Algoritmi si Structuri de Date

- seminar IV -

RMQ.

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

Se da un vector cu *N* elemente. Scrieti un program care raspunde la *M* intrebari de tipul "Care este elementul minim din intervalul [*x*,*y*]?"

Input:	Output:	Explicatie:
12 4	0	2 9 **0** 5 1 8
2 9 0 5 1 8 7 3 10 4 11 6	1	**1** 8 7
0 5	2	**2** 9
4 6	3	7 **3** 10 4 11 6
0 1	6	11 **6**
6 10		
10 11		

Algoritm brute-force.

Pentru fiecare intrebare parcurgem intervalul respectiv si la final afisam valoarea minima.
Complexitate: O(M * N) (in cel mai rau caz primim intrebari de tipul 0 N-1).

Square root decomposition.

Rezolvare: Vom imparti sirul initial in parti de $\sqrt{N}$ si pentru fiecare parte vom memora care este elementul minim din acea parte. Cand vom modifica un element, vom verifica daca trebuie sa modificam si valoarea care se afla in partea respectiva.

2	9	0	5	1	8	7	3	10	4	11	6
0			1			3			4		

Exemplu: Operatia 0 5

2	9	0	5	1	8	7	3	10	4	11	6
0			1			3			4		

Pasi:

- Calculam din ce parte face primul index.
  - Indexul 0 se afla in partea 0 / $\sqrt{N}$ = 0 / 3 = 0.
  - Partea se termina pe pozitia 0 + $\sqrt{N} - 1$ = 0 + 3 – 1 = 2.
  - Noi vrem sa aflam minimul din intervalul [0, **5**], 2 <= 5, putem updata minimul
- Minim = 0
- Verificam daca putem sa sarim peste urmatoarea parte
  - Urmatoarea parte incepe pozitia 3.
  - Partea se termina pe pozitia 3 + $\sqrt{N}$ - 1 = **5**
  - **5** <= **5**, deci putem updata minimul
- Minim = min (Minim, 1) = 0

Acest algoritm poate fi modificat usor pentru a suporta si operatii dinamice (elementele se pot modifica si se cere elementul minim dupa ce a avut loc o modificare).

Modificarile constau in actualizarea noului element si daca acesta este egal cu valoarea din vectorul de jos, se va parcurge partea respectiva, recalula noul minim si se va salva in partea respectiva.

