Arbori de intervale

Apostol Ilie-Daniel
Popescu Ştefan-Alexandru

Cuprins

- Prezentare generala arbori de intervale
- Lazy Propagation
- Arbori de intervale impliciti
- Arbori de intervale 2D
- Arbori de intervale persistenti

Arbori de intervale (AINT) / Motivatie

Point Update & Range Minimum

$a_1 \quad a_2 \quad a_3 \quad \ldots \quad a_N$

update (pos, val) -> a_{pos} = val

query (left, right) -> min($a_{left...right}$) = ?

Arbori de intervale (AINT) / Motivatie

Point Update & Range Minimum

$a_1 \quad a_2 \quad a_3 \quad \dots \quad a_N$

update (pos val) -> a_{pos} = val

query (left, right) -> min($a_{left...right}$) = ?

Solutii:

O(N^2) pe update si O(1) pe query

Arbori de intervale (AINT) / Motivatie

Point Update & Range Minimum

$a_1 \quad a_2 \quad a_3 \quad \dots \quad a_N$

update (pos val) -> a_{pos} = val

query (left, right) -> min($a_{left...right}$) = ?

Solutii:

O(N^2) pe update si O(1) pe query

O(1) pe update si O(N) pe query

Arbori de intervale (AINT) / Motivatie

Point Update & Range Minimum

$a_1 \quad a_2 \quad a_3 \quad \dots \quad a_N$

update (pos val) -> a_{pos} = val

query (left, right) -> min($a_{left...right}$) = ?

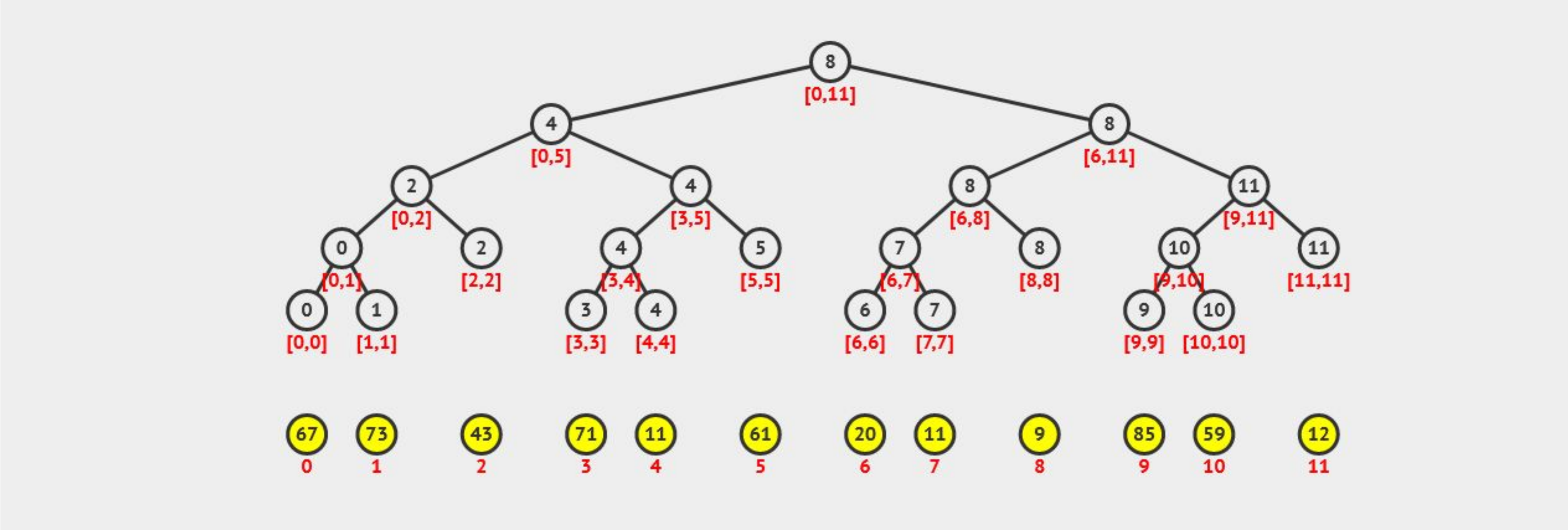
Solutii:

