Elementul minim dintr-un interval (RMQ)

Careja Alexandru-Cristian

Facultatea de Automatica si Calculatoare, Universitatea Politehnica Bucuresti
Splaiul Independentei 290, 060029 Bucuresti, Romania
careja.alexandru@gmail.com

Rezumat. Fiind dat un vector cu N elemente intregi, RMQ cere elementul minim dintr-un subvector al vectorului initial. In acest studiu voi compara principalii algoritmi de rezolvare ai problemei gasirii elementului minim dintr-un interval. Voi explica algoritmii in cauza si voi da exemple de cateva interogari si rezultatul care ar trebui returnat. Dupa aceasta etapa, voi trece la implementarea efectiva a solutiilor (in Python) si verificarea acestora cu un set de teste care va pune la incercare eficienta si corectitudinea algoritmilor. Voi analiza complexitatea solutiilor si voi prezenta principalele avantaje si dezavantaje ale fiecareria. In urma analizei, voi incheia cu felul in care as aborda problema in practica si in ce situatii as opta pentru una din solutiile alese.

Cuvinte cheie: RMQ, LCA, interogare, interval, vector, matrice, arbore, complexitate, eficienta, asimptota, optim.

1 Introducere

1.1 Cerinta

Dat fiind un vector A cu N elemente de tip intreg, trebuie raspuns eficient la intrebarea: “Care este elementul minim in intervalul care incepe la pozitia x si se termina la pozitia y?”. Se considera ca se da vectorul si apoi se fac M interogari fara a se modifica intre timp vectorul.

Observatie: Voi considera pozitia 0 ca fiind pozitia primului element din vector

Tabel 1. Exemple de interogari RMQ

Vector input	X, Y	Elementul minim
10, 2, 3, 49, 33, 2, 5, 78, 12, 11, 90	**0,10**	**2**
10, 2, 3, 49, 33, 2, 5, 78, 12, 11, 90	**7, 10**	**11**
1, 2, 3, 4, 5, 6, 54, 213, -4, 40, 9, 9	**3, 11**	**-4**
1001, 2002, 900, 1200, 6000, 555	**0, 1**	**1001**
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ,13 ,14	**3,3**	**4**

1.2 Aplicatii practice

Range Minimum Query este un algoritm foarte des folosit pentru rezolvarea problemei LCA (Lowest Common Ancestor – gasirea celei mai mic stramos intr-un arbore, RMQ si LCA fiind doua probleme echivalente), dar nu numai. RMQ mai este folosit si in preprocesarea stringurilor si la suffix arrays, o noua structura de date care suporta cautari de stringuri aproape la fel de rapide ca suffix trees, dar care foloseste mai putina memorie si necesita mai putin efort de programare.

1.3 Solutii de rezolvare

- Sparse Table
- Cartesian Tree & algoritmul Farach-Colton and Bender
- Segment Tree

1.4 Evaluarea solutiilor

In vederea intocmirii unui set de teste, voi genera un set de date de intrare, cu diverse valori pentru N si M. Pentru fiecare fisier de intrare voi genera random N numere intregi si M indici x si y, cu conditia ca perechile de indici sa fie diferite pentru fiecare test.

Validarea corectitudinii o voi face prin compararea rezultatului generat de fiecare solutie analizata cu rezultatul obtinut prin metoda banala(cea care raspunde in O(n) la fiecare interogare), folosind o functie de comparare a fisierelor.

Eficienta solutiilor va fi evaluata pe de o parte, pe foaie prin calcularea complexitatii algoritmului, cat si pe calculator prin masurarea timpului de executie pe cateva fisiere de intrare mari, si prin compararea cu timpul algoritmului banal.