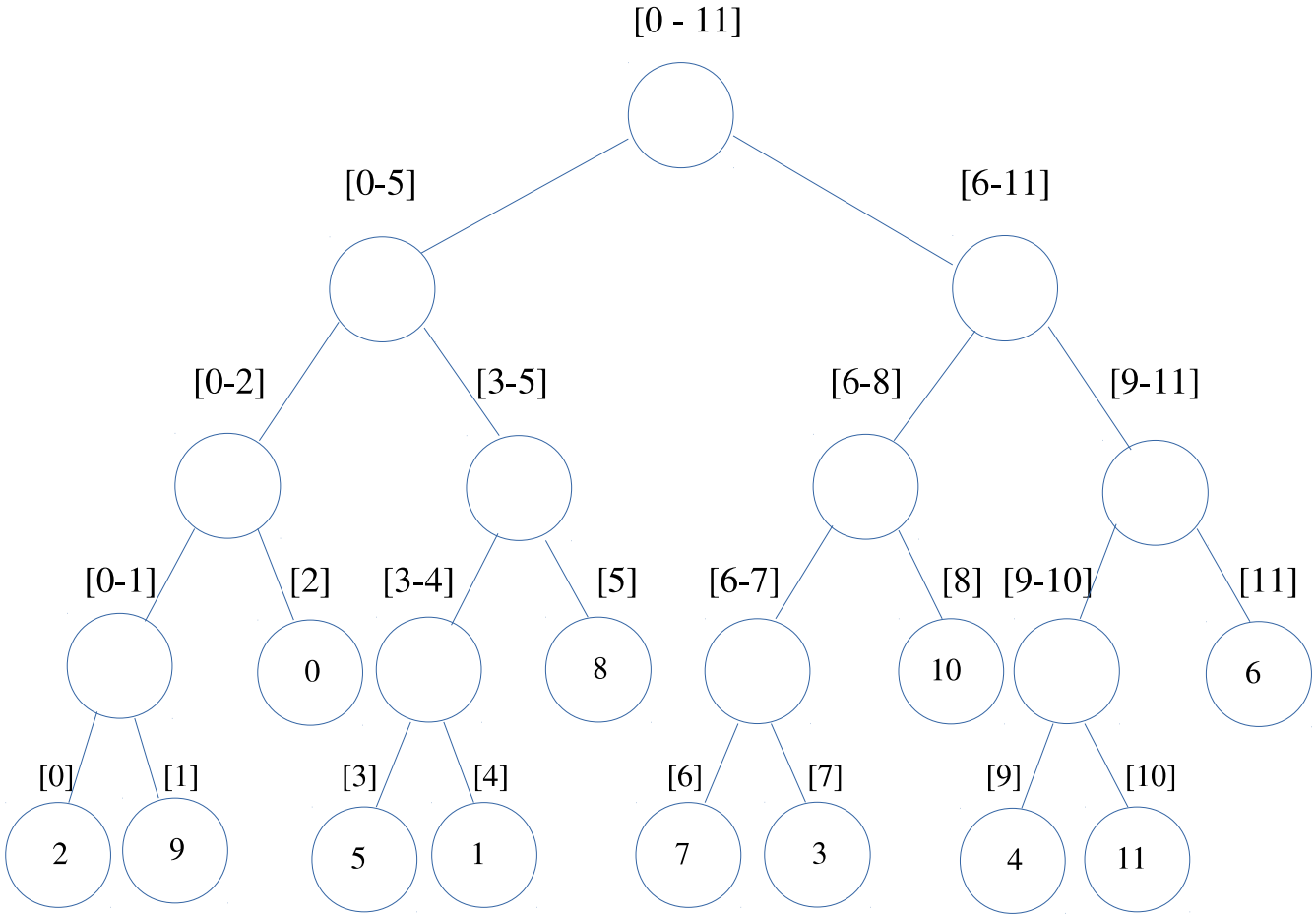
Complexitate: O (N + M * $\sqrt{N}$)

Arbori de intervale.

