Codul Huffmann - Arborele de codificare Huffmann

Algoritmul Huffmann este folosit pentru compresia datelor fără pierdere de informaţie. Presupunând că se codifică un text alcătuit din caractere ascii, algoritmul Huffmann utilizează un tablou de frecvenţe, în care se specifică pentru fiecare caracter o valoare care reprezintă frecvenţa de apariţie a acestuia. Lungimea secvenţei de codificare (în biţi) pentru fiecare caracter depinde de frecvenţa acestiua. Cu cât un caracter apare mai frecvent în text, cu atât lungimea codului utilizată pentru el este mai mică.

Coduri prefix (prefix-codes)

Un cod prefix este un cod în care nici un element al codului nu reprezintă un prefix pentru al element. Acest lucru permite codificare/decodificare fără ambiguităţi.

Exemplu:

Se consideră alfabetul alcătuit din primele 6 litere {a, b, c, d, e, f} şi următorul tabel de codificare:

	a	b	c	d	e	f
Frecvenţa (la 1000 caractere în lb. engl.)	45	13	12	16	9	5
Codificare cu cod de lungime fixă (3 biţi) – cod1	000	001	010	011	100	101
Codificare cu cod de lungime variabilă – cod2	0	101	100	111	1101	1100

Codificarea cu codul 2 00101100110110100 reprezintă 0. 0. 101. 100. 1101. 101.0.0 adică textul aabcebaa

Arborele Huffman

Un mod de construcţie a unui astfel de cod prefix de lungime variabilă se realizează cu ajutorului arborelui Huffman. În acest arbore, frunzele reprezintă caracterele de codificat. Arcul de la un nod părinte la descendentul stâng se marchează cu caracterul 0, iar cel către descendentul drept se marchează cu 1.

Codul corespunzător unui caracter dintr-o frunză se obţine prin concatenarea caracterelor care se află pre drumul de la rădăcină către frunza respectivă.

Observaţie: un cod optimal se obţine pe baza unui arbore binar în care fiecare nod neterminal are exact doi descendenţi.

Construcţia arborelui de codificare Huffman:

Se plasează fiecare caracter într-o frunză şi se plasează într-o coadă de prioritate (min) de noduri. Deci în vârf va fi nodul de prioritate minimă.
Cât timp în coadă se găsesc cel puţin 2 noduri: se extrag primele două minime, se leagă ca fii de către un nou nod părinte, a cărui frecvenţă va fi dată de suma frecvenţelor celor 2 copii. Acest nod nou se inserează în coada de prioritate.

După obţinerea arborelui Huffman se poate construi un look-up table în care fiecărui caracter i se asociază codul obţinut pe baza arborelui. Acest tabel se foloseşte apoi pentru codificare.

Decodificarea: pentru a putea decodifica corect documentul, este necesar ca atât cel care codifică, cât şi cel care decodifică să utilizeze acelaşi arbore. Acest lucru se realizează ori prin folosirea de comun acord a aceluiaşi set de caractere cu aceeaşi frecvenţă sau prin transmiterea împreună cu textul codificat şi a arborelui de codificare. În literatură sunt propuse diferite moduri de soluţionare a acestei probleme.

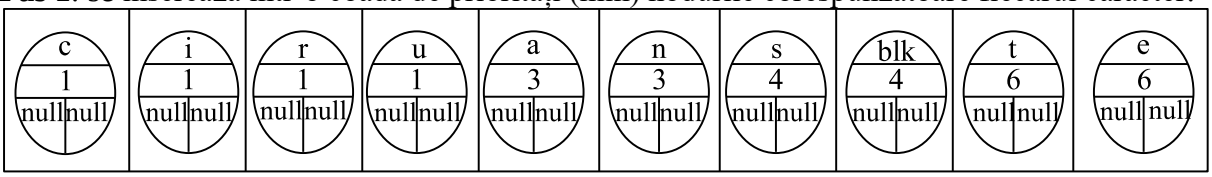
Exemplu:

Considerăm următorul text: acesta este un test interesant.
Avem următorul tablou de frecvențe:

a	c	e	i	n	R	s	t	u	Blank
3	1	6	1	3	1	4	6	1	4

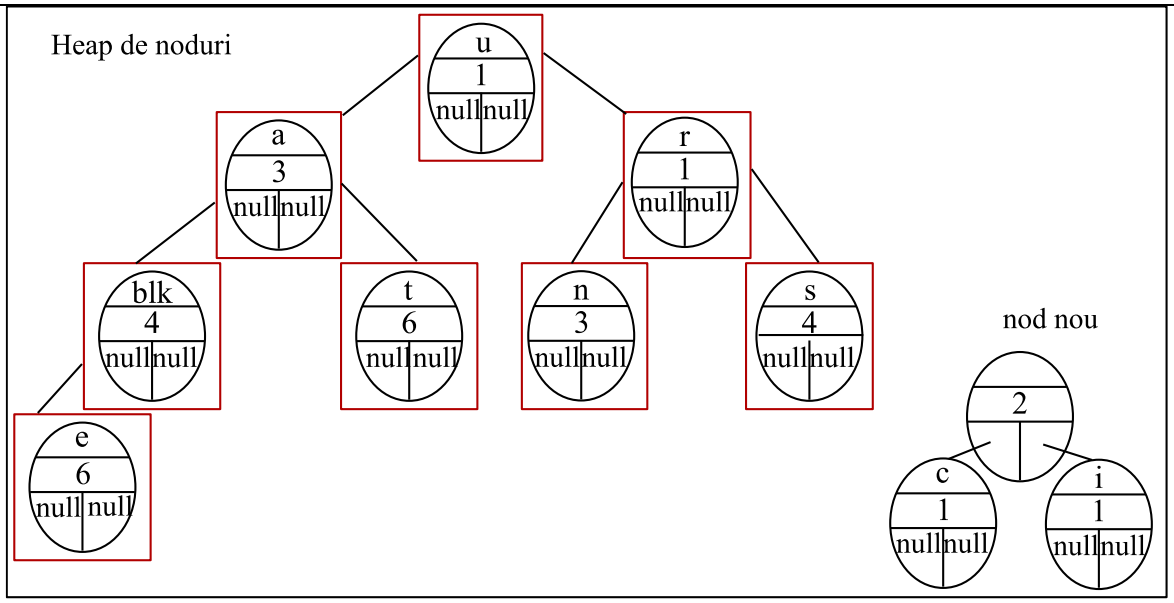
Construim un arbore în care fiecare nod are: un câmp frecvență, un câmp caracter și câmpuri de legătură către fii stâng și drept și către părinte (eventual). Dacă algoritmul de parcurgere al arborelui pentru determinarea codurilor este recursiv, nu este nevoie de legătura la părinte.