2 Prezentarea Solutiilor

2.1 Sparse Table

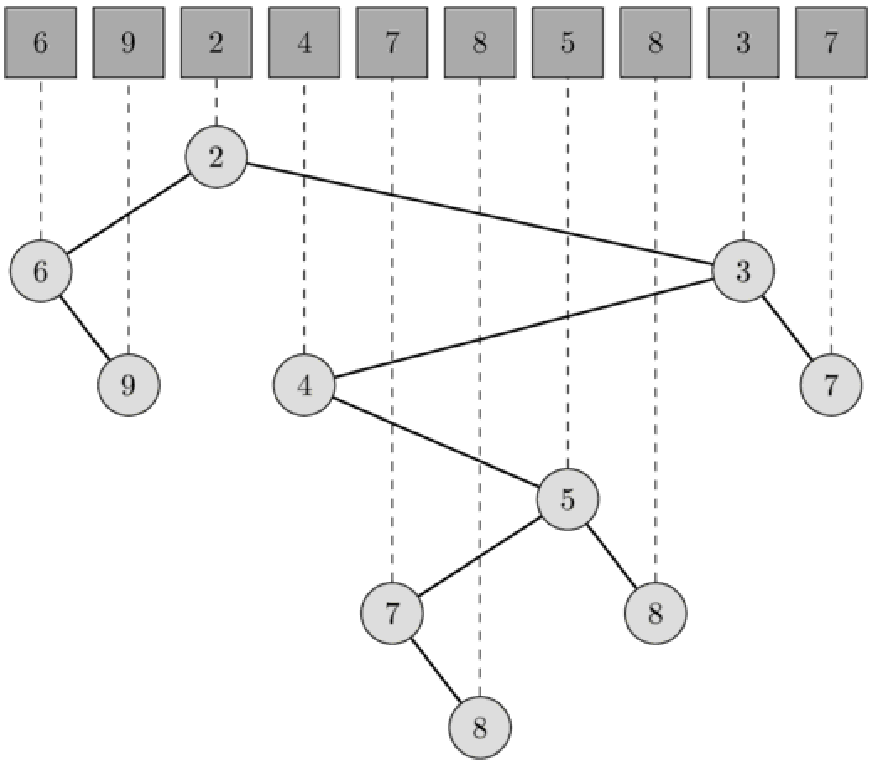
Este foarte bine cunoscut faptul ca orice numar nenegativ poate fi scris intr-un mod unic ca o suma de puteri descrescatoare ale lui doi (de exemplu 13 = 1101 in baza 2, adica 13 = 8 + 4 + 1). Pentru un numar n, exista maximum $\log_2 N$ termeni in suma. De asemenea, orice vector poate fi reprezentat in mod unic ca o reuniune de vectori de lungimi care sunt puteri descrescatoare ale lui doi (de exemplu [2: 23] = [2: 17] U [18: 21] U [22: 23], unde vectorul complet are lungimea de 22 si vectorii in care l-am descompus au lungimile 16, 4, 2). La uniunea vectorilor sunt maxim $\log_2 N$ vectori. Principala idee din spatele metodei Sparse Table este sa precomputeze toate raspunsurile de RMQ posibile pentru fiecare vector de lungime egala cu o putere de-a lui doi.

Raspunsul la o interogare diferita de cele precomputate pana acum se precomputeaza prin impartirea vectorului dat in vectori de lungime egala cu puteri ale lui doi, si uitandu-ne la raspunsurile deja precomputate si combinandu-le, ajungand la un raspuns final. In rezolvarea problemei RMQ prin metoda Sparse Table pornim de la ipoteza ca vectorul nu se va modifica de-a lungul seriei de interogari.

