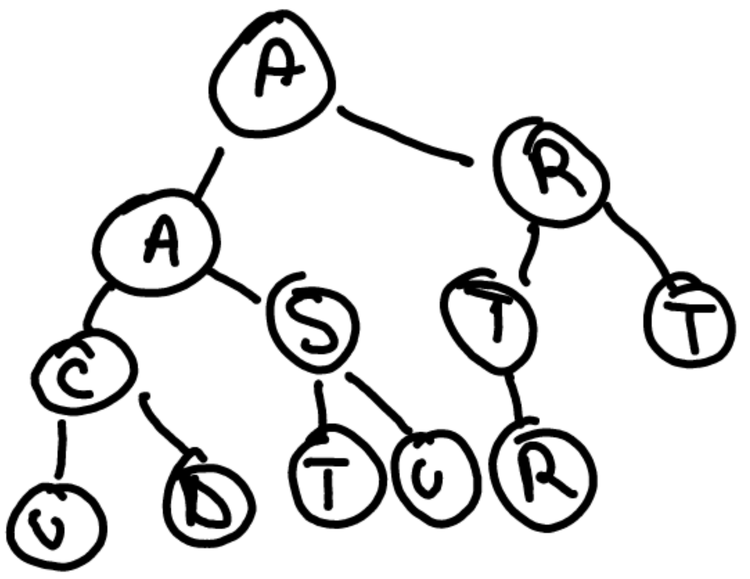


se adaugă V

se adaugă C

C < D => înălțare



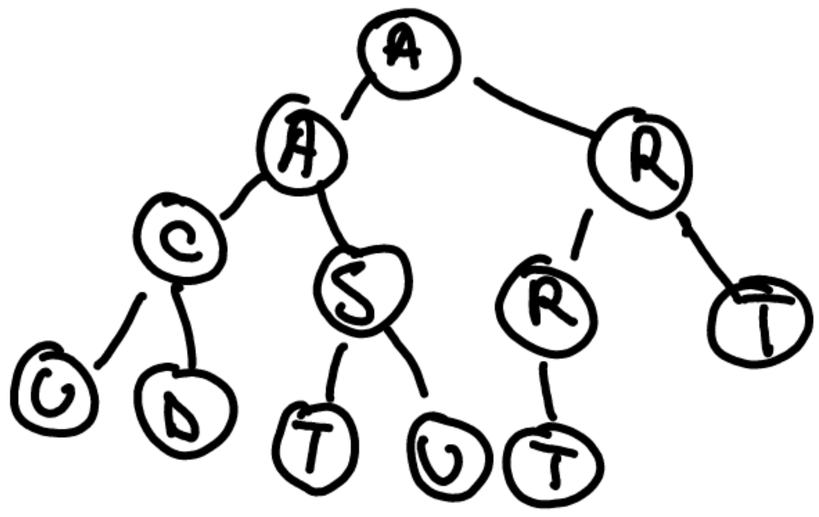


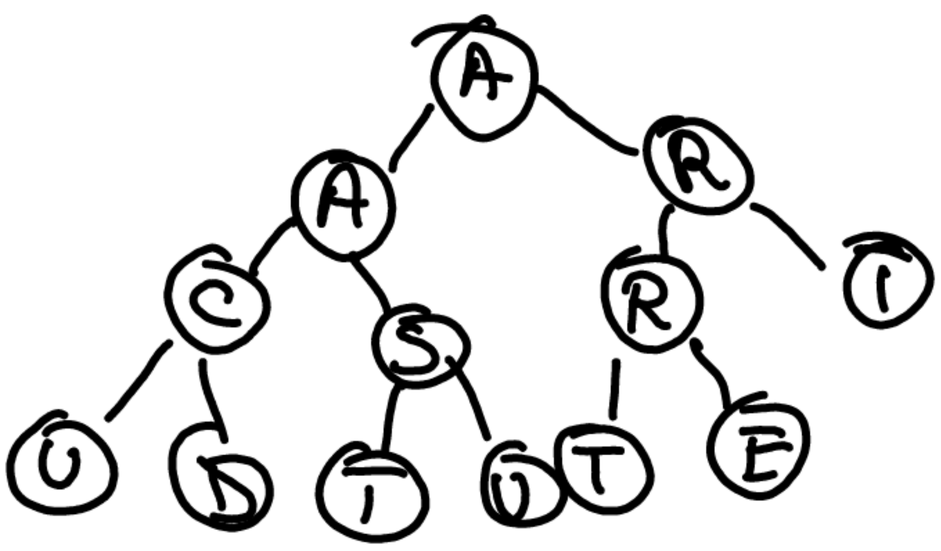
se adaugă T

se adaugă V

se adaugă A

R < T => înălțare



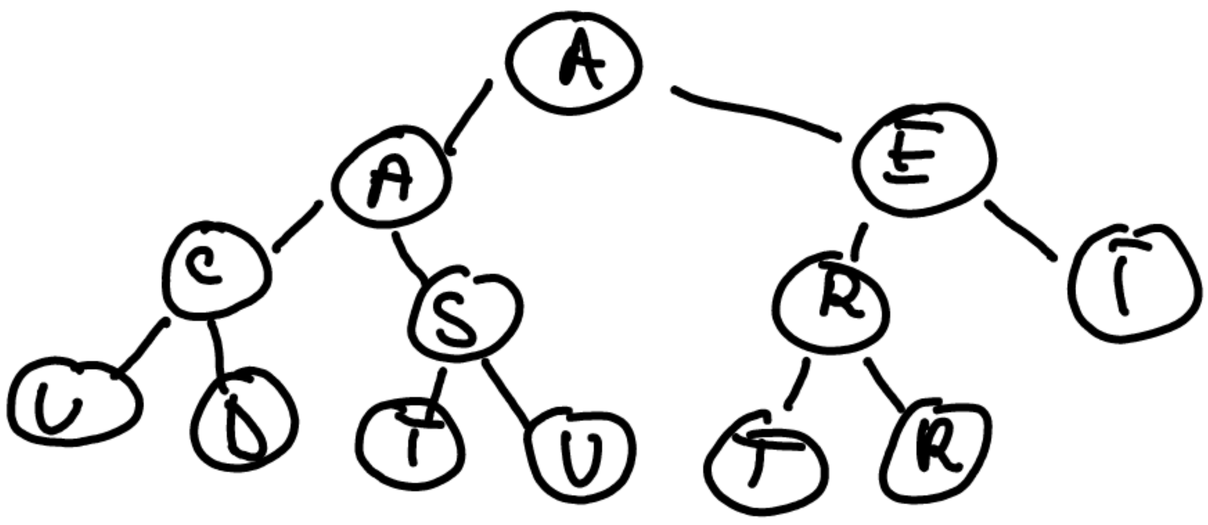


se adaugă E

E < R => înălțare

E < R => înălțare

=



c1. Nr. minim de noduri într-un arbore plin de adâncime 3 este: 15 e)

Acest lucru se datorează faptului că:

Toate cele 4 niveluri sunt pline:

0 -> 1 nod

1 -> 2 noduri

2 -> 4 noduri

3 -> 8 noduri

În total: $1+2+4+8 = 3+12 = 15$ noduri

(se spune că arborele este PLIN, nu APROAPE PLIN)

c2. Costul defavorabil pt căutare liniară într-un vector ordonat este c) timp logaritmic, deoarece, în cazul în care elementul nu se găsește în vector, algoritmul împarte succesiv lista în două până când poziția finală va fi mai mică decât cea inițială $\Rightarrow T(n) = T(n/2) + 1$

Notăm $n = 2^k$

$T(2^k) = T(2^{k-1}) + 1$

$T(2^{k-1}) = T(2^{k-2}) + 1$

$\vdots$

$T(1) = 1$

---

① $T(2^k) = \underbrace{1+1+\dots+1}_{k \text{ ori}} = k+1$

Dar $n = 2^k \Rightarrow k = \log_2 n$ $\Big| \Rightarrow T(n) = \log_2 n + 1 \in \Theta(\log_2 n)$

5.

