Arbori de intervale

Arbori de Intervale

Problemă. Se dă un vector cu n numere și operații de genul:

- Adăugăm la poziția i valoarea x (x poate fi și negativ)
- Cerem maximul pe intervalul i, j (ex 3 6)

0	1	2	3	4	5	6	7	8
3	9	2	**5**	**7**	**34**	**6**	11	8

Cum putem face asta?

Șmenul lui Batog

Problemă. Se dă un vector cu n numere și operații de genul:

- Adăugăm la poziția i valoarea x (x poate fi și negativ)
- Cerem minimul pe intervalul i, j (ex 3 6)

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr><td>3</td><td>9</td><td>2</td><td>5</td><td>7</td><td>34</td><td>6</td><td>11</td><td>8</td></tr>
<tr><td colspan="3">9</td><td colspan="3">34</td><td colspan="3">11</td></tr>
</table>

Împărțim vectorul în zone de L (?) și calculăm minimul pe fiecare zonă în parte.

Șmenul lui Batog

Problemă. Se dă un vector cu n numere și operații de genul:

- Adăugăm la poziția i valoarea x (x poate fi și negativ)
  - Pentru că, dacă facem maximul mai mic, trebuie să găsim noul maxim **O(sqrt(n))**
- Cerem maximul pe intervalul i, j (ex 3 6)

0	**1**	**2**	3	4	5	**6**	**7**	8
3	**9**	**2**	5	7	3	**6**	**11**	8
9			**7**			11		

Cum răspundem la 0, 8? Dar la 0, 4? Dar la 1, 7 ?

Care este complexitatea ?

Șmenul lui Batog

Complexitate query: Împărțim în n/L zone de lungime L

O(n/L(nr de zone) + 2 * L(2 zone le pot itera aproape complet)) → L = sqrt(n)

O (sqrt(n) + 2 * sqrt(n)) = **O(sqrt(n))**

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr><td>3</td><td>9</td><td>2</td><td>5</td><td>7</td><td>3</td><td>6</td><td>11</td><td>8</td></tr>
<tr><td colspan="3">9</td><td colspan="3">7</td><td colspan="3">11</td></tr>
</table>

Șmenul lui Batog

Împărțim în zone de:

- sqrt(n) sau…
- sqrt(n)/2
- sqrt(n) * 2 .. și
- Variațiuni
- De ce?
  - Pentru că, în practică, nu sqrt(n) va fi cel mai rapid. Totuși, sqrt(n) este o alegere buna în general.

Șmenul lui Batog

Problemă. Se dă un vector cu n numere. Sortați-l!

problemă: https://leetcode.com/problems/sort-an-array/submissions/

cod: https://pastebin.com/bFHYephh

0	1	2	3	4	5	6	7	8
3	9	50001	5	7	34	6	11	8
3			**5**			**6**		

Arbori de Intervale

Problemă. Se dă un vector cu n numere și operații de genul:

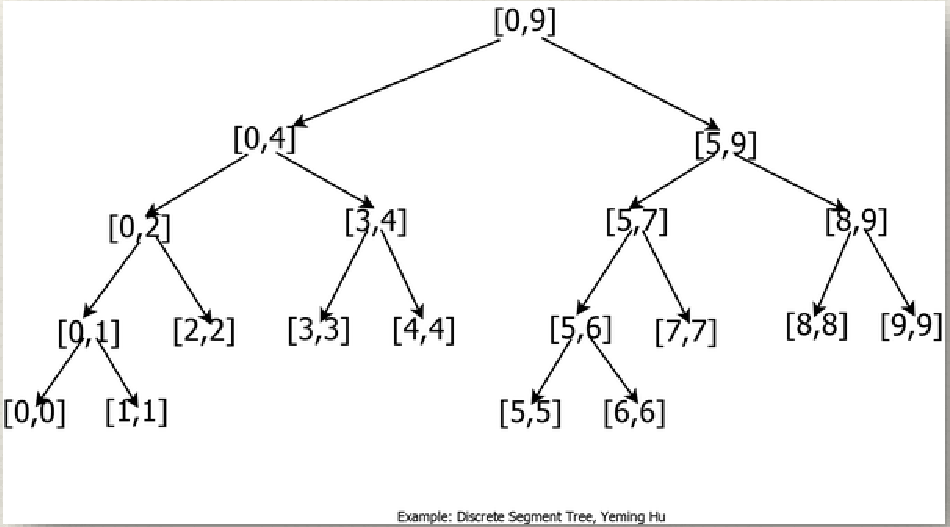
- Adăugăm la poziția i valoarea x (x poate fi și negativ)
- Cerem minimul pe intervalul i, j (ex 3 6)

0	1	2	3	4	5	6	7	8	9
3	9	2	**5**	**7**	**34**	**6**	11	8	44

Arbori de Intervale

Arbore cu rădăcina ținând intervalul [0,n)

Pentru un nod ce ține intervalul [L, R] → fiul stâng ține [L, (L+R)/2], cel drept [(L+R)/2 + 1, R]

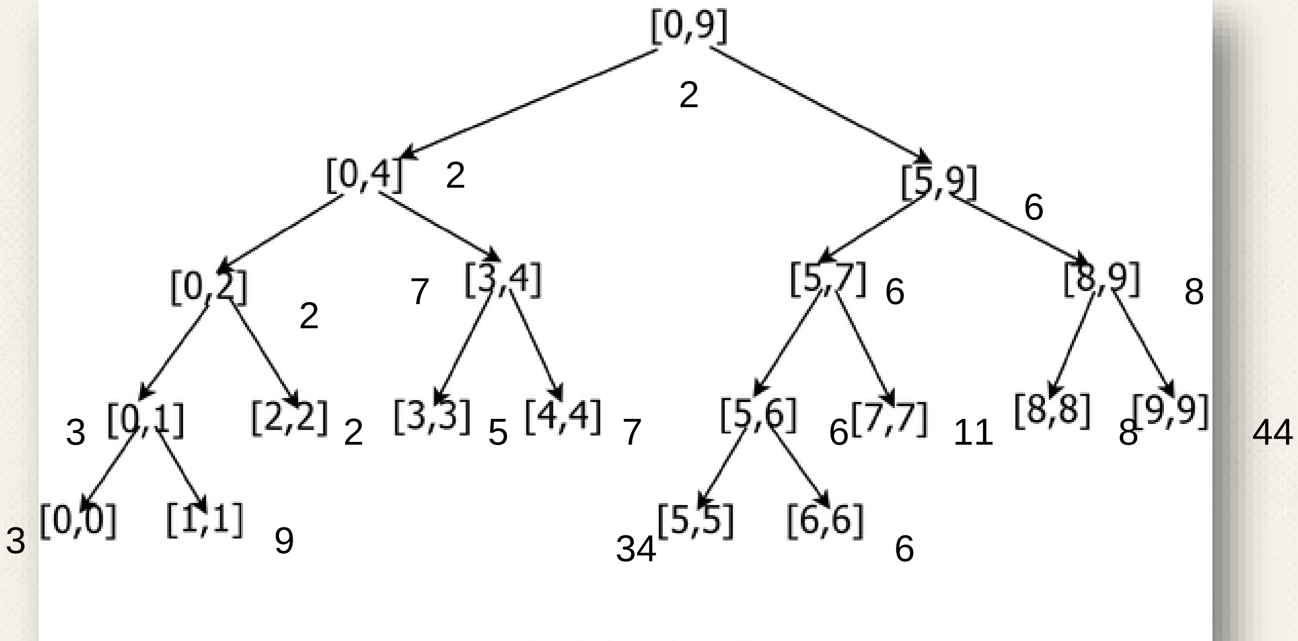


0	1	2	3	4	5	6	7	8	9
3	9	2	**5**	**7**	**34**	**6**	11	8	44

Arbori de Intervale

0	1	2	3	4	5	6	7	8	9
3	9	2	**5**	**7**	**34**	**6**	11	8	44

Ținem minimul!