2.2 Cartesian Tree & algoritmul Farach-Colton and Bender

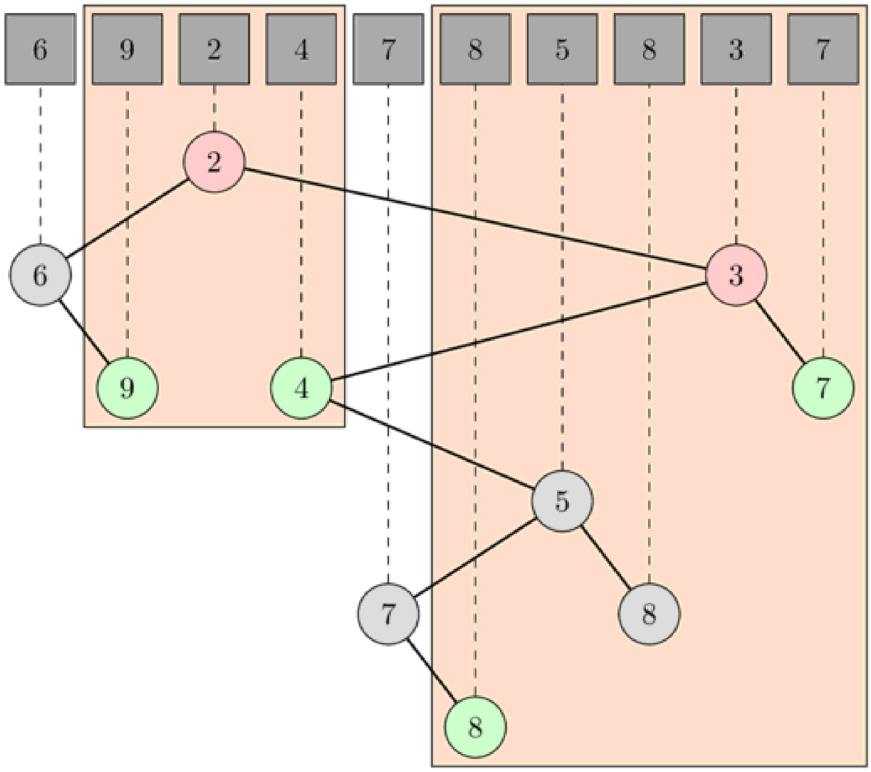
Pornim de la construirea unul arbore Cartezian din vectorul dat. Un arbore Cartezian este un arbore binar cu proprietatea de min-Heap (valoarea nodului parinte este mai mica decat decat valorile nodurilor copii ai sai) astfel incat parcurgerea inordine a arborelui sa viziteze nodurile in ordinea in care se gasesc in vectorul dat.

Imagine 1. Vector de 10 elemente si arborele Cartezian corespunzator



Valoarea minima din intervalul [l, r] este echivalenta cu cel mai mic stramos comun al nodului corespondent elementului de la indexul l si nodului corespondent elementului de la indexul r. In urmatoare imagine se poate vedea rezolvarea interogarii LCA (cel mai mic stramos comun) pentru inputul [1, 3] si [5, 9].

Imagine 2. Ilustrare a celui mai mic stramos comun



Construirea arborelui Cartezian se face adaugand elemente succesiv, avand in vedere ca dupa fiecare element adaugat sa pastram proprietatea de arbore Cartezian. Adaugarea un element A[i] poate modifica doar nodurile cele mai din dreapta incepand de la radacina si luand copilul din dreapta in mod repetat. Subarborele nodului cu cea mai mica valoare, dar mai mare sau egala decat elementul curent A[i] devine subarborele stang al nodului curent si arborele cu radacina A[i] va deveni noul subarbore drept al nodului cu cea mai mare valoare mai mica decat A[i].

Algoritmul Farach-Colton & Bender face turul Euler pe arborele Cartezian si il memoreaza intr-un vector E. Imparte vectorul E in blocuri de $0.5\log_2 N$, unde N este lungimea lui E. Pentru fiecare bloc, calculeaza elementul minim si il memoreaza intr-un vector M si construieste un Sparse Table pentru M.