Subalgoritm suma (v, k) este

{ pre: v ∈ Intreg[]
k ∈ Intreg
post: se returneaza suma a celor mai mari
k numere din lista
@ exceptie daca nu exista k nr. in lista }

Daca dim(v) < k atunci

@ arunca exceptie "Nu sunt suficiente nr"

SF Daca

creeaza (am -> "<=", k)

{s-a creat un ansamblu initializat cu -1, de
dimensiune k (incepe de la indice 1) }

elem - min <- -1
@ocupat <- 0
it <- iterator(v)

prim(it)

CâtTimp valid(it) atunci
a <- element(it)
Daca elem - min < a atunci
adauga (am, a, ocupat, k)
Daca am[k] ≠ -1 atunci
elem - min <- am[k]
altfel
elem - min <- -1
SFDaca
urmator(it)
SFCâtTimp

s ← 0
Pentru i = 1, n executa
    s ← s + am[i]
SfPentru
suma ← s
SfSubalgoritm

Functia adauga(am, n, ocupat, k)este
{pre: am este un ansamblu de dimn n ∈ Intreg
    a ∈ Intreg, a ≠ -1
    ocupat ∈ Intreg
  post: am' ← am + a
        ocupat' ← ocupat + 1}
Daca' ocupat >= n atunci
    sterge(am, n, ocupat)
SfDaca'

ocupat ← ocupat + 1
am[ocupat] ← a
poz ← ocupat

Cat Timp [poz/2] >= 1 ∧ am[poz] < am[[poz/2]]ex
swap { aux ← am[poz]
       am[poz] ← am[[poz/2]]
       am[[poz/2]] ← aux

    poz ← [poz/2]
SfCatTimp
SfFunctia

Se elimină întotdeauna elem. cel mai mic

```
Functia sterge (an, k, ocupat) este
    { pre: an este un ansamblu de k elem.
           k ∈ Intreg
           ocupat ∈ Intreg
      post: ocupat' <- ocupat - 1
            se elimină rădăcina
    }

    an[1] <- an[ocupat]
    ocupat <- ocupat - 1

    poz <- 1
    pozd <- poz * 2
    Dacă pozd < ocupat ∧ an[pozd+1] < an[pozd] at
        pozd <- pozd + 1
    SfDacă

    Cât timp pozd ≤ ocupat ∧ an[pozd] < an[poz] ex.
        aux <- an[pozd]
        an[pozd] <- an[poz]
        an[poz] <- aux

        poz <- pozd
        pozd <- poz * 2
        Dacă pozd < ocupat ∧ an[pozd+1] < an[pozd] at
            pozd <- pozd + 1
        SfDacă
    SfCâtTimp
SfFunctia
```

Alte funcții utilizate:

dim(v) -> returnează dimensiunea listei v

iterator(v) -> creează iterator pt. lista v

element(it) -> returnează elementul de la it, dacă iteratorul este valid

următor(it) -> trece la "poziția" următoare în listă

valid(it) -> verifică dacă iteratorul este valid

creare(am, "c=", k) -> creează un ansamblu cu relația c=, de dimensiune k.

Funcția adaugă are o complexitate $O(\log_2 k)$
(adăugarea într-un ansamblu -> cost delete $O(h)$, $h = \log_2 n$)

Funcția șterge are tot o complexitate $O(\log_2 k)$
(1 tot operație de căutare pe ansamblu)

în subalgoritm se parcurg toate cele n elemente și se adaugă / adaugă + șterg din ansamblu

$\sum_{i=1}^{n} \log_2 k = n \cdot \log_2 k \Rightarrow O(n \log_2 k)$