prob → Range minimum query (RMQ)

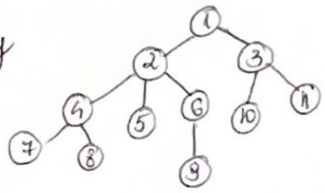
ex: Se dă un vector. Care este cel mai mic element din intervalul i, j?

prob → Lowest common ancestor (LCA)

ex: Se dă un arbore. Se dau două noduri într-un arbore. Găsiți cel mai apropiat strămoș comun.

prob → Lowest ancestor (LA)

ex: Se dă un arbore. Se dă un nod și un întreg k. Care este strămoșul de nivel k al nodului dat.



O(h): din tată în tată la fiecare query

O(1): dacă reții strămoșul de nivel j al lui i

→ memorie: O(n * h)

2 1 → 1
9 1 → 6
9 2 → 2

- O(n) query, memorie
- O(√n) query, O(n) memorie (Batog)
- O(log n) query, O(n log n) memorie
  - → pt. fiecare nod ține minte tații de ordinul 1, 2, 4, 8, 16, ...
    (pt 6: 2, 1, -1, -1 ...)
    (1) (2) (4) (8)
  - → al k-lea strămoș
    - → similar cu căutarea binară
    - → sărim cu puterea lui 2 cea mai mare
      7 3 → sărim 2 pași până la 2
      2 1 → sărim 1 pas → 1

rez → RMQ

- O(n) query
- Șirul lui Batog - O(√n) pe query
- ținem pt. fiecare element puterile lui 2 și răspundem similar LA în log n

ex: Se dă un nr $n \leq 10^9$. Vrem calculez $\log n$ în $O(1)$

LCA → RMQ

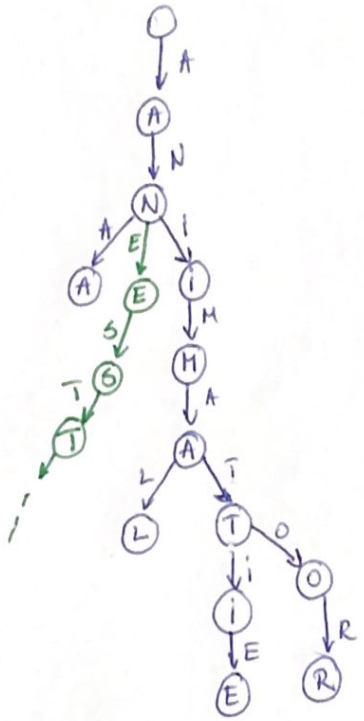
↳ parcurgere RSD din rădăcină și scriem fiecare nod de fiecare dată când trecem prin el

↳ pt. fiecare nod - reținem și distanța de la el la rădăcină
- prima sa apariție în parcurgerea Euler

↳ LCA(i, j) = RMQ(first[i], first[j])

Curs 14 → TRIE

Trie cu cuvintele ana, animator, animatie, animal



Memorare

↳ fiecare nod - 26 vecini, pt. fiecare literă
↳ alfabet mare : fiecare nod tine un hash-map — pt. fiecare literă tine pointerul către nodul cu acea literă

Inserare O(1)

↳ pornim din rădăcină și, la fiecare literă, mergem cu nodul coresp. literei, eventual creăm acel nod
(inserăm anestezie)

Căutare O(1)

↳ pornim de la rădăcină și mergem, la fiecare pas, pe litera coresp.
↳ prefix max : căutăm ultimu. până nu găsim nod coresp. acelei litere