# 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)

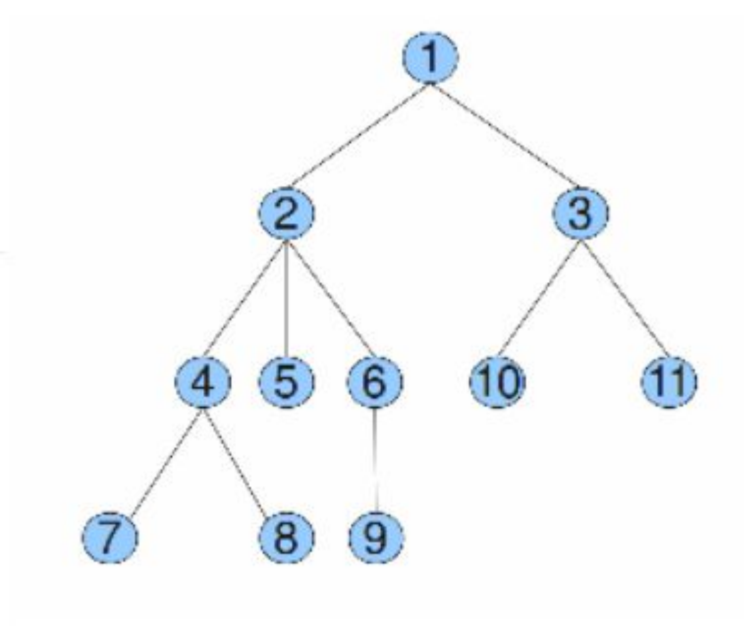
4 9 → 2

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

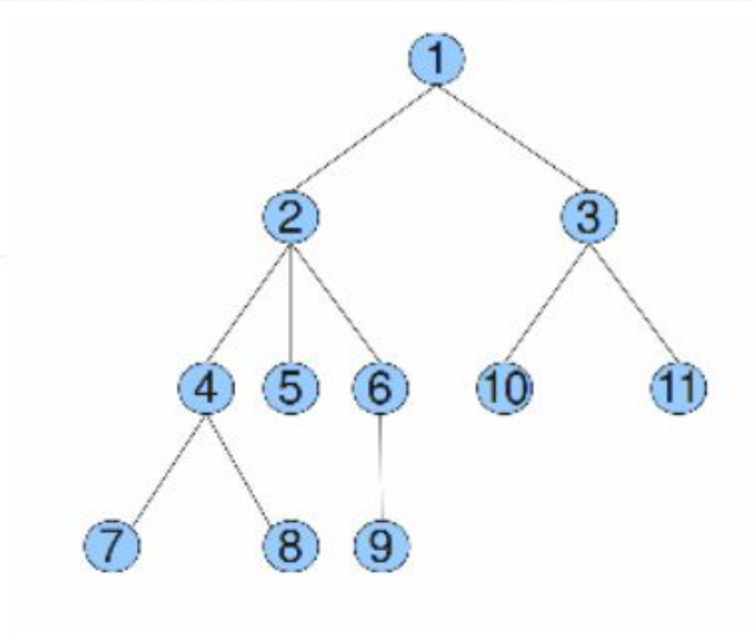
9 1 → 6

9 2 -> 2

9 3 -> 1

6 4 → -1

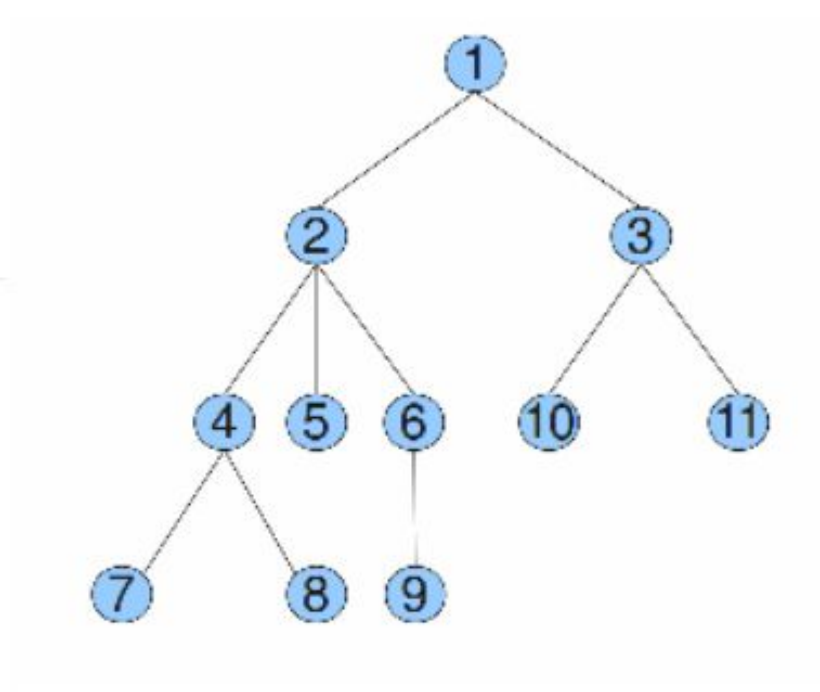
10 1 → 3



# 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(n), parcurgand din tata in tata la fiecare querry.

Sau 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}`