Pas 1: se inserează într-o coadă de priorități (min) nodurile corespunzătoare fiecărui caracter:

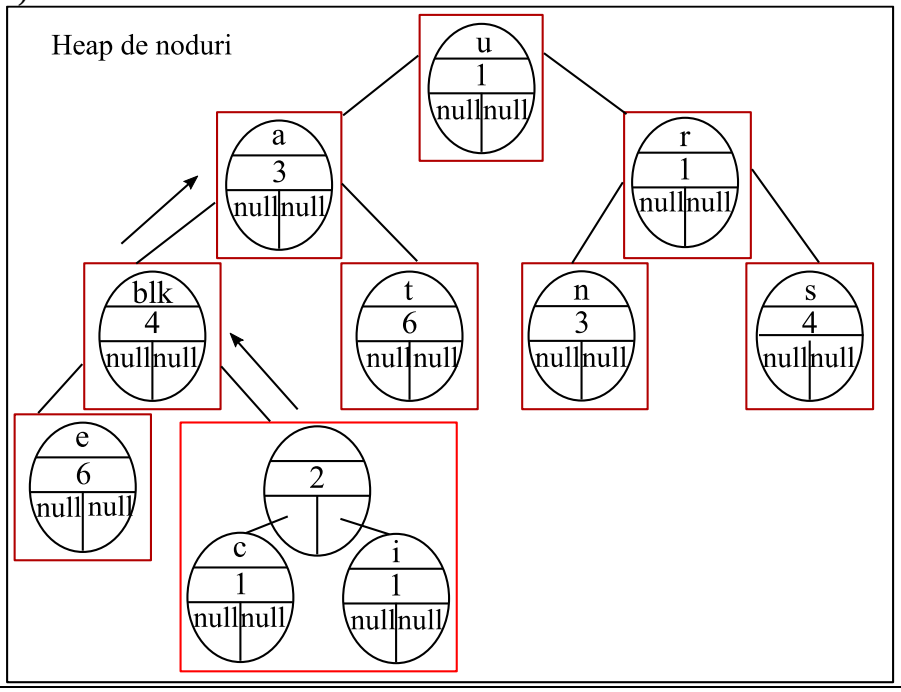


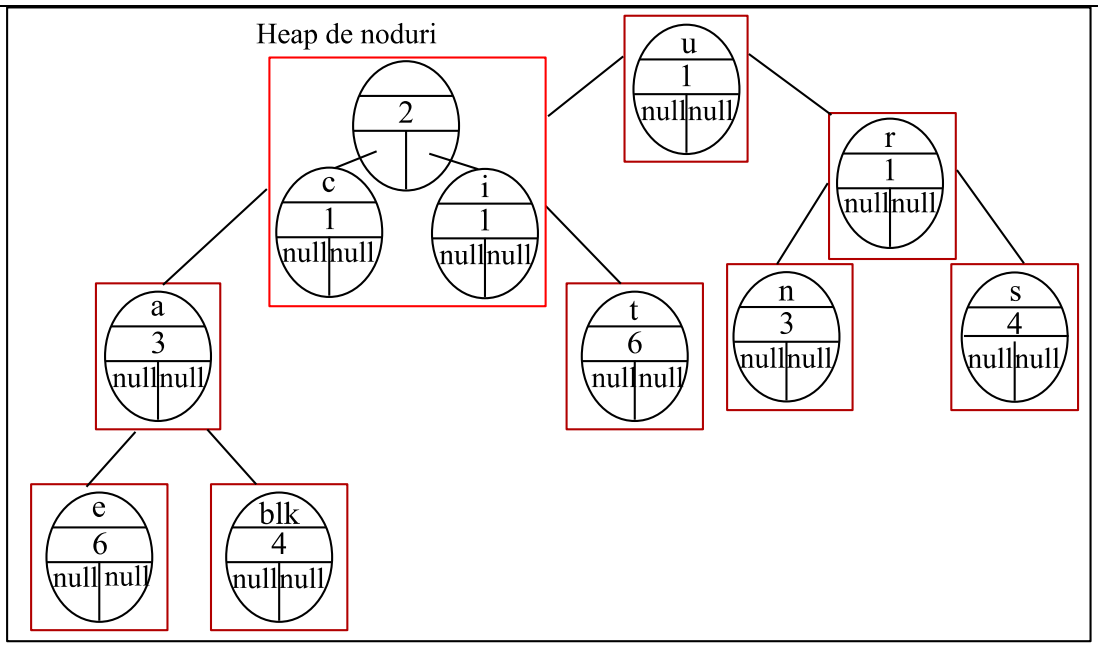
Pas 2: cât timp în coadă există mai mult decât un nod, se procedează în modul următor: se extrag primele noduri (cu frecvența minimă) din coadă, se creează un nou nod care are ca frecvență suma frecvențelor celor două noduri, se leagă cele două noduri ca fii ai noului nod și apoi se inserează apoi în coada de priorități. Dimensiuna heap-ului se reduce cu 1.

1)


Heap de noduri
u
1
null null
a
3
null null
r
1
null null
blk
4
null null
t
6
null null
n
3
null null
s
4
null null
e
6
null null
nod nou
2
c
1
null null
i
1
null null

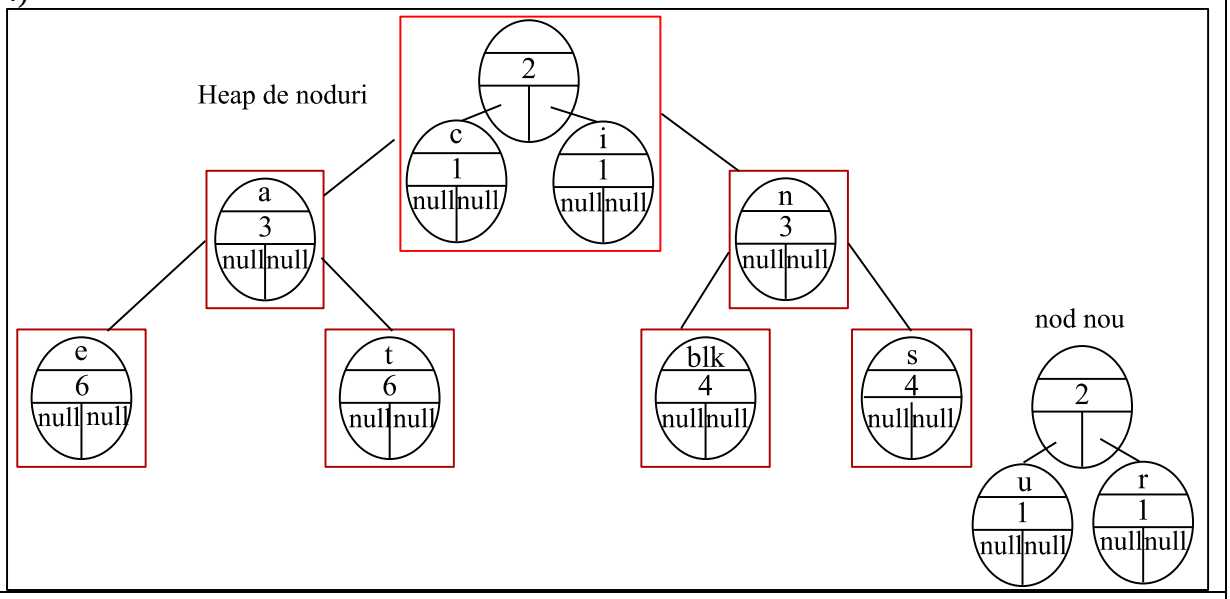
2)

