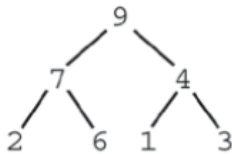


Capitolul IC.09. Autoevaluare

Observație

Există întrebări cu mai multe răspunsuri corecte

1. Se consideră arborele MaxHeap cu implementarea statică [9, 7, 4, 2, 6, 1, 3] și reprezentarea



Care va fi conținutul tabloului după ștergerea din heap a valorii maxime?

- a) [7, 4, 2, 6, 1, 3]
- b) [7, 4, 6, 2, 1, 3]
- c) [7, 4, 6, 2, 3, 1]
- d) [7, 6, 4, 2, 1, 3]
- e) [7, 6, 4, 2, 3, 1]

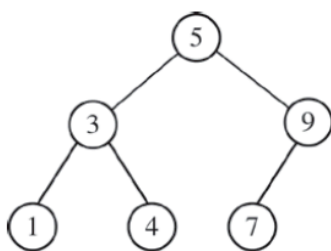
2. Într-un arbore binar plin, toate nodurile interne au exact doi descendenți. Câte noduri interne sunt într-un arbore binar plin cu 512 frunze?

- a) 256
- b) 511
- c) 512
- d) 513
- e) 1024

3. Care dintre următoarele structuri de date este cea mai potrivită pentru stocarea simbolurilor și a cuvintelor rezervate specifice unui limbaj de programare?

- a) tabelă de dispersie
- b) listă înlănțuită
- c) arbore binar
- d) coadă de priorități
- e) tablou unidimensional

4. Se consideră arborele binar de căutare de mai jos.



Pornind de la un arbore gol, inserarea cărei secvențe de chei întregi conduce la BST de mai sus?

- (A) 5, 9, 1, 7, 3, 4
- (B) 5, 7, 4, 9, 3, 1
- (C) 5, 4, 7, 3, 9, 1
- (D) 5, 3, 4, 9, 1, 7
- (E) 5, 3, 1, 7, 9, 4

5. Care dintre structurile de mai jos este cea mai potrivită pentru a implementa o colecție de valori cu următoarele caracteristici:

- Elementele sunt inserate, regăsite și șterse în mod FIFO;
 - Numărul de elemente a colecției nu este cunoscut apriori;
 - Este necesară alocarea unei zone de memorie relativ mare pentru fiecare element al colecției.
- a) Listă liniară simplu înlănțuită, cu pointeri la primul și ultimul element
 - b) Listă dublu înlănțuită, cu pointer la unul din capete
 - c) Tablou
 - d) Arbore binar
 - e) Tabelă de dispersie

6. Care este ordinul de complexitate pentru următoarea secvență?

```
while (i>0)
{
    for (j = 0 ; j < n/2 ; j++)
        cout << i << j << endl;
    i--;
```

- a) $O(n^2)$ b) $O(n \log n)$ c) $O(\log n)$ d) $O(n)$

7. Se consideră tabloul V ce conține reprezentarea implicită a unui arbore binar:

60	25	45	20	15	30	40	10	18	12	5	22
----	----	----	----	----	----	----	----	----	----	---	----

Arborele este:

- a) AVL b) BST c) Heap d) nicio variantă nu este corectă

8. Fie un arbore heap cu N elemente și k nivele complete, implementat static “fără goluri”. Atunci:

- a. Pe ultimul nivel sunt $N-2^{k+1}+1$ noduri
- b. Pe ultimul nivel sunt $2^{k+1}-N+1$ goluri
- c. Pe ultimul nivel sunt $N-2^k-1$ noduri
- d. Pe ultimul nivel sunt $2^{k+1}-N-1$ goluri

9. Care din afirmațiile de mai jos sunt corecte?

- a) Dacă un arbore binar de căutare și un arbore B conțin același număr de chei, atunci cei doi arbori au adâncimi egale.
- b) Ordinul de complexitate pentru operația de inserare a unui nod într-o coadă liniară alocată dinamic este $O(1)$.
- c) Căutarea binară este de complexitate mai mică decât căutarea secvențială.
- d) Dacă un arbore B este de ordin 3, atunci orice pagină cuprinde cel mult 3 chei.

