Studiu comparativ al algoritmilor de rezolvare a problemei Range Minimm Query

Anca Millio

Calculatoare si Tehologia Informatiei, anul II

1 Introducere

1.1 Descrierea problemei rezolvate

In domeniul IT, un Range Minimum Query (RMQ) rezolva problema gasirii valorii minime dintr-un subvector al unui vector de elemente comparabile. De exemplu: Fiind dat un vector A si doi indici, $i \leqslant j$, care este cel mai mic element dintre: A[i], A[i + 1], ... , A[j - 1], A[j]?

1.2 Exemple de aplicatii practice pentru problema aleasa

RMQ sunt utilizate in diferite cazuri, cum ar fi problema celui mai mic stramos comun (lowest cmmon ancestor LCA) sau problema celui mai lung prefix comun (longest common prefix problem LCP).

1.3 Specificarea solutiilor alese

1. Sparse Table

   Acest concept este utilizat pentru interogarile rapide pe un set static de date(elementele nu se schimba). Este utilizata o functie care trebuie sa fie asociativa. Procesarea este facuta astfel incat interogarile primesc raspuns in mod eficient.

2. Sqrt-decomposition

   Imparte intervalul dat in blocuri separate de cate sqrt(n) elemente, apoi calculeaza minimul din fiecare bloc si stocheaza rezultatele. Este un algoritm care permite schimbarea valorilor din interval intre interogari. Raspunde la fiecare interogare in O(sqrt(N)), realizeaza preprocesarea in O(N).

3. Segment Tree

   Frunzele arborelui sunt elementele vectorului dat, iar fiecare dintre nodurile interne reprezinta minimul tuturor frunzelor de sub acesta. De asemenea, acesta este un algoritm ce permite schimbarea valorilor vectorului in timpul interogarii. Raspunde la fiecare interogare in O(log N), realizeaza preprocesarea in O(N).

1.4 Criteriile de evaluare pentru solutiile propuse

1. Se vor crea fisiere de input, care apoi vor fi pasate atat algoritmilor mentionati mai sus, cat si unui algoritm banal, apoi se vor compara fisierele de output, fiind evident ca output-ul algoritmului banal va fi corect, deorece este usor de implmentat si astfel sunt evitate greselile. Daca nu exista diferente, rezulta ca implementarile celor trei solutii sunt corecte. Datele vor fi generate aleator.

2. Pentru analizarea eficientei si a complexitatii solutiilor, se vor crea fisiere de test cu dimensiuni variabile ale vectorului de numere, dar si a numarului de query-uri solicitate. Variatia duratei rularii in functie de marimea intrarii va fi stocata intr-un alt fisier, pe baza caruia vor fi apoi alcatuite grafice care sa ilustreze complexitatea fiecarui algoritm testat. Se vor alege vectori cu dimensiuni de 10, 100, 1000, etc elemente, iar pentru fiecare dintre aceste lungimi, cate un test cu 10 query-uri si unul cu 100 de query-uri. Numerele vor fi generate aleator.

2 Prezentarea solutiilor

2.1 Descrierea modului in care functioneaza algoritmii alesi

1. Sparse Table

   Ideea pe care se bazeaza aceasta solutie este de a precalcula minimul tuturor subvectorilor de lungime 2^j, unde j variaza de la 0 la Log n, unde n este lungimea vectorului. Se realizeaza o matrice de cautare. Elementul [i][j] va contine minimul intervalului incepand de la i si de marimea 2^j. De exemplu, cautarea [0][3] va contine minimul din intervalul [0, 7] (incepand cu 0 si dimensiunea 2^3).

   Pentru orice interval arbitrar [L, R], trebuie s folosim intervale cu lungimi puteri ale lui 2. Se foloseste cea mai apropiata putere de 2. Trebuie intotdeauna sa facem cel mult o comparatie (compararea minimului a doua

intervale care sunt de lungimi puteri ale lui 2). Un interval incepe cu L si se termina cu "L + cea mai apropiata putere de 2". Celalalt interval se termina cu R si incepe cu "R - aceeasi putere + 1". De exemplu, daca intervalul dat este [2, 10], comparam minimul a dou intervale [2, 9] si [3, 10].

2. Sqrt-decomposition

   Se imparte intervalul [0, n-1] (ce reprezinta indicii elementelor din vectorul de lungime n), in blocuri de cate n elemente fiecare.

   Se calculeaza minimul fiecarui bloc de dimensiune n si se stocheaza intr-un vector rezultatele.

   Pentru a interoga un interval [L, R], luam minimul dintre valorile minime ale blocurilor care se afla in intregime in acest interval. Pentru blocurile care se suprapun partial cu intervalul dat, se parcurg liniar pentru a gsi minimul. La final se alege minimul dintre aceste valori.