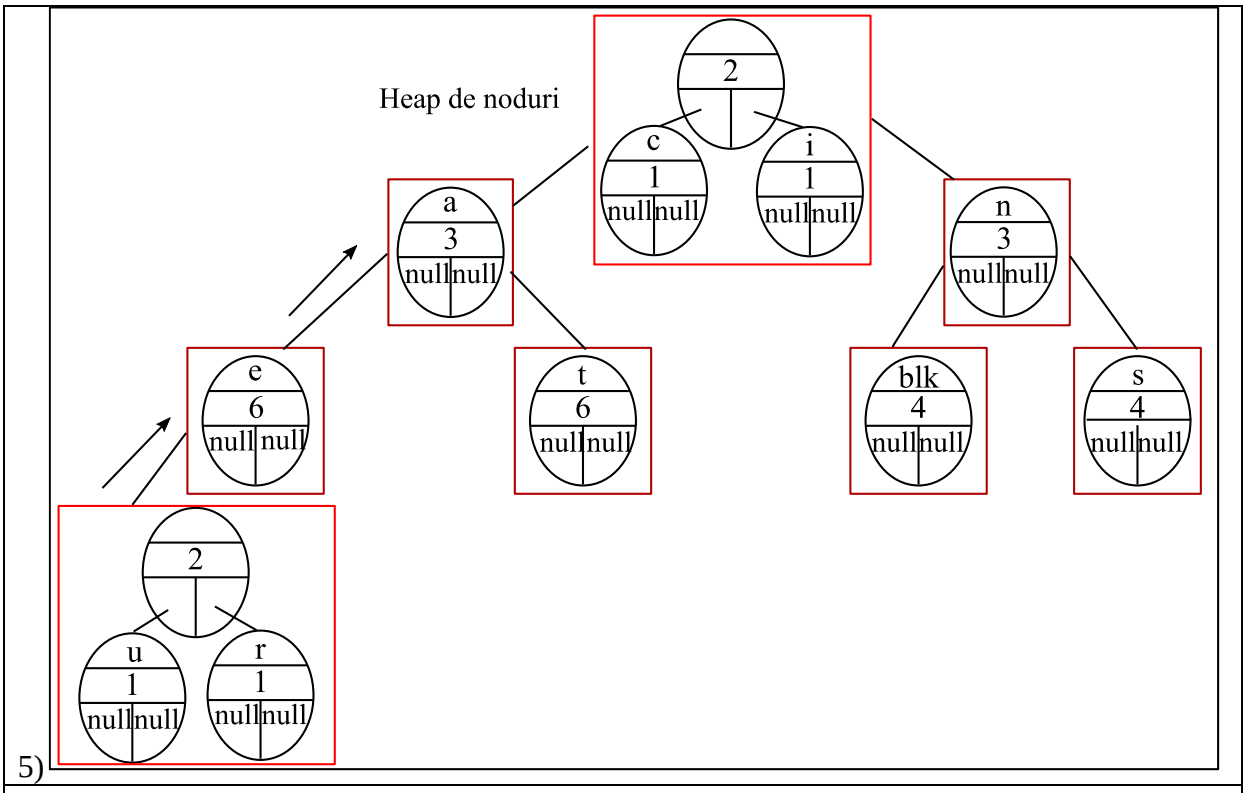

Heap de noduri
u
1
null null
a
3
null null
r
1
null null
blk
4
null null
t
6
null null
n
3
null null
s
4
null null
e
6
null null
2
c
1
null null
i
1
null null

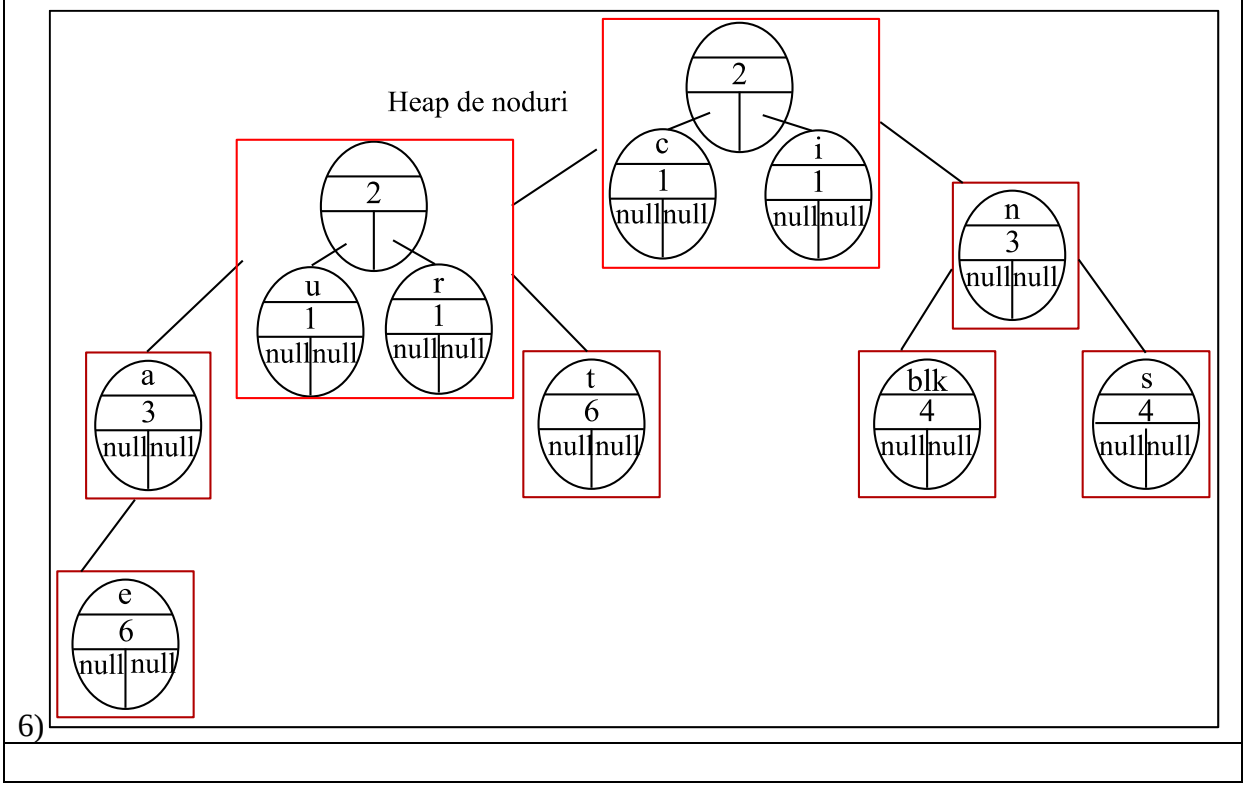

Heap de noduri
2
c
1
null null
i
1
null null
u
1
null null
a
3
null null
t
6
null null
r
1
null null
n
3
null null
s
4
null null
e
6
null null
blk
4
null null

3)

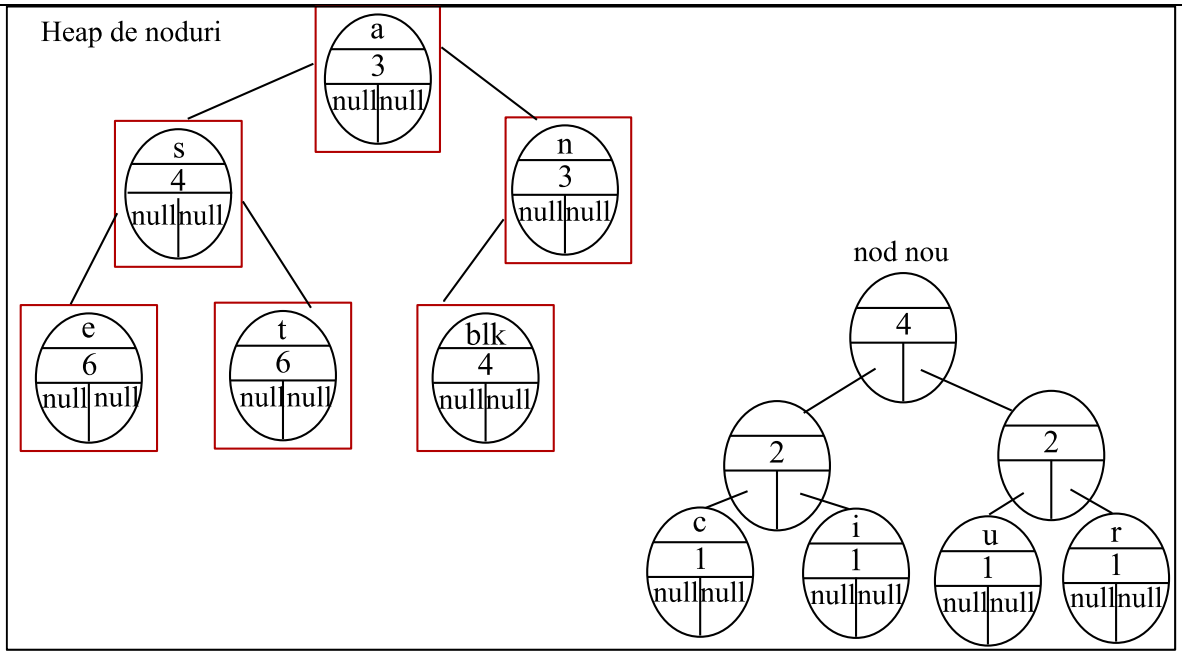
4)