O(N^2) pe update si O(1) pe query

O(1) pe update si O(N) pe query

O(logN) pe update si O(logN) pe query

Arbori de intervale (AINT) / Ilustrare



Credits

Arbori de intervale (AINT) / Structura

Arbori de intervale (AINT) / Structura

- Update (functie recursiva)

Arbori de intervale (AINT) / Structura

- Update (functie recursiva)
  - Conditie de oprire (si aplicarea operatiei asupra frunzei)

Arbori de intervale (AINT) / Structura

- Update (functie recursiva)
  - Conditie de oprire (si aplicarea operatiei asupra frunzei)
  - Aflarea fiului in care ne ducem

Arbori de intervale (AINT) / Structura

- Update (functie recursiva)
  - Conditie de oprire (si aplicarea operatiei asupra frunzei)
  - Aflarea fiului in care ne ducem
  - Combinarea raspunsurilor din subarbori (PULL)

Arbori de intervale (AINT) / Structura

- Update (functie recursiva)
  - Conditie de oprire (si aplicarea operatiei asupra frunzei)
  - Aflarea fiului in care ne ducem
  - Combinarea raspunsurilor din subarbori (PULL)
- Query (functie recursiva)

Arbori de intervale (AINT) / Structura

- Update (functie recursiva)
  - Conditie de oprire (si aplicarea operatiei asupra frunzei)
  - Aflarea fiului in care ne ducem
  - Combinarea raspunsurilor din subarbori (PULL)
- Query (functie recursiva)
  - Conditie de oprire si intoarcerea raspunsului pentru interval

Arbori de intervale (AINT) / Structura

- Update (functie recursiva)
  - Conditie de oprire (si aplicarea operatiei asupra frunzei)
  - Aflarea fiului in care ne ducem
  - Combinarea raspunsurilor din subarbori (PULL)
- Query (functie recursiva)
  - Conditie de oprire si intoarcerea raspunsului pentru interval
  - Aflarea subarborilor care contin intervalul de query

Arbori de intervale (AINT) / Structura

- Update (functie recursiva)
  - Conditie de oprire (si aplicarea operatiei asupra frunzei)
  - Aflarea fiului in care ne ducem
  - Combinarea raspunsurilor din subarbori (PULL)
- Query (functie recursiva)
  - Conditie de oprire si intoarcerea raspunsului pentru interval
  - Aflarea subarborilor care contin intervalul de query
  - (posibil) Combinarea raspunsurilor

Arbori de intervale (AINT) / Alte probleme

Range Update & Point Query

$a_1 \quad a_2 \quad a_3 \quad \dots \quad a_N$

update (left, right, val) -> $a_{left..right}$ += val

query (pos) -> a_{pos} = ?

Arbori de intervale (AINT) / Alte probleme

Point Update Minimum Point Query

$a_1 \quad a_2 \quad a_3 \quad \ldots \quad a_N$

update (pos, val) -> a_{pos} += val

query () -> minimum pos, $a_{pos} > 0$

Lazy Propagation / Motivatie

Range Update & Range Minimum

$a_1 \quad a_2 \quad a_3 \quad \dots \quad a_N$

update (left, right, val) -> $a_{left..right}$ += val

query (left, right) -> min($a_{left...right}$) = ?

Lazy Propagation / Motivatie

Range Update & Range Minimum

a_1 a_2 a_3 ... a_N

update (left, right, val) -> $a_{left..right}$ += val

query (left, right) -> min($a_{left...right}$) = ?

Ce inseamna lazy?

Lazy Propagation / Motivatie

Range Update & Range Minimum

$a_1 \quad a_2 \quad a_3 \quad \dots \quad a_N$

update (left, right, val) -> $a_{left..right}$ += val

query (left, right) -> min($a_{left...right}$) = ?