10. Care este ordinul de complexitate pentru următoarea secvență?

```
for (i = 1 ; i < n ; i=i*2)
    for (j = 0 ; j < n; j++)
        cout << i << j << endl;
```

- a) $O(n^2)$ b) $O(n \log n)$ c) $O(\log n)$ d) $O(n)$

11. Dată fiind Q o coadă circulară alocată static, Q.vect este vectorul ce conține elementele cozii, Q.head și Q.tail reprezintă indexul primului element respectiv indexul primului loc liber în care se poate insera un element și funcția următoare:

```

put(Q,a)
  if Q.tail=Nextpoz(Q.head) then
    print "Coadă plină"
  elseif
    Q.vect[Q.tail] ← a
    Q.tail ← Nextpoz(Q.tail)
  endif
end

```

Unde este o eroare în secvența dată:

- a) Pe ramura **then**
- b) Pe ramura **elseif**
- c) Testul de la **if**
- d) Secvența este corectă

12. Completați spațiile libere:

Într-un arbore B inserarea se face întotdeauna (precizați locul)

Într-un arbore binar complet de adâncime h, numărul total de noduri este

Căutarea nodului de cheie minimă într-un BST, **în cazul cel mai defavorabil**, este de complexitate

Într-un arbore B, nodurile sunt grupate în, cheile fiind ordonate strict crescător.

Algoritmul de sortare prin inserție este de complexitate

Într-un arbore binar complet de adâncime h, numărul total de noduri este.....

Căutarea nodului de cheie minimă într-un BST, **în cazul mediu**, este de complexitate

13. Care trebuie să fie ordinea introducerii elementelor 2,4,5,6,7,9 într-un arbore binar de căutare astfel încât, la parcurgerea acestuia în preordine să se obțină următoarea secvență
4 2 7 5 6 9

- a. 4 7 9 2 5 6
- b. 4 9 7 6 5 2
- c. 4 2 7 5 6 9
- d. 7 2 4 5 9 6

14. Precizați care din afirmațiile următoare sunt adevărate:

- a) Complexitatea inserării unui nod într-o coadă alocată dinamic este $O(n)$;
- b) Algoritmul de sortare QuickSort este de complexitate mai mică decât BubbleSort;
- c) Un arbore BST cu $n > 2$ nu poate fi HEAP.

15. Fie o listă circulară simplu înlănțuită alocată dinamic unde pointerul l conține adresa primului element din listă iar pointerul x conține adresa unui element din lista l . Secvența

- | | |
|--|--|
| $y \leftarrow \text{succ}(x)$ | poate fi utilizată pentru: |
| $\text{succ}(x) \leftarrow \text{succ}(y)$ | a. ștergerea din listă a primului nod |
| $\text{free_sp}(y)$ | b. ștergerea din listă a nodului indicat de x |
| | c. ștergerea din listă a nodului ce urmează celui indicat de x |

16. Fie S o stivă ordonată în care ***S.vect*** este vectorul de dimensiune N în care se reprezintă elementele stivei ***S*** iar ***S.virf*** este indicele vârfului stivei. Atomii sunt de tip **Atom** iar stiva de tip **Stack**. Funcția **push**:

```

void push(Stack &S, Atom a)
{
  if (S.virf >= N)
    printf("\n Stiva plină");
  else
    S.vect[S.virf++] = a;
}

```

- a. Este corectă
- b. Prototip eronat
- c. Eroare pe ramura Else
- d. Eroare condiție if

