Structuri de date - Test restanțe / măriri

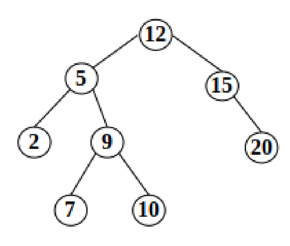
Întrebări

Considerăm procedura de mai jos. Care din următoarele afirmații sunt adevărate?

```
procedure sort(a, n)
begin
    for i ← 0 to n-1 do
        for j ← n-1 downto i do
            if a[j] < a[j-1] then
                swap(a[j], a[j-1])
end
```

- [ ] Algoritmul de sortare prin inserție necesită mai multe comparații decât algoritmul sort() în cazul cel mai favorabil.
- [ ] Complexitatea timp a algoritmului sort() de mai sus în cazul cel mai favorabil este O(log n).
- [x] La finalul fiecărei iterări a lui i, valorile a[0], ...a[i] sunt ordonate crescător.
- [ ] La finalul fiecărei iterări a lui i, valorile a[n-1-i], ...a[n-1] sunt ordonate crescător.
- [ ] Algoritmul de sortare este stabil.

Considerați următorul arbore AVL echilibrat. Ștergeți valoarea 15. Care din următoarele afirmații sunt adevărate?



- [ ] Nu sunt necesare operații de rotație pentru a restabili proprietatea AVL.
- [x] Parcurgerea pe nivele a arborelui final este 9, 5, 12, 2, 7, 10, 20.
- [ ] Factorul de echilibare al nodului 5 înainte de ștergere este 1.
- [x] Este necesară o operație de rotație dublă dreapta pentru a restabili proprietatea AVL.
- [ ] Este necesară o operație de rotație simplă la dreapta pentru a restabili proprietatea AVL.

Considerăm următorul vector [8, 5, 7, 9, 2, 10, 1, 4, 6]. Aplicați etapa 1 (construcția MaxHeap-ului) din cadrul algoritmului de sortare prin selecție sistematică. Care din următoarele afirmații sunt adevărate?

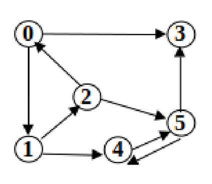
- [ ] După aceasta etapă, vectorul este [10, 5, 7, 9, 2, 8, 1, 4, 6].
- [ ] În vectorul inițial, elementele de la poziția 3 încolo satisfac proprietatea de MaxHeap.
- [ ] După aceasta etapă, valoarea maximă se află pe prima poziție în vector.
- [ ] Complexitatea algoritmului de sortare prin selecție sistematică este O(n*n) în cazul cel mai nefavorabil.

Fie următoarea secvență de pseudocod în care S este o stivă nevidă, iar C este o coadă. Care dintre următoarele afirmații sunt adevărate?

```
C <- coadaVida()
while (not esteVida(S)) do {
      aux <- top(S); pop(S); insereaza(C, aux)
}
while (not esteVida(C)) do {
      aux <- citeste(C); push(S, aux)
}
```

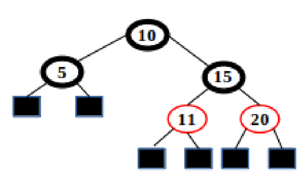
- [ ] Algoritmul rulează la nesfârșit.
- [ ] Algoritmul lasă stiva S nemodificată.
- [x] Algoritmul inversează ordinea elementelor în stiva S.
- [ ] După execuția algoritmului, stiva S este vidă.

Fie digraful D=(V,A) de mai jos, reprezentat prin liste de adiacență (nodurile sunt reprezentate în ordine crescătoare în cadrul listelor de adiacență).



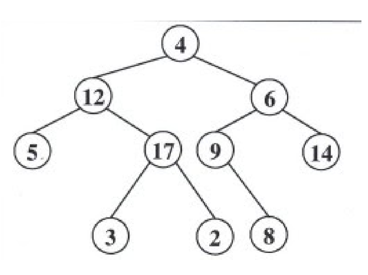
- [ ] Parcurgerea BFS din nodul 0 este 0, 1, 3, 2, 5, 4.
- [x] Componentele tare conexe sunt {0,1,2}, {3}, {4,5}.
- [x] Parcurgerea în adâncime din nodul 0 este 0,1,2,5,3,4.
- [ ] Digraful este tare conex.
- [ ] Complexitatea timp a algoritmului de parcurgere în adâncime este O(|V|*|V|).

Fie următorul arbore bicolor. Inserăm valoarea 12. Care din următoarele afirmații sunt adevărate?



- [x] Nodul inserat se coloreaza în roșu.
- [x] Nodurile 11 și 20 sunt recolorate în negru.
- [ ] Arborele care rezultă după inserare e un arbore bicolor valid și nu trebuie aplicată nici o operație.
- [ ] Pentru a restabili proprietățile, este necesară o operație de rotație și recolorări.
- [ ] Înălțimea neagră a nodului rădăcină după inserare este 3.

Se consideră arborele binar din figura de mai jos. Care din următoarele afirmații sunt false?



- [x] Parcugerea inordine a arborelui este: 2, 3, 4, 5, 6, 8, 9, 12, 14, 17.
- [x] Frontiera arborelui este formată din 3 vârfuri.
- [ ] Ultimile două vârfuri din parcugerea BFS a arborelui sunt 2 și 8.
- [ ] Primele două vârfuri din parcugerea postordine a arborelui sunt 5 și 3.
- [x] Ultimile două vârfuri din parcugerea preordine a arborelui sunt 6 și 14.