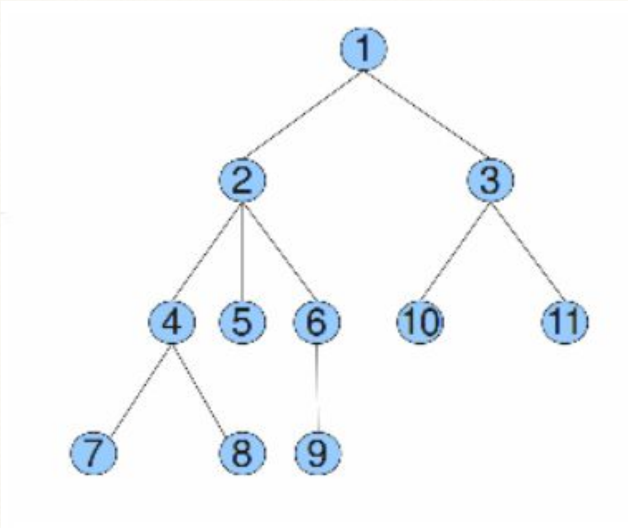
# Lowest Ancestor - soluții

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 $O(n^2)$ și răspuns $O(1)$.

# Lowest Ancestor - soluții

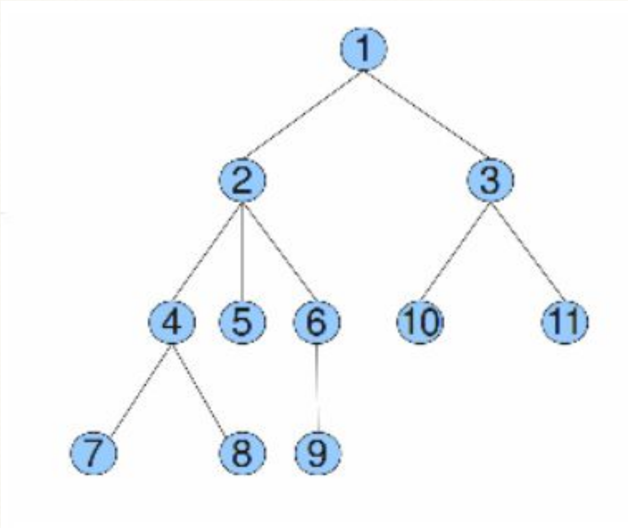
Se dă un arbore. Răspundeți cât mai eficient la întrebări de genul: **Se dă un nod și un întreg k. Care este strămoșul de nivel k al nodului dat?**

Sau pot folosi sqrt decomposition:

Țin tatăl de ordin radical din n.

