Range Minimum Query

Proiect Analiza Algoritmilor

Koleci Alexandru-Paul

1 Noiembrie 2021

Universitatea Politehnica Bucuresti
Facultatea de Automatica si Calculatoare

1 Introducere

1.1 Descrierea Problemei Rezolvate

"Range Minimum Query" este o problema ce consta in aflarea elementului minim dintr-un interval. Pentru un vector de N intregi, trebuie sa gasim minimul dintr-un sub-interval dat, facandu-se M interogari.

Dificultatea acestei probleme apare datorita complexitatii algoritmului: solutia simpla, in care parcurgem tot vectorul de fiecare data si comparam elementele, nu este destul de eficienta, asadar trebuie folosite alte metode.

1.2 Exemple de aplicatii practice pentru problema aleasa

O aplicatie practica in care RMQ este folosit o reprezinta algoritmul de "string-matching" (gasirea unui sub-string intr-un string mai mare sau intr-un text). De asemenea, RMQ mai este folosit si in algoritmul de aflare al celui mai mic stramos comun al doua noduri (Lowest Common Ancestor).

1.3 Specificarea solutiilor alese

Pentru a rezolva problema RMQ, voi folosi 3 algoritmi: Segment Tree, Sparse Table si Square Root Decomposition.

1.4 Specificarea criteriilor de evaluare alese pentru validarea solutiilor

Pentru validarea solutiilor vor exista doua criterii: corectitudinea si eficienta. Cei 3 algoritmi trebuie sa dea acelasi rezultat pentru acelasi test, si vor fi comparati in functie de eficienta si, implicit, de complexitatea lor. Algoritmii for fi testati atat pe cazul general, in care nu se da update, cat si in cazul in care se poate da update.

2 Prezentarea solutiilor

2.1 Descrierea modului in care functioneaza algoritmii alesi

Segment Tree - este folosit un arbore binar pentru a stoca elementele vectorului. Incepem prin salvarea elementului minim din vector in primul nod, dupa care intervalul este injumatatit. Dupa aceea, operatia se repeta pentru nodurile copil, nodului stang ii revine prima jumatate a intervalului, iar nodului drept cea de-a doua jumatate. La fiecare pas salvam elementul minim din intervalul ramas si injumatatim din nou intervalul. Intr-un final, nodurile frunza vor reprezenta valorile vectorului.

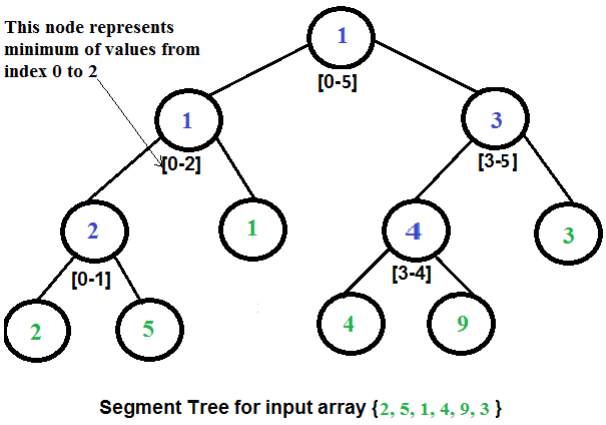

Fig. 1. Reprezentare Segment Tree

Sparse Table - este folosita o matrice pentru a stoca valorile minime din toate sub-intervalele de lungime 2^k, unde 2^k e mai mic decat N, k numar natural. Dupa aceea, in functie de sub-intervalul cerut, calculam minimul dintre intervalele de lungime puteri ale lui 2 din care acesta este alcatuit.

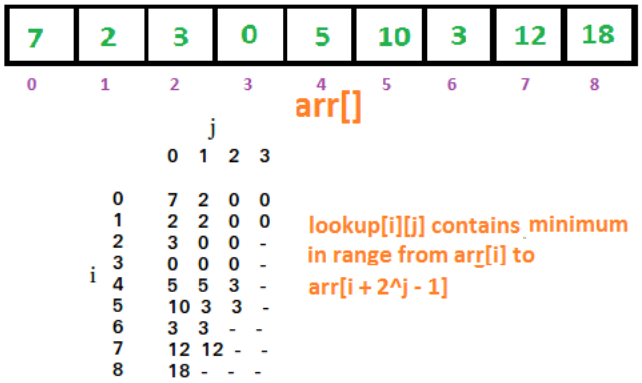

Fig. 2. Reprezentare Sparse Table

Square Root Decomposition - intervalul este impartit in sub-intervale de lungime $\sqrt{N}$. Pentru fiecare din acestea este calculat minimul si salvat index-ul acestuia. In final, se calculeaza minimul intervalului cerut.