3. Segment Tree

   In Segment Tree, frunzele sunt elementele vectorului de intrare, iar fiecare nod intern reprezinta minimul tuturor frunzelor de sub el.

   Se incepe cu un vector: v[0. . . n-1], care se imparte recursiv in jumatati. Se imparte de fiecare data segmentul curent in doua jumatati (daca acesta nu a devenit inca un segment cu lungimea 1), si apoi se apeleaza aceeasi procedura pe ambele jumatati. Pentru fiecare astfel de segment, stocam valoarea minima intr-un nod al arborelui.

   Odata ce arborele a fost construit, se pot face interogari. Urmatorul algoritm in pseudocod ilustreaza obtinerea rezultatului unei interogari.

   $qs \rightarrow$ indicele de inceput al intervalului interogarii, $qe \rightarrow$ indicele de final al intervalului

```
int RMQ (nod, qs, qe)
{
    daca intervalul corespunzator nodului curent coincide cu
    intervalul interogarii(qs, qe)
        se returneaza valoarea nodului
    altfel daca intervalul corespunzator nodului curent este
    complet in afara interoarii(qs, qe)
        se returneaza infinit
```

```
    altfel
        se returneaza min(RMQ (copilul stng al nodului,
            qs, qe), RMQ (copilul drept al nodului, qs, qe))
}
```

2.2 Analiza complexitatii solutiilor

1. Sparse Table

   Deoarece este nevoie de cel mult o comparatie pentru fiecare interogare, dupa cum am precizat mai explicit la sectiunea "Descrierea modului in care functioneaza algoritmii alesi", rezulta ca aceasta solutie are complexitatea interogarii de O(1).

   Complexitatea etapei de preprocesare a datelor poate fi calculata astfel:

   In functia care creeaza matricea utilizata la interogare, bucla externa(for) are complexitatea O(k), deoarece se alege un k minim, astfel incat $2^{k+1} > n$, asadar O(k) = O(log(n)). Bucla interna(for) itereaza de la 0 la n-1(unde n e lungimea vectorului) si contine doar o comparatie, deci are complexitatea O(n). In concluzie, obtinem complexitatea O(n * k) = O(n * log(n)).

   Matricea formata are dimensiunea $n * log(n)$, ceea ce determina o complexitate spatiala de O(n log(n)).

2. Sqrt-decomposition

   Aici vectorul este impartit in $\sqrt{n}$ intervale de cate $sqrtn$ elemente ale vectorului, pri urmare complexitatea preprocesarii este de O($\sqrt{n} * \sqrt{n}$) = O(n), iar complexitatea spatiala este O($\sqrt{n}$), deoarece este nevoie de crearea unui vector care stocheaza minimul din fiecare bloc.

   Complexitatea interogarii este O($\sqrt{n}$). Minimul blocurilor din interiorul intervalului dat este direct accesibil si pot exista cel mult $\sqrt{n}$ blocuri de acest tip. Pot fi maxim doua blocuri care nu se suprapun complet cu intervalul dat si trebuie parcurse, asa ca va exista o complexitate de $2 * O(\sqrt{n})$ pentru acest proces. Asadar, complexitatea totala va fi O(n).

3. Segment Tree

   Complexitatea pentru constructia arborelui(preprocesarea) este O(n). Numarul de noduri din arbore este de 2n-1, iar valoarea fiecarui nod este cal-

culata o singura data in constructia arborelui.

Complexitatea interogarii este O(Logn). Pentru a interoga minimul unui interval, se proceseaza cel mult doua noduri la fiecare nivel, iar numarul de nivele este O(Log(n))(inaltimea arborelui este log(n)).

Complexitatea spatiala este determinata de stocarea arborelui, deci apartine O(n).

2.3 Prezentarea principalelor avantaje si dezavantaje pentru solutiile luate in considerare

1. Sparse Table

   – Avantaje:

   Utilizeaza o structura de date simpla, aceasta fiind matricea pentru aflarea raspusurilor la interogari.
   Un alt avantaj consta in viteza de raspuns la interogari (O(1)), fiind cea mai rapida solutie dintre cele trei, din acest punct de vedere.

   – Dezavantaje:

   Daca vectorul de input sufera modificari, costul repetarii peprocesarii va reprezenta un dezavantaj, fiind cel mai ineficient ca durata dintre cei trei algoritmi comparati(O(n log(n)).

2. Sqrt-decomposition

   – Avantaje:

   Implementarea este usoara, in comparatie cu celelalte doua solutii, avand nevoie de o structura de date foarte simpla, mai exact vectorul in care se stocheaza minimele blocurilor vectorului de input (O($sqrtn$)).
   Preprocesarea este mai putin costisitoare decat in cazul "Sparse table".

   – Dezavantaje:

   Complexitatea interogarii reprezinta un dezavantaj, fiind cea mai lenta comparativ cu celelalte solutii alese(O($\sqrt{n}$)).