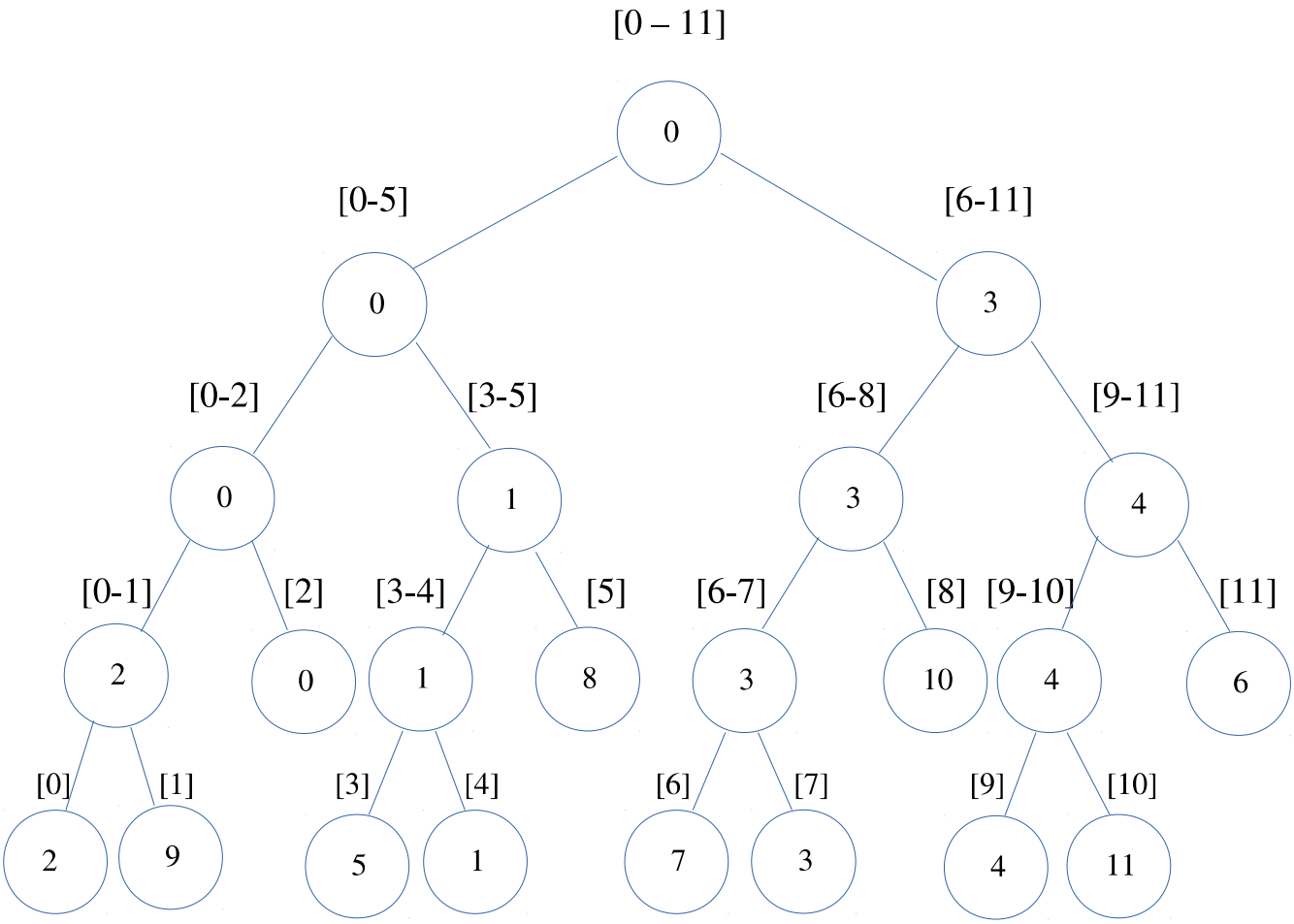
Un arbore de intervale este un arbore binar in care fiecare nod contine un interval si o valoare asociata acelui interval.

0	1	2	3	4	5	6	7	8	9	10	11
2	9	0	5	1	8	7	3	10	4	11	6

Pentru a construi arborele, in primul rand vom pune in fiecare frunza cate un element din vectorul de mai sus (in functie de pozitia sa in vector).



Vom completa restul nodurilor folosind operatia care ne intereseaza aplicata pe cei doi fii (in cazul de fata operatia **min**) si vom continua pana ajungem la radacina.



Daca avem un query pe un anumit interval, vom pleca din radacina si vom restrange cautarea pana intervalul cautat este inclus intr-un nod. Daca acest lucru nu se intampla, inseamna ca rezultatul nostru se poate obtine combinand valorile din mai multe noduri, a caror intervale reunite ne dau intervalul cautat.