17. Se consideră un arbore AVL. În urma efectuării operațiilor de echilibrare, ordinea afișării nodurilor arborelui este aceeași pentru:
- Parcurgerea în preordine
 - Parcurgerea în inordine
 - Parcurgerea în postordine
 - Toate tipurile de parcurgeri
18. Fie o listă simplu înlănțuită alocată dinamic, pointerul q conține adresa unui element din listă, iar p adresa unui element ce se dorește a fi inserat. Secvența următoare
- | | |
|-------------------|---|
| succ(p) ← succ(q) | poate fi utilizată pentru: |
| succ(q) ← p | a. Inserarea în listă a data(p) după data(q) |
| aux ← data(q) | b. Inserarea în listă a data(p) după succesorul nodului q |
| data(q) ← data(p) | c. Inserarea în listă a data(p) înainte de data(q) |
| data(p) ← aux | d. secvență eronată pentru scopul propus |
19. Ce rotații trebuie aplicate asupra arborelui binar de căutare pentru care s-au introdus elementele 4 2 8 6 9 7
- Rotație dublă dreapta
 - Rotație dublă stânga
 - Rotație simplă stânga
 - Rotație simplă dreapta
20. Asupra căror structuri poate fi folosită următoarea funcție de parcurgere în inordine?
- ```
void inordine(int i)
{
 if (i <= n && (A[i] != '-')) {
 inordine(i*2);
 prelucrare(A[i]);
 inordine(i*2+1);
 }
}
```
- Reprezentarea implicită a arborilor binari de căutare
  - Arbori heap
  - Nu poate fi folosită, fiind greșită.
21. Pentru două structuri de date predefinite tip coadă și stivă s-au implementat **numai** funcțiile pentru inserare, ștergere și testare dacă structura este goală. Se implementează o aplicație care numără elementele din structura A (stivă sau coadă) aflată în stare inițială utilizând o structură auxiliară B (de tip stivă sau coadă). Structurile A și B pot fi utilizate în orice combinație stivă-coadă, dar după numărare, structura A trebuie să se fie în starea inițială. Pentru care din combinațiile I-III de mai jos, numărarea elementelor din A este posibilă în condițiile enunțate mai sus:
- A este coadă, B este stivă
  - A este stivă, B este stivă
  - A este stivă, B este coadă
- a) Nici una    b) Numai II și III    c) Numai I și II    d) I, II și III    e) Numai II și III

22. Într-un arbore binar plin, toate nodurile interne au exact doi descendenți. Câte noduri interne sunt într-un arbore binar plin cu 128 frunze?

- a) 256                      b) 127                      c) 128                      d) 129                      e) 64

23. Ce fel de rotații pot fi implementate utilizând următorul cod:

```
Nod* Rot(Nod* p)
{
 Nod* aux;
 aux = p->stg;
 p->stg = aux->drt;
 aux->stg = p;
 p->fact = 0;
 aux->fact = 0;
 return aux;
}
```

- a. Rotație dublă dreapta  
b. Rotație simplă stânga  
c. Rotație simplă dreapta  
d. Codul nu va efectua nici o rotație, fiind greșit.

24. Complexitatea timp a algoritmului de generare a unui heap (fără a utiliza memorie suplimentară) pornind de la un vector cu  $n$  elemente poate fi

- a. Întotdeauna  $O(\log n)$   
b.  $O(n \log n)$  sau  $O(n)$   
c. Numai  $O(n)$   
d. Întotdeauna  $O(n \log n)$

25. Care din afirmațiile de mai jos sunt corecte?

- e) Dacă un arbore binar de căutare și un arbore B conțin același număr de chei, atunci cei doi arbori au adâncimi egale.  
f) Ordinul de complexitate pentru operația de inserare a unui nod într-o coadă liniară alocată dinamic este  $O(1)$ .  
g) Căutarea binară este de complexitate mai mică decât căutarea secvențială.  
h) Dacă un arbore B este de ordin 3, atunci orice pagină cuprinde cel mult 3 chei.

26. Se consideră funcția de dispersie  $h(x) = x \bmod m$  care asociază unui întreg  $x$  o intrare într-o tabelă de dimensiune  $m$ . Pentru care din următoarele valori ale lui  $m$  funcția  $h$  este o funcție de dispersie perfectă peste setul de chei  $\{0, 6, 9, 12, 22, 31\}$ ?