Dacă radical din n este 100 și eu țin din 100 în 100:

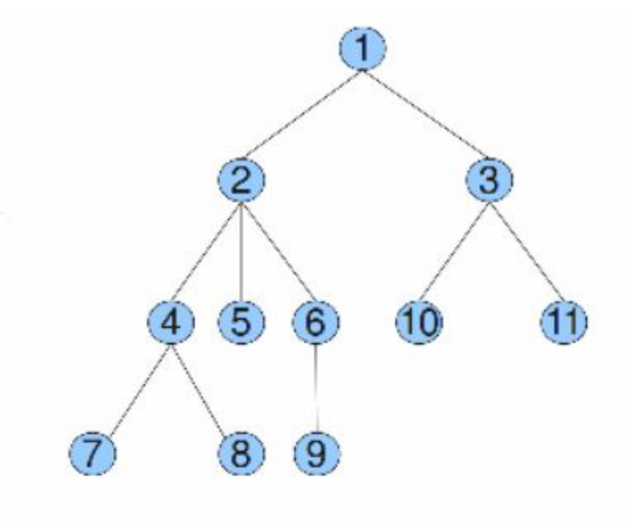
Tatăl 300 este `tata100[tata100[tata100[x]]];`



# 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 și 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.

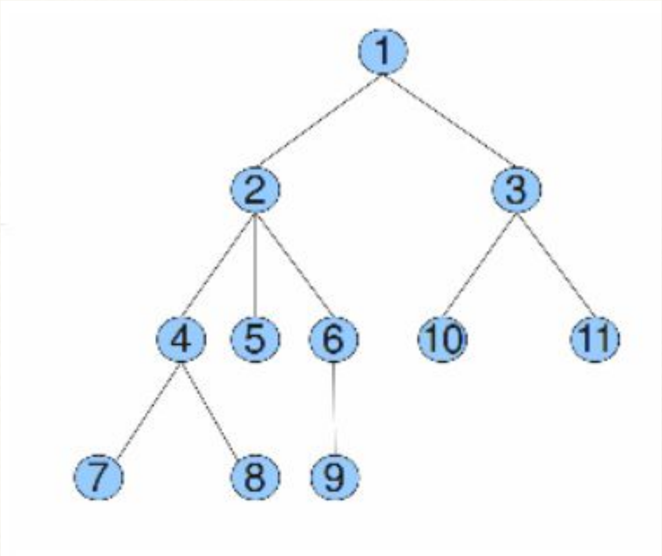
# Lowest Ancestor - soluții

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

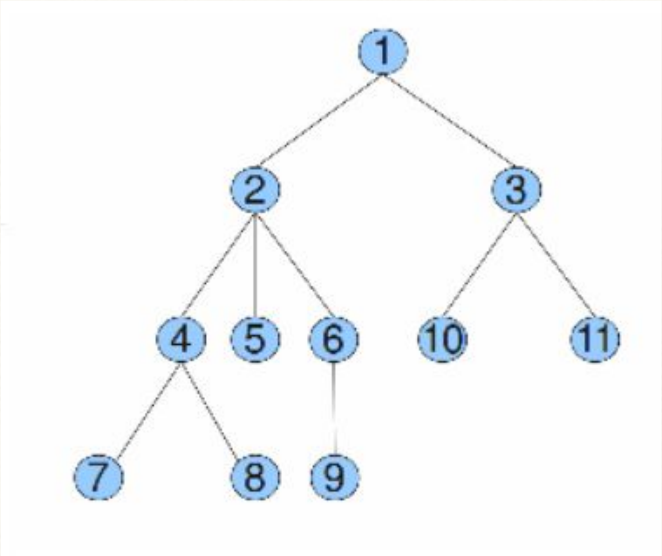
## O(log n) query și O(n log n) memorie

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?