Heap de noduri
2
c
1
null null
i
1
null null
a
3
null null
n
3
null null
e
6
null null
t
6
null null
blk
4
null null
s
4
null null
nod nou
2
u
1
null null
r
1
null null

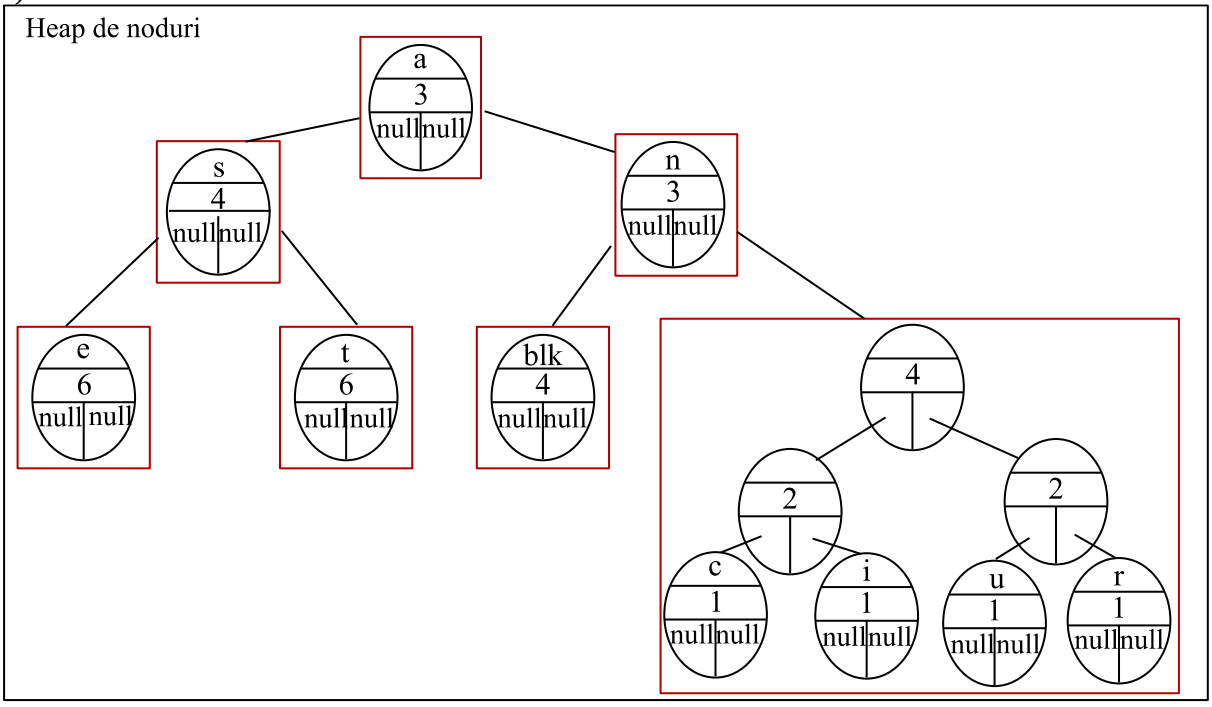

Heap de noduri
2
c
1
null null
i
1
null null
a
3
null null
n
3
null null
e
6
null null
t
6
null null
blk
4
null null
s
4
null null
2
u
1
null null
r
1
null null
5)


Heap de noduri
2
c
1
null null
i
1
null null
2
u
1
null null
r
1
null null
n
3
null null
a
3
null null
t
6
null null
blk
4
null null
s
4
null null
e
6
null null
6)

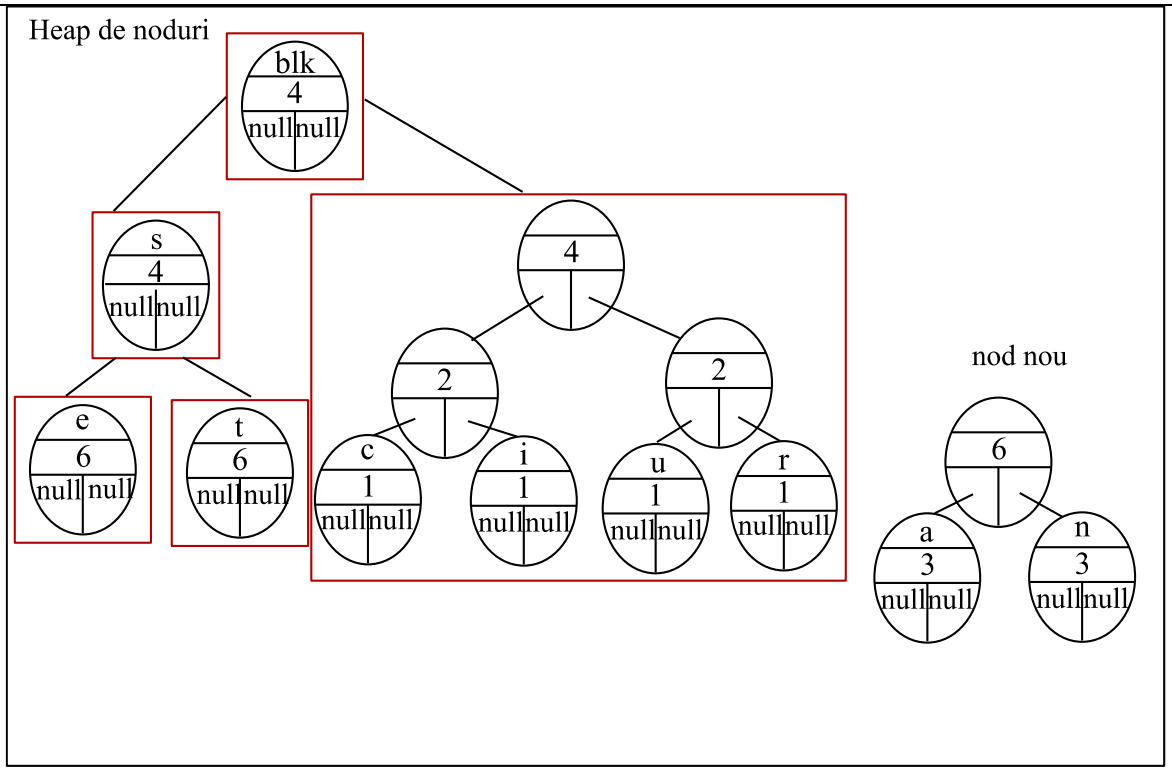
7)


Heap de noduri
a
3
null null
s
4
null null
n
3
null null
e
6
null null
t
6
null null
blk
4
null null
nod nou
4
2
2
c
1
null null
i
1
null null
u
1
null null
r
1
null null