Ce inseamna lazy?

=> O noua metoda : PUSH

Lazy Propagation / Aplicatie

Reuniune de dreptunghiuri

Se dau N dreptunghiuri (care se afla intr-un chenar de marime HxW)care au laturile paralele cu axele de coordonate. Acopera reuniunea lor tot chenarul?

Lazy Propagation / Aplicatie

Reuniune de dreptunghiuri

Se dau N dreptunghiuri (care se afla intr-un chenar de marime HxW)care au laturile paralele cu axele de coordonate. Acopera reuniunea lor tot chenarul?

Baleiere dupa o coordonata, fie x aceasta

Lazy Propagation / Aplicatie

Reuniune de dreptunghiuri

Se dau N dreptunghiuri (care se afla intr-un chenar de marime HxW)care au laturile paralele cu axele de coordonate. Acopera reuniunea lor tot chenarul?

Baleiere dupa o coordonata, fie x aceasta

=> Arbore de intervale cu lazy propagation

Lazy Propagation / Aplicatie

Reuniune de dreptunghiuri

Se dau N dreptunghiuri (care se afla intr-un chenar de marime HxW)care au laturile paralele cu axele de coordonate. Acopera reuniunea lor tot chenarul?

Baleiere dupa o coordonata, fie pe ox
=> Arbore de intervale cu lazy propagation
La momentul x:
updateSum(y1, y2, +1) daca x1 = x
updateSum(y1, y2, -1) daca x2 + 1 = x
queryMin(1, W) > 0 ?

Lazy Propagation / Limitari

Range Min & Max Update | Range Sum Query

$a_1 \quad a_2 \quad a_3 \quad \dots \quad a_N$

updateMin(left, right, val) -> $a_{left..right}$ min= val

updateMax(left, right, val) -> $a_{left..right}$ max= val

querySum(left, right) -> sum($a_{left...right}$) = ?

Lazy Propagation / Limitari

Range Min & Max Update | Range Sum Query

$a_1 \quad a_2 \quad a_3 \quad \dots \quad a_N$

updateMin(left, right, val) -> $a_{left..right}$ min= val

updateMax(left, right, val) -> $a_{left..right}$ max= val

querySum(left, right) -> sum($a_{left...right}$) = ?

Segment Tree Beats

Implementare cu pointeri / AINT implicit

Sa consideram urmatoarea situatie:

Avem un sir a de dimensiune n, care initial contine doar valori de 0. Avem 2 tipuri de query-uri:

1) set p val -> a[p] = val
2) sum l r -> a[l] + [l + 1] + ... + a[r]

Implementare cu pointeri / AINT implicit

Sa consideram urmatoarea situatie:

Avem un sir a de dimensiune n, care initial contine doar valori de 0. Avem 2 tipuri de query-uri:

1) set p val -> a[p] = val
2) sum l r -> a[l] + [l + 1] + ... + a[r]

Restrictii:

1) N <= 1e9

Implementare cu pointeri / AINT implicit

Sa consideram urmatoarea situatie:

Avem un sir a de dimensiune n, care initial contine doar valori de 0. Avem 2 tipuri de query-uri:

1) set p val -> a[p] = val
2) sum l r -> a[l] + [l + 1] + … + a[r]

Restrictii:

1) N <= 1e9
2) Rezolvare online

Implementare cu pointeri / AINT implicit

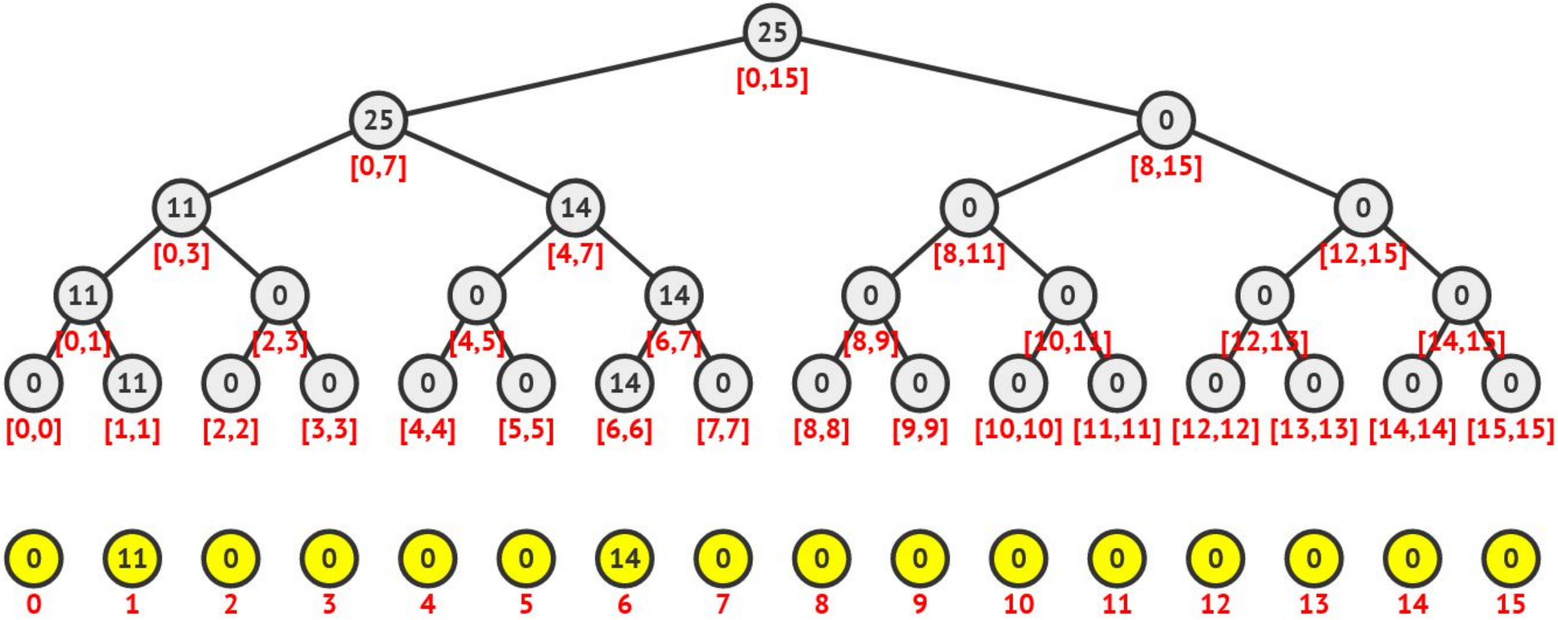
Idei:

Avand in vedere ca n <= 10^9, nu putem salva intreg arborele.

Fiecare element din sir e initial 0, deci putem pastra din arborele de intervale doar nodurile a caror valoare a fost schimbata la un moment dat. Daca modificam o valoare ce a fost initial 0, putem crea noduri noi pe drumul de la radacina pana la frunza asociata ei.