Daca se da o interogare l, r si l si r sunt in blocuri diferite, atunci raspunsul va fi compus din sufixul blocului lui l, incepand de la el, prefixul blocului lui r, terminandu-se la r, si minimul dintre blocurile dintre l si r. Valorile din vectorul E difera cu o unitate una fata de cea de langa, astfel ca daca scadem din fiecare bloc valoarea primului element, vom avea o secventa formata numai din 1 si -1. Iar pentru ca blocurile sunt de o dimensiune atat de mica, numarul de secvente posibile este unul redus.

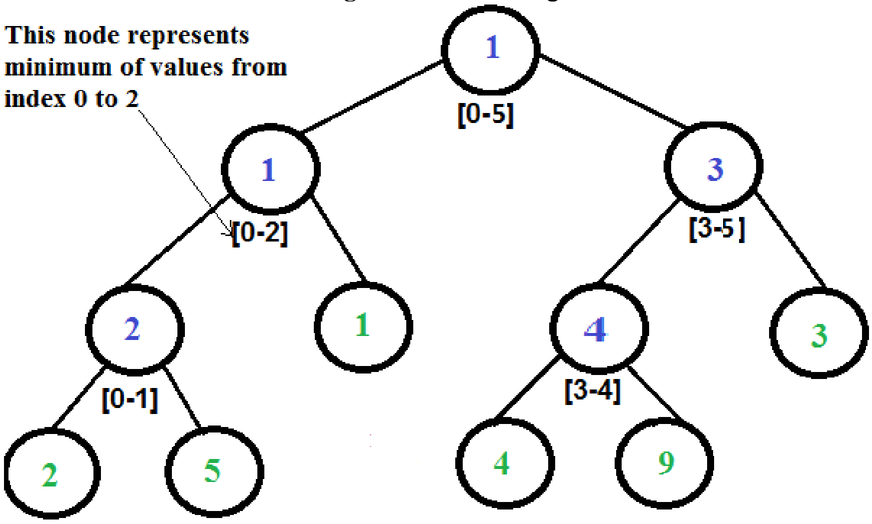
Precomputeaza rezultatul RMQ pentru fiecare bloc (prin metoda Sparse). Pentru implementare, fiecare bloc este caracterizat de o masca de biti si se memoreaza indexul elementului minim.

2.3 Segment Tree

Reprezentarea Arborelui Segment:

- Nodurile frunza sunt elementele din vector
- Fiecare nod intern reprezinta minimul dintre toate frunzele de sub el

Imagine 3. Ilustrare Segment Tree



Segment Tree for input array {2, 5, 1, 4, 9, 3 }

Pornim cu un segment al vectorului A[0 ... n-1] si de fiecare data impartim segmentul curent in doua jumatati (pana cand avem segmente de lungime 1) si apoi apelam procedura pe fiecare jumatate. Memoram minimul intr-un nod de arbore Segment. O data terminata preprocesarea (construirea arborelui), pentru raspunderea la interogari algoritmul va urma urmatoarul pseudocod:

```
RMQ(nod, st, dr){
    Daca intervalul asociat nodului este intre indicii st si dr
        Atunci returneaza valoarea nodului
    Daca intervalul asociat nodului este complet in afara indicilor st si dr
        Atunci returneaza infinit
    Altfel
        Returneaza minimul dintre RMQ(copilul stang al nodului, st dr) si
        RMQ(copilul drept al nodului, st, dr)
        }
```

2.4 Analiza complexitatii solutiilor

- Sparse Table
  - Preprocesare
- Vom folosi un vector bidimensional pentru a memora rezultatele precomputate pentru interogari.ST[i][j] va stoca raspunsurile pentru intervalul [i, i + 2^j - 1] de lungime 2^j. Dimensiunea vectorului bidimensional va fi N x (K + 1), unde N este lungimea vectorului, iar K este [$\log_2 N$] + 1. Complexitatea preprocesarii algoritmului este O(N $\log N$).
  - Raspunsul la interogari
- Raspunderea la interogari se realizeaza intr-un timp O(M * 1) intrucat trebuie doar acceseze o valoare deja precomputata din ST si se realizeaza M interogari.

