Range Minimum Query

Oprea Olivia Maria-Magdalena 323CA

INTRODUCERE

Descrierea problemei rezolvate

Subiectul abordat este “Range Minimum Query”, care se axeaza pe gasirea eficienta a elementului minim de la indexul x la indexul y (inceputul respectiv finalul intervalului) dintr-un vector dat. O aplicatie practica a acestui subiect este “Problema celui mai mic stramos comun” si “Problema celui mai lung prefix comun”.

Exemple de probleme practice

RMQ poate fi folosit, asa cum am specificat si mai sus, la rezolvarea problemelor celui mai mic stramos comun si cel mai lung prefix comun.

Problema celui mai mic stramos comun este folosita in aproximarea si gasirea linilor de caractere exacte, pe scurt ctrl + f de pe calculatoare.

Problema celui mai lung prefix comun este folosita in comprimarea arborelui de sufixe, traversarea eficienta de sus in jos si de jos in sus al arborelui de sufixe si grabeste potrivirea cuvintelor pe tabloul de sufixe.

Solutii alese

1. **Square Root Decomposition**

   Folosim aceasta metoda pentru a reduce spatiul cerut in felul urmator: vom impartii vectorul in blocuri diferite de lungime aproximativ egale cu sqrt(n), iar apoi vom stoca pozitia celui mai mic element din fiecare bloc.

   Preprocesarea dureaza O(n), iar spatiul alocat este O(sqrt(n)).

2. **Sparse Table Algorithm**

   Aceasta metoda necesita O(sqrt(n)) spatiu, dar timpul de interogare este O(sqrt(n)). Tot aceasta metoda accepta timpul de interogare O(1) cu un spatiu suplimentar de O(n log n).

   Creem o matrice cu n linii si cel mai apropiat numar de log n +1 coloane. Matricea se completeaza astfel: pornim de la pozitia i si cautam elemental minim a 2^j elemente si stocam pozitia elementului

minim. Daca nu se pot compara cele 2^j elemente, pozitia din matrice a elementului minim va ramane goala.

Pentru a gasi elementul minim dintr-un interval se calculeaza lungimea acestuia si se compara primul element cu urmatoarele $2^{\log n}$elemente, iar daca diferenta dintre n si urmatoarele elemente este mai mare decat 0 algoritmul se reia.

3. **Segment Tree**

Am ales aceasta solutie deoarece nu avem voie sa folosim solutia banala.

Timpul de preprocesare este O(n), iar timpul pentru gasirea elementului minim este O(log n), spatiul necesar in plus este O(n) pentru a stoca metoda.

Metoda se construieste de la un vector dat. Impartim segmentul curent in 2 jumatati (daca nu este deja un segment de lungime 1), iar apoi apelam aceeasi procedura pe cele 2 jumatati si pentru fiecare astfel de segment stocam valoarea minima intr-un segment tree. Inafara de ultimul nivel, arborele este completat in totalitate si este un arbore binar.

Criterii de evaluare

Deoarece cautam elementul minim dintr-un interval vom avea nevoie de diferiti vectori pentru a valida algoritmul nostru si pentru a-i testa corectitudinea.

Vectorii alesi vor fi umpluti pe rand cu numere positive, negative, ambele, numere ordonate crescator, descrescator, piramida (crescator pana intr-un punct, apoi descrescator), partial crescatoare sau partial descrescatoare, elemente random.

Testele au fost generate cu ajutorul unui program java care primea ca imput cat de mari sa fie numerele si cate interogari sa se creeze pe ele. Numerele sunt generate aleatoriu si ajuta la verificarea eficientei si performantei algoritmilor.

PREZENTAREA SOLUTIILOR

1. **Square Root Decomposition:**

Acest algoritm primeste un vector si apoi il imparte in diferite blocuri de marime (sqrt(n) (in cazul meu lg, deoarece au rezultate destull de apropiate). Dupa aceea algoritmul calculeaza minimul fiecarui bloc in parte si il stocheaza.
Intreaga complexitate al acestui algoritm este de O((sqrt(n)) si este la fel si in cazul in care trebuie sa verificam si marginile vectorului (2*O((sqrt(n))).
Acest algoritm ne permite sa reducem semnificativ numarul de operatii, ocupa mai putin spatiu, dar timpul de rezolvare este mult mai mare.

