Probleme pentru acasă

Acest fișier va fi actualizat în fiecare zi de vineri cu probleme utile pentru laboratorul de săptămâna următoare.
Rezolvarea acestor probleme este opțională și nu se punctează.

Liste

1. Atunci când un sens de mers cu două benzi se îngustează într-o singură bandă, traficul cel mai fluid se obține prin metoda fermoarului: mașinile înaintează alternativ de pe cele două benzi pe singura rămasă. Implementați o funcție care interclasează două liste cu elemente de tip `int` ce pot avea dimensiuni diferite.
2. Scrieți o funcție care interclasează două liste cu elemente de tip `int`. Fiecare listă are elementele în ordine crescătoare. Lista rezultată trebuie să fie de asemenea ordonată și nu trebuie să conțină duplicate. Soluția nu trebuie să folosească memorie suplimentară față de cele două liste.
3. Creați o funcție care șterge dintr-o listă al k-lea element începând cu sfârșitul acesteia. Complexitatea soluției trebuie să fie $O(n)$.
4. Implementați o listă dublu înlănțuită cu santinelă la final.
5. Transformați o listă dublu înlănțuită cu santinelă la final într-o listă circulară.

Stive. Cozi

1. Concatenați două stive implementate cu listă. Care este complexitatea soluției?
2. Implementați o stivă folosind două cozi.
3. Implementați o coadă folosind două stive.
4. Implementați o coadă folosind o listă dublu înlănuțuită circulară cu santinelă.
5. Inversați cuvintele dintr-o propoziție folosind o stivă.
6. Inversați ordinea elementelor dintr-o stivă folosind doar operațiile `push` și `pop`.
7. Scrieți o funcție care calculează lungimea celui mai lung subșir de numere crescătoare consecutive dintr-un șir de numere întregi, folosind o stivă.
8. Creați un algoritm care convertește o expresie în notație infix (notația standard a expresiilor matematice) în notația postfix (denumită și Reverse Polish Notation, în care operatorul apare după operanzi).

Recursivitate

1. Implementați recursiv algoritmii buble sort, insertion sort și selection sort.
2. Implementați recursiv ștergerea unei listei.

Arbori binari

1. Eliminați spațiul ocupat de un arbore binar.
   - **Recursiv**. Se traversează în postordine și se elibrează spațiul pentru nodurile din stânga și dreapta înainte de a elibera spațiul ocupat de rădăcină.
   - **Iterativ**. Se parcurge arborele pe niveluri. Se șterge un nod din coadă după ce au fost adăugați în coadă copiii acelui nod. Se eliberează întâi nodurile cele mai de sus, ținând minte adresele copiilor.
2. Ștergeți un subarbore dintr-un arbore. Rădăcina subarborelui este dată după adresă sau după valoare (caz în care considerați că valorile stocate în arbore sunt diferite).
3. Verificați dacă un nod (dat prin valoarea stocată) se găsește într-un arbore binar. Numărați de câte ori apare stocată aceeași valoare.
4. Găsiți cel mai apropiat strămoș comun (Lowest Common Ancestor – LCA).
   LCA a două noduri a și b este cel mai adânc (de jos) nod care îi are pe a și b ca descendenți.
   - **Recursiv**. Trebuie găsit nodul din arbore care îl are pe a într-un subarbore și b în celălalt subarbore. Se folosește parcurgerea în postordine.
   - **Iterativ**. Se salvează căile de la rădăcină la a și de la rădăcină la b în 2 cozi. Se parcurg cele 2 cozi până când se găsesc 2 valori diferite. Ultima valoare egală este cea a LCA.
5. Verificați dacă un arbore binar este simetric (ca dispunere a nodurilor, nu dpdv al valorilor). Arborele este simetric dacă subarborii stâng și drept sunt imagini în oglindă, ca în Figura 1:
   - Amândoi arborii există sau nu
   - Subarborele stâng arată ca cel drept în oglindă
   - Subarborele drept arată ca cel stâng în oglindă

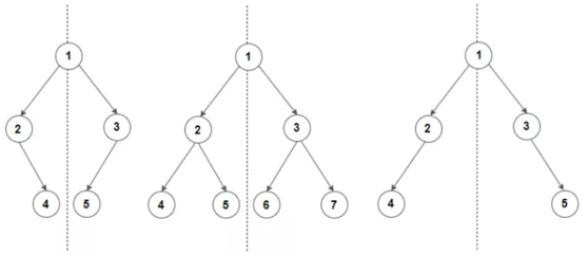

Figura 1: Arbori oglindă

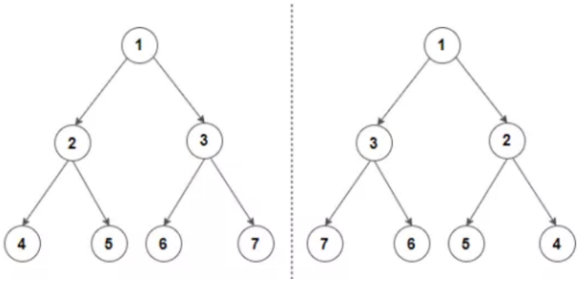

Figura 2: Inversare arbore

6. Convertiți un arbore binar în versiunea sa în oglindă.
   Parcurgeți arborele în postordine (SDR) și pentru fiecare nod inversați copiii, ca în Figura 2.