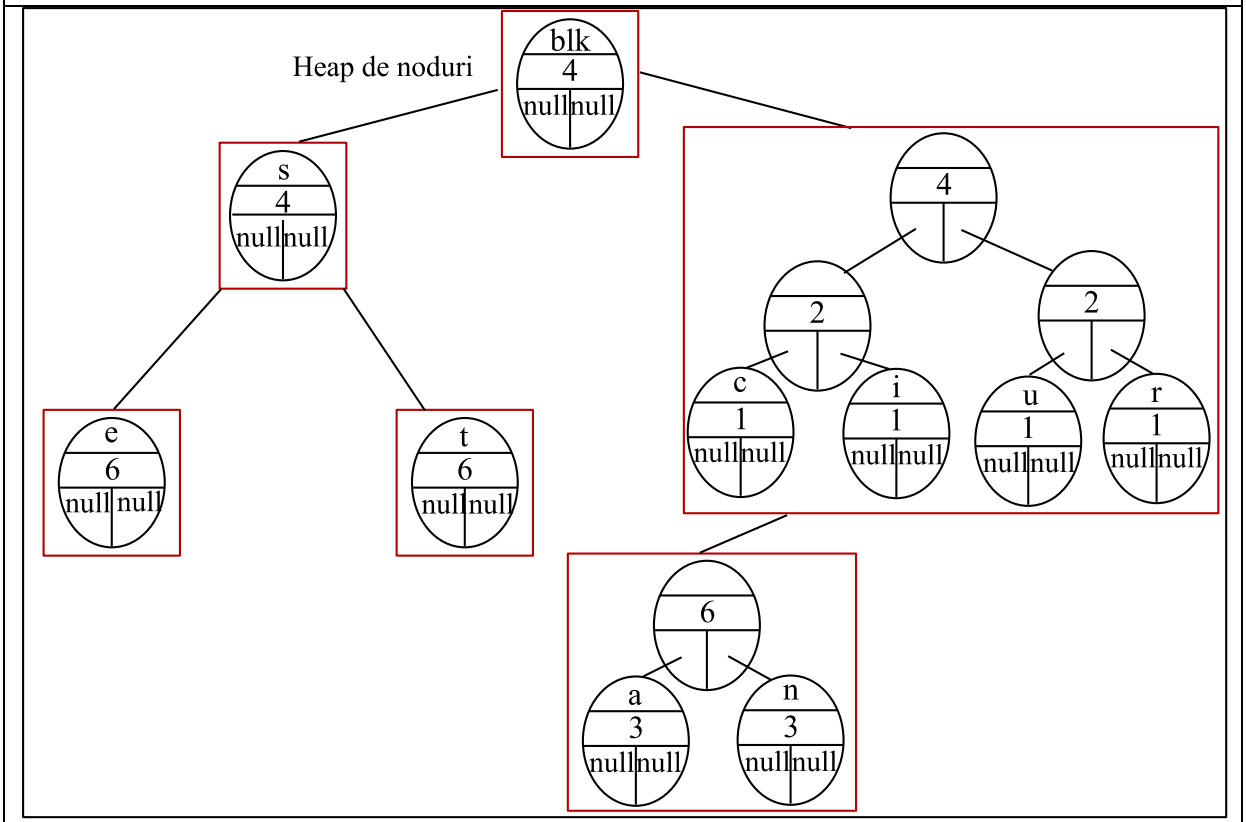
8)

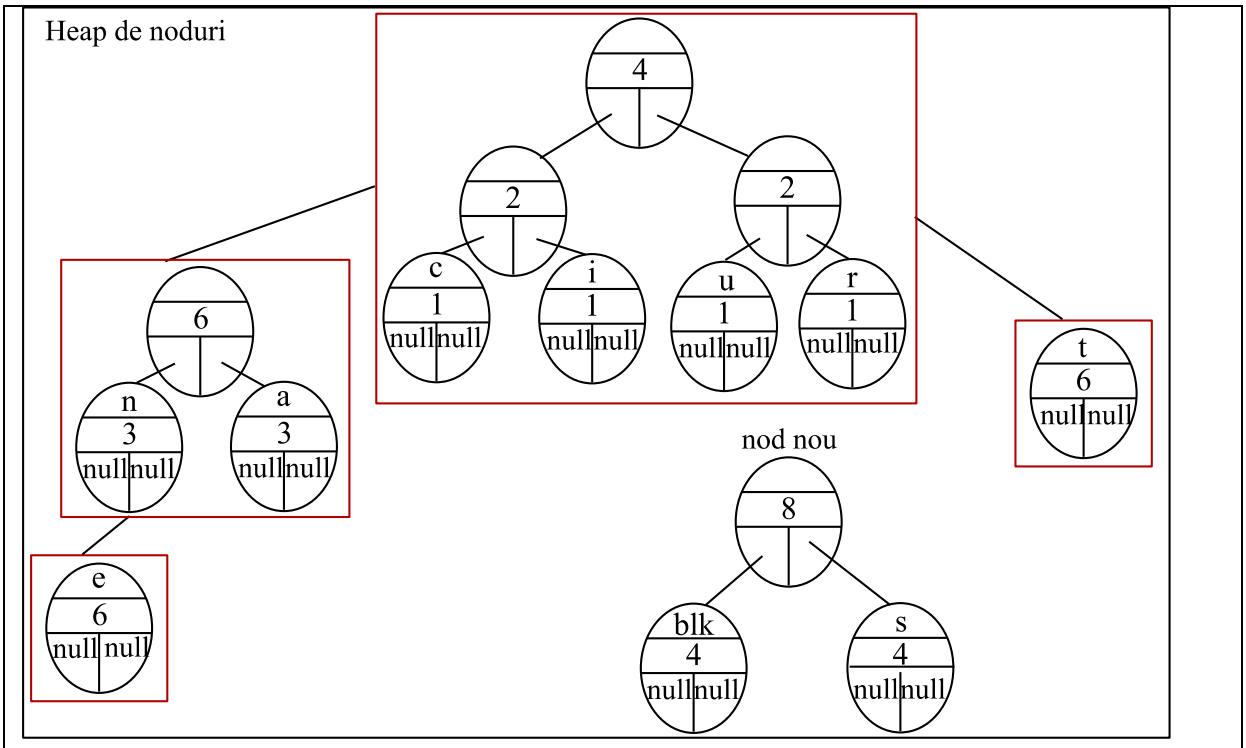

Heap de noduri
a
3
null null
s
4
null null
n
3
null null
e
6
null null
t
6
null null
blk
4
null null
4
2
2
c
1
null null
i
1
null null
u
1
null null
r
1
null null

9)

