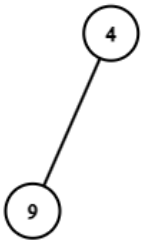
Subiectul 1

Partea 1

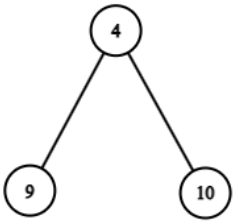
1. **Într-un min-heap faceți operațiile I(9), I(4), I(10), I(2), delete min, I(17), I(3), I(19), I(26) delete min, delete min. Arată arborele după fiecare operație.**

I(9)

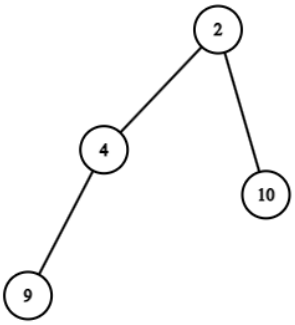
I(4)



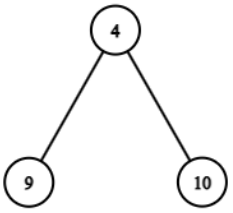
I(10)



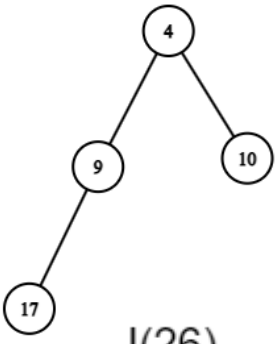
I(2)



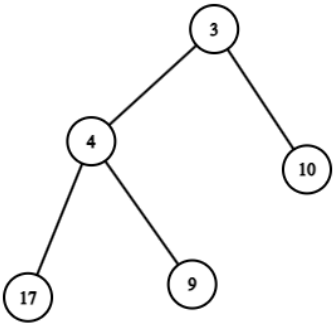
delete min



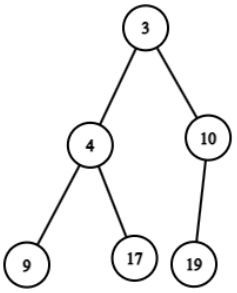
I(17)



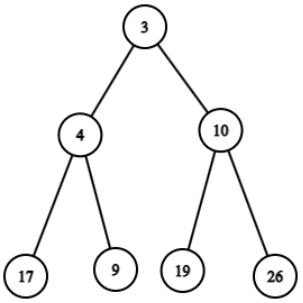
I(3)



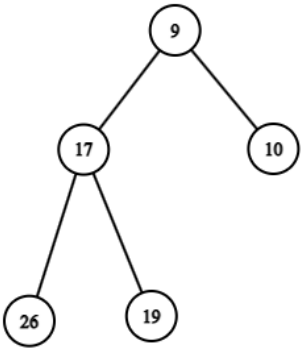
I(19)



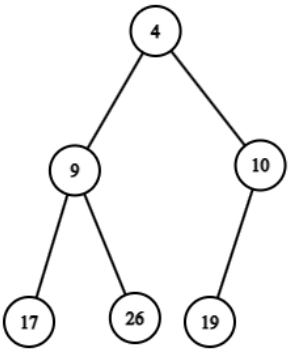
I(26)



delete min

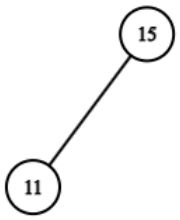


delete min

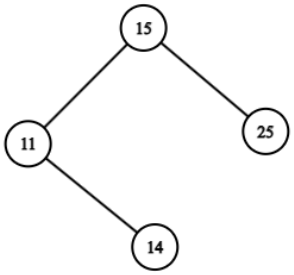


2. **Într-un arbore binar de căutare faceți operațiile I(15), I(11), I(14), I(25), del (11), I(7), I(11), I(9), I(5), del(16), I(8), I(10), I(6), del(7). Aratati arborele după fiecare 2 operații.**

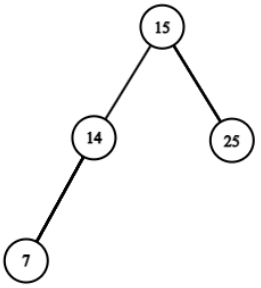
I(15), I(11)