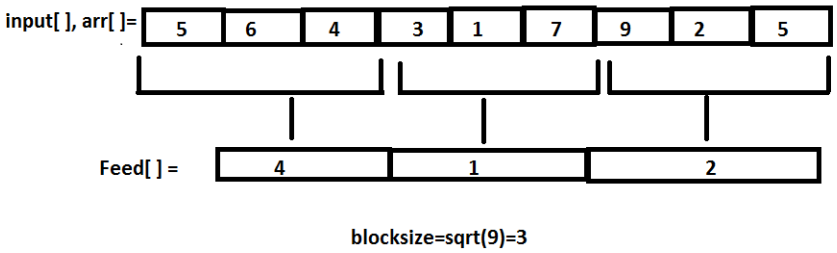

Fig. 3. Reprezentare Sqrt Decomposition

2.2 Analiza complexitatii solutiilor

Sparse Table
Query: O(1) - odata ce matricea este facuta, se cauta pur si simplu valoarea elementului de la un anumit index. Nu depinde de N.
Update: O(NlogN) - trebuie refacuta complet matricea, asadar durata este data de marimea matricii: N linii si logN coloane.

Segment Tree
Query: O(logN) - odata ce tree-ul este facut, pentru a ajunge la minimul dorit, trebuie sa trecem prin nodurile care contin minimurile din diferite subintervale. Este practic o suma de logN-uri, de aceea complexitatea este logN.
Update: O(logN) - trebuie urmarit un singur drum de la radacina pana la frunza dorita, ceea ce dureaza logN.

Square Root Decomposition
Query: O($\sqrt{N}$) - vectorul se imparte in $\sqrt{N}$ subintervale, pentru care se face minimul, asa ca trebuie traversate $\sqrt{N}$ elemente pentru fiecare bloc.
Update: O(1) - se gaseste blocul in care se afla elementul de la indexul respectiv si doar se calculeaza minimul pentru acel bloc. Nu depinde de N.

2.3 Prezentarea principalelor avantaje si dezavantaje ale solutiilor luate in considerare

Sparse Table
Avantaje: Datorita faptului ca are complexitatea O(1), este cel mai rapid dintre algoritmi, asa ca, pe cazul general, este cel mai bun.
Dezvantaje: Este cel mai slab algoritm in cazul in care trebuie sa dam update, deoarece trebuie refacuta intreaga structura in care sunt retinute valorile query-urilor. Pentru fiecare update, complexitatea refacerii matricii are complexitate O(NlogN) (N linii, logN coloane).

Segment Tree
Avantaje: Are complexitate O(logN), adica destul de buna, si este bun atat pe cazul general, cat si pe cazul in care se da update, pentru ca se da relativ repede update la un nod, update avand complexitatea O(logN).
Dezvantaje: Este mai lent decat sparse table pe cazul general, iar in cazul in care se da update, cresterea duratei de executie este mai mare decat in cazul algoritmului square root decomposition. Pentru valori foarte mari ale lui N, va fi din ce in ce mai lent programul la update.

Square Root Decomposition
Avantaje: Update are complexitate O(1), asa ca nu e aproape nicio diferenta intre cazul in care se da update si cazul in care nu se da.
Dezvantaje: Este considerabil mai lent decat ceilalti 2 algoritmi pe cazul general, asa ca ar trebui folosit doar daca stim ca se da update de multe ori.

3 Evaluare

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

Pentru evaluarea algoritmilor, voi alcatui un set de teste care sa verifice atat corectitudinea, cat si eficienta algoritmilor propusi. Fiecare test va contine un vector pentru care trebuie aflat minimul dintr-un interval al sau, asadar corectitudinea este usor de verificat. In ceea ce priveste eficienta, testele vor urmari o serie de cazuri.

Fie M numarul de interogari si N lungimea vectorului.

Testele 1-7 abordeaza cazul N proportional cu M, initial fiind valori relativ mici, pentru a se vedea usor corectitudinea, si la final ajungand sa fie N=M=1000000.

Testele 8-14 sunt pentru cazul in care N>>M, N ajungand sa fie 1000000 si M mic.

Testele 15-21 sunt pentru cazul in care M>>N, M fiind maxim 1000000.

Testele 22 si 23 sunt menite sa verifice partea de update. Testul 22 este un test simplu, pe care se poate urmari usor raspunsul, iar testul 23 este un stress test, avand N si M mari, si multe update-uri.

Testele au fost create folosind un program python scris de mine.