- Cartesian Tree & algoritmul Farach-Colton and Bender
  - Preprocesare
- Construirea arborelui Cartezian se realizeaza in timp O(N), iterand o singura data prin vector. Algoritmul Farach-Colton and Bender poate preprocesa arborele in timp O(N) astfel:

  Imparte vectorul turului euler in blocuri de dimensiune K = 0.5$\log_2 N$. Pentru fiecare bloc, calcueaza elementul minim si il memoreaza intr-un vector B de dimensiune $\frac{N}{K}$ si construieste un Sparse Table din vectorul B. Complexitatea in timp va fi:

$$\frac{N}{K}\log\frac{N}{K} = \frac{2N}{\log N}\log\frac{2N}{\log N} = \frac{2N}{\log N}\left(1+\log\frac{N}{\log N}\right) \le \frac{2N}{\log N} + 2\text{N} = \text{O(N)}.$$

  In vectorul E, atunci cand scadem primul element, numarul de secvente posibile este egal cu:

$$2^{K-1} = 2^{0.5\log(N)-1} = 0.5\left(2^{\log(N)}\right)^{0.5} = 0.5\sqrt{N}.$$

  Deci, numarul de posibile blocuri diferite este O($\sqrt{N}$), asadar putem precomputa rezultatul RMQ in interiorul fiecarui bloc in O($\sqrt{N}K^2$) = O($\sqrt{N}(\log N)^2$) = O(N). Astfel, preprocesarea ia un timp de O(N). Algoritmul este unul Asimptotic optim.
  - Raspunsul la interogari
- Raspunderea la interogari se realizeaza intr-un timp O(M * 1) , folosind maximum 4 valori precomputate: elementul minim din blocul care contine indicele st, elementul minim din blocul care contine indicele dr, si cele doua minime ale segmentelor suprapuse ale blocurilor dintre st si dr.
- Segment Tree
  - Preprocesare
- Toate nivelele arborelui segment vor fi complet pline cu exceptia ultimului nivel. De asemenea, arborele va avea proprietatea de Full Binary Tree deoarece mereu impartim segmentele in doua jumatati. Deci, arborele construit va avea N frunze, si implicit N-1 noduri interne. Numarul total de noduri va fi 2N – 1, iar valoarea fiecarui nod este calculata o singura data in procesul de construire a arborelui. Inaltimea arborelui segment va fi $\log_2 N$. Complexitatea temporala a construirii

arborelui segment este O(N). Pentru a afla raspunsul, trebuie sa procesam maxim doua noduri la nivel de intrare

- Raspunsul la interogari

❖ Complexitatea la interogare este O($M * \log N$) deoarece la fiecare nivel putem apela maxim doua noduri, iar numarul de nivele este O($\log N$) si se realizeaza M interogari.

2.5 Avantaje si dezavantaje

Sparse Table-Raspunde fiecarei interogari in O(1), preprocesare in O($N \log N$).

-Avantaje: Structura de date foarte simpla si complexitate in timp buna.

-Dezavantaje: Pentru seturi de date foarte mari, se va comporta foarte incet in comparatie cu alti algoritmi mai puternici; Nu se mai poate aplica algoritmul daca vectorul se poate modifica intre 2 interogari.

Cartesian Tree & algoritmul Farach-Colton and Bender -Raspunde fiecarei interogari in O(1), preprocesare in O(N).

-Avantaje: Complexitate optima.

-Dezavantaje: Pentru seturi de date cu N < 10^6, realizeaza preprocesarea mai incet decat metoda Sparse Table; Foarte mult cod; Nu se mai poate aplica algoritmul daca vectorul se poate modifica intre 2 interogari.

Segment Tree -Raspunde fiecarei interogari in O($\log N$), preprocesare in O(N).