# Lowest Ancestor

## O(log n) query și O(n log n) memorie

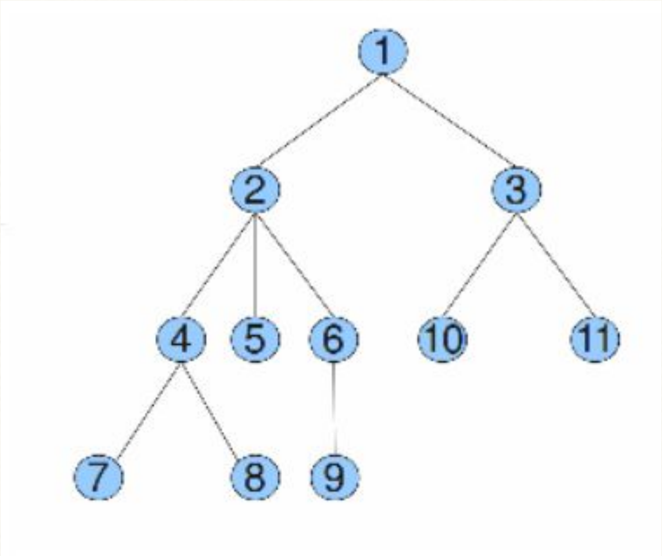
Pentru fiecare nod, țin tații de înălțime 1, 2, 4, 8, 16…