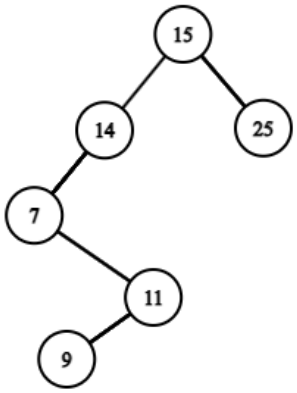
I(14), I(25)



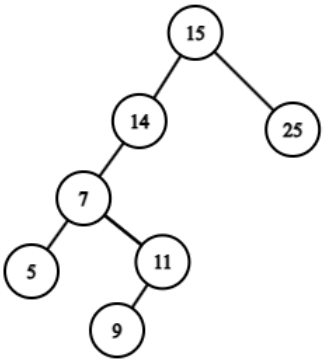
del (11), I(7)



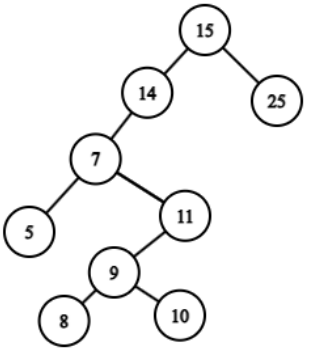
I(11), I(9)



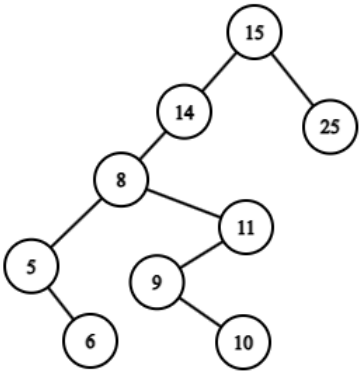
I(5), del(16)



I(8), I(10)



I(6), del(7)



3. **Care dintre următoarele NU este o caracteristică a unei funcții hash bune?**
    - **Răspuns corect: c) Rată ridicată de coliziuni**
    - **Explicație:** O funcție hash bună minimizează coliziunile, distribuie uniform valorile hash, are cost computațional redus și produce rezultate deterministe.

Partea 2:

4. Care dintre următoarele secvențe de operații este invalidă într-o stivă care are inițial trei elemente?

- **Răspuns corect: c), f)**
- **Explicație:** Într-o stivă, operațiile POP nu pot fi efectuate dacă stiva este goală, ceea ce face ca c) și f) să fie secvențe invalide.

5. Care dintre următoarele afirmații sunt adevărate despre ștergerea unui nod dintr-un arbore binar de căutare (BST)?

- **Răspuns corect: a), b), c), d)**
- **De ce nu sunt corecte celelalte opțiuni:**
  - e) Ștergerea rădăcinii unui BST este imposibilă. Explicație: Această afirmație este falsă, deoarece ștergerea rădăcinii unui BST este posibilă și implică rearanjarea arborelui pentru a menține proprietățile BST.

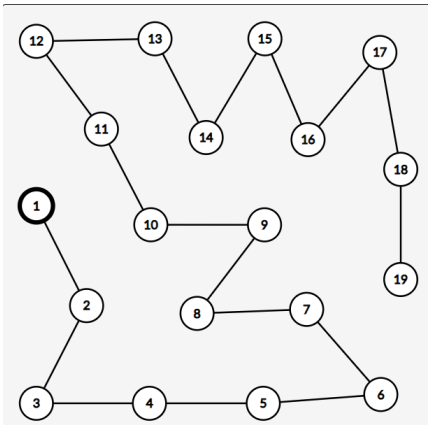
6. Care din următoarele structuri de date permit operații de Insert, Search, și Delete în O(log n)?

- **Răspuns corect:a), b), e)**
- **De ce nu sunt corecte celelalte opțiuni:**
  - c) Fibonacci Heap: Permite operații în timp amortizat O(log n) doar pentru unele operații, nu pentru toate trei.
  - d) Binary Heap: Are operații de căutare ineficiente (O(n)).
  - f) Deque: Operațiile de căutare și ștergere sunt ineficiente (O(n)).