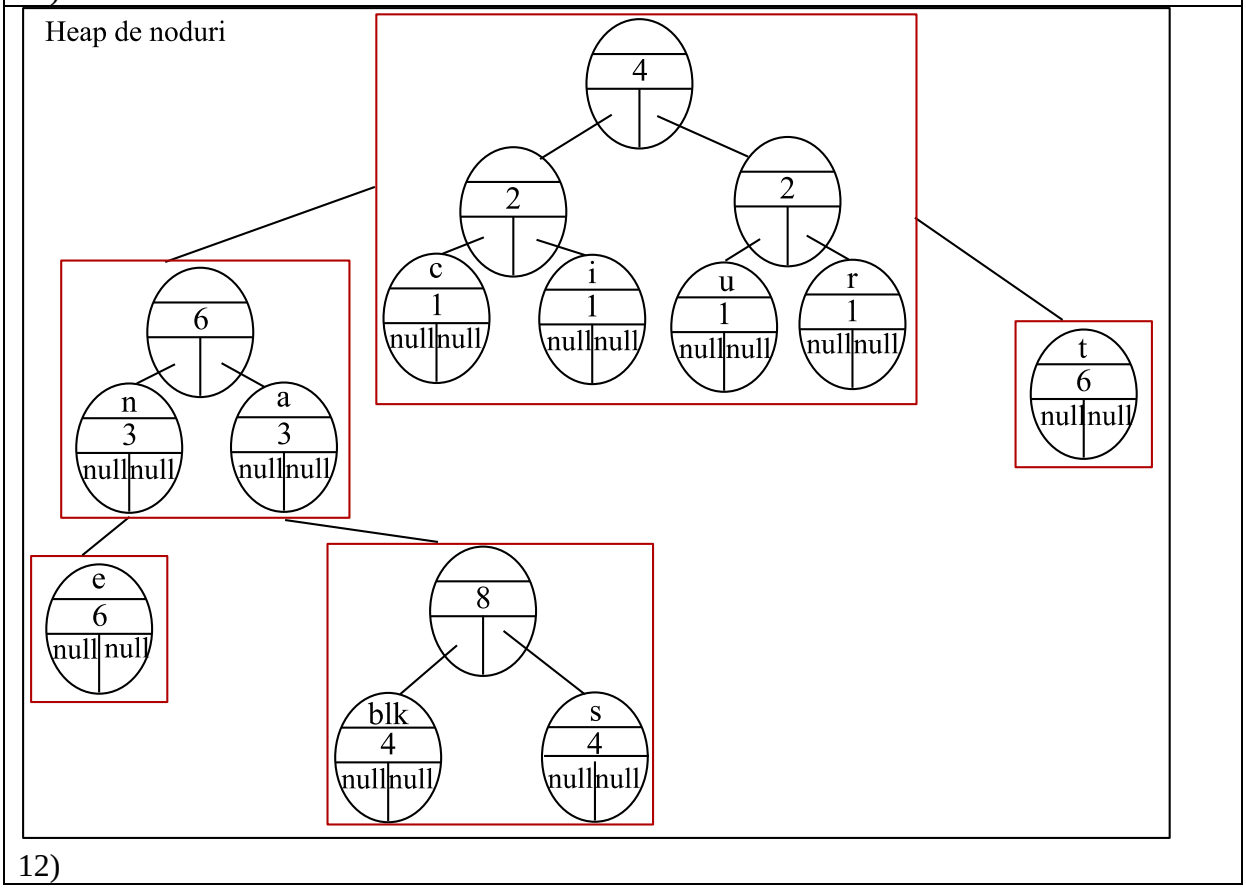

Heap de noduri
blk
4
null null
s
4
null null
e
6
null null
t
6
null null
4
2
2
c
1
null null
i
1
null null
u
1
null null
r
1
null null
nod nou
6
a
3
null null
n
3
null null

10)

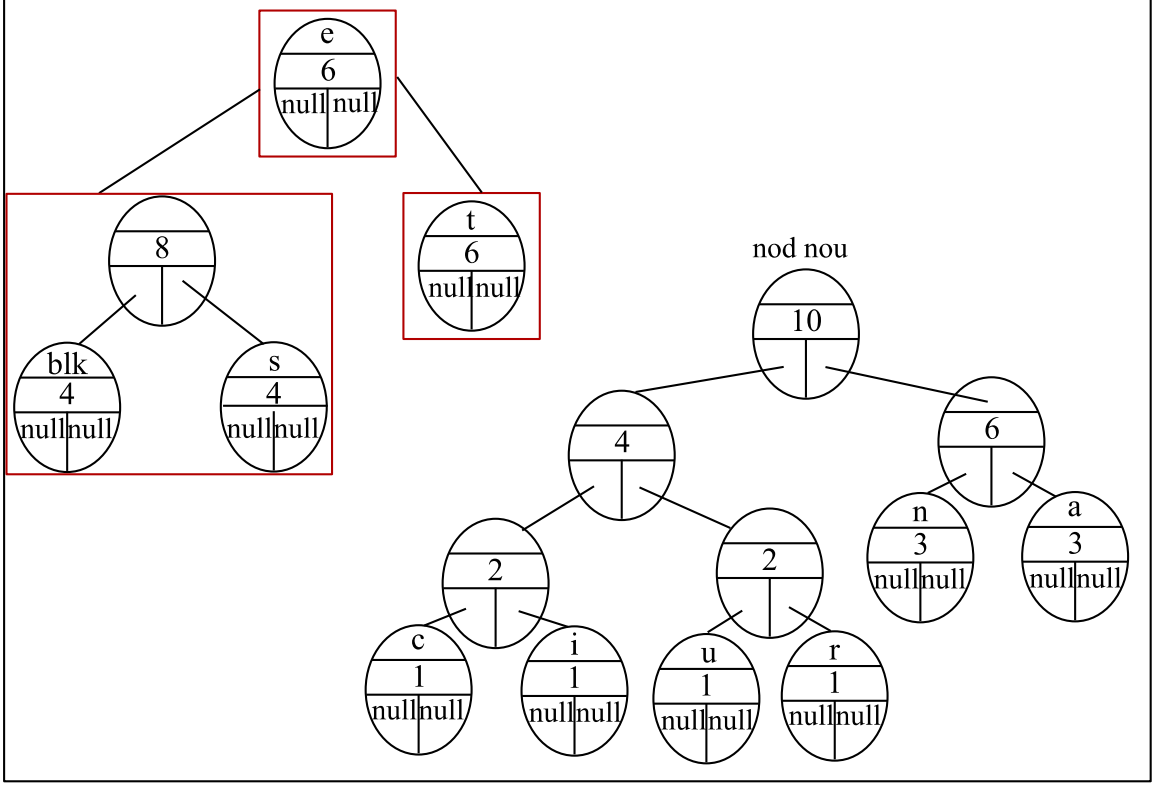

Heap de noduri
blk
4
null null
s
4
null null
e
6
null null
t
6
null null
4
2
2
c
1
null null
i
1
null null
u
1
null null
r
1
null null
6
a
3
null null
n
3
null null


Heap de noduri
4
2
2
c
1
null null
i
1
null null
u
1
null null
r
1
null null
6
n
3
null null
a
3
null null
t
6
null null
e
6
null null
nod nou
8
blk
4
null null
s
4
null null

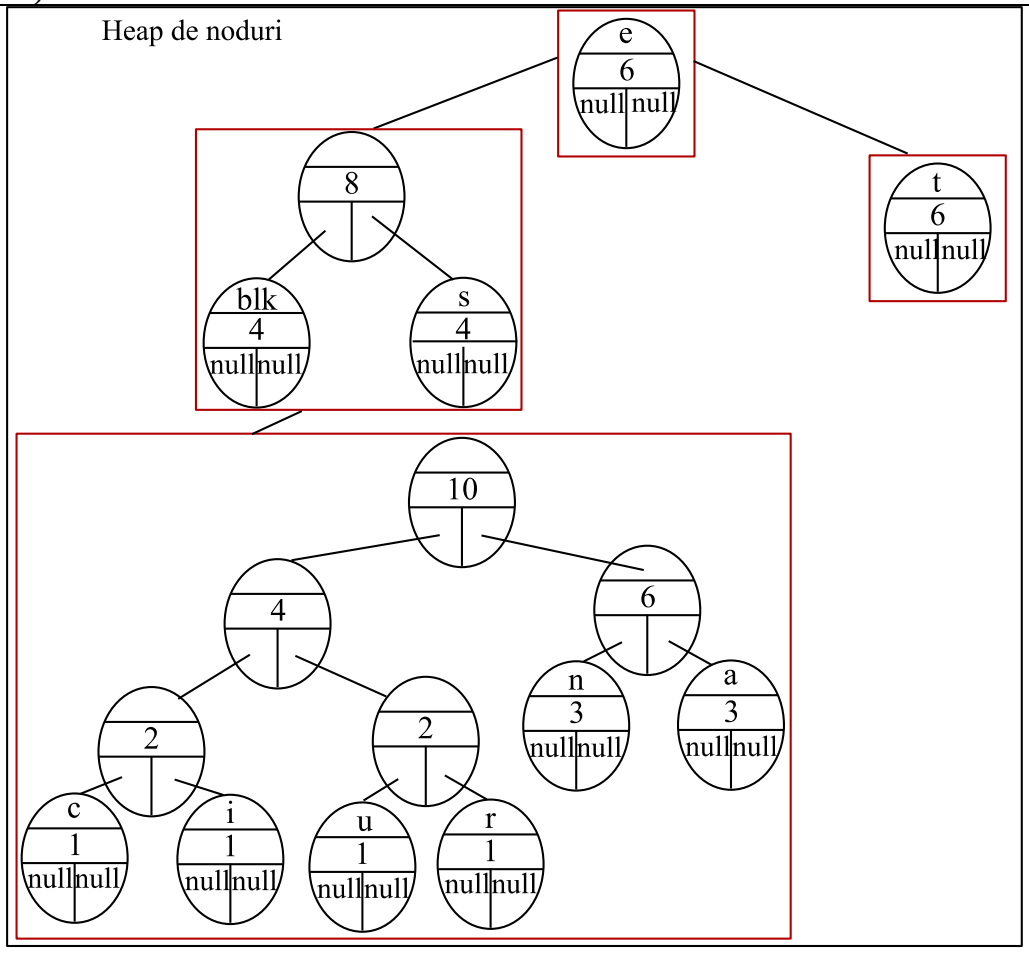
11)