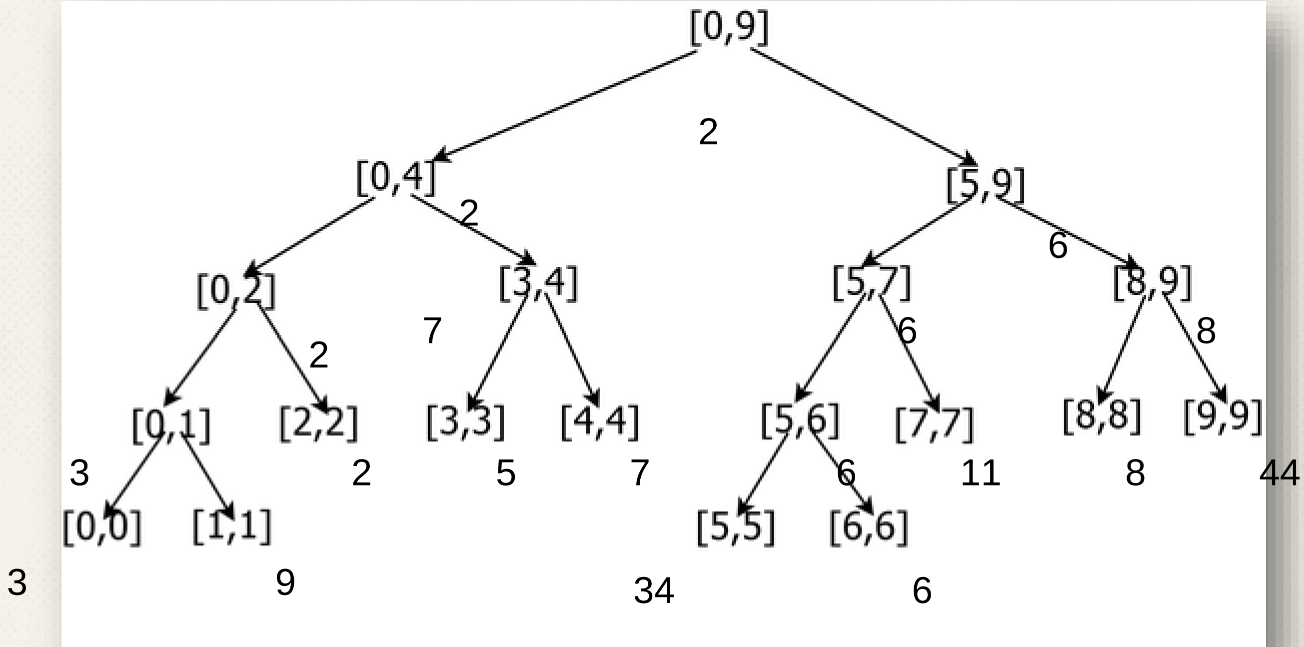


Example: Discrete Segment Tree, Yeming Hu

Arbori de Intervale

0	1	2	3	4	5	6	7	8	9
3	9	2	**5**	**7**	**34**	**6**	11	8	44

Cum îl implementăm?



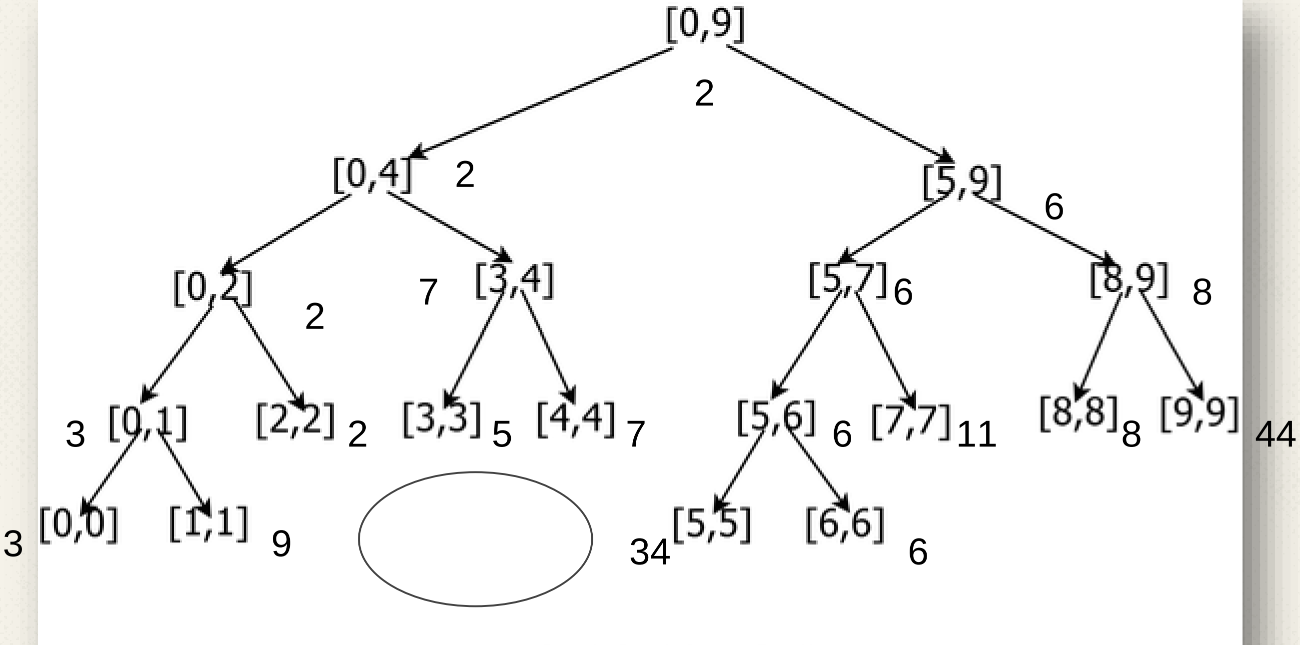
Example: Discrete Segment Tree, Yeming Hu

Arbori de Intervale

0	1	2	3	4	5	6	7	8	9
3	9	2	**5**	**7**	**34**	**6**	11	8	44

Cum îl implementăm?

- Arbore like
- **Vector!**

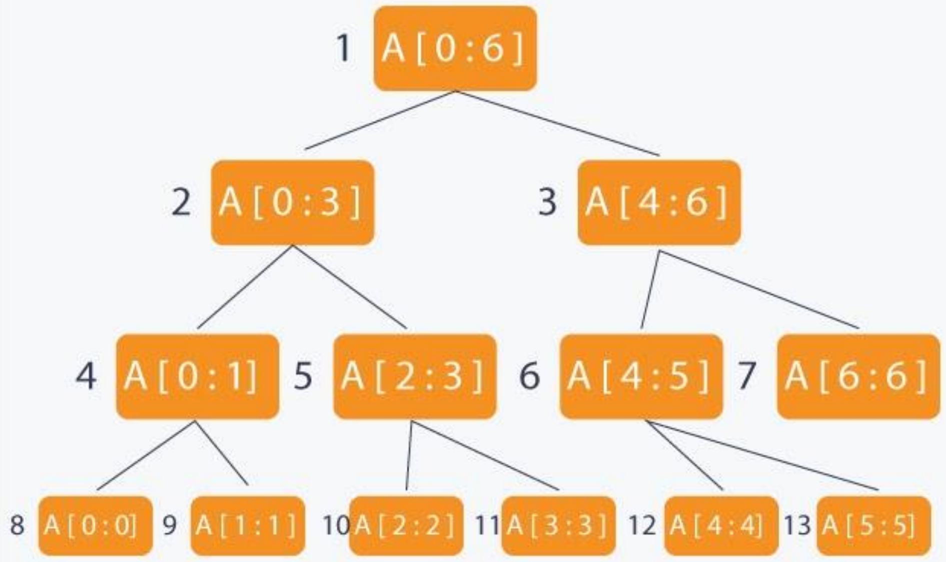


Example: Discrete Segment Tree, Yeming Hu

Arbori de Intervale

tree [1] = A[0:6]
tree [2] = A[0:3]
tree [3] = A[4:6]
tree [4] = A[0:1]
tree [5] = A[2:3]
tree [6] = A[4:5]
tree [7] = A[6:6]
tree [8] = A[0:0]
tree [9] = A[1:1]
tree [10] = A[2:2]
tree [11] = A[3:3]
tree [12] = A[4:4]
tree [13] = A[5:5]

Segment Tree represented as linear array



Segment Tree

Arbori de Intervale

Reprezentare similară cu heapul:

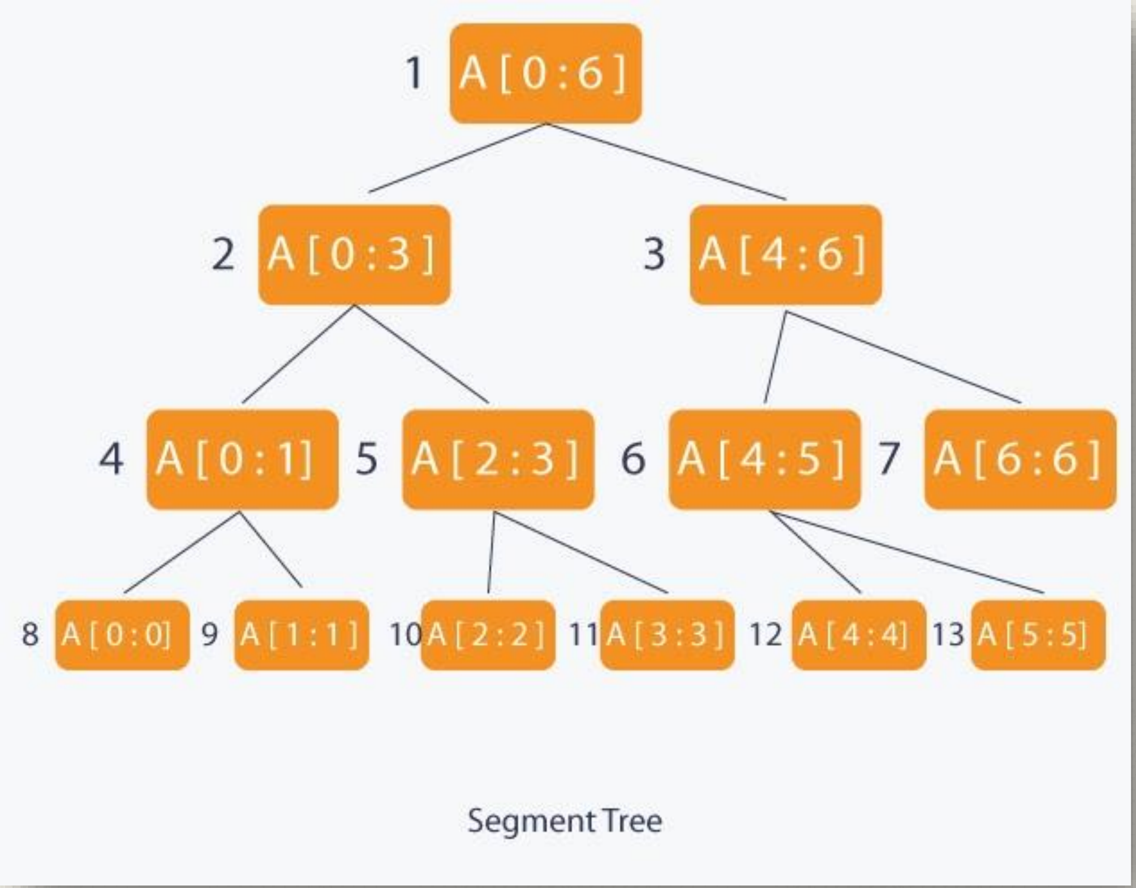
- Rădăcina (1 de multe ori) are intervalul [0,n) [L,R)
  - Fiul stâng are [L, (L+R)/2]; el are poziția în vector i*2
  - Fiul drept are [(L+R)/2 + 1, R]; el are poziția în vector i*2+1
  - Vectorul poate avea niște elemente lipsă pe ultimul rând (vezi 2 slide-uri mai sus).

În total vectorul are 2*n noduri “active”, dar avem nevoie de mai mult de 2*n memorie. 4*n e safe

O(n) memorie.

Operații

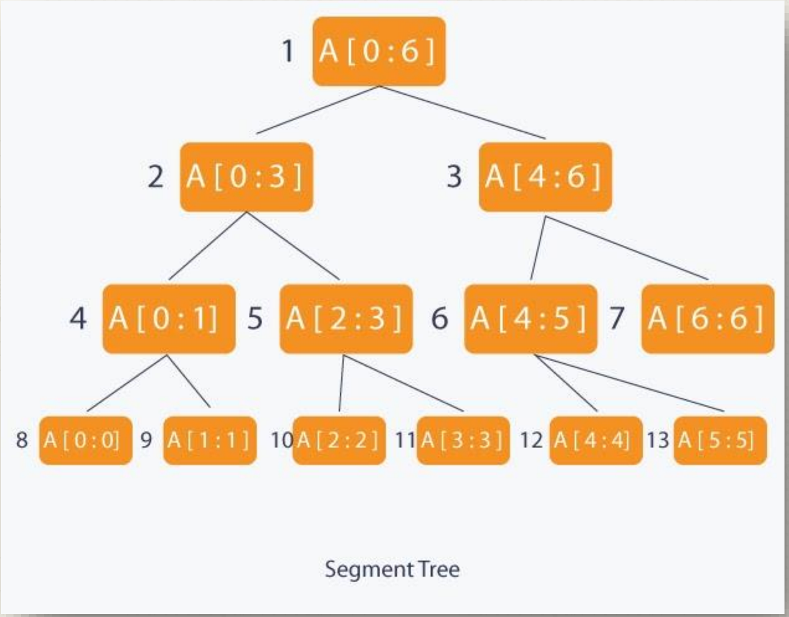
- Query pe index
- Query pe interval
  - Min
  - Sum
- Modificare element
- Modificare interval



Operații

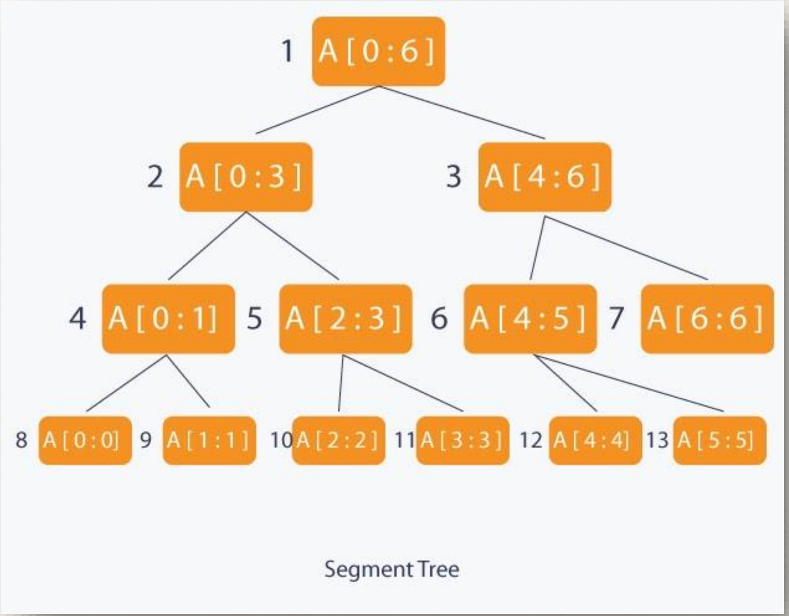
- Query pe index
  - Ori avem “pointeri” spre frunze și răspund direct
  - Ori pornim top down

```
getValue(vector<int> arb_int, int index, int n) {
    int L = 0, R = n, poz = 1;
    while (L != R) {
        if (index > (L + R)/2) {
            L = (L +R)/2, poz = poz*2 + 1;
        }
        else {
            R = (L+R)/2, poz *=2;
        }
    return arb_int[poz]; // L = R;
}
```



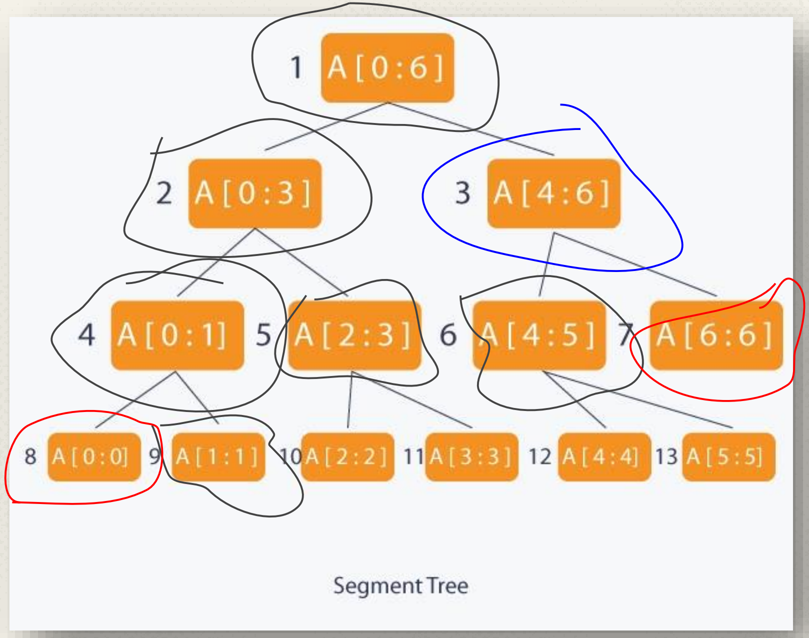
Operații

- Query pe interval
  - Evident, nu luăm toate valorile; ar putea fi liniar
  - Q(1,5) min



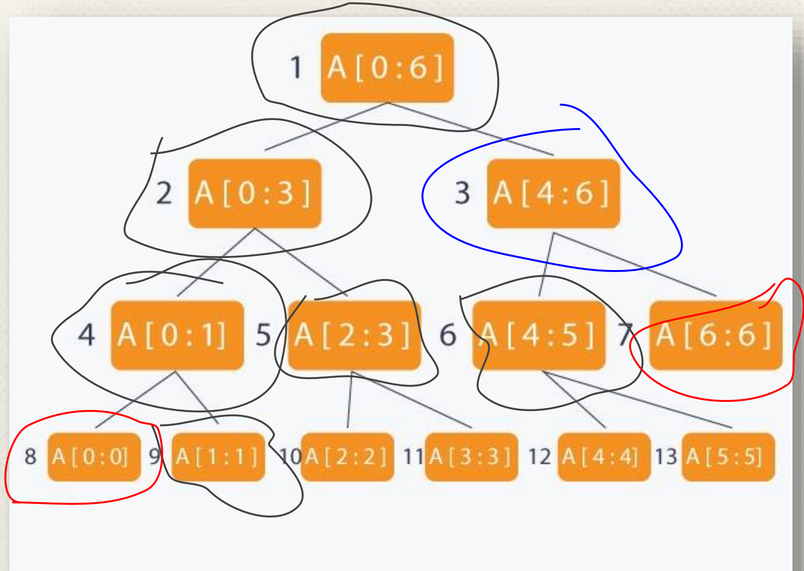
Operații

- Query pe interval
  - Evident, nu luăm toate valorile; ar putea fi liniar
  - Q(1,5) min
  - Pornim din rădăcina și mergem recursiv și L și R
  - **Dacă intervalul nodului nu se intersectează, oprim**
  - **Dacă intervalul e inclus complet, luăm info & ne oprim**
  - Câte noduri putem parcurge?