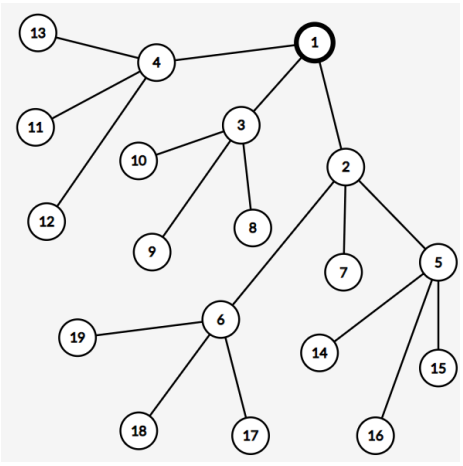
7. Un arbore ternar cu 19 noduri poate avea înălțimea ? (Considerăm ca un arbore cu 1 nod are înălțimea 0)

- **Răspuns corect: c), d), e), f), g), h)**
- **De ce nu sunt corecte celelalte opțiuni:**
- a), b): Aceste valori sunt prea mici pentru a găzdui 19 noduri într-un arbore ternar care poate avea noduri pe mai multe niveluri.

arborele cu adancime 18



arborele cu adancime 3



Orice arboere cu adancime intre cele doua numere se poate obtine.

8. Dacă vrem sa sortam 10^7 numere naturale mai mici decât 10^6 ce algoritm de sortare ar fi bine sa folosim ?

- **Răspuns corect: d) Counting Sort** (are complexitate O(n+max), n=10^7, max=10^6)
- **De ce nu sunt corecte celelalte opțiuni:**
  - a) Radix Sort (baza 2^6): Mai lent din cauza numărului mai mare de pași. Complexitate O(log(baza 2^6)(max)*(n+*baza*))
  - b) Quick Sort: În cel mai rău caz O(n^2), deși în medie O(n log n).
  - c) Merge Sort: O(n log n) și necesită spațiu suplimentar.
  - e) Timsort: O(n log n) și necesită spațiu suplimentar.
  - f) Radix Sort (baza 2^16): Ineficient pentru intervale mari de valori.Complexitate O(log(baza 2^16)(max)*(n+*baza(2)*))

9. Să presupunem că avem numerele între 1 și 1000 inserate într-un arbore binar de căutare și că dorim să căutăm valoarea 363. Care dintre următoarele secvențe nu ar putea fi secventa de noduri examinate?

- **Răspuns corect: c), e)**
- **Explicație:** Aceste secvențe nu respectă proprietatea de ordine a unui arbore binar de căutare pentru valoarea 363.
- c) numarul trebuie sa fie mai mic decat 911, dar la un moment dat se intalneste 912
- e) numarul trebuie sa fie mai mare decat 347 dar mai mic decat 299

10. În ce complexitate putem construi cât mai eficient un heap dintr-un vector de n elemente?

- **Răspuns corect: a) O(n)**
- **Explicație:** Construirea unui heap dintr-un vector de n elemente se face în O(n).

11. Care dintre următoarele afirmații sunt adevărate despre un heap binar?

- **Răspuns corect: a), d)**
- **De ce nu sunt corecte celelalte opțiuni:**
  - b) Poate avea mai mult de un nivel incomplet: Incorect, deoarece un heap binar complet poate avea doar ultimul nivel incomplet.
  - c) Poate avea mai multe rădăcini la un moment dat: Incorect, un heap binar are o singură rădăcină.
  - e) Suportă aceleași operații pe care le suportă un arbore binar echilibrat în aceeași complexitate: Incorect, deoarece operațiile pe un heap și un arbore binar echilibrat au complexități diferite si heapul nu suporta search sau succesor decat in O(n).

12. Fie H un max-heap care conține 80 valori distincte. În câte poziții diferite se poate afla elementul minim?

- **Răspuns corect: b) 40**

- **Explicație:** Elementul minim dintr-un max-heap poate fi în 40 de poziții diferite, adică la nivelurile inferioare ale heap-ului.

13. Care este limita inferioară a complexității în timp (în cel mai rău caz) pentru orice algoritm de sortare care folosește doar comparații între elemente, pentru a sorta un șir de n elemente?

- **Răspuns corect:c) Ω(n log n)**
- **Explicație:** Limita inferioară a complexității în timp pentru orice algoritm de sortare care folosește doar comparații este Ω(n log n).

14. Diferența de înălțime dintre două frunze într-un max-heap poate fi?

- **Răspuns corect:b), d)**
- **Explicație:** Diferența de înălțime dintre două frunze într-un max-heap poate fi 1 sau 0.