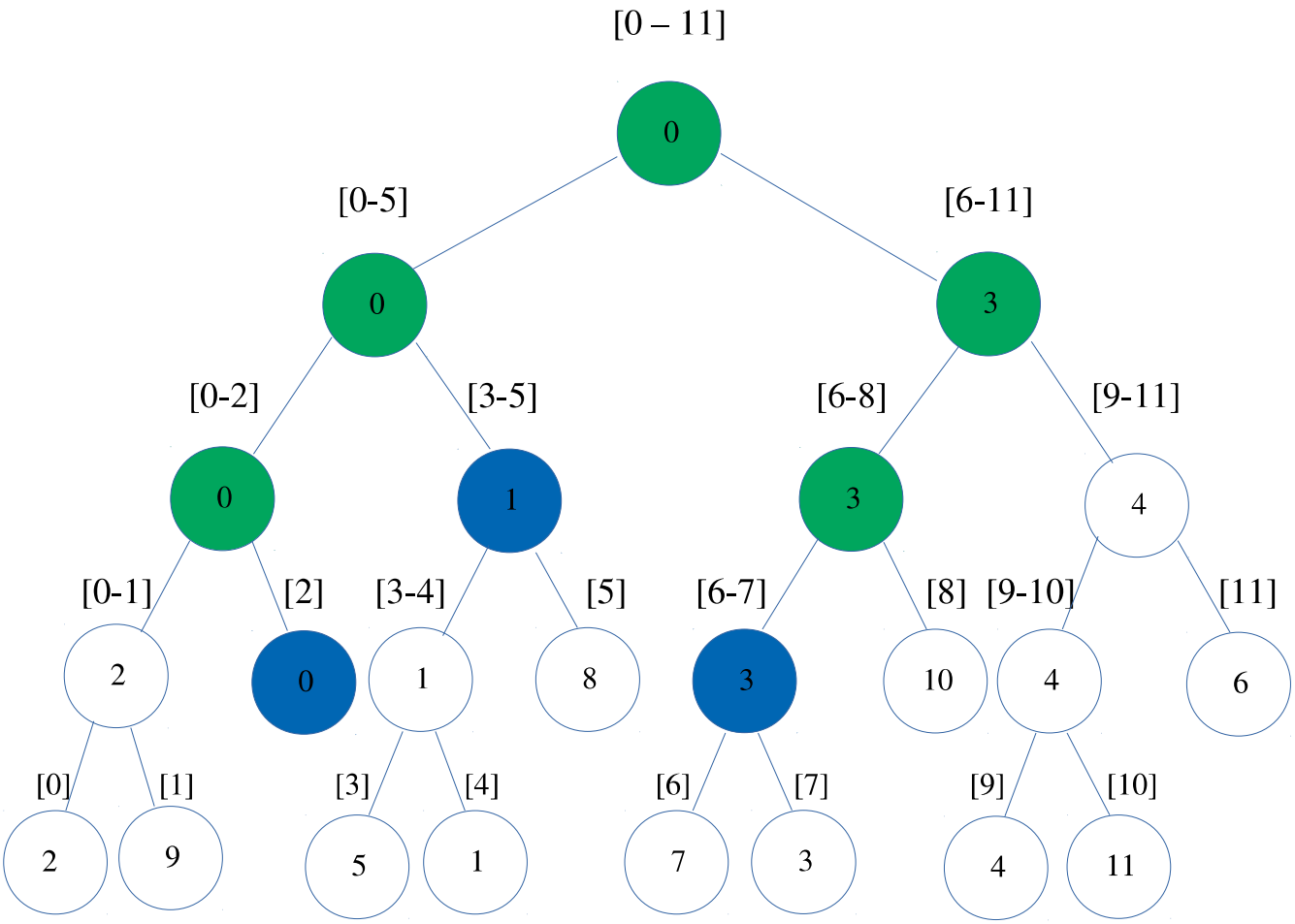
Exemplu I: Intervalul **0 – 5**

- Pornim de la intervalul radacinii **0 – 11**
- Este **0 – 11** inclus in **0 – 5**? Nu, mijlocul intervalului **0 – 11** este (**0 + 11**) / 2 = **5**
  - Se intersecteaza **0 – 5** cu **0 – 5**? Da, continuam recursiv:
    - Este **0 – 5** inclus in **0 – 5**? Da, returnam valoarea din nodul **0 – 5** → **0**
  - Se intersecteaza **0 – 5** cu **6 – 11**? Nu, nu vom merge mai departe in arbore.
- Elementul minim din intervalul **0 – 5** este **0**.