Heap de noduri
4
2
2
c
1
null null
i
1
null null
u
1
null null
r
1
null null
6
n
3
null null
a
3
null null
t
6
null null
e
6
null null
8
blk
4
null null
s
4
null null

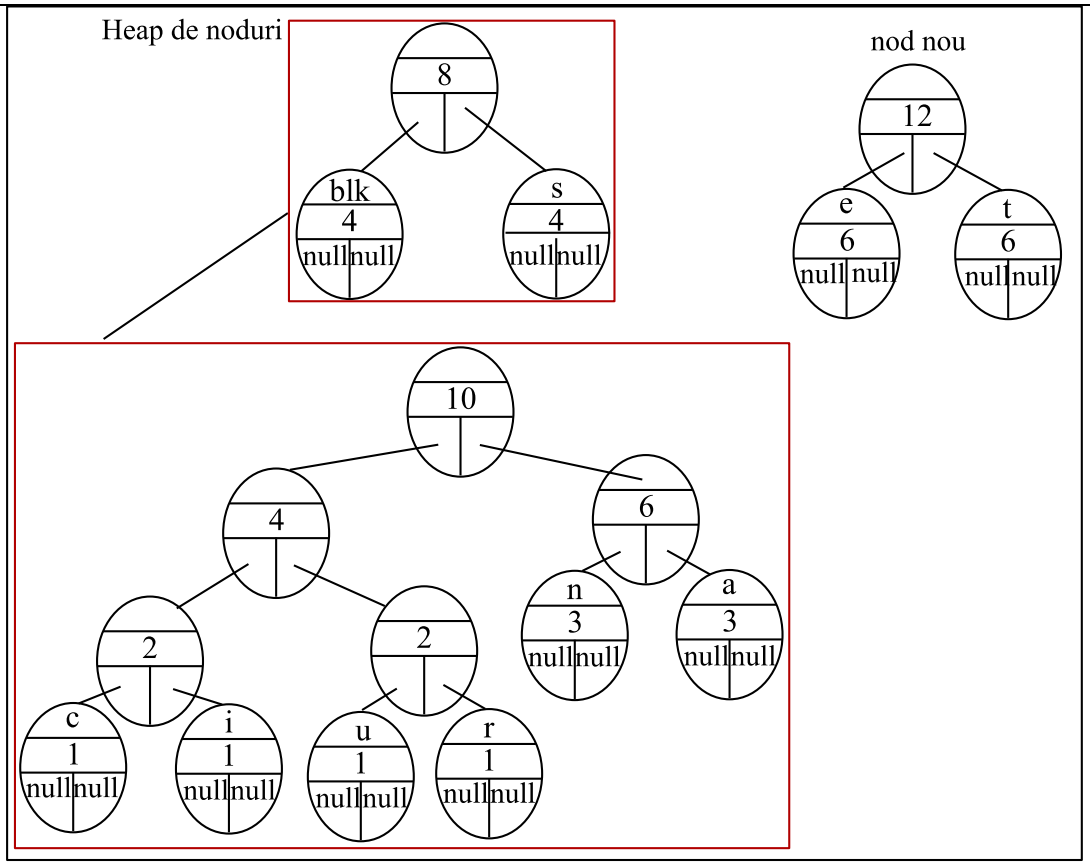
12)


Heap de noduri
e
6
null null
8
t
6
null null
blk
4
null null
s
4
null null
nod nou
10
4
6
2
2
n
3
null null
a
3
null null
c
1
null null
i
1
null null
u
1
null null
r
1
null null

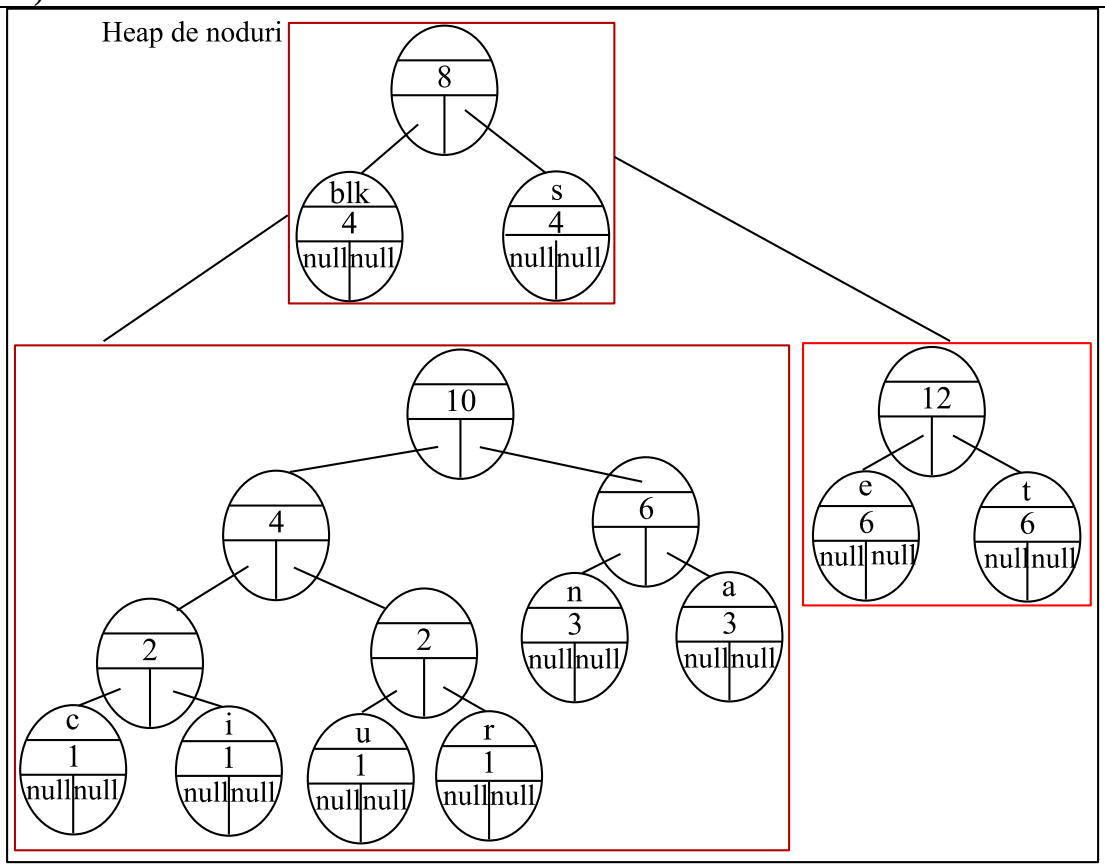
13)


Heap de noduri
e
6
null null
8
t
6
null null
blk
4
null null
s
4
null null
10
4
6
2
2
n
3
null null
a
3
null null
c
1
null null
i
1
null null
u
1
null null
r
1
null null