3.2 Specificatiile sistemului de calcul

Procesor: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz
Memorie: 16GB RAM

3.3 Ilustrarea rezultatelor evaluarii solutiilor

SEGMENT TREE			SPARSE TABLE			SQRT DECOMP	
Nr test	Timp (s)		Nr test	Timp (s)		Nr test	Timp (s)
1	0		1	0		1	0
2	0		2	0		2	0
3	0		3	0		3	0
4	0		4	0		4	0
5	0.015625		5	0		5	0.015625
6	0.09375		6	0.0626		6	0.15625
7	1.140625		7	0.546875		7	3.46875
8	0		8	0		8	0
9	0		9	0		9	0
10	0		10	0		10	0
11	0		11	0		11	0
12	0		12	0		12	0
13	0		13	0		13	0
14	0		14	0		14	0
15	0		15	0		15	0
16	0		16	0		16	0
17	0.140625		17	0.078125		17	0.109375
18	0		18	0		18	0.15625
19	0.015625		19	0.015625		19	0.078125
20	0.296875		20	0.1875		20	0.3125
21	0.4375		21	0.3125		21	0.421875
22	0		22	0		22	0
23	1.34375		23	2.171875		23	3.515625

Fig. 4. Timpul de executie al celor trei programe

3.4 Prezentarea valorilor obtinute pe teste

Observatie: Durata executiei de 0 secunde reprezinta o durata mai mica decat 0.0000001 secunde (Am calculat doar cu 6 zecimale).

Pe testele 1-21, algoritmul Sparse Table este mai rapid decat celelalte doua, iar la testul 22 se da doar un update, ceea ce nu dureaza mult timp. Pentru testul 23 de la Sparse Table, am dat update doar de 10 ori, nu de 1000000, ca in ceilalti doi algoritmi, asadar durata testului 23 (2.171875 secunde) este doar pentru 10 update-uri. Pentru 100 de update-uri, de exemplu, algoritmul rula timp de 21 de secunde pe testul 23, deci nu este bun cazul in care putem schimba elementele vectorului.

Segment Tree este, in aproape fiecare test, mai rapid decat algoritmul Square Root Decomposition. Testele 7 si 23 au amandoua valorile N=M=1000000, si se observa aici diferenta intre testul in care se da update (testul 23) si cel in care nu se da (testul 7). Pentru testul 23, Segment Tree are durata de 1.34375 secunde, spre deosebire de testul 7, in care durata este de 1.140625 secunde. O intarziere de 0.20 secunde, iar la unele rulari poate fii chiar mai mare diferenta.

In schimb, Square Root Decomposition dureaza 3.515625 secunde in testul 23 si 3.46875 in testul 7, o intarziere de doar 0.05 secunde. In unele cazuri, testul 23 rula chiar mai repede decat testul 7, demonstrand ca update-urile nu afectau deloc eficienta programului. De asemenea, este mai rapid decat Segment Tree in testele 17 si 21, ceea ce arata ca, in unele cazuri, sunt apropiate ca eficienta.

Testele 8-14 au durata de executie foarte mica (aproximativ 0) pentru toti cei 3 algoritmi, de unde reiese ca marimea vectorului nu este principalul factor care influenteaza eficienta, ci numarul de query-uri, caci fiecare algoritm se va repeta de M ori.

4 Concluzii

4.1 Abordarea problemei in practica

Pentru abordarea problemei, daca nu exista posibilitatea de a da update unei valori din vector, as folosi algoritmul Sparse Table. Comparativ cu celelalte doua, este extrem de rapid, singura etapa care dureaza este cea de preprocesare, de creare a matricei din care se calculeaza valorile minime din query-uri. Este foarte rapid chiar si in cazurile in care sunt multe elemente in vector sau sunt multe query-uri. Insa in cazul in care ar exista optiunea de a da update unui element, as folosi algoritmul Segment Tree daca vectorul de numere nu ar fi foarte mare si daca nu s-ar da multe update-uri. Pentru fiecare update, complexitatea este O(logn), asa ca daca N e mare si sunt multe update-uri, nu va fi la fel de eficient. Daca sunt numere multe in vector si se dau multe update-uri, as folosi algoritmul Square Root Decomposition, care, avand complexitate O(1) la update, nu va avea intarzieri in acest caz.

5 Bibliografie

1. http://www.topcoder.com/thrive/articles/Range%20Minimum%20Query%20and%20Lowest%20Common%20Ancestor, ultima accesare 02/11/2021

2. https://www.geeksforgeeks.org/range-minimum-query-for-static-array/?ref=lbp, ultima accesare 02/11/2021

3. https://drops.dagstuhl.de/opus/volltexte/2017/7615/pdf/LIPIcs-SEA-2017-12.pdf, ultima accesare 02/11/2021

4. https://iq.opengenus.org/range-minimum-query-square-root-decomposition, ultima accesare 02/11/2021

5. https://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query, ultima accesare 02/11/2021

6. Fischer, Johannes; Heun, Volker (2007). A New Succinct Representation of RMQ-Information and Improvements in the Enhanced Suffix Array