Exemplu II: Intervalul **2 – 7**

- Pornim de la intervalul radacinii **0 – 11**
- Este **0 – 11** inclus in **2 – 7**? Nu, mijlocul intervalului **0 – 11** este (**0 + 11**) / 2 = **5**
  - Se intersecteaza **2 – 7** cu **0 – 5**? Da, continuam recursiv:
    - Este **0 – 5** inclus in **2 – 7**? Nu, mijlocul lui **0 – 5** este (0 + 5) / 2 = **2**
      - Se intersecteaza **2 – 7** cu **0 – 2**? Da, continuam recursiv:
        - Este **0 – 2** inclus in **2 – 7**? Nu, mijlocul lui **0 – 2** este (0 + 2) / 2 = **1**
          - Se intersecteaza **2 – 7** cu **0 – 1**? Nu, nu mergem mai departe.
          - Se intersecteaza **2 – 7** cu **2 – 2**? Da, returnam valoarea din [2] → **0**.
      - Se intersecteaza **2 – 7** cu **3 – 5**? Da, continuam recursiv:
        - Este **3 – 5** inclus in **2 – 7**? Da, returnam valoarea din [3-5] → **1**.
      - Am terminat cu intervalul **0 – 5**, returnam minimul dintre **0** si **1**, adica **0**.
  - Se intersecteaza **2 – 7** cu **6 – 11**? Da, continuam recursiv:
    - Este **6 – 11** inclus in **2 – 7**? Nu, mijlocul lui **6 – 11** este (6 + 11) / 2 = **8**
      - Se intersecteaza **2 – 7** cu **6 – 8**? Da, continuam recursiv:
        - Este **6 – 8** inclus in **2 – 7**? Nu, mijlocul lui **6 – 8** este (6 + 8) / 2 = **7**
          - Se intersecteaza **2 – 7** cu **6 – 7**? Da, continuam recursiv:
            - Este **6 – 7** inclus in **2 – 7**? Da, return valoarea din [6-7] → **3**
          - Se intersecteaza **2 – 7** cu **8 – 8**? Nu, nu mergem mai departe.
          - Am terminat intervalul **6 – 8**, returnam valoarea minima gasita **3**.
      - Se intersecteaza **2 – 7** cu **9 – 11**? Nu, nu mergem mai departe.
      - Am terminat intervalul **6 – 11**, returnam valoarea minima **3**.
  - Am terminat intervalul **0 – 11**. Returnam valoarea minima combinata de pe cele doua noduri, adica min(**0**, **3**).
- Minimul pe intervalul **2 – 7** este 0.

Puteti vedea mai jos cu **albastru** care au fost nodurile care au fost inspectate pentru a gasi minimul pe intervalul 2-7 si cu **verde** prin ce noduri a trecut algoritmul.



De asemenea, aceasta solutie functioneaza si daca au loc operatii dinamice de modificare a elementelor. Practic, trebuie sa modificam un element care se afla intr-o frunza si apoi modificam toate nodurile care se afla pe drumul dintre radacina si frunza respectiva. Cum arborele este echilibrat, au loc maxim O(log n) operatii (atat pentru o modificare cat si pentru un query).

Complexitate: O(N log N + M log N)

Range minimum query.

Un algoritm mai eficent de a rezolva aceasta problema este de a precalcula pentru fiecare index care este elementul minim intre indexul respectiv si index + 2, index + 4, index + 8 si asa mai departe si cand are loc un query, vom returna minimul dintre doua parti care se intersecteaza.

Vom construi o matrice RMQ[i][j] cu semnificatia:

- prima componenta este puterea la care il ridicam pe 2
- a doua componenta este indexul de la care incepem
- RMQ[i][j] = minimul pe intervalul A[j, j + 2^i -1]

Prima linie a matricei este vectorul de intrare (se cere minimul de pe pozitiile A[j, j + 2^0 -1]). Pe a doua linie vom scrie pentru fiecare index care este valoarea minima intre pozitiile j si j + 2^1-1, adica minimul intre j si j + 1. Pe a treia linie vom scrie care este valoarea minima intre pozitiile j si j + 2^2-1, deci minimul intre j si j + 3. Pe a patra linie, scriem minimul intre j si j + 7.