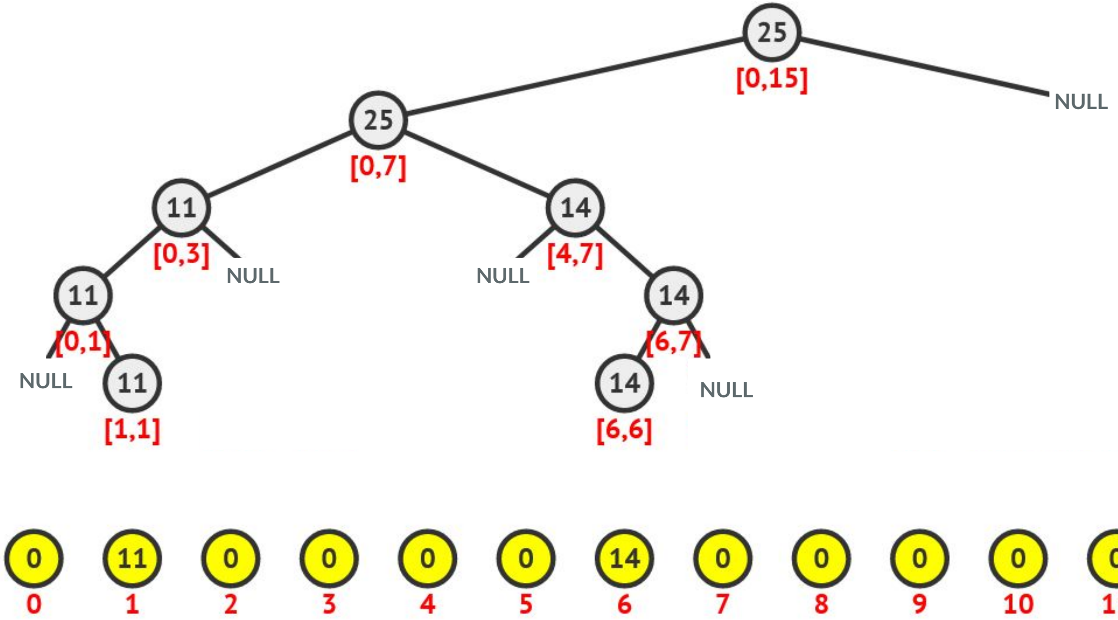
Implementare cu pointeri / AINT implicit

Idei:
Avand in vedere ca fiecare element din sir e initial 0, putem pastra din arborele de intervale doar nodurile a caror valoare a fost schimbata. Astfel, arborele:



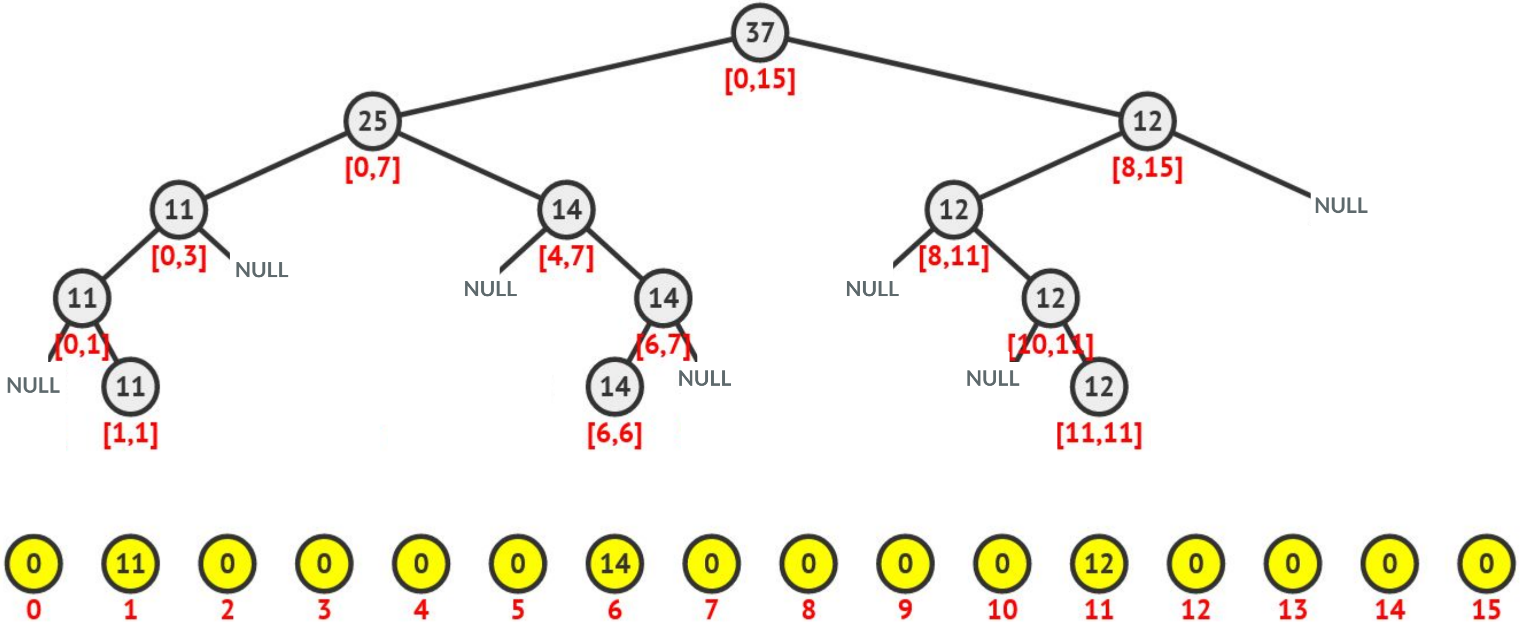
Implementare cu pointeri / AINT implicit