Operații

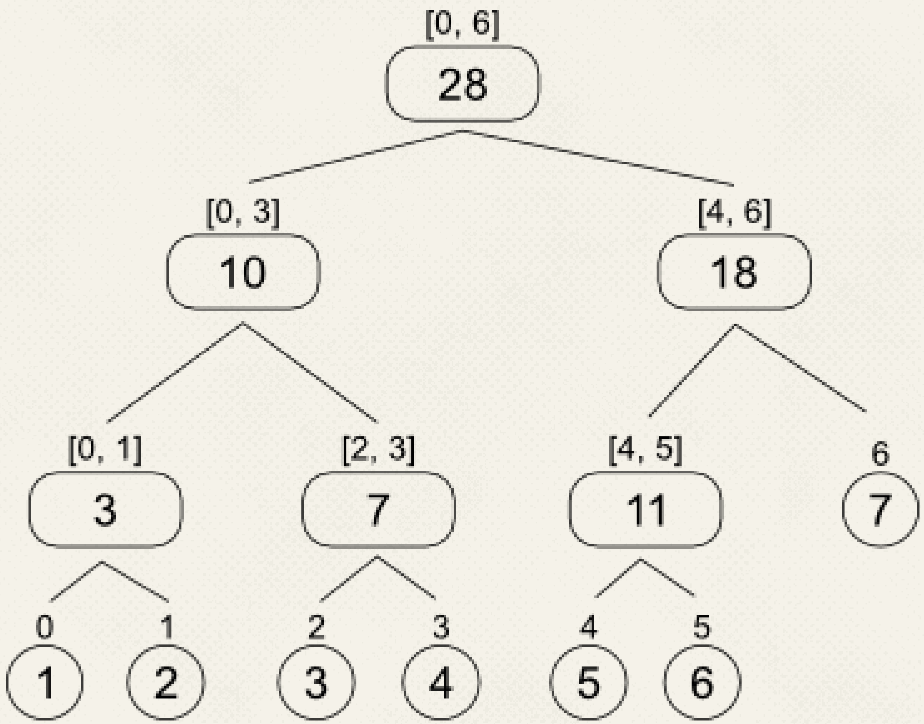
- Query pe interval
  - Evident, nu luăm toate valorile; ar putea fi liniar
  - Q(1,5) min
  - Pornim din rădăcina și mergem recursiv și L și R
  - Caz I — Dacă intervalul nodului nu se intersectează, oprim
  - Caz II — **Dacă intervalul e inclus complet, luăm info & ne oprim**
  - Câte noduri putem parcurge?
    - Doar 4*log n
    - Coborâm pe o ramură până facem un split
      - După split, în fiecare parte, unul dintre fii va fi ori cazul I, ori cazul II, deci se va coborî pe maxim 2 drumuri până jos.



Segment Tree

Operații

- Query pe index
- Query pe interval
  - **Sum (1,5)**
    - **?**

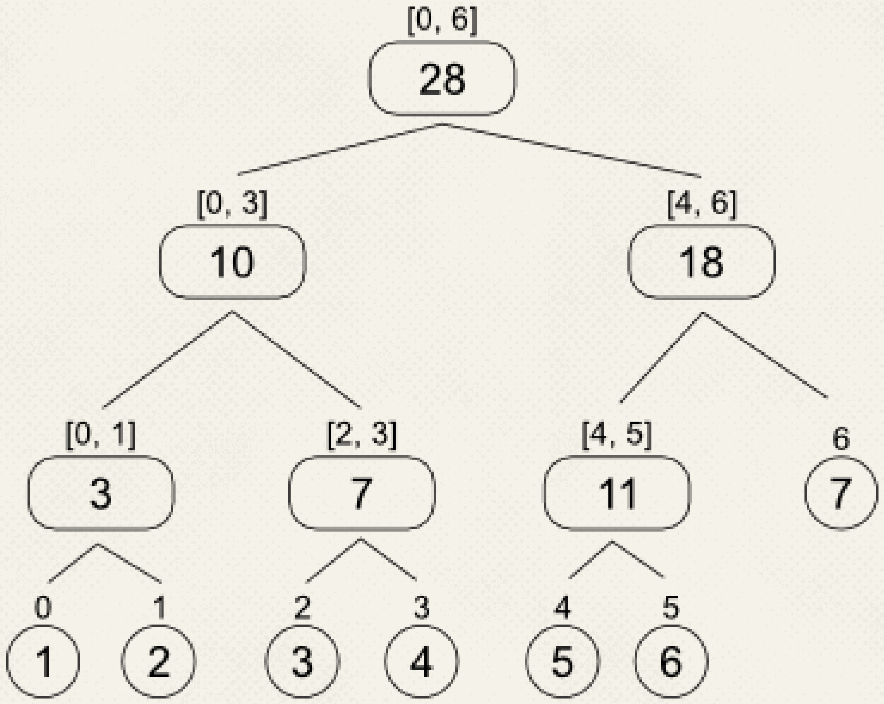


Operații

- Modificare element
  - Dacă țin suma, pot face top-down
  - Dacă țin minim, pot face ori:
    - Top down up
      - coborâm din rădăcină până găsim frunza pe care o modificăm
      - La urcare, facem update tata = min(cei 2 fii)
    - Bottom up
      - Exact ca mai sus, dar avem deja indexul ținut
  - Înapoi la sortare (https://leetcode.com/problems/sort-an-array/submissions/)

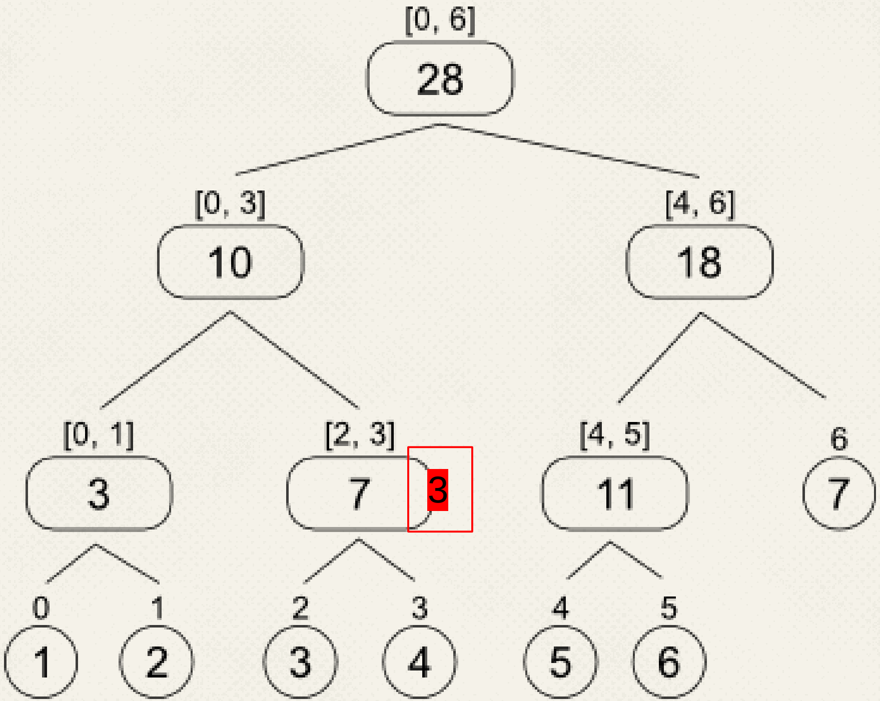
Operații

- Modificare pe interval
  - Similar cu query pe interval
  - Merg recursiv în ambii fii
    - Mă opresc, dacă nu am intersecție
    - Modific doar nodul actual, dacă este inclus de tot în interval
      - Aici trebuie să ținem în nod o informație suplimentară (toate nodurile cresc cu o anumită valoare)
    - Cobor dacă e intersectie parțială