2	9	0	5	1	8	7	3	10	4	11	6
2	0	0	1	1	7	3	3	4	4	6	
0	0	0	1	1	3	3	3	4			
0	0	0	1	1							

RMQ[3][4] = min (RMQ[2][4], RMQ[2][4 + 2^{3-1}])

Aceasta matrice se poate construi dupa urmatoarea relatie recurenta:

RMQ[i][j] = A[j] , daca i = 0

min (RMQ[i - 1][j], RMQ[i – 1][j + 2^{i-1}]) , altfel

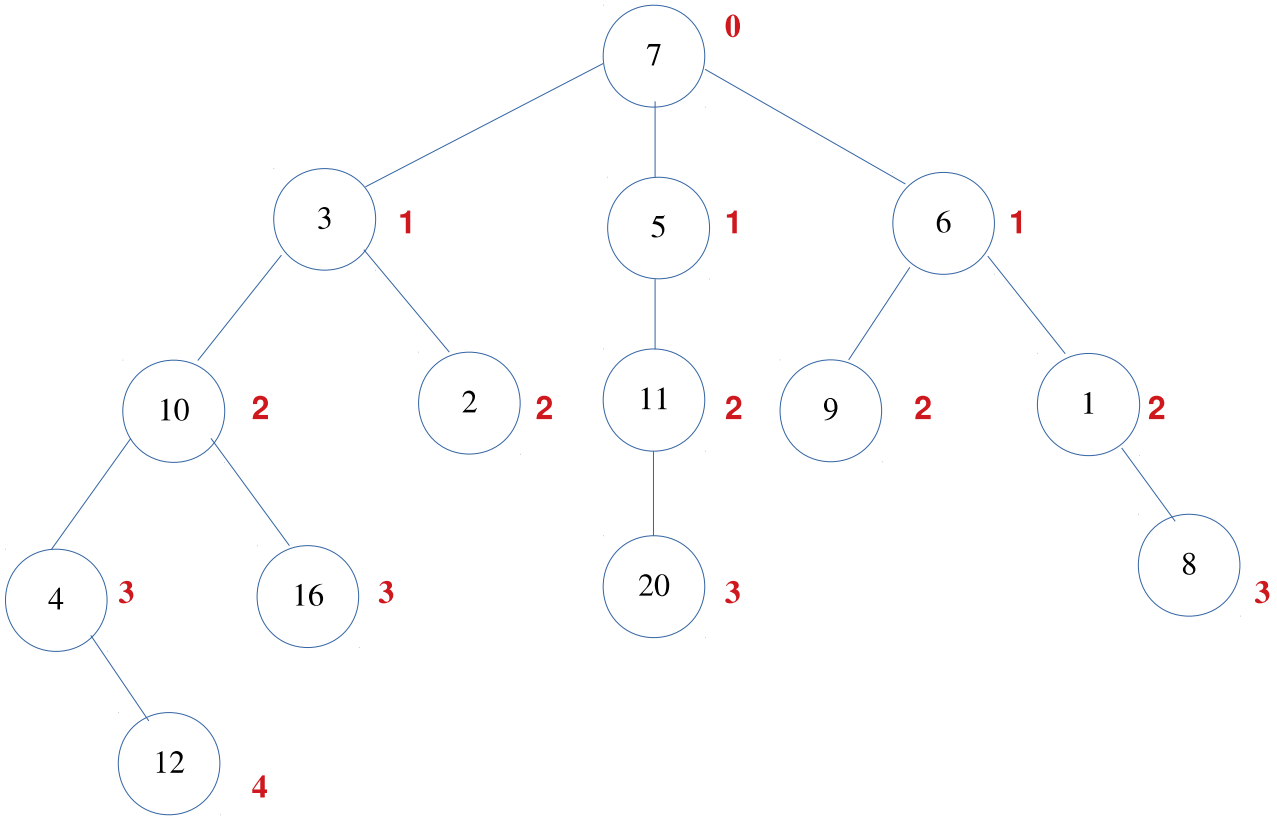
Pentru un query, de exemplu 2 7, trebuie sa gasim doua intervale intercalate a caror reuniune este intervalul 2 7. Pentru aceasta, gasim cea mai mica putere a lui 2 care nu depaseste intervalul ($\log_2$(7 – 2)), care este 2, si returnam valoarea minima dintre RMQ[2][2] si RMQ[2][7 – 2^2 + 1] care este min(0, 1) = 0.
Complexitate: O(N log N + M)