Devine:



Implementare cu pointeri / AINT implicit

Daca vrem sa setam o valoare noua, pur si simplu adaugam nodurile de pe drumul de update si modificam nodurile deja existente:



Implementare cu pointeri / AINT implicit

Complexitate spatiu / timp: O(Q * log (N))

AINT 2D

Consideram urmatoarea situatie:

Avem o matrice 2D de dimensiune N x M asupra careia efectuam diverse operatii:

1) set x y val ->mat[x][y] = val
2) sum x1, y1, x2, y2 -> sum(plan[x1..x2][y1..y2])

AINT 2D

Consideram urmatoarea situatie:
Avem o matrice 2D de dimensiune N x M asupra careia efectuam diverse operatii:

1) set x y val ->mat[x][y] = val
2) sum x1, y1, x2, y2 -> sum(plan[x1..x2][y1..y2])

Restrictii:

1) $0 <= N, M <= 10^9$
2) Rezolvare online

AINT 2D

Idei:

1) Pentru fiecare linie am putea pastra cate un arbore de intervale (dinamic / implicit) care partitioneaza coloanele.
2) Date fiind dimensiunile matricii, liniile ar trebui si ele tinute dinamic (map <int, AINT>).
3) Complexitatea per update ar fi $O(\log(M) + \log(N))$, iar cea pe query ar fi $O(\log(N) + N * \log(M))$.

AINT 2D

Idei:

1) Pentru fiecare linie am putea pastra cate un arbore de intervale (dinamic / implicit) care partitioneaza coloanele.
2) Date fiind dimensiunile matricii, liniile ar trebui si ele tinute dinamic (map <int, AINT>).
3) Complexitatea per update ar fi $O(\log(M) + \log(N))$, iar cea pe query ar fi $O(\log(N) + N * \log(M))$.

4) Observam ca iterarea prin linii reaminteste de rezolvarea bruta a problemei prezentate la inceput (RMQ).
5) Astfel, apare ideea de a tine un arbore de intervale care sa partitioneze liniile, iar in fiecare nod al acestui arbore am putea tine un arbore de intervale care sa partitioneze coloanele.

AINT 2D

6) Problema in acest caz e urmatoarea: daca efectuam o operatie de update si ajungem intr-o frunza asociata arborelui unui nod din primul arbore ce salveaza un interval de lungime > 1, atunci nu prea stim de unde sa facem pull pentru valori.

AINT 2D

Solutie:

Consideram multimile partitionarilor coloanelor si liniilor (in sensul unui AINT) si facem produs cartezian intre ele. Astfel, obtinem noduri asociate unor submatrici [x1..x2] x [y1..y2] ([x1..x2] poate aparea ca interval intr-un AINT contruit pe linii, iar [y1..y2] poate aparea ca interval intr-un AINT construit pe coloane). Fiecare nod va tine 4 pointeri, 2 catre copiii obtinuti prin injumatatirea submatricii printr-o linie verticala, 2 catre copiii obtinuti prin injumatatirea submatricii printr-o linie orizontala.