15. Care dintre următoarele afirmații sunt adevărate despre stive și cozi?

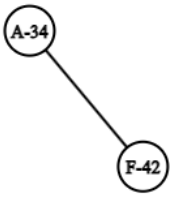
- **Răspuns corect: a), d), e)**
- **De ce nu sunt corecte celelalte opțiuni:**
  - b) Ambele respectă principiul LIFO (Last In, First Out): Incorect, deoarece stiva respectă principiul LIFO, dar coada nu.
  - c) Ambele respectă principiul FIFO (First In, First Out): Incorect, deoarece coada respectă principiul FIFO, dar stiva nu.

Partea 3

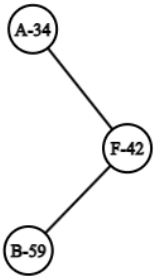
16. Inserati următoarele chei și priorități într-un treap (cu heap de minim): (A, 34), (F, 42), (B, 59), C(87), (J, 77), (L, 10). Aratati arborele după fiecare inserare.

(A, 34)

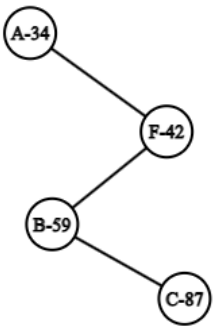
(F, 42)



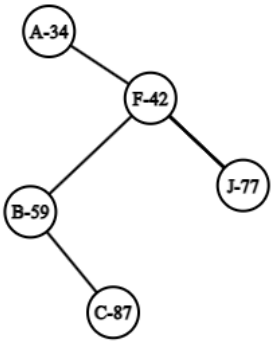
(B, 59)



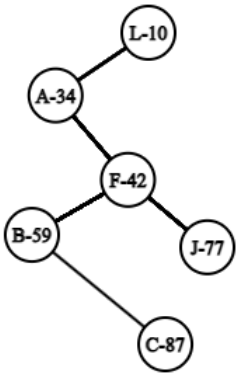
(C,87)



(J, 77)



(L, 10)



17. Construiți sparse table-ul (matricea din algoritmul RMQ) pentru șirul: 4 1 8 2 5 9 13 10. Presupunem ca în cadrul unui query am vrea sa determinam minimul pe un interval.