LCA.

Site infoarena: https://infoarena.ro/problema/lca

Se da un arbore cu radacina T. Cel mai apropiat stramos comun a doua noduri u si v este nodul w care este stramos al ambelor noduri u si v si are cea mai mare adancime în T.

Exemplu:



LCA (3, 2) = 3	LCA (4, 2) = 3	LCA (5, 6) = 7
LCA (2, 3) = 3	LCA (11, 5) = 5	LCA (12, 8) = 7
LCA (3, 11) = 7	LCA (9, 8) = 6	LCA (12, 16) = 10

Algoritm brute-force.

Un algoritm brute-force care rezolva aceasta problema este acela care in primul rand face o parcurgere BFS a grafului si ii atribuie fiecarui nod o adancime (indicata cu rosu). Apoi daca primim de exemplu query-ul LCA(12, 8), observam ca 12 se afla la adancime 4 si 8 se afla la adancime 3, urcam cu elementul care se afla la o adancime mai mare, 12 → 4 si apoi mergem simultan cate un nivel cu cele doua noduri (4 si 8) si verificam daca elementele sunt egale:

- LCA(12, 8) =
- LCA(4, 8) =
- LCA(10, 1) =
- LCA(3, 6) =
- LCA(7, 7) = 7

Complexitate query: O(N)

Arbori binari de cautare.

Pentru arborii binari de cautare putem gasi un algoritm mai eficient datorita proprietatii ca in partea stanga a unui nod se afla elemente mai mici si in partea dreapta se afla elemente mai mari.

Cand primim un query, vom pleca din radacina si in functie de valoarea din radacina si cele doua valori ale query-ului vom alege urmatorul pas astfel:

- daca cele doua valori ale query-ului sunt mai mici, LCA-ul se afla in partea stanga si vom continua recursiv cu fiul stang;
- daca cele doua valori ale query-ului sunt mai mari, LCA-ul se afla in partea dreapta si vom continua recursiv cu fiul drept;
- daca cele doua valori se afla de o parte si de alta a valorii din radacina, sau un element are aceeasi valoare ca nodul curent, returnam nodul fiind LCA-ul cautat.

Complexitate query: O(h)

Arbori oarecare (Square root decomposition).

O alta solutie mai eficienta care functioneaza pentru arbori oarecare o gasim folosind tehnica square root decomposition. Astfel fiecare nod din arbore mai contine un pointer catre nodul care se afla mai sus cu $\sqrt{N}$ pozitii.

Algoritmul incepe ca cel brute-force, marcand pentru fiecare nod adancimea sa si apoi in loc sa urcam cate un element putem sa sarim peste $\sqrt{N}$ elemente daca cele doua valori sunt diferite. Daca observam ca, daca am incerca sa sarim $\sqrt{N}$ elemente, obtinem acelasi element, atunci avansam cate o pozitie, intrucat LCA-ul se afla la o distanta mai mica sau egala cu $\sqrt{N}$.

Complexitate query: O($\sqrt{N}$)

Arbori oarecare (Cautare binara).

Pentru a putea obtine o solutie mai buna, am putea salva intr-o matrice cine este cel de-al 2^i-lea stramos al fiecarui nod (acest lucru presupune ca valorile din noduri sunt destul de mici – daca nu sunt putem face o normalizare).