AINT 2D - Update

```
update_y (nod, lx, rx, ly, ry, q):
    if ly == ry:
        if lx == rx:
            nod.val = q.v
        else:
            nod.val = combine(nod.lcx, nod.rcx)
    else:
        my = (ly + ry) / 2
        if q.y <= my:
            update_y(nod.lcy, lx, rx, ly, my, q)
        else:
            update_y (nod.rcy, lx, rx, my + 1, ry, q)
        nod.val = combine(nod.lcy, nod.rcy)
```

```
update_x(nod, lx, rx, ly, ry, q):
    if lx == rx:
        update_y(nod, lx, rx, ly, q)
    else:
        mx = (lx + rx) / 2
        if q.x <= mx:
            update_x(nod.lcx, lx, mx, ly, ry, q)
        else:
            update_x(nod.rcx, mx + 1, rx, ly, ry, q)
        update_y(nod, lx, rx, ly, q)
```

AINT Persistent

Se considera urmatoarea problema:

Avem un sir a de dimensiune n, ce contine valori cuprinse intre 1 si n. Avem un singur tip de query:

1) kth l r -> a k-a valoare in ordine crescatoare din subsirul a[l..r]

AINT Persistent

Idei:

Sa consideram o varianta simplificata a problemei, in care query-urile arata in felul urmator:

1) kth 1 r -> a k-a valoare in ordine crescatoare din subsirul a[1..r]; r creste de la un query la altul

AINT Persistent

Idei:

Sa consideram o varianta simplificata a problemei, in care query-urile arata in felul urmator:

1) kth 1 r -> a k-a valoare in ordine crescatoare din subsirul a[1..r]; r creste de la un query la altul

In aceasta situatie, am putea tine un arbore de intervale care sa asocieze fiecarei valori frecventa ei in subsirul a[1..r_actual]. Daca avem de rezolvat un query si r > r_actual, crestem r_actual si incrementam si frecventa valorilor noi adaugate la subsir.

AINT Persistent

Idei:
Sa consideram o varianta simplificata a problemei, in care query-urile arata in felul urmator:

1) kth 1 r -> a k-a valoare in ordine crescatoare din subsirul a[1..r]; r creste de la un query la altul

In aceasta situatie, am putea tine un arbore de intervale care sa asocieze fiecarei valori frecventa ei in subsirul a[1..r_actual]. Daca avem de rezolvat un query si r > r_actual, crestem r_actual si incrementam si frecventa valorilor noi adaugate la subsir.

Avand arborele de intervale, cum putem determina al k-lea element?

AINT Persistent

Revenind la problema initiala, stim ca putem rezolva un query daca avem un aint ce pastreaza informatie despre intervalul asociat query-ului. De asemenea, stim ca putem extinde intervalul pe care e construit un aint actualizand in acelasi timp si aint-ul intr-un mod eficient.

AINT Persistent

Revenind la problema initiala, stim ca putem rezolva un query daca avem un aint ce pastreaza informatie despre intervalul asociat query-ului. De asemenea, stim ca putem extinde intervalul pe care e construit un aint actualizand in acelasi timp si aint-ul intr-un mod eficient.

OBS:
Daca avem un aint asociat intervalului [1..r] si unul asociat intervalului [1..l - 1] si avand in vedere faptul ca lucram cu frecvente, atunci putem defini aint-ul asociat intervalului [l..r] ca fiind diferenta dintre celelalte 2 (practic, cand incrementam capatul din dreapta, fiecare nod al aint-ului se comporta precum o suma partiala).

AINT Persistent

Revenind la problema initiala, stim ca putem rezolva un query daca avem un aint ce pastreaza informatie despre intervalul asociat query-ului. De asemenea, stim ca putem extinde intervalul pe care e construit un aint actualizand in acelasi timp si aint-ul intr-un mod eficient.

OBS:
Daca avem un aint asociat intervalului [1..r] si unul asociat intervalului [1..l - 1] si avand in vedere faptul ca lucram cu frecvente, atunci putem defini aint-ul asociat intervalului [l..r] ca fiind diferenta dintre celelalte 2 (practic, cand incrementam capatul din dreapta, fiecare nod al aint-ului se comporta precum o suma partiala).