-Avantaje: Complexitate foarte buna; Permite schimbarea vectorului intre interogari; Pentru seturi de date foarte mari (N = $10^6 \sim 10^7$), este cel mai rapid adunand timpii de preprocesare si de raspuns la interogari.

-Dezavantaje: Mult cod in comparatie cu alti algoritmi; Nu raspunde instant la interogari, precum ceilalti doi algoritmi.

3 Evaluare

3.1 Construirea setului de test

Folosind un program scris in python am generat 30 de fisiere de intrare pentru testarea corectitudinii: 10 care au N = 40 elemente si M = random(0, 1000) interogari (generate de asemenea random intre 0 si 40) si 20 de teste cu N = 500 si M = random(0, 4000) interogari(tot generate random). Cele N elemente sunt si ele generate random si au valori cuprinse intre 0 si 10^6. De asemenea, am mai creat 4 teste foarte mari cu N de orinul milioanelor pentru observarea diferentei dintre timpii de executie ai fiecarui algoritm si alte 26 de teste folositoare la crearea graficelor in care voi evidentia eficienta algoritmilor. Zece dintre aceste teste au N, numarul de elemente din vector mare si M, numarul de interogari mic (N = 10k * 2^i, unde i este numarul testului si M = 500), iar alte 8 teste au N mic si M mare(N = 500, M = 10k * 2^i). Ultimele 8 teste au N = M si variaza de la ordinul zecilor de mii, pana la un milion.

Pentru verificarea corectitudinii algoritmilor, am generat pentru fiecare fisier de intrare (din cele 30) un fisier de iesire de referinta folosind algoritmul banal care raspunde unei interogari in O(N).

3.2 Specificatiile sistemului de calcul

Sistemul de calcul pe care am rulat si generat testele beneficiaza de urmatoarele performante:

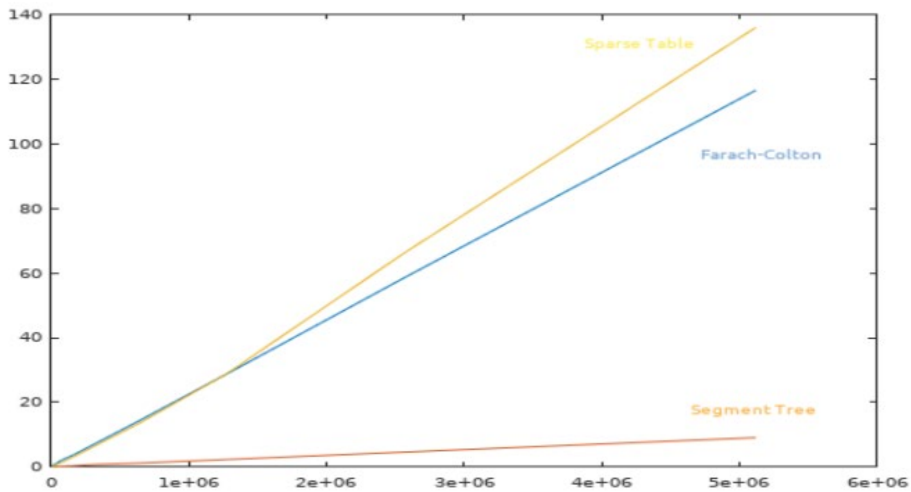
- Procesor: Intel Core i5 – 8300h @2.30GHz, 4 nuclee
- Memorie RAM: 8GB DDR4 – 2666

Insa, testarea se realizeaza pe o masina virtuala de Ubuntu 16.04 careia ii sunt alocati 4 GB de memorie RAM.