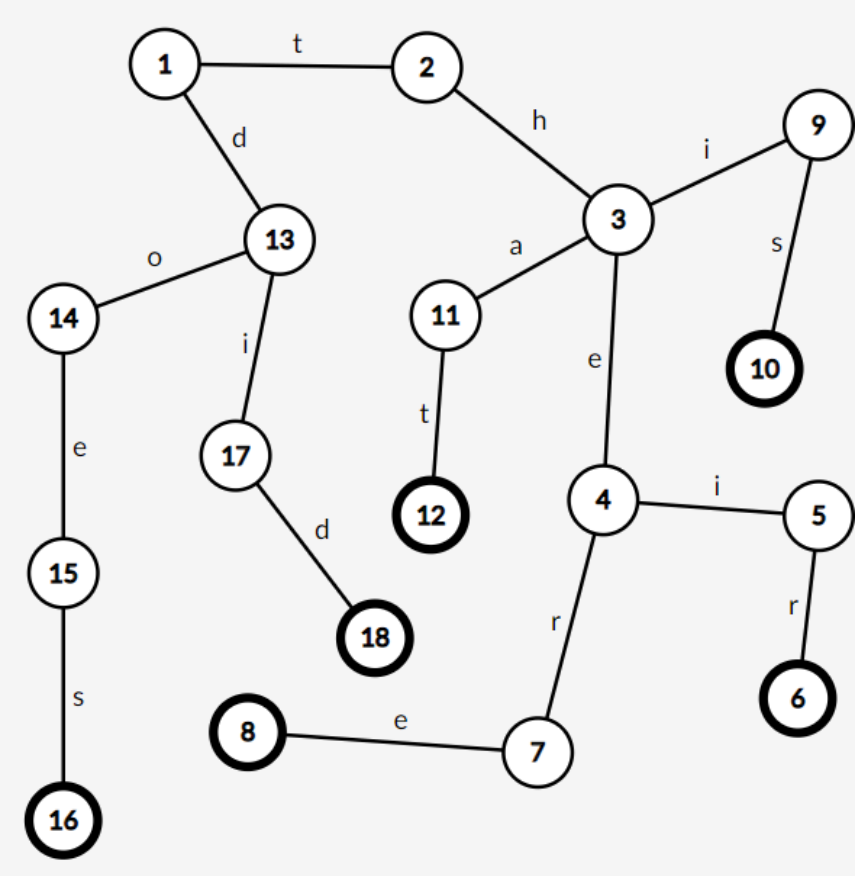
4 1 8 2 5 9 13 10

1 1 2 2 5 9 10

1 1 2 2 5

1

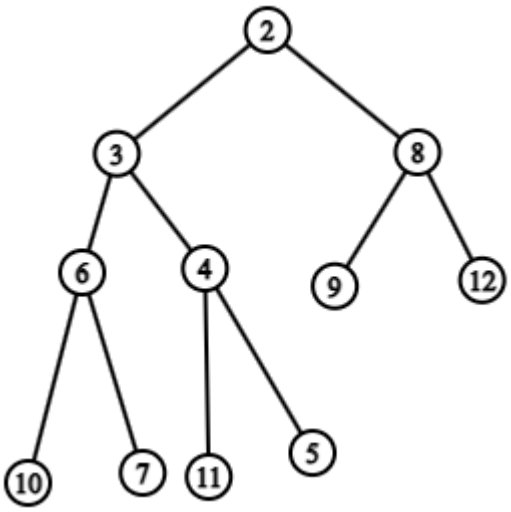
18. Inserati într-o trie cuvintele: their, there, this, that, does, did. Explicați cum puteți sorta aceste cuvinte în ordine lexicografică folosind trie-ul construit. Nodurile incercuite in negru sunt terminale



Odată ce trie-ul este construit, sortarea în ordine lexicografică poate fi realizată printr-un traversare în preordine (preorder traversal). Într-un trie, o traversare în preordine va vizita nodurile în ordine lexicografică, deoarece:

- Nodurile sunt vizitate de la rădăcină spre frunze.
- La fiecare nivel, vizităm nodurile în ordinea alfabetică a literelor.

19. **Construiți un min-heap cu 11 noduri în care pe ultimul nivel se afla printre altele valorile 11, 5 și 7.**



20. **Inserati în skip list următoarele valori: 13, 17, 1, 6, 8, 21, 23, 4, 9. La aruncarea banului obțineți următoarele valori: B, S, S, S, B, B, S, S, B, S, B, S, S, B, S, S, S, B, S, B, S, B. Cand obtineti B va opriți și inserați până la acel nivel. Cand obtineti S continuați la nivelul următor.**

Level 3: 17----------23

Level 2: 6----------17----21------23

Level 1: 4-----6----8------9----------17-----21-----23

Level 0: 1-----4-----6-----8-----9-----13-----17-----21-----23

Partea 4

21. Care este numărul minim de elemente dintr-un heap binar de înălțime k? Vom presupune că rădăcina heap-ului se află la înălțime 0.

- **Răspuns corect: 2^k**
- **Explicație:** Un heap binar complet de înălțime k are minim 2^k elemente. 1+2^1+2^2+...+2^(k-1)+1 = 2^k

22. Fie T un arbore binar de căutare și x un nod din T cu 2 fii. Care este numărul maxim de fii pe care îi poate avea predecesorul lui x?

- **Răspuns corect: 1**
- **Explicație:** Predecesorul lui x într-un BST este in arborele stang a lui x, maxim in dreapta, deci nu poate avea copil drept. Asadar, poate avea maxim 1 fiu, pe cel stang.

23. Care este complexitatea în timp, în cel mai rău caz, pentru a uni (merge) două heap-uri binomiale de dimensiuni m și n, într-un nou heap binomial?

- **Răspuns corect: log(m+n)**
- **Explicație:** Combinarea a două heap-uri binomiale are complexitatea O(log(m+n)) deoarece vom avea maxim log(m) arbori in primul si log(n) arbori cel de al doilea iar in combinarea lor vom avea maxim log(m+n) arbori si cum combinarea a doi arbori de dimensiune egala putere al lui doi se face in o(1) complexitatea este o(log(m + n))

24. Se poate găsi elementul maxim dintr-un min-heap în O(log(n))?

- **Răspuns corect: Nu**
- **Explicație:** Elementul minim într-un max-heap pe orice frunza, necesitând parcurgerea a cel puțin n/2 elemente.