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];
}
```



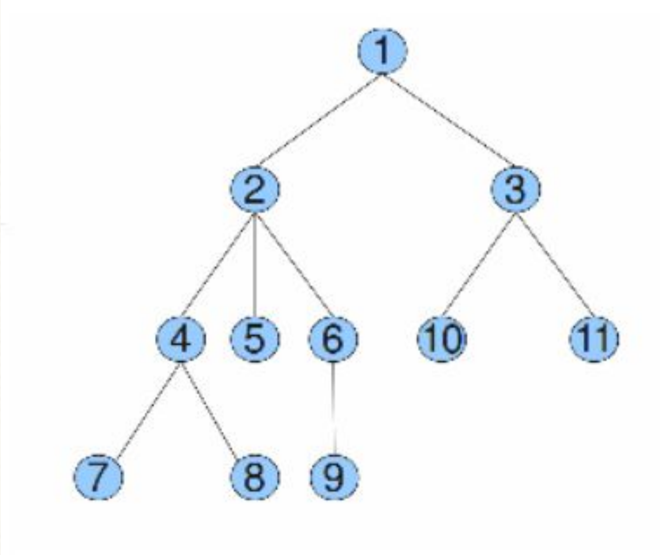
# Lowest Ancestor

## 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…

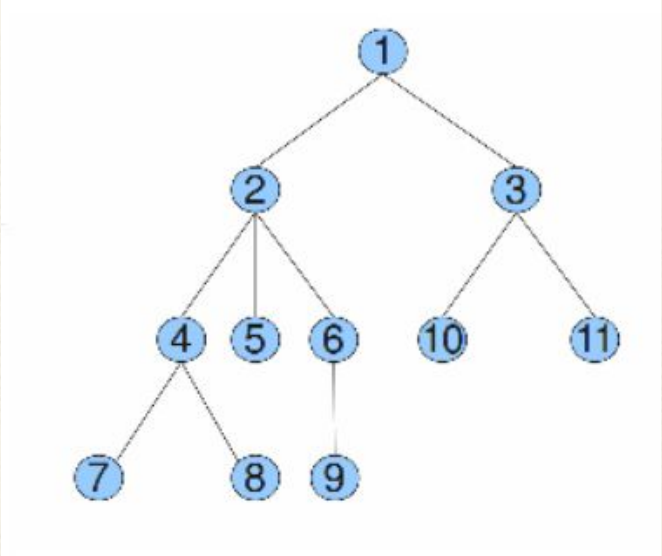
Pentru 7 → 4, 2, -1, -1 ….

Pentru 6 → 2, 1, -1, -1 ….

Cum calculăm al k-lea strămoș?

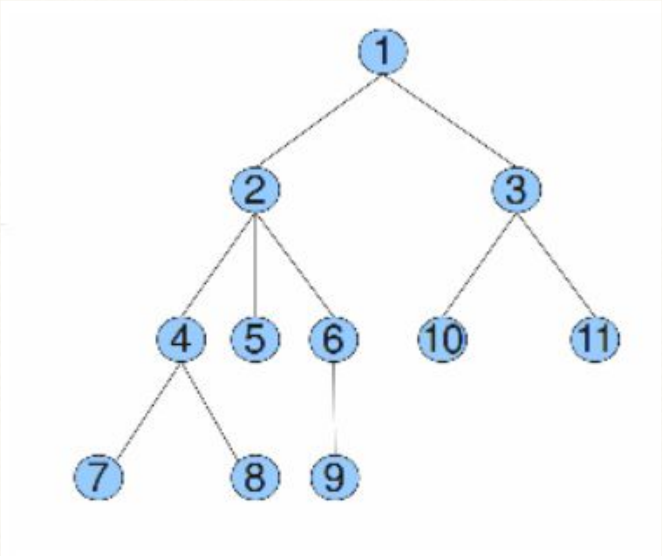
tata(x, 14) = tata(tata8[x], 6) = tata(tata4[tata8[x]],2)

= tata2[tata4[tata8[x]]



# Lowest Ancestor
## O(log n) query și O(n log n) memorie

Complexitate ?

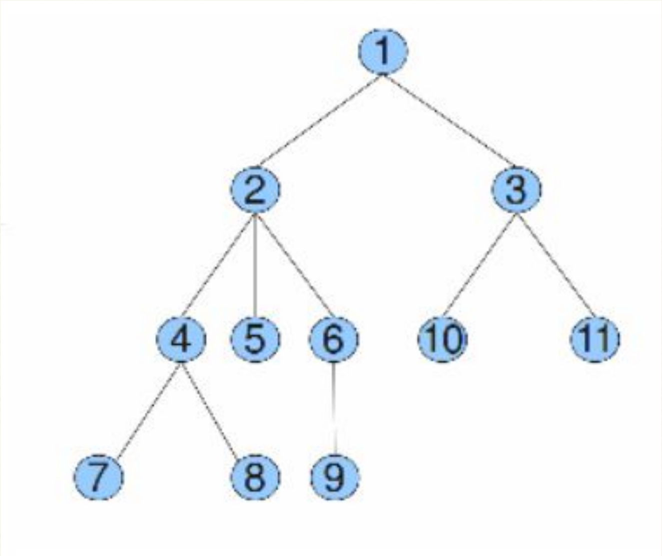


# Lowest Ancestor

## 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 | 3 | 9 | 2 | 8 | 5 | 3 | 8 | 7 | 6 | 11 |
| min2 | 3 | 2 | 2 | 5 | 3 | 3 | 7 | 6 | 6 | 11 |
| min4 | | | | | | | | | | |
| min8 | | | | | | | | | | |

# 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
  - 2 9
  - 3 6

# 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 marime 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 marime 512
  - **Atentie ideea