- a)  $m = 11$   
b)  $m = 4$   
c)  $m = 7$   
d)  $m = 13$

27. Precizați care din afirmațiile următoare sunt adevărate:

- a. Un arbore HEAP este sigur și AVL;  
b. Se consideră o listă alocată dinamic, identificată prin pointerul  $p$ . Operația de inserare a unui nod în listă, înaintea elementului pointat de  $p$  este de complexitate  $O(n)$ ;  
c. Complexitatea algoritmului de înmulțire a două matrici este  $O(n^2)$ ;  
d. Algoritmul QuickSort este de complexitate  $O(n \log(n))$ .

28. Fie un arbore binar de căutare cu structura **Nod**:

|                              |                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------|
| <b>Nod* f(Nod* &amp;nod)</b> | Funcția f implementează operația prin care:                                         |
| {Nod* p;                     | a. se șterge din arbore nodul de valoare minimă și întoarce pointer spre acesta;    |
| if(nod->drt==0)              | b. se caută nodul de valoare maximă și întoarce pointer spre acesta ;               |
| {p=nod;                      | c. se detașează din arbore nodul de valoare minimă și întoarce pointer spre acesta; |
| nod=nod->stg;                | d. se detașează din arbore nodul de valoare maximă și întoarce pointer spre acesta; |
| return p;                    | e. Nici una din variantele anterioare.                                              |
| }                            |                                                                                     |
| else                         |                                                                                     |
| return f(nod->drt);}         |                                                                                     |

29. Se consideră operația de ștergere a elementului de valoare maximă dintr-o listă liniară alocată dinamic (complexitate  $O_{\text{lista}}$ ), dintr-un arbore heap (complexitate  $O_{\text{heap}}$ ) și dintr-un arbore binar de căutare (complexitate  $O_{\text{BST}}$ ). Care din următoarele relații este corectă:

- $O_{\text{lista}} = O_{\text{BST}} = O_{\text{heap}}$
- $O_{\text{lista}} < O_{\text{BST}} < O_{\text{heap}}$
- $O_{\text{heap}} = O_{\text{BST}} < O_{\text{lista}}$
- $O_{\text{heap}} < O_{\text{BST}} < O_{\text{lista}}$

30. Fie T un arbore binar. Dacă notăm:

$n_2$  - numărul de noduri de grad 2 din arborele binar

T,

$n_1$  - numărul de noduri de grad 1 din arborele binar

T,

$n_0$  - numărul de noduri terminale (frunze) din arborele binar T,

care din următoarele afirmații este corectă?

- $n_0 = n_2 + 1$  .
- $n_2 = n_0 + 1$  .
- $n_1 = n_2 + 1$  .

31. Se consideră o tabelă de dispersie cu valori în  $\{0, 1, \dots, m-1\}$ . Tabela poate conține mai mult de  $m$  înregistrări atunci când

- rezolvarea coliziunilor se face prin adresare deschisă
- rezolvarea coliziunilor se face prin înlănțuire
- rezolvarea coliziunilor se face prin dispersie dublă
- funcția de dispersie realizează o dispersie perfectă

31. Operațiile care trebuie făcute în cazul unei rotații duble la dreapta într-un arbore AVL sunt următoarele:

- |                                         |   |                              |                              |
|-----------------------------------------|---|------------------------------|------------------------------|
| a. lchild(a)                            | = | b. b = lchild(a)             | c. b = lchild(a)             |
| rchild(c)                               |   | c = rchild(b)                | c = rchild(b)                |
| b = lchild(a)                           |   | lchild(a) = rchild(c)        | rchild(b) = lchild(c)        |
| rchild(b) = lchild(c)                   |   | rchild(b) = lchild(c)        | rchild(c) = a                |
| c = rchild(b)                           |   | rchild(c) = a                | lchild(c) = b                |
| rchild(c) = a                           |   | lchild(c) = b                | a = c // se schimbă rădăcina |
| lchild(c) = b                           |   | a = c // se schimbă rădăcina | arborelui                    |
| a = c // se schimbă rădăcina arborelui. |   |                              |                              |

32. Într-un graf complet cu 6 noduri sunt maxim:

- 360 cicluri hamiltoniene;
- 720 cicluri hamiltoniene;
- 12 cicluri hamiltoniene;
- 120 cicluri hamiltoniene.