2. **Sparse Table Algorithm:**

Algoritmul primeste un vector si creaza o matrice.

Pe prima linie contine elementele vectorului (copie). A doua linie contine minimul dintre fiecare 2 elemente din vector. A treia linie contine minimul dintre fiecare 4 elemente din vector si asa mai departe.
Acest algoritm are avantajul ca este foarte rapid cu timpul de interogare O(1), dar destul de lent necesitand un spatiu mult mai mare de O(n log n).

3. Segment Tree:
Algoritmul primeste un vector si creaza un arbore binar.
Vectorul este impartit in 2, iar apoi jumatatile sunt impartite si ele la randul lor in 2 si tot asa, iar minimul elementelor este stocat mai sus ca parinte ale celor 2 noduri in arbore. Arborele este binar si complet.
Acest algoritm are avantajul ca este destul de rapid, dar este foarte complex si greu de implementat.

EVALUARE
Testele au fost generate aleatoriu de un program java cu care puteam sa specific dimensiunea lor. Pentru verificarea corectitudinii algoritmilor am rulat testele pe cei 3 algoritmi si am comparat rezultatele, acestea fiind indentice. Deci voi presupune ca sunt corecte.
Testele contin numere de dimensiuni mici, mari si foarte mari pentru a testa eficienta si timpul de executie pe mai multe cazuri.

Testele au fost rulate pe un sistem de calcul cu urmatoarele specificatii:
Tip: Linux Ubuntu de pe masina virtuala;
Operating System: Windows 10 Education 64-bit;
Procesor: Intel® Core™ i7-7700Hq CPU @ 2.80GHz 2.80GHz;
Memorie: 2048 MB RAM.

Timpii de rulare se pot observa in acest table:

	Square Root Decomposition	Sparse Table	Segment Tree
Test10.in	168	520	699
Test1.in	5	29	35
Test2.in	10	52	55
Test3.in	58	90	122
Test4.in	204	564	712
Test5.in	521	1222	15645
Test6.in	354	711	945
Test7.in	497	1200	13525
Test8.in	854	152200	58111

Test9.in	30	87	140

Aici sper ca timpii dau destul de aproape de realitate deoarece am avut mari dificultati in creerea unui script pentru a vedea timpii de rulare. Sper ca nu am calculatorul prea incet, dar totusi se pot observa caracteristicile temporale de la o marime al imputului la alta.
Asa cum ma asteptam ultimul algoritm fiind cel mai complex si complicat are un timp foarte mare de executie fata de ceilalti algoritmi, iar primul algorit pare cel mai rapid.

CONCLUZIE

Daca ar fii sa ma intalnesc cu o problema de RMQ in viata de zi cu zi, as alege un algoritm pe urmatoarele criterii: performantele temporale, limitarile de memorie, complexitatea implementarii. In functie de caz vom alege algoritmul cel mai potrivit.

Pentru embedded trebuie sa consume cat mai putina memorie. Pentru eficienta o sa facem tradeoff memorie-raspuns in timp.

Dupa mine Sparce Table ar fii o alegere buna din punct de vedere al ambelor probleme.

BIBLIOGRAFIE

https://www.geeksforgeeks.org/range-minimum-query-for-static-array/

https://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query/

https://www.geeksforgeeks.org/sqrt-square-root-decomposition-technique-set-1-introduction/

https://en.wikipedia.org/wiki/Range_minimum_query

https://cp-algorithms.com/sequences/rmq.html

https://web.stanford.edu/class/cs166/lectures/01/Small01.pdf

https://www.youtube.com/watch?v=ZBHKZF5w4YU

https://www.youtube.com/watch?v=c5O7E_PDO4U

https://www.youtube.com/watch?v=0CPBgO4Mg-E