25.) Fie T un arbore binar cu n noduri. Este posibil sa verificati în O(n) dacă T este arbore binar de căutare?Justificati.

- **Răspuns corect: Da**
- **Explicație:** Se face parcurgerea in inordine si se verifica daca e crescatoare

Partea 5

26) Fie P o permutare cu n elemente. Scopul este sa determinam aceasta permutare știind pentru fiecare poziție i (1 <= i <= n) cate elemente aflate înaintea ei sunt mai mici decat P[i]. Dacă exista mai multe permutări posibile, se va afla cea minim lexicografica. Input: N = 4 0 1 1 0 Output: 2 4 3 1

Solutie:

Fie cnt[i] = vectorul citit în datele de intrare, și p[i] = permutarea pe care trebuie să o aflăm. În primul rând, soluția, dacă există, este unică, iar condițiile de existență sunt că 0 <= cnt[i] <= i - 1 pentru orice i de la 1 la n. Dacă condiția menționată anterior nu se respectă, soluția nu există.

Presupunem ca solutia exista. Vom afla pe rând elementele în ordine descrescătoare a indicilor (de la n la 1). Ultimul element poate fi găsit instantaneu, acesta fiind egal cu cnt[n] + 1.

Inițial, mulțimea elementelor posibile pe care acestea le pot pune pe pozitia curenta este egală cu {1, 2, 3, …, n - 1, n}, iar după ce aflăm că p[n] = cnt[n] + 1, voi șterge acest element din mulțime (deoarepe p este o permutare si la un pas ulterior nu voi putea pune din nou alelasi element). Pentru valoarea n - 1, trebuie să găsim al 1 + cnt[n-1] -lea cel mai mic element din mulțimea rămasă, și după aceea să ștergem p[n-1]. Deci, procesul este următorul:

Notăm cu S(i) = mulțimea elementelor de la pasul i din proces. Inițial S(n) = {1, 2, 3, …, n - 1, n} iar mai apoi avem că p[i] = al 1 + cnt[i] - lea element din mulțimea S(i), iar S(i - 1) = S(i) / {p[i]}. De aici ne dăm seama și de unde vine condiția de existență și de ce soluția este unică. Procesul poate fi simulat cu un AINT și utilizând căutarea binară pe AINT în log(n) în felul următor: inițial facem: addAint(v = 1, tl = 1, tr = n, pos = 1, val = +1), addAint(v = 1, tl = 1, tr = n, pos = 2, val = +1), …, addAint(v = 1, tl = 1, tr = n, pos = n - 1, val = +1), addAint(v = 1, tl = 1, tr = n, pos = n, val = +1). Iar pe parcurs ce vrem să găsim al k-lea element din mulțimea rămasă trebuie doar să căutăm binar până unde suma elementelor este <= k - 1, iar elementul nostru este fix următorul. După ce ștergem un element din mulțime facem: addAint(v = 1, tl = 1, tr = n, pos = p[i], val = -1).

Complexitatea: O(n*log(n)).

27) Se considera un șir S de paranteze rotunde (închise sau deschise) de lungime n și q query-uri (i, j). Pentru fiecare query, sa se raspunda dacă subsecventa de la i la j (S[i…j]) este corect parantezata.

Solutie:

Initial, trebuie să verificăm dacă numărul de paranteze deschise este egal cu numărul de paranteze închise. Acest lucru se poate face folosind sume parțiale. Definim bal(i) ca „balanța pe care o are parantezarea în primele i elemente”, adică numărul de '(' minus numărul de ')' din primele i elemente. Avem că bal(i) = bal(i - 1) + 1 dacă al i-lea element este '(', sau bal(i) = bal(i - 1) - 1 dacă al i-lea element este ')'. Ca să avem un număr egal de paranteze deschise și închise în parantezarea de la i la j, trebuie ca bal[j] - bal[i - 1] = 0.