3.3 Eficienta solutiilor

Primul test la care am supus algoritmii a fost un set de 10 teste cu N = $10^4 : 5 * 10^6$ si am luat in considerare doar timpul de preprocesare pe care l-am expus in graficul de mai jos:

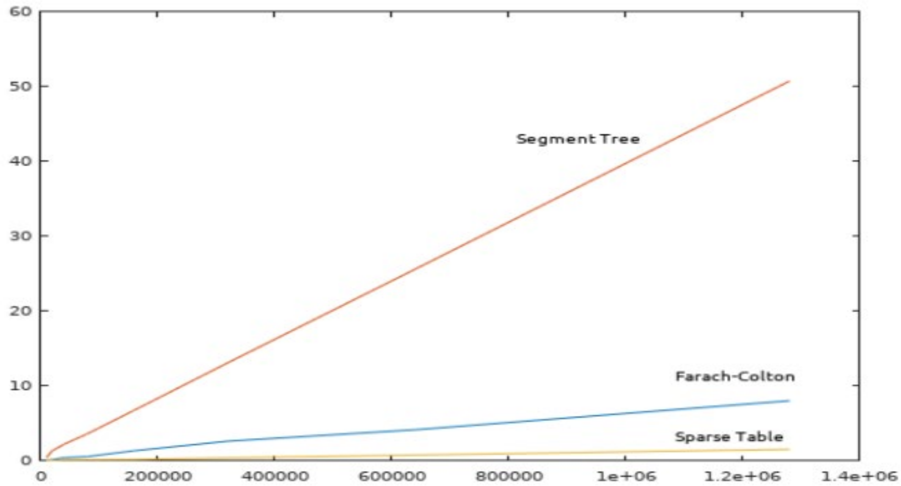
Figura 1. Eficienta algoritmilor la preprocesare



Axa OY a graficului reprezinta timpul de executie al preprocesarii, iar axa OX reprezinta numarul de elemente din vector. Se poate observa foarte usor diferenta dintre un algoritm cu preprocesare in timp O(N),Segment Tree, si algoritmul cu preprocesare O(Nlog N),Sparse Table, judecand numai dupa pantele fiecarei functii. Panta functiei asociate algoritmului Segment Tree este constanta, in timp ce panta functiei asociate algoritmului Sparse Table este in continua crestere. Pe de alta parte, functia asociata algoritmului Farach-Colton and Bender are o panta mare initial, si se comporta mai slab decat Sparse Table, insa panta acestei functii este in continua descrestere si va tinde **asimptotic** la aceeasi valoare ca si panta funcitei asociate algoritmului Segment Tree . Acest lucru ar putea fi observat si mai usor daca am putea figura timpii de executie pe seturi de date si mai mari (N > 10^8). Functiie asociate algoritmilor Farach-Colton si Sparse Table se intalnesc in punctul aproximativ N = 10^6, moment din in care algoritmul Farach-Colton se va comporta mai bine decat Sparse Table.

Al doilea test la care am supus algoritmii alesi este un set de 8 teste cu M=10^4 : 10^6 Interogari si N = 10^3.

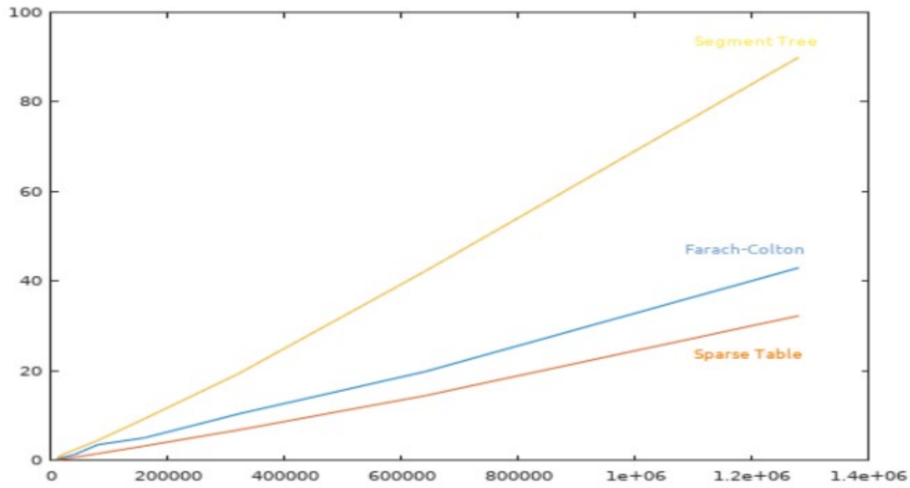
Figura 2. Eficienta algoritmilor la interogare