functioneaza doar pentru minim**, nu si pentru suma deoarece o parte din interval (489, 531) este inclus in ambele query-uri. Daca vrem sa calculam minimul acest lucru nu este o problema dar pentru sume da!
  - Pentru suma trebuie sa 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 suplimentare, si O(n) constructie.

# 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 ca problema nu cerea minim, prin urmare nu se putea rezolva in O(1) pe query va dau doua rezolvari diferite
        - cu batog: https://pastebin.com/5RUrVpVi
        - Totusi problema se rezolva cu sume partiale in O(1) pe query

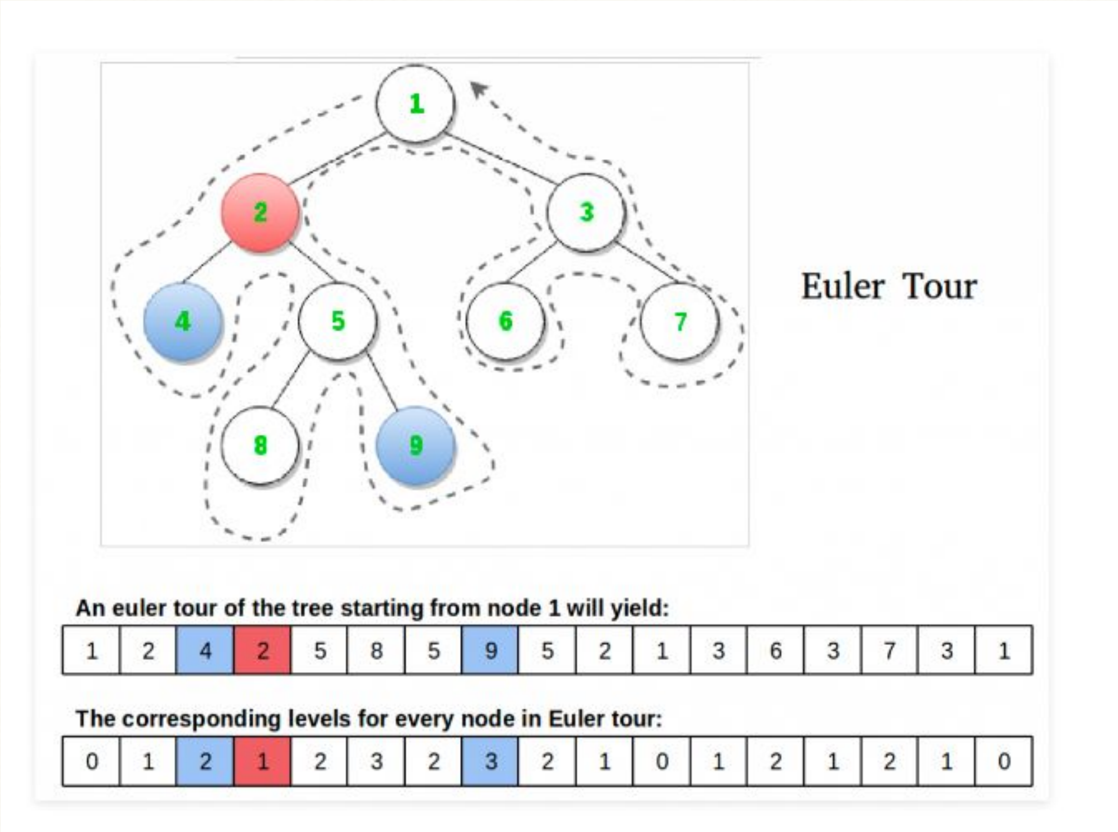
# LCA -> RMQ

Problema LCA se poate reduce la RMQ

- Descriere pe larg
- Principiul este o liniarizare a arborelui
- 

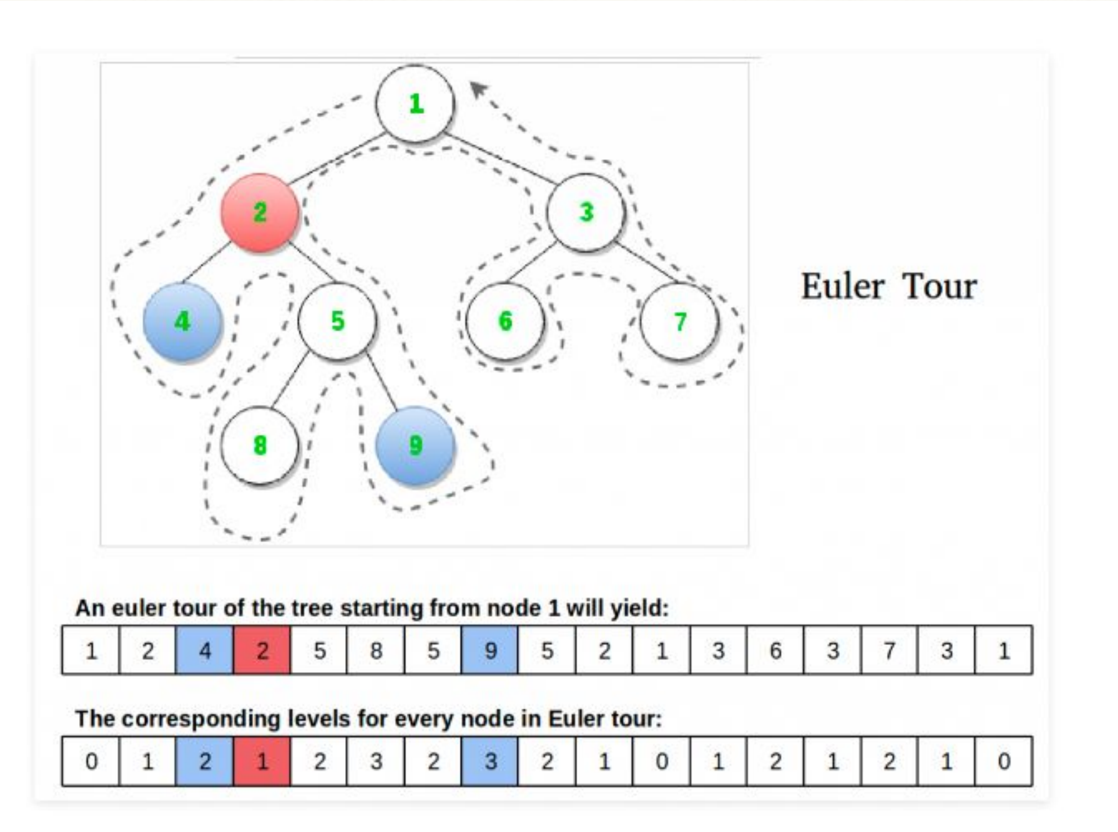
# LCA -> RMQ

- Incepem o parcurgere RSD din radacina si scriem fiecare nod **de fiecare data cand trecem prin el.**
- Pentru fiecare nod retinem si distanta de la el la radacina.



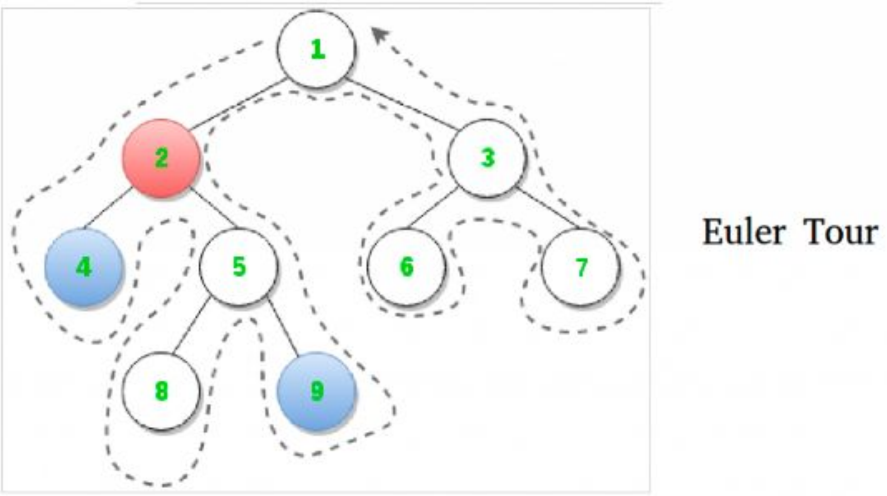
# LCA -> RMQ

- Incepem o parcurgere RSD din radacina si scriem fiecare nod **de fiecare data cand trecem prin el.**
- Pentru fiecare nod retinem si distanta de la el la radacina.
- Pentru fiecare nod mai retinem si prima sa aparitie in pargurgerea Euler…
- De exemplu pentru 4 e pozitia 2, pentru 9 e 7



# LCA -> RMQ

- LCA(i,j) este RMQ(first[i], first[j])… LCA(4,9) va fi RMQ pe parcurgerea Euler intre primele aparitii a lui 4 si 9
- Deci RMQ(2,7)…
- RMQ se va face pe vectorul de distante pana la radacine (2, 7), prin umrare obtinem distanta 1 catre radacina care corespunde nodului 2.
- Orice drum intre 4 si 9 trece prin 2, dar nu mai sus de 2!




An euler tour of the tree starting from node 1 will yield:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 2 | 5 | 8 | 5 | 9 | 5 | 2 | 1 | 3 | 6 | 3 | 7 | 3 | 1 |

The corresponding levels for every node in Euler tour:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 1 | 2 | 3 | 2 | 3 | 2 | 1 | 0 | 1 | 2 | 1 | 2 | 1 | 0 |