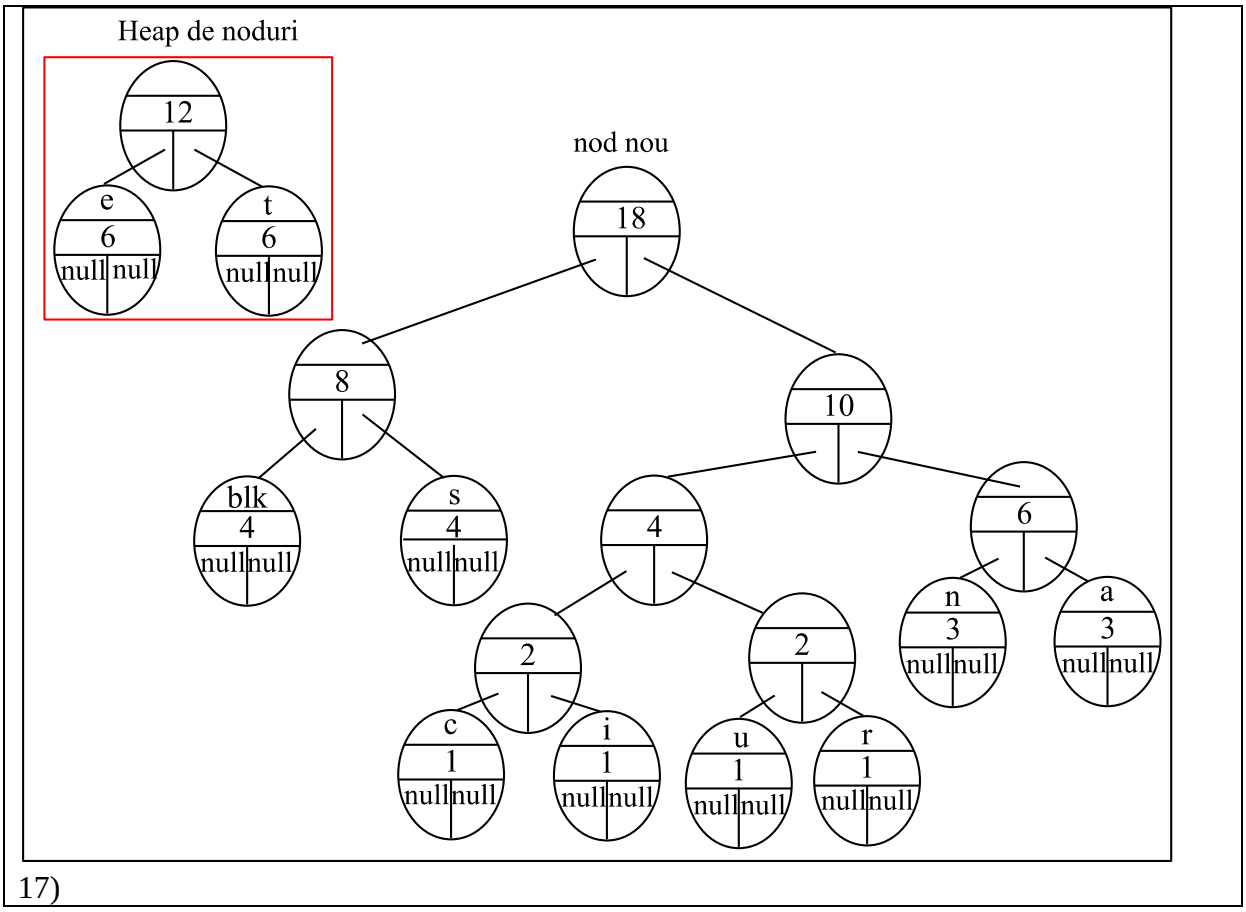
14)


Heap de noduri
8
blk
4
null null
s
4
null null
nod nou
12
e
6
null null
t
6
null null
10
4
6
2
2
n
3
null null
a
3
null null
c
1
null null
i
1
null null
u
1
null null
r
1
null null

15)

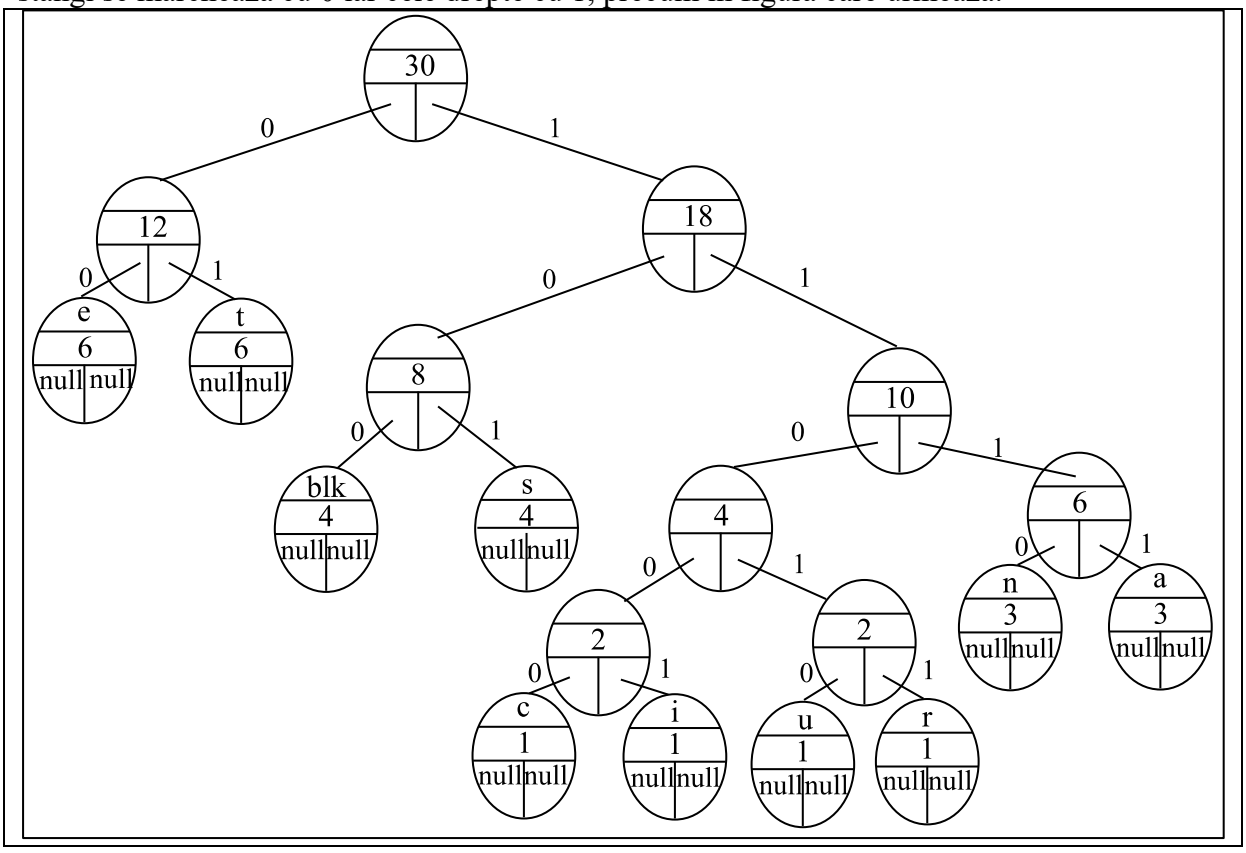

Heap de noduri
8
blk
4
null null
s
4
null null
10
4
6
2
2
n
3
null null
a
3
null null
c
1
null null
i
1
null null
u
1
null null
r
1
null null
12
e
6
null null
t
6
null null

16)



17)

În final, după procesarea ultimelor două noduri se obține arborele Huffmann în care ramurile stângi se marchează cu 0 iar cele drepte cu 1, precum în figura care urmează:



Codurile pentru fiecare caracter dintr-o frunză se obțin prin alăturarea caracterelor de 0 și 1, care se întâlnesc pe drumul de la rădăcină la frunza respectivă.
Astfel pentru arborele de mai sus caracterele vor avea codurile:

Car	a	c	e	i	n	r	s	t	u	Blank
Fecv	3	1	6	1	3	1	4	6	1	4
Cod	1111	11000	00	11001	1110	11011	101	01	11010	100