7. Transformați un arbore în arborele sumă echivalent.
   În arborele sumă, valoarea fiecărui element care nu este frunză este suma elementelor din subarborii drept și stâng, ca în Figura 3.
   Parcurgeți arborele în postordine. Pentru fiecare nod care nu este frunză rețineți vechea valoare și stocați în locul ei suma elementelor din subarborii drept și stâng. Returnați suma subarborilor + valoarea veche.

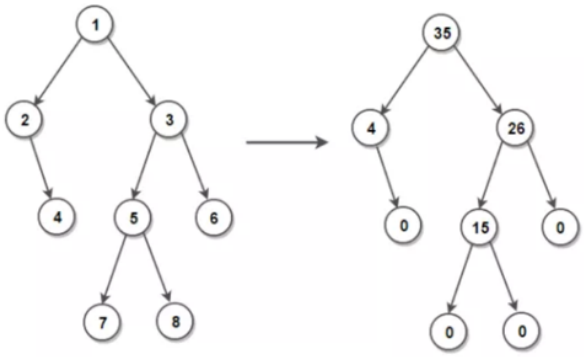

Figura 3: Arbore sumă

8. Verificați dacă un arbore este arbore sumă sau nu.

Arbori binari de căutare

1. Scrieți o funcție care verifică dacă un arbore este binar de căutare sau nu.

2. Scrieți o funcție care verifică dacă doi BST conțin aceleași valori.

3. Presupunând că într-un arbore binar de căutare nodurile nu stochează și informația despre înălțime, implementați o funcție eficientă care verifică dacă arborele este echilibrat.

4. Verificați dacă într-un arbore binar de căutare (sau AVL) există o cale de la rădăcină la frunze care are suma nodurilor egală cu o valoare s dată.

5. Dându-se un arbore binar de căutare, scrieți o funcție care calculează cea mai mică diferență (în modul) dintre oricare două noduri ale arborelui.
6. Scrieți o funcție care primește un arbore binar de căutare și o valoare. Funcția returnează cel mai mare număr din arbore care este mai mic decât valoarea primită (predecesorul).
   Scrieți o funcție care returnează succesorul.
7. Convertiți un arbore binar de căutare într-o listă dublu înlănțuită ordonată.
8. Ștergeți dintr-un arbore binar de căutare elementele care sunt mai mici decât o valoare dată, x_{min}, respectiv mai mari decât o valoare dată, x_{max}.
9. Scrieți o funcție care afișează toate valorile dintr-un arbore binar de căutare cuprinse într-un interval citit de la tastatură.
10. Găsiți toate tripletele de noduri dintr-un arbore binar de căutare a căror sumă este egală cu o valoare dată, s.

Grafuri

1. Implementați parcurgerea în adâncime nerecursiv.
2. Implementați parcurgerea în lățime utilizând reprezentarea grafurilor cu liste de adiacență.
3. Scrieți o funcție care găsește cel mai scurt drum între două vârfuri dintr-un graf.
4. Scrieți un program care determină dacă un graf orientat este tare conex, anume dacă există drum între oricare două vârfuri, de exemplu de la A la B și de la B la A.
5. Scrieți o funcție care verifică dacă un graf este bipartit. Soluția se poate baza pe următoarea idee: atunci când se întâlnește un vârf care nu a mai fost vizitat, acesta se colorează cu o culoare opusă celei ale părintelui. Dacă arborele este bipartit, fiecare muchie va lega două vârfuri de culori diferite.
6. Scrieți o funcție care verifică dacă un graf orientat are cicluri.
7. Scrieți o funcție care testează dacă există muchie între nodurile x și y ale unui graf neorientat reprezentat prin liste de adiacență.
8. Scrieți o funcție care calculează gradul de intrare al unui nod x dintr-un graf orientat reprezentat prin liste de adiacență.
9. Implementați algoritmul lui Prim pentru crearea arborelui minim de acoperire.

Algoritmi greedy

1. Scrieți un program are rezolvă problema poștașului (curs 11, slide 5).
2. Rezolvați problema fracționară a rucsacului (curs 11, slide 8).
3. Scrieți soluția pentru problema spectacolelor (curs 11, slide 9).
4. Rezolvați problema Cel mai mare număr.
5. Rezolvați problema Permutări condiționate.
6. Rezolvați problema Minimax.