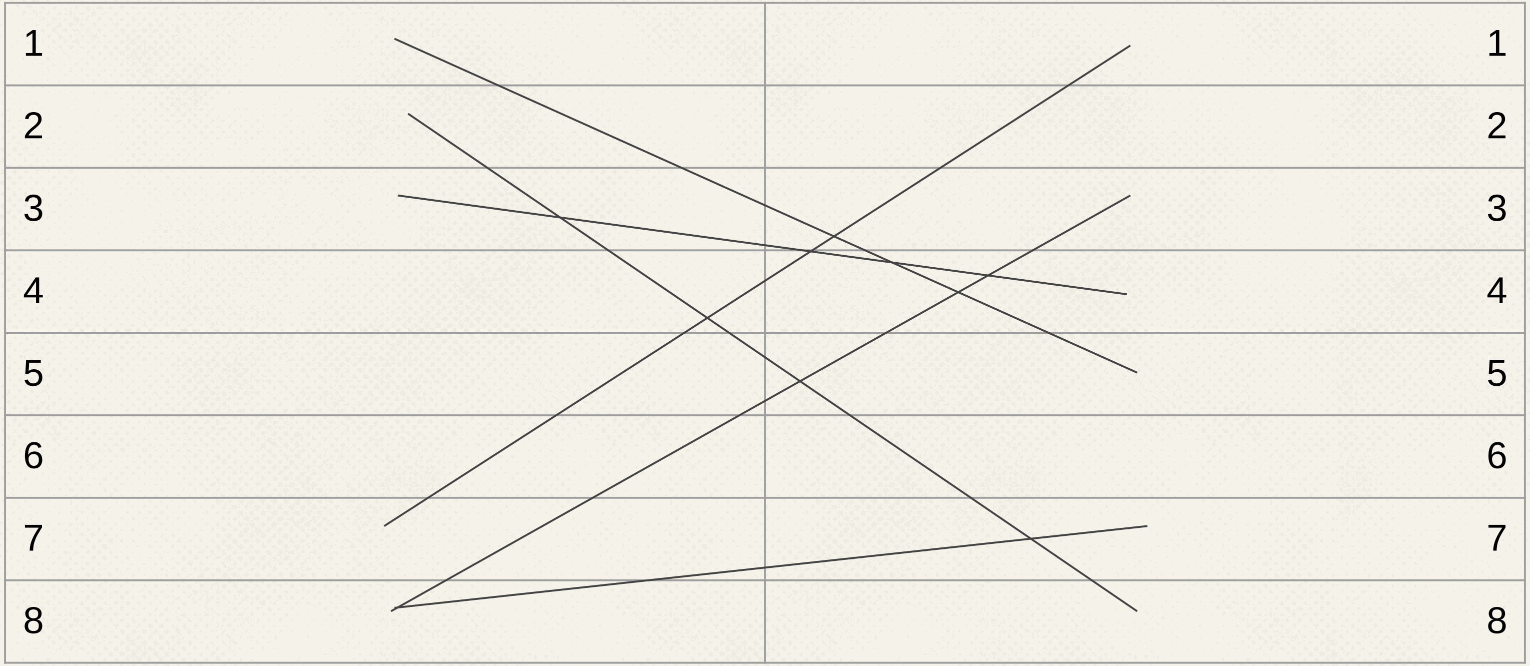


Operații

- Modificare pe interval
  - Add(3, 1, 3) (adaugă 3 la fiecare element din intervalul 1, 3)
  - O mică atenție la query-uri



Câte intersecții am?



Problemă pentru seminar 6!

1 **5**, 2 **8,** 3 **4**, 7 1, 8 3, 8 7

Implementare

- https://www.hackerearth.com/practice/data-structures/advanced-data-structures/segment-trees/tutorial/

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 100100;
int n, q, arb[4 * N];

void update(int nod, int st, int dr, int idx, int val){
    if (st == dr){
        arb[nod] = val;
        return;
    }
    int mid = (st + dr) >> 1;
    if (idx <= mid) update(2 * nod, st, mid, idx, val);
    else update(2 * nod + 1, mid + 1, dr, idx, val);
    arb[nod] = min(arb[2 * nod], arb[2 * nod + 1]);
}

int query(int nod, int st, int dr, int l, int r){
    if (st >= l && dr <= r)
        return arb[nod];
    int mid = (st + dr) >> 1;
    int left_side = 1e9, right_side = 1e9;
    if (l <= mid) left_side = query(2 * nod, st, mid, l, r);
    if (r > mid) right_side = query(2 * nod + 1, mid + 1, dr, l, r);
    return min(left_side, right_side);
}

```