Fie t adresa rădăcinii unui arbore binar implementat cu structuri înlănțuite și care conține în noduri valori întregi distincte din intervalul [0, 1000]. Ce calculează apelul test(t, 0, 1000), unde funcția test este descrisă mai jos?

```
function test(t, l, r)
begin
    if t != NULL then
        if t→inf < l or t→inf > r then
            return 0
        else return test(t→stg, l, t→inf–1) and test(t→dr, t→inf +1, r)
    else return 1
end
```

- [ ] Returnează numărul de elementele din arbore care se găsesc într-un interval.
- [ ] Funcția nu este corectă.
- [ ] Verifică dacă un arbore binar este AVL echilibrat.
- [x] Verifică dacă un arbore binar este arbore binar de căutare.
- [ ] Returnează înălțimea unui arbore binar de căutare.

Se consideră subprogramul de mai jos unde x și n sunt doi parametri întregi și pozitivi. În care din următoarele clase poate fi încadrată complexitatea timp a acestui subprogram?

```
function f(x, n)
begin
   if (n == 0) then return 1
   else return x * f(x, n-1)
end
```

- [ ] Ω(n)
- [ ] Θ(x!)
- [x] O(n)
- [x] O(2^n)
- [ ] Θ(2^n)

Fie următorul MaxHeap implementat cu tablouri: (65, 47, 23, 13, 1, 5, 20). După o operație de inserare a cheii 58, urmată de o operație de eliminare, tabloul asociat MaxHeap-ului va conține:

- [ ] (58, 23, 47, 20, 5, 1, 13)
- [ ] (58, 47, 23, 20, 13, 5, 1)
- [x] (58, 47, 23, 13, 1, 5, 20)
- [ ] Niciuna, deoarece tabloul inițial nu satisface proprietatea de MaxHeap.
- [ ] (65, 58, 23, 47, 1, 5, 13)

Se consideră funcția f de mai jos, unde n este un parametru întreg și pozitiv. Care dintre următoarele afirmații sunt adevărate?

```
function f(n)
begin
   x <- 0;  i <- n
   while (i > 1) do {
      for j <- 1 to n do x <- x + 1
      i <- i / 2
   }
   return x
end
```

- [ ] Complexitatea algoritmului poate fi încadrată în clasa O(n).
- [x] Apelul f(9) returnează valoarea 27.
- [ ] Apelul f(4) returnează valoarea 16.
- [ ] Apelul f(9) returnează valoarea 24.
- [x] Complexitatea algoritmului poate fi încadrată în clasa Θ(n log n).

Fie următorul algoritm pentru inserarea unui element e într-o listă liniară L implementată cu structuri înlănțuite. Vom considera poziția k o valoare între 0 și n-1 (n fiind numărul de elemente din listă), iar lista nevidă. Care din următoarele afirmații sunt adevărate?

```
procedure insereaza(L, k, e)
begin
   new(q); q->elt <- e
   if k == 0 then { q->succ <- L.prim;  L.prim <- q }
   else {
      p <- L.prim; j <- 0
      while (j < k-1 and p != L.ultim ) do
         {p <- p->succ; j <- j+1}
      q->succ <- p->succ; p->succ <- q
      if (p == L.ultim) then L.ultim <- q
   }
end
```

- [ ] Algoritmul este corect și inserează elementul k pe poziția e.
- [x] Algoritmul este corect și inserează elementul e pe poziția k.
- [ ] Algoritmul este greșit deoarece inserează elementul e pe poziția k-1.
- [ ] Algoritmul este greșit deoarece inserează elementul e pe poziția k+1.
- [ ] Algoritmul este greșit deoarece se pierd o parte din elementele listei.

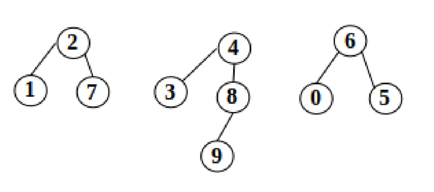
Timpul de execuție al unui algoritm recursiv se exprimă prin relația de recurență T(n) = 3T (n/2) + n^2 . În care din următoarele clase de complexitate poate fi incadrată soluția acestei recurențe? (Toți logaritmii de mai jos sunt în baza 2.)

- [ ] T(n) = Θ(n^(log 3) log n)
- [ ] T(n) = Θ(n^(log 3))
- [ ] T(n) = Θ(n^2 log n)
- [ ] T(n) = Θ(log n)
- [x] T(n) = Θ(n^2)

Fie o tabelă de dispersie externă de dimensiune 7 și h(k) = k mod 7. Inserați în ordine valorile 18, 13, 25, 17, 20.

- [x] Valorile 18 și 25 sunt inserate în lista asociată indicelui 4.
- [ ] Avem coliziune atunci când cheile sunt egale iar funcția de dispersie returnează valori diferite.
- [ ] Factorul de încărcare după ce au fost inserate valorile este egal cu 1.
- [ ] Numărul maxim de elemente care pot fi inserate în tabela de dispersie de mai sus este egal cu 7.
- [ ] Complexitatea timp în cazul cel mai nefavorabil pentru a insera n valori într-o tabelă de dispersie externă inițial vidă, unde elementele în cadrul listelor sunt păstrate în ordine crescătoare, este O(n*n).

Fie structura union-find ponderată de mai jos. După apelul funcției union(9,1), avem:



- [ ] Vectorul de părinți este [6, 2, 4, 4, -1, 6, -3, 2, 4, 4].
- [ ] Obținem două colecții cu 7, respectiv 3 elemente disjuncte.
- [ ] Părintele lui 9 este 2.
- [ ] 1 este adăugat ca fiu al lui 9.

Înapoi | Trimiteți