Axa OY reprezinta timpul in care se raspunde la M interogari, iar axa OX reprezinta M, numarul de interogari. In figura 2 observam complexitatile temporale ale algoritmilor cand vine vorba de interogari. Algoritmii Farach-Colton si Sparse Table raspund interogarilor in O(M), insa in algoritmul Farach-Colton se efectueaza cu cateva mai multe operatii, dar inca integrandu-se in aceasta categorie(O(M)). Algoritmul Segment Tree raspunde fiecari interogari in timp logaritmic si raspunde tuturor interogarilor cu o complexitate O($M \log N$), iar acest lucru este usor observabil in figura 2, fiind comparat cu ceilalti doi algoritmi.

Al treilea test, si ultimul, supune algoritmii la un set de teste cu N = M cu valori intre 10^4 si $1.28 * 10^6$.

Figura 3. Eficienta algoritmilor per total



In cazul N = M se observa ca algoritmul Segment Tree se comporta cel mai slab, iar acest lucru este cauzat de complexitatile sale de O(N) la preprocesare si O(Mlog N) la interogare. Pe de alta parte, un rezultat neasteptat, cel putin la prima vedere, este cel al algoritmului Sparse Table care pare sa se comporte cel mai bine dintre toti in acest caz. Acest lucru se intampla doar pana la un anumit punct, in care algoritmul Farach-Colton and Bender il va intrece datorita complexitatii sale optime asimptotice. Totodata, la infinit, functiile asociate Sparse Table si Segement Tree vor avea aceeasi panta, intrucat au aceeasi complexitate pentru preprocesare + interogare (doar in cazul N = M).

4 Concluzii

Care algoritm este mai bun?

Raspunsul la aceasta intrebare este depinde. Depinde de ce vrem ca algoritmul nostru sa faca, depinde si de numarul de elemente din vectorul nostru, si depinde si de cat de des dorim sa interogam RMQ.

Daca stim ca vom interoga RMQ de mult mai multe ori decat elemente sunt in vector, iar vectorul nu este de o dimensiune mai mare de 10^6 elemente, atunci cea mai buna optiune ar fi Sparse Table. Este usor de implementat, raspunde cel mai rapid, si are un timp de preprocesare mai bun decat Farach-Colton. Daca vectorul este de o dimensiune mult mai mare de 10^6 elemente, atunci algoritmul Farach-Colton and Benders care foloseste un Cartesian Tree este optiunea mai buna.

Pe de alta parte, daca stim ca nu se vor efectua un numar de interogari comparabil cu numarul de elemente din vector, atunci Segment Tree este alegerea cea mai buna, indiferent de dimensiunea vectorului. In plus, Algoritmul Segment Tree permite schimbarea vectorului intre interogari, in timp ce algoritmii Sparse Table si Farach-Colton ar trebui sa preproceseze tot vectorul din nou.

Referinte

1. Autori: Hao Yuan, Mikhail J. Atallah : Data Structures for Range Minimum Queries in Multidimensional (2014)
2. Forumul www.geeksforgeeks.org , ultima accesare 13/12/2019
3. https://cp-algorithms.com/sequences/rmq.html , ultima accesare 13/12/2019
4. Blogul TopCoder www.topcoder.com , ultima accesare 12/12/2019
5. Platforma online medium.com , ultima accesare 13/12/2019
6. www.hackerearth.com , ultima accesare 13/12/2019