```cpp
int main(){
    cin >> n >> q;
    for (int i=1; i<=n; i++){
        int x; cin >> x;
        update(1, 1, n, i, x);
    }
    while (q--){
        char c; int x, y;
        cin >> c >> x >> y;
        if (c == 'q')
            cout << query(1, 1, n, x, y) << '\n';
        else
            update(1, 1, n, x, y);
    }
    return 0;
}
```

RMQ, LCA, LA

Definirea problemelor

Range Minimum Query (RMQ):

Se dă un vector. Răspundeți cât mai eficient la întrebări de genul: **Care este cel mai mic element din intervalul i, j?**

0	1	2	3	4	5	6	7	8	9
3	9	2	8	5	3	8	7	6	11

https://www.infoarena.ro/problema/rmq

0 3 → 2

5 9 → 3

LCA

Lowest Common Ancestor (LCA):

Se dă un arbore. Răspundeți cât mai eficient la întrebări de genul: **Se dau două noduri într-un arbore. Găsiți cel mai apropiat strămoș comun.**
(https://www.infoarena.ro/problema/lca)

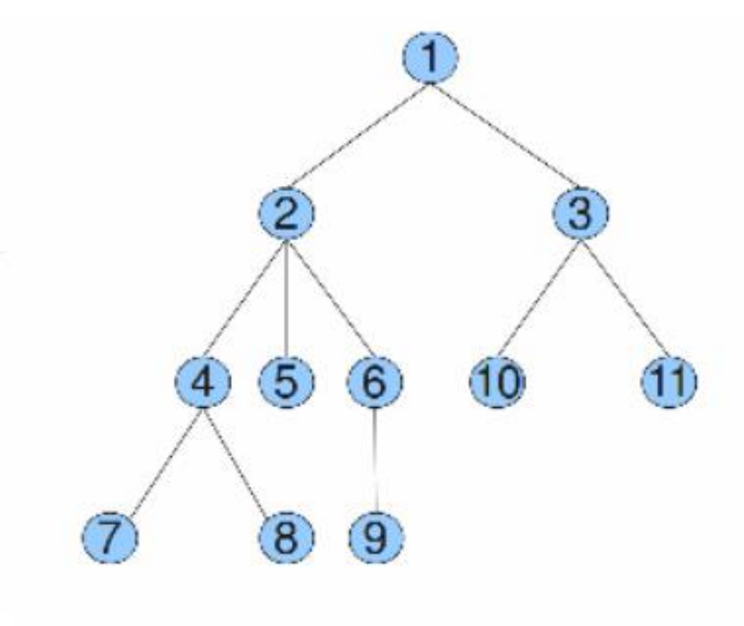
4 9 → 2

4 11 → 1

7 6 → 2

8 9 → 2

8 4 → 4



Lowest Ancestor

Se dă un arbore. Răspundeți cât mai eficient la întrebări de genul: **Se dă un nod și un întreg k. Care este strămoșul de nivel k al nodului dat?**

https://www.infoarena.ro/problema/stramosi (adăugată cu 1 punct la temă)

2 1 → 1

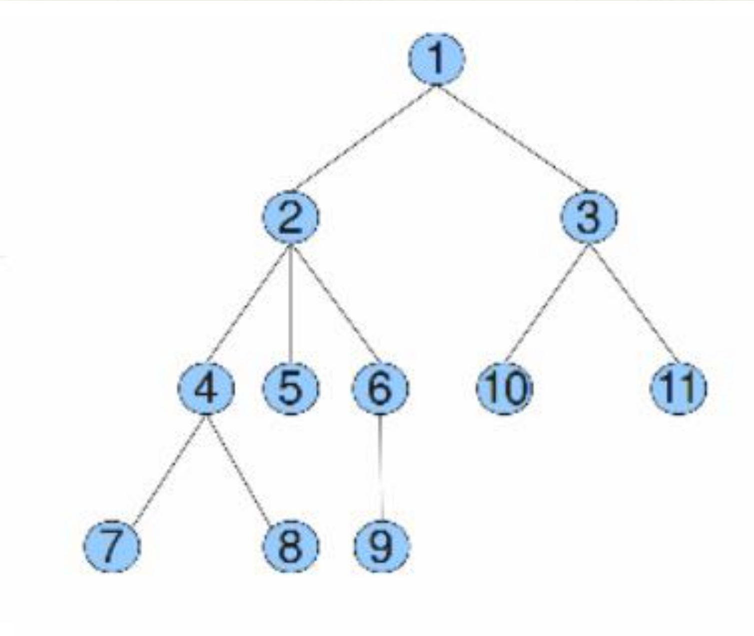
9 1 → 6

9 2 -> 2

9 3 -> 1

6 4 → -1

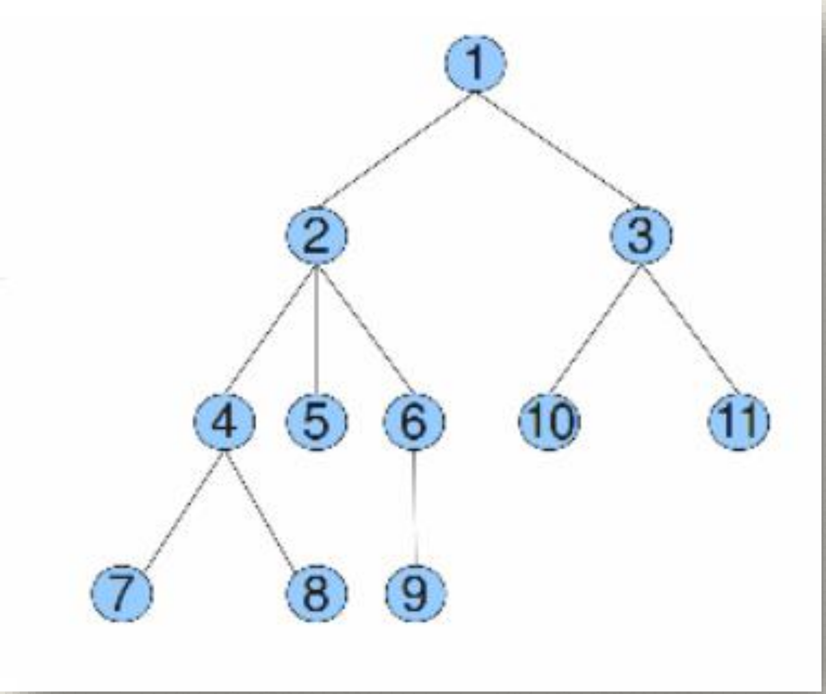
10 1 → 3



Lowest Ancestor - soluții

Se dă un arbore. Răspundeți cât mai eficient la întrebări de genul: **Se dă un nod și un întreg k. Care este strămoșul de nivel k al nodului dat?**

2 1 → 1 9 1 → 6



Cum facem?

- Putem răspunde în O(h), parcurgând din tată în tată la fiecare query.
- Putem răspunde în O(1), dacă pentru fiecare nod rețin

`D[i][j]` = strămoșul de nivel j a lui i

`D[9] = {9, 6, 2, 1}`

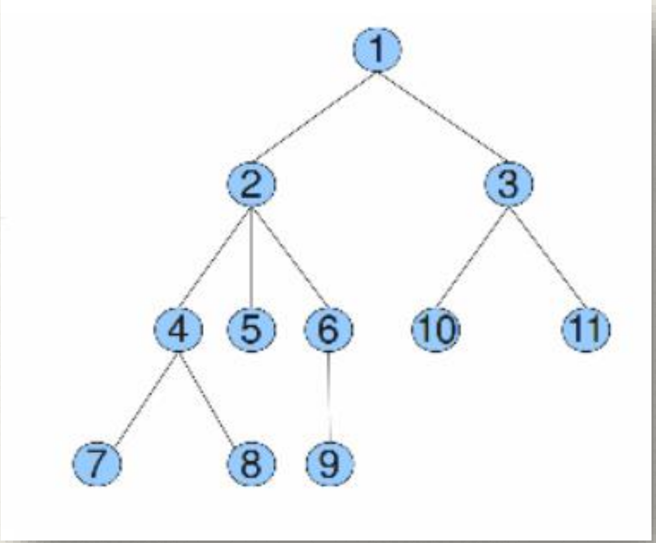
Lowest Ancestor - soluții

Se dă un arbore. Răspundeți cât mai eficient la întrebări de genul: **Se dă un nod și un întreg k. Care este strămoșul de nivel k al nodului dat?**

2 1 → 1 9 1 → 6

`D[i][j]` = strămoșul de nivel j a lui i

`D[9] = {9, 6, 2, 1}`



Memorie și preprocesare O(n*h) și răspuns O(1).

Lowest Ancestor - soluții

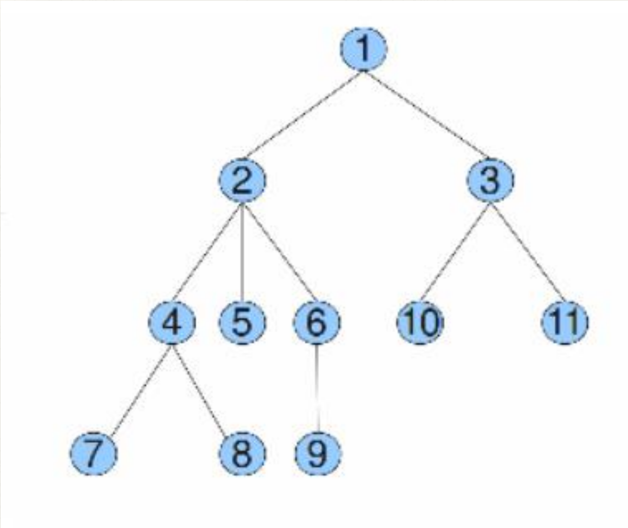
Se dă un arbore. Răspundeți cât mai eficient la întrebări de genul: **Se dă un nod și un întreg k. Care este strămoșul de nivel k al nodului dat?**

Sau pot folosi sqrt decomposition:

Țin tatăl de ordin radical din n,

dacă radical din n este 100, iar eu țin din 100 în 100:

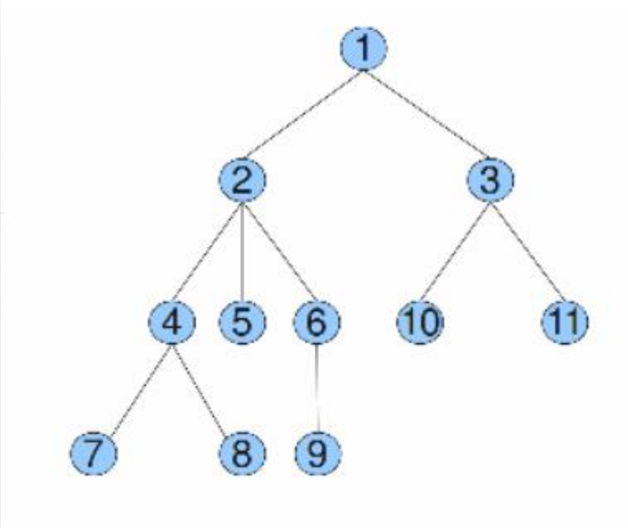
Tatăl 300 este `tata100[tata100[tata100[x]]];`



Lowest Ancestor - soluții

Se dă un arbore. Răspundeți cât mai eficient la întrebări de genul: **Se dă un nod și un întreg k. Care este strămoșul de nivel k al nodului dat?**

2 1 → 1 9 1 → 6



Țin tatăl de ordin radical din n,

dacă radical din n este 100, iar eu țin din 100 în 100:

Tatăl 301 este

```
tata[tata100[tata100[tata100[x]]]];
```

Soluție cu **O(n)** memorie suplimentară,

O(1) pe nod și **O(sqrt(n))** pe query.

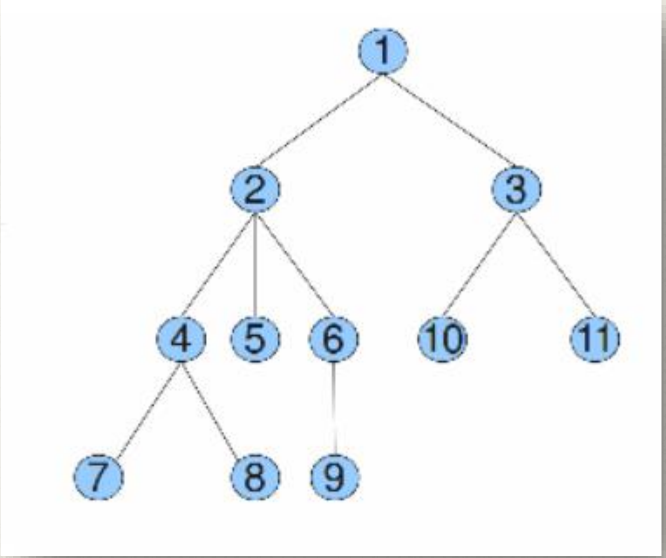
Lowest Ancestor - soluții

Se dă un arbore. Răspundeți cât mai eficient la întrebări de genul: **Se dă un nod și un întreg k. Care este strămoșul de nivel k al nodului dat?**

2 1 → 1 9 1 → 6

Cum facem ?

- O(n) query, O(1) memorie
- O(sqrt n) query și O(n) memorie (Batog)
- **O(log n) query și O(n log n) memorie**



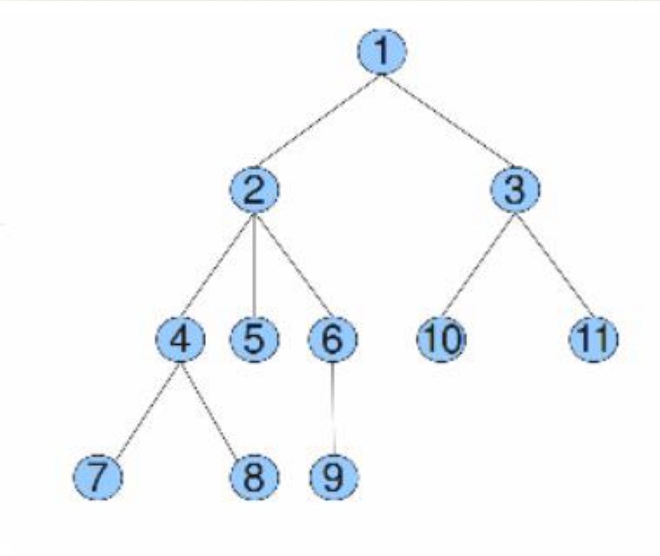
Lowest Ancestor

O(log n) query și O(n log n) memorie

Pentru fiecare nod, țin tații de înălțime 1, 2, 4, 8, 16…

Pentru 7 → 4, 2, -1, -1 ….

Pentru 6 → 2, 1, -1, -1 ….



Cum calculăm vectorul de tați?