Acum trebuie și să nu avem niciodată un k cu proprietatea că k aparține intervalului {i, i+1, …, j - 1, j} cu bal[k] - bal[i - 1] < 0, adică să nu avem niciodată un prefix cu proprietatea că avem mai multe paranteze închise decât deschise pe el. Dacă nu există niciun astfel de k și parantezarea este echilibrată, atunci există soluție. Stim că abs(bal[i] - bal[i - 1]) = 1. Deci, dacă există un k cu proprietatea că bal[k] - bal[i - 1] < 0, atunci există unul cu proprietatea că bal[k] = bal[i - 1] - 1. Dacă pentru fiecare indice z găsim următorul indice u[z] (u[z] > z și bal[u[z]] = bal[z] - 1). u[z] reprezinta urmatoarea pozitie la dreapta cu balansul mai mic decat pozitia curenta. Atunci trebuie doar să verificăm că u[z] > j pentru a nu avea un balans negativ pe parcursul secvenței de la i la j. Acest indice z se poate să fie găsit fie cu o stivă (de la capăt la început), fie cu un vector de frecvență care menține ultimul indice de o anumită valoare, abuzând de faptul că bal are valori <= n în modul.

Complexitatea fiind in ambele cazuri: O(n)

Solutie 2:

Alternativ, se poate folosi si rmq. Ideea de balans se pastreaza (bal[j] - bal[i - 1] trebuie sa fie egal cu 0) ceea ce se verifica folosind sume partiale, dar in loc de stiva folosim

un rmq pentru pentru a afla balansul minim pe fiecare interval si se verifica daca acesta e >= bal[i-1].

Complexitatea: n*log(n)

28) Se da un vector A cu N elemente. 1 <= A[i] <= N. pentru fiecare i sa se gaseasca j minim a.i. A[j] < A[i].

Soluție:

Vom profita de faptul că 1 <= A[i] <= N și vom utiliza un vector de frecvență în care, pe poziția i, vom pune indicele minim al unui element care are valoarea egală cu i. După aceea, vom face un nou vector sau vom reconstrui vectorul nostru în așa fel încât acesta să mențină indicele minim al unei poziții care are valoarea <= i, acest lucru realizându-se foarte ușor, doar înlocuind vectorul cu minimul dat pe prefix. Dacă numim acest vector cu T[i], atunci în momentul în care trebuie să răspundem cu soluția pentru indicele i, va trebui să afișăm T[a[i]-1].

Complexitatea: O(n).

Soluție 2:

Alternativ, putem folosi un arbore de intervale. Pentru poziția i, cel mai mic j cu proprietatea A[j]<A[i] este query(început=1,final= i). Apoi adăugăm în arbore pe poziția A[i] elementul i, dacă pe poziția A[i] nu se află deja ceva (add(poz=A[i] val= i)).

Complexitatea: n*log(n)

29) Se da un vector A cu N elemente. 1 <= A[i] <= N. Să se numere tripletele (i, j, k) a.i. 1 <= i < j < k <= N si A[i] < A[j] < A[k].

Soluție:

Observăm următorul fapt de care putem să abuzăm: odată ce fixăm J, adică indicele din mijloc, calculele pe care le facem la stânga sunt complet independente de calculele pe care le facem la dreapta, adică trebuie doar să numărăm câți indici i există la stânga cu proprietatea că a(i) < a(j) și trebuie să numărăm câți indici k există cu proprietatea că a(k) > a(j). Odată ce calculăm aceste două numere și le numim x(j) și y(j), trebuie să adăugăm la soluția noastră x(j) * y(j), deci soluția finală va fi sum(j=1…n | x(j) * y(j)).

Voi descrie cum se poate calcula x(j), y(j) calculându-se aproape la fel, mai puțin că vom parcurge vectorul în ordine inversă și că în loc să numărăm câte sunt mai mici, trebuie să numărăm câte sunt mai mari. Voi calcula vectorul X cu un arbore de intervale: parcurgem vectorul de la stânga la dreapta și în momentul în care trecem prin dreptul indicelui i, va trebui să zicem că x(i) = query(1, a(i)-1), iar după aceea facem add(a(i), +1), profitând de faptul că elementele sunt între 1 și n.

Vectorul Y se calculează asemănător, dar cu un alt arbore de intervale. Parcurgem vectorul de la dreapta la stânga și în momentul în care trecem prin dreptul indicelui i, y(i) = query(a(i)-1,n), iar după aceea facem add(a(i), +1), profitând de faptul că elementele sunt între 1 și n.

Complexitatea: O(n*log(n)).