33. Fie  $f : \mathbb{N} \rightarrow \mathbb{N}$  și  $g : \mathbb{N} \rightarrow \mathbb{N}$  două funcții. Care din definițiile următoare este corectă:

- Spunem că  $f \in O(g)$  și notăm  $f = O(g)$  dacă și numai dacă  $\exists c \in \mathbb{R}$  și  $\forall n_0 \in \mathbb{N}$  astfel încât pentru  $\forall n > n_0 : f(n) < c \cdot g(n)$
- Spunem că  $f \in O(g)$  și notăm  $f = O(g)$  dacă și numai dacă pentru  $\forall c \in \mathbb{R}$  și  $\forall n_0 \in \mathbb{N}$ ,  $\exists n \in \mathbb{N}$  astfel încât  $f(n) < c \cdot g(n)$
- Spunem că  $f \in O(g)$  și notăm  $f = O(g)$  dacă și numai dacă  $\exists c \in \mathbb{R}$  și  $\exists n_0 \in \mathbb{N}$  astfel încât pentru  $\forall n > n_0 : f(n) < c \cdot g(n)$

34. Fie  $S$  o stivă înlănțuită și  $f, g, h$  implementări ale celor trei operații de bază (push, pop, top):

| <b>f(S,a)</b>                  | <b>g(S)</b>                     | <b>h(S)</b>                   |
|--------------------------------|---------------------------------|-------------------------------|
| $p \leftarrow \text{get\_sp};$ | <b>if</b> $S=0$ <b>then</b>     | <b>if</b> $S=0$ <b>then</b>   |
| $\text{data}(p) \leftarrow a;$ | eroare("Stiva vida")            | eroare("Stiva vida")          |
| $\text{succ}(p) \leftarrow S$  | <b>else</b>                     | <b>else</b>                   |
| $S \leftarrow p$               | $\text{return}(\text{data}(S))$ | $p \leftarrow S;$             |
| <b>Endf</b>                    | <b>endif</b>                    | $S \leftarrow \text{succ}(S)$ |
|                                | <b>endg</b>                     | $\text{Free\_sp}(p)$          |
|                                |                                 | <b>endif</b>                  |
|                                |                                 | <b>end</b>                    |

Funcțiile  $f, g, h$  implementează respectiv operațiile

- pop, top, push
- top, push, pop
- push, top, pop

35. Dată fiind  $Q$  o coadă circulară alocată static,  $Q.\text{vect}$  este vectorul ce conține elementele cozii,  $Q.\text{head}$  și  $Q.\text{tail}$  reprezintă indexul primului element respectiv indexul primului loc liber în care se poate insera un element și funcția următoare:

```

put(Q,a)
 if
 $Q.\text{tail} = \text{Nextpoz}(Q.\text{head})$
 then
 print "Coadă plină"
 elseif
 $Q.\text{vect}[Q.\text{tail}] \leftarrow a$
 $Q.\text{tail} \leftarrow \text{Nextpoz}(Q.\text{tail})$
 endif
end

```

Unde lipsește ceva pentru ca funcția **put** să lucreze corect în orice situație:

- Pe ramura **then**
- Pe ramura **elseif**
- Testul de la **if**

36. Fie  $G=(V,E)$  un graf reprezentat prin matricea de adiacență  $A$ :

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

- a) Graful  $G$  este arbore      b) Graful  $G$  nu este arbore      c) Graful este neorientat  
d) Graful este conex      e) Graful este complet