```
for (int i = 1; i < log n; ++i) {
  for (int j = 1; j < n; ++j)
     tata[j][i] = tata[tata[j][i-1]][i-1];
}
```

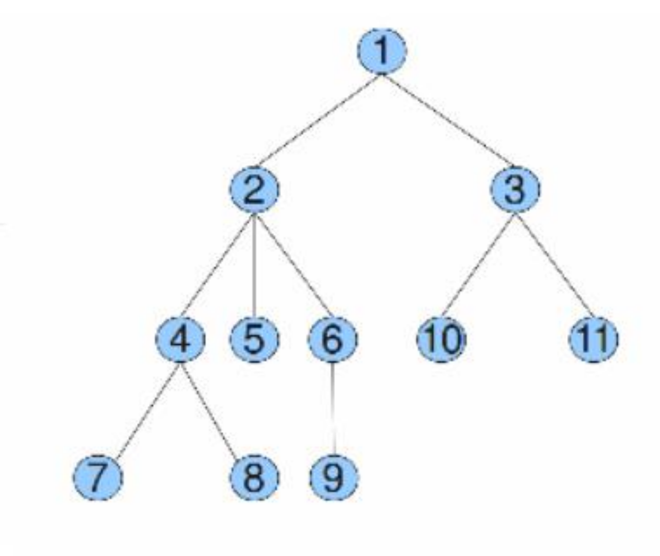
Lowest Ancestor

O(log n) query și O(n log n) memorie

Pentru fiecare nod, țin tații de înălțime 1, 2, 4, 8, 16…

Pentru 7 → 4, 2, -1, -1 ….

Pentru 6 → 2, 1, -1, -1 ….



Cum calculăm al k-lea strămoș?

- Similar cu căutarea binară discutată la curs
- Sărim cu puterea lui 2 cea mai mare

7 3 → 7 sărim 2 pași până la 2

Apoi 2 1 → sărim 1 pas → 1

Lowest Ancestor

O(log n) query și O(n log n) memorie

Pentru fiecare nod, țin tații de înălțime 1, 2, 4, 8, 16…

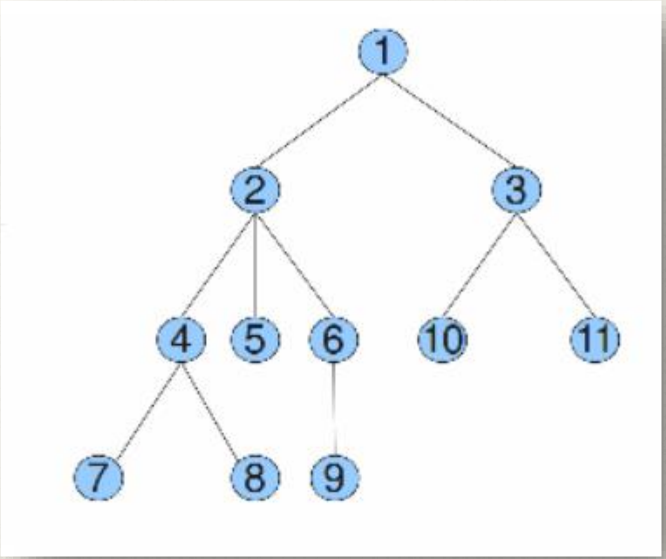
Pentru 7 → 4, 2, -1, -1 ….

Pentru 6 → 2, 1, -1, -1 ….

Cum calculăm al k-lea strămoș?

tata(x, 14) = tata(tata8[x], 6) = tata(tata4[tata8[x]],2)

= tata2[tata4[tata8[x]]

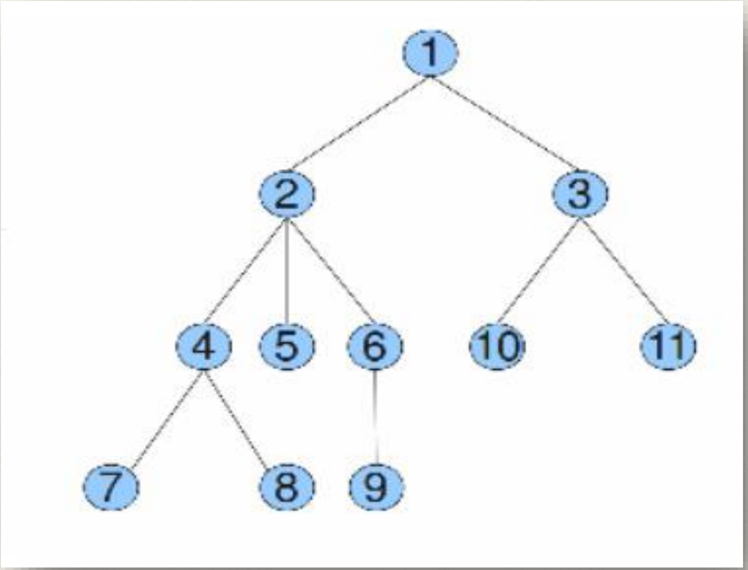


Lowest Ancestor

O(log n) query și O(n log n) memorie

Complexitate ?

- O(n log n) preprocesoare
- O(n log n) memorie suplimentară
- O(log n) pe query
- Se poate obține O(n) memorie suplimentară
    - (vezi cursul de la MIT)



Range Minimum Query Soluții

Range Minimum Query (RMQ):

Se dă un vector. Răspundeți cât mai eficient la întrebări de genul: **Care este cel mai mic element din intervalul i,j?**

0	1	2	3	4	5	6	7	8	9
3	9	2	8	5	3	8	7	6	11

Soluții ?

- O(n) pe query
- Șmenul lui Batog - O(sqrt (n)) pe query
- **Ținem pentru fiecare element puterile lui 2 și răspundem similar LA în log n.**

Range Minimum Query Soluții

- Ținem pentru fiecare element puterile lui 2 și răspundem similar LA în log n.

	0	1	2	3	4	5	6	7	8	9
min										
min2	3	2	2	5	3	3	7	6	6	11
min4	2	2	2	3	3	3	6	6	6	11
min8	2	2	2	3	3	3	6	6	6	11

Range Minimum Query Soluții

- **Ținem pentru fiecare element puterile lui 2 și răspundem similar LA în log n.**

	0	1	2	3	4	5	6	7	8	9
min	3	9	2	8	5	3	8	7	6	11
min2	3	2	2	5	3	3	7	6	6	11
min4	2	2	2	3	3	3	6	6	6	11
min8	2	2	2	3	3	3	6	6	6	11

- Query în **log(n)**
  - 1 6 min4(1) 1-4 + min2(5) 5-6
  - 2 9 min8(2)
  - 3 6 min4(3) ->alte exemplu și aici

Range Minimum Query Exemplul 2

- **Ținem pentru fiecare element puterile lui 2 și răspundem similar LA în log n.**

	0	1	2	3	4	5	6	7	8	9
min	3	10	2	11	5	3	13	7	1	0
min2	3	2	2	5	3	3	7	1	0	0
min4	2	2	2	3	3	1	0	0	0	0
min8	2	1	0	0	0	0	0	0	0	0

Problemă adițională

Se dă un nr $n \leqslant 10^9$. Cum calculez logn în O(1) ?

Problemă adițională

Se dă un nr $n \leqslant 10^9$. Cum calculez logn în O(1) ?

- Pot ține, pentru fiecare număr de la 1 la 256, care e cel mai semnificativ bit
  - $14 \rightarrow 8$
  - $230 \rightarrow 128$
  - ....
- Pentru un număr pe 32 de biți, găsesc primul byte > 0 și aplic ce am calculat mai sus