D[i][j] = cel de-al 2^i-lea stramos al nodului j.

Pentru exemplul de mai sus, matricea D ar arata astfel:

D[0][j] = p_j, pentru orice nod j, D[0][j] este parintele acelui nod

D[1][10]	= 7	D[1][4]	= 3	D[2][12]	= 7
D[1][2]	= 7	D[1][16]	= 3	D[2][j]	= 7, ∀ nod j
D[1][11]	= 7	D[1][20]	= 5		
D[1][9]	= 7	D[1][8]	= 6		
D[1][1]	= 7	D]1][12]	= 10		

Daca dorim sa aflam LCA-ul a doua noduri, vom ajunge mai intai la acelasi nivel, si apoi vom efectua o cautare binara (daca verificam valoarea din mijloc si aceasta are aceeasi valoare pentru ambele noduri, vom restrange cautarea binara in stanga – in jos; altfel in dreapta – in sus).

Ex: LCA (12, 8) = LCA(D[0][12], 8) = LCA(4, 8) // am urcat la acelasi nivel
efectuam cautare binara pentru stanga = 0, dreapta = 2; mijloc = **1**
D[1][4] = 3 | valorile sunt diferite, deci vom
D[1][8] = 6 | restrange cautarea la dreapta
efectuam cautare binara pentru stanga = 2, dreapta = 2; mijloc = **2**
D[2][4] = 7 | valorile sunt indentice, deci vom
D[2][8] = 7 | restrange cautarea la stanga
LCA(12, 8) = 7.

LCA(4, 16) // sunt deja la acelasi nivel
efectuam cautare binara pentru stanga = 0, dreapta = 2; mijloc = **1**
D[1][4] = 3 | valorile sunt indentice, deci vom
D[1][16] = 3 | restrange cautarea la stanga

efectuam cautare binara pentru stanga = 0, dreapta = 0; mijloc = **0**

D[0][4] = 10 | valorile sunt indentice, deci vom

D[0][16] = 10 | restrange cautarea la stanga

LCA(4, 16) = 10.

Pentru a obtine matricea D putem sa folosim urmatoarea recurenta:

D[**i**][**j**] = D[**i – 1**][D[**i – 1**][**j**]]

"Cel de-al 2^i stramos al nodului **j** este cel de-al 2^{i-1} stramos al nodului care se afla la 2^{i-1} niveluri mai sus de nodul **j**."

Complexitate query: O(log N)

Arbori oarecare (Eficient).

Pentru a rezolva eficient aceasta problema avem nevoie de reprezentarea Euler a arborelui (aceasta reprezentare se poate obtine efectuand o parcurgere DFS a arborelui, dar de fiecare data cand trecem printr-un nod, il mai scriem odata; astfel obtinem o parcurgere "continua" a arborelui).

Vom scrie mai jos parcurgerea Euler si nivelul pe care se afla fiecare nod:

7	3	10	**4**	**12**	**4**	**10**	**16**	**10**	**3**	**2**	3	7	5	**11**	**20**	**11**	**5**	**7**	**6**	9	6	1	8
0	1	2	**3**	**4**	**3**	**2**	**3**	**2**	**1**	**2**	1	0	1	**2**	**3**	**2**	**1**	**0**	**1**	2	1	2	3

Daca dorim sa aflam LCA-ul a doua noduri, vom gasi prima aparitie a celor doua noduri si vom efectua RMQ pe acea portiune:

LCA(4, 2) = 3 // nivelurile sunt: **3 4 3 2 3 2 1 2**; elementul minim este 1 => nod 3

LCA(12, 4) = 4 // nivelurile sunt: **3 4**; elementul minim este 3 => nod 4

LCA(11, 6) = 7 // nivelurile sunt: **2 3 2 1 0 1**; elementul minim este 0 => nod 7

Complexitate query: O(1)