37. Funcția **căutare-nod** ar trebui să implementeze operația de căutare a unui nod cu cheia  $k$  într-un arbore binar de căutare. Funcția este incompletă:

```
Căutare_nod(rădăcină,k) // rădăcină — pointer la rădăcină arborelui
if (rădăcină = 0) then // primul if
 return NULL
else
 if key(data(rădăcină)) > k then // al doilea if
 return căutare_nod (fîu_stînga(rădăcină))
 else
 if key(data(rădăcină)) < k then // al treilea if
 return căutare_nod (fîu_dreapta(rădăcină))
 endif
 endif
endif
end{Căutare_nod}
```

Unde lipsește ceva:

- pe ramura **then** a primului **if**
- pe ramura **then** a celui de-al doilea **if**
- pe ramura **else** a celui de-al treilea **if** (lipsește dar ar trebui să fie prezentă și să conțină ceva)

38. Fie  $T=(V,E)$  un arbore heap incomplet eventual doar pe ultimul nivel și în care valoarea de nod este chiar cheia acestuia. Care este varianta corectă pentru operația de **inserare a unui nod în heap-ul  $T$**  ?

- Se inserează nodul pe nivelul incomplet;**  
**Se reorganizează structura arborelui** : se detașează din structura arborelui rădăcina, se interschimba nodul detașat cu nodul de pe nivelul incomplet și apoi valoarea rădăcinii este retrogradată pîna cînd structura heap-ului este realizată.
- Nodul se inserează pe nivelul incomplet, după ultima frunză;**  
**Se reorganizează structura arborelui** : se promovează nodul inserat de la stînga la dreapta, prin interschimbare cu părintele acestuia pînă cînd structura heap-ului este realizată.
- Se inserează nodul pe nivelul incomplet, după ultima frunză;**  
**Se reorganizează structura arborelui** : se interschimbă nodul cu rădăcina, după care valoarea rădăcinii este retrogradată pîna cînd structura heap-ului este realizată.



39. Fie matricea de adiacență A:

|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |

Această matrice corespunde:

- a. unui graf    b. unui digraf    c. unui graf conex    d. unui graf ce nu este arbore

40. Se consideră un arbore AVL. În urma efectuării operațiilor de echilibrare, ordinea afișării nodurilor arborelui este aceeași pentru:

- a) Parcurgerea în preordine    b) Parcurgerea în inordine    c) Parcurgerea în postordine    d) Toate tipurile de parcurgeri

41. Ce tip de rotație trebuie aplicată pentru un arbore AVL echilibrat dacă s-au introdus elementele {4 2 8 6 9 7} exact în ordinea dată?

- a) Rotație dublă dreapta    b) Rotație dublă stânga    c) Rotație simplă stânga    d) Rotație simplă dreapta

42. Complexitatea timp, în cazul cel mai defavorabil, pentru căutarea nodului de cheie minimă într-un arbore binar de căutare este:

- a.  $O(n^2)$   
b.  $O(n)$   
c.  $O(n \log n)$   
d.  $O(\log n)$

43. Complexitatea operațiilor de inserare și ștergere într-un arbore heap cu  $n$  noduri este:

- a)  $O(n \log n)$   
b)  $O(n)$   
c)  $O(\log n)$   
d)  $O(n^2)$

44. Fie  $T=(V,E)$  un arbore binar cu reprezentarea implicită:

|    |    |    |    |    |    |    |   |    |    |    |    |    |    |
|----|----|----|----|----|----|----|---|----|----|----|----|----|----|
| 40 | 25 | 70 | 10 | 30 | 60 | 80 | 5 | 20 | 28 | 35 | 50 | 65 | 75 |
|----|----|----|----|----|----|----|---|----|----|----|----|----|----|

Arborele T este :

- a. Arbore binar de căutare  
b. Arbore heap  
c. Arbore binar complet

45. Nodurile unui arbore binar de căutare pot fi afișate în ordinea crescătoare a cheilor prin:

- a) Parcurgere pe linie  
b) Parcurgere în inordine  
c) Parcurgere în preordine  
d) Parcurgere în postordine