3. Segment Tree

   – Avantaje:

     Interogarile se fac mai rapid decat in cazul Sqrt-decomposition, iar memoria utilizata si viteza de preprocesare depasesc performanta Sparse Table, ceea ce duce la concluzia ca acest algoritm este recomandat in cazul in care se fac modificari ale vectorului de intrare intre interogari, dar se doreste si o viteza relativ buna de raspuns la interogari.

   – Dezavantaje:

     Cantitatea de cod este mai mare decat la celelalte solutii, facand implementarea mai anevoiasa.

3 Evaluare

3.1 Descrierea modalitatii de construire a setului de teste folosite pentru validare

In afara de cele trei solutii, am implementat si un algoritm cu rezolvarea banala, pentru a putea compara corecttudinea celor trei solutii.

Pentru realizarea testelor am creat un fisier(generator.cpp) in care am implementat un algoritm care genereaza un vector de numere, dar si un vector de perechi de numere(pentru interogari) in mod aleator, apoi am scris intr-un fisier bash(Tests), comenzi care dau ca parametrii codului generator, numarul de elemente din vector si numarul de interogari dorite si redirecteaza vectorul de rezultate intr-un fisier corespunzator din folder-ul "out", precum si durata interogarilor in alt fisier($Algo\_Time$). Acest ultim fisier este utilizat la construirea graficelor care ilustreaza performanta comparativa a solutiilor alese.

Am realizat teste pe diferite lungimi ale vectorului de intrare(10, 100, 1000, ...), precum si numar variabil de interogari(10, 100) pentru fiecare lungime a vectorului, apoi am aplicat fiecare test fiecarui algoritm(algo1 = Sparse Table, algo2 = Sqrt-decomposition, algo3 = Segmnt Tree, $basic\_algo$ = solutia banala).

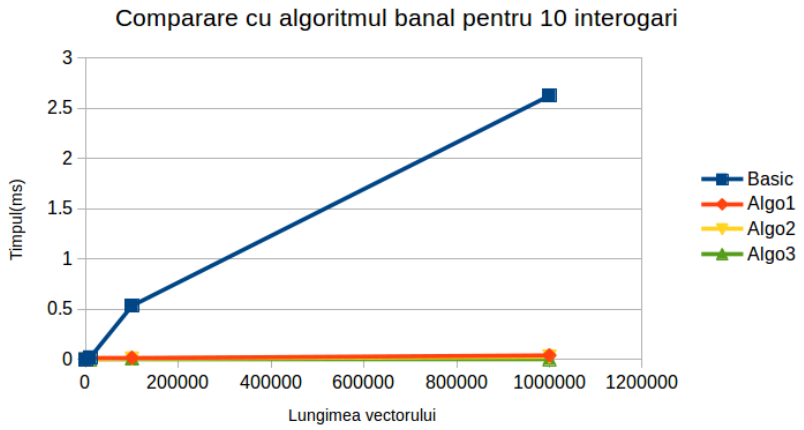
3.2 Specificatiile sistemului de calcul pe care au fost rulate testele

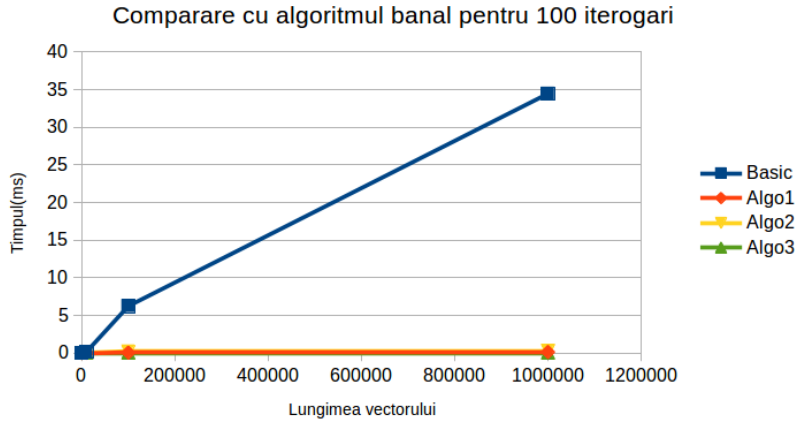
Procesor:
Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz

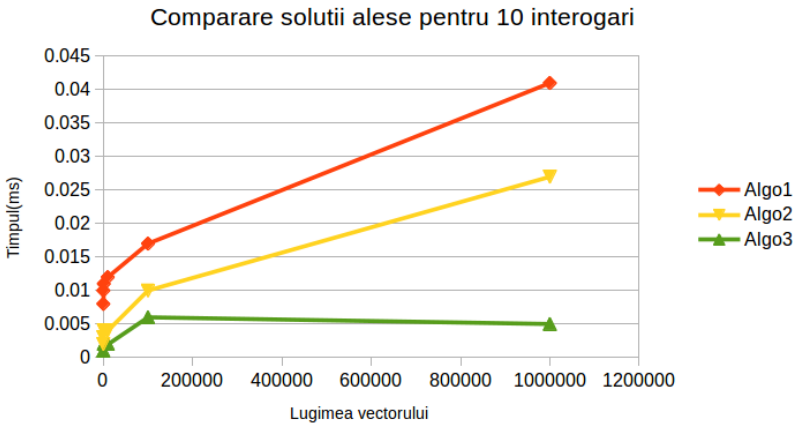
Memorie disponibila:
8 GB

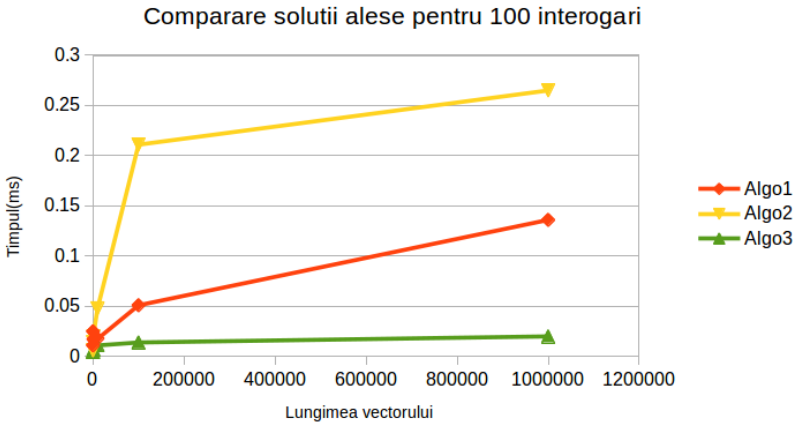
3.3 Ilustrarea, folosind grafice, a rezultatelor evaluarii solutiilor pe setul de teste

Se poate observa ca cele trei solutii sunt mult mai bune decat algoritmul banal pentru oricate interogari.






Comparare solutii alese pentru 10 interogari
Timpul(ms)
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
200000
400000
600000
800000
1000000
1200000
Lugimea vectorului
Algo1
Algo2
Algo3


Comparare solutii alese pentru 100 interogari
Timpul(ms)
0.3
0.25
0.2
0.15
0.1
0.05
0
0
200000
400000
600000
800000
1000000
1200000
Lungimea vectorului
Algo1
Algo2
Algo3

3.4 Prezentarea succinta a valorilor obtinute pe teste

Rezultatele sunt cele asteptate in privinta performantei algoritmilor 2 si 3, atat pentru 10, cat si pentru 100 de interogari, algoritmul 3 fiind mai rapid decat 2.

Este neasteptat faptul ca pentru 10 interogari, algortmul 1 este cle mai ineficient, iar pentu 100 de interogari, acesta se situeaza pe locul doi. Ne-am fi asteptat ca el sa fie pe locul intai in toate situatiile.

Analizand codul, am observat ca pentru fiecare interogare se calculeaza un logaritm, iar acest calcul consuma foarte mult timp. Pentru a verifica aceasta ipoteza, am comentat toate liniile din functia query a algoritmului 1 (Sparse Table), in afara de calculul logaritmului si am observat ca timpii de executie sunt sensibil egali cu varianta in care codul este complet. Pentru a aduce performanta la rezultatele asteptate, ar trebui construit un vector care sa stocheze rezultatele calculelor logaritmilor respectivi in faza de preprocesare.

4 Concluzii

In practica nu exista o solutie perfecta(cea mai buna). Este esential ca aceasta sa fie aleasa in functie de caracteristicile datelor de intrare.

Daca vectorul de input se modifica des(de exemplu de 100 de ori pentru 1000 de interogari) sau se dau foarte putine interogari, atunci ar fi optim sa se aleaga solutia cu cel mai bun timp de preprocesare. Segment Tree si Sqrt-decomposition au cea mai buna complexitate de preprocesare(fata de Sparse Table) mai exact O(n), deci preferabil ar fi sa alegem Segment Tree, deoarece acesta are o performanta mai buna la interogari.

Daca vectorul de input nu se modifica, dar se fac multe interogari(de la ordinul zecilor de mii in sus), cel mai potrivit ar fi Sparse Table, deoarece interogarile se fac cel mai rapid(O(1)), comparativ cu celelalte doua solutii.

5 Bibliografie

Data ultimei accesari a urmatoarelor surse este: 13.12.2018.

1. https://web.stanford.edu/class/cs166/lectures/01/Slides01.pdf
2. https://www.geeksforgeeks.org/range-minimum-query-for-static-array/
3. https://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query/