- Pot ține rezultatul pt 2 bytes și atunci am nevoie de doar 2 operații

Range Minimum Query Soluții

- **Ținem pentru fiecare element puterile lui 2 și răspundem în O(1)**

	0	1	2	3	4	5	6	7	8	9
min	3	9	2	8	5	3	8	7	6	11
min2	3	2	2	5	3	3	7	6	6	11
min4	2	2	2	3	3	3	6	6	6	11
min8	2	2	2	3	3	3	6	6	6	11

- Query în **O(1)**? Cum?
  - 1 6 → min(min(1,4), min(3,6)) - prin urmare, putem face 2 query-uri [a, a + log(b-a)], [b - log(b-a) + 1, b].
  - 20, 1000 → min [Q(20, 531), Q(489, 1000)] → 2 query-uri de mărime 512

Range Minimum Query Soluții

- **Ținem pentru fiecare element puterile lui 2 și răspundem în O(1)**

- Query în **O(1)**? Cum?
  - 1 6 → min(min(1,4), min(3,6)) - prin urmare, putem face 2 query-uri [a, a + log(b-a)], [b - log(b-a) + 1, b].
  - 20, 1000 → min [Q(20, 531), Q(489, 1000)] → 2 query-uri de mărime 512
  - **Atenție! Ideea funcționează doar pentru minim**, nu și pentru sumă, deoarece o parte din interval (489, 531) este inclus în ambele query-uri. Dacă vrem să calculăm minimul, acest lucru nu este o problemă, dar pentru sume, da!
  - Pentru sumă, trebuie să facem O(logn) query-uri, deci probabil arborii de intervale sunt mai buni, deoarece au tot O(log n) pe query, dar au O(n) memorie suplimentară și O(n) construcție.

Range Minimum Query Soluții

- Complexitate **O(n log n)** memorie și preprocesare și **O(1)** query
  - Se poate obține O(n) preprocesare și memorie suplimentară și O(1) pe query.
    - Link
    - Implementare
      - RMQ pe Infoarena: https://pastebin.com/7a8uVdtP
      - https://leetcode.com/problems/range-sum-query-immutable/
        - am realizat la un seminar că problema nu cerea minim, prin urmare nu se putea rezolva în O(1) pe query. Vă dau două rezolvări diferite
        - cu Batog: https://pastebin.com/5RUrVpVi
        - Totuși, problema se rezolvă cu sume parțiale în O(1) pe query

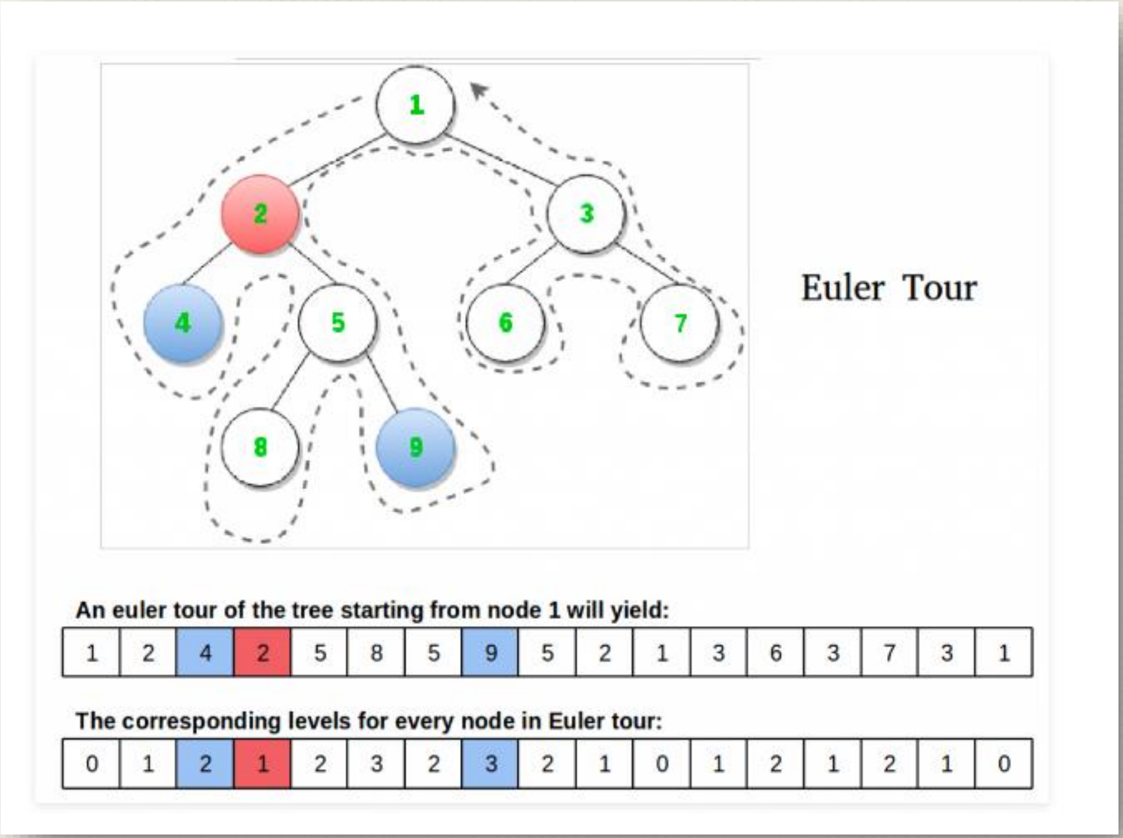
LCA → RMQ

Problema LCA se poate reduce la RMQ

- Descriere pe larg
- Principiul este o liniarizare a arborelui

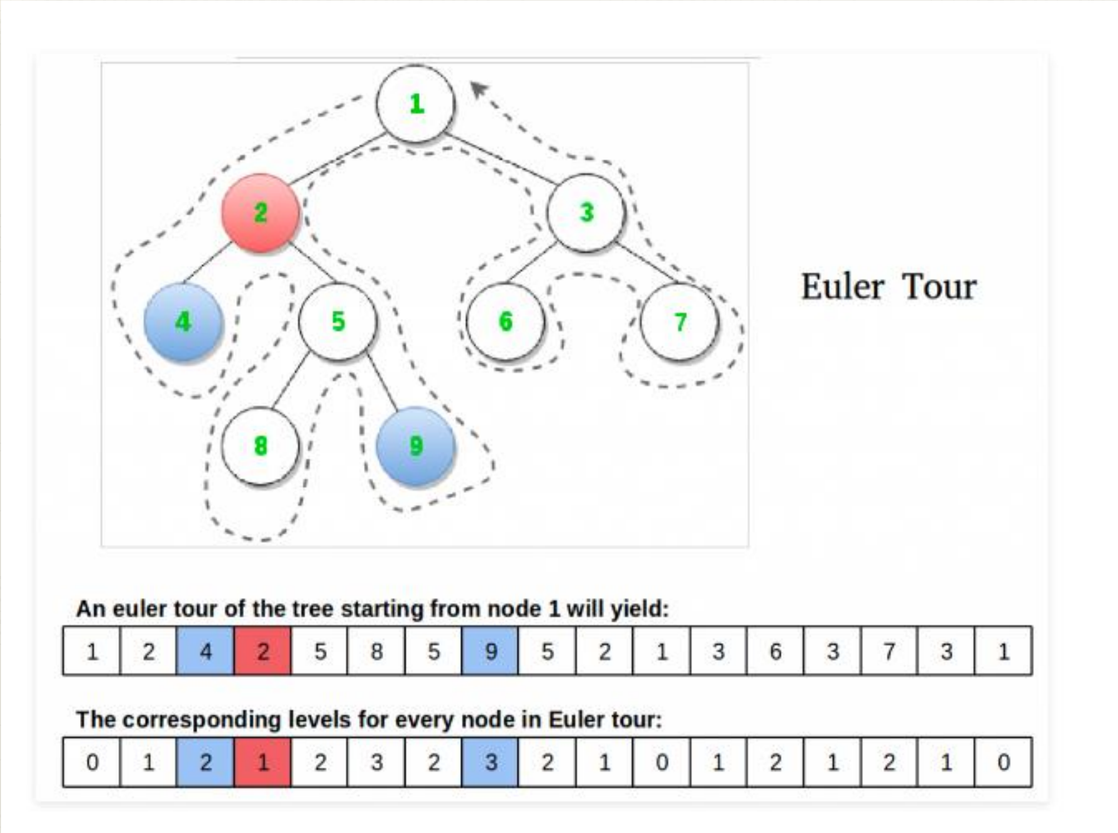
LCA → RMQ

- Începem o parcurgere RSD din rădăcină și scriem fiecare nod **de fiecare dată când trecem prin el.**
- Pentru fiecare nod, reținem și distanța de la el la rădăcină.
- Pentru fiecare nod, mai reținem și prima sa apariție în parcurgerea Euler…
- De exemplu, pentru 4 e poziția 2, pentru 9 este 7.



LCA → RMQ

- LCA(i,j) este RMQ(first[i], first[j])...
- LCA(4,9) va fi RMQ pe parcurgerea Euler între primele apariții ale lui 4 și 9
- Deci RMQ(2,7)...
- RMQ se va face pe vectorul de distanțe, până la rădăcină (2, 7), prin urmare obținem distanța 1 către rădăcina care corespunde nodului 2.
- Orice drum între 4 și 9 trece prin 2, dar nu mai sus de 2!



Final