Range Minimum Query

Brebu Costin-Bogdan

Facultatea de automatiă și calculatoare - 321 CD

- **Introducere**

  În stiința computerelor, problema elementului minim dintr-un interval se ocupă cu găsirea valorii minime dintr-un subvector al unui vector cu elemente comparabile.

  Există multe aplicații in legătură cu aceasta problemă, dar cele mai cunoscute sunt:

  - The lowest common ancestor problem. Un exemplu relevant este analiza ADN pentru a gasi stramoșul comun a două specii.
  - The longest common prefix problem.Gasirea celui mai lung prefix comun a doua stringuri.

  Metodele alese de mine pentru rezolvarea problemei de „Range minimum query" sunt: Sparse Table, Segment Tree, Sqrt-decomposition.

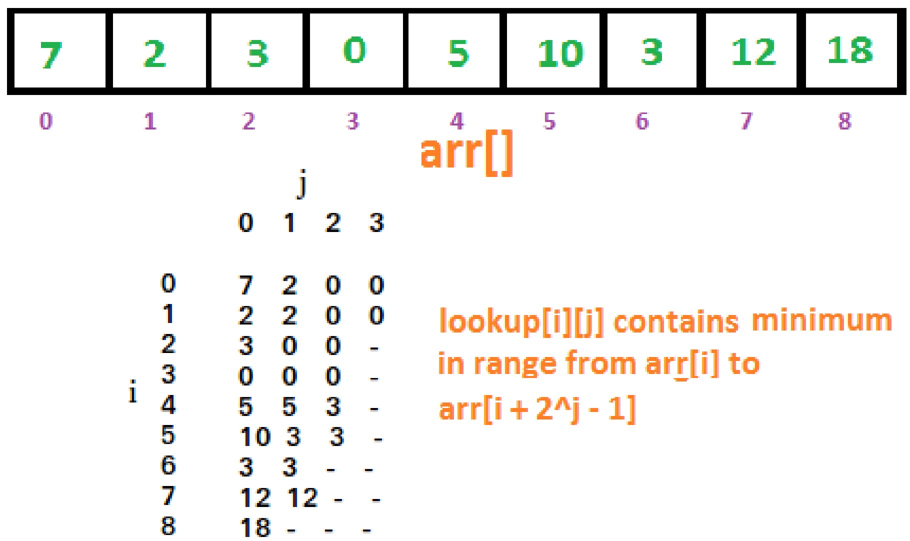
  Testarea corectitudinii algoritmilor aleși se face pe baza evaluării ouputului rezultat prin rularea unor teste suficient de complexe si de variate. Voi valida corectitudinea outputului comparand rezultatele algorimilor între ele avân aceleași inputuri.

- **Prezentarea soluțiilor**
  - **Sparse Table**

    Un sparse table este o matrice a cărei elemente respecta regula:

$$st_{ij} = query\left(i, i + 2^j - 1\right)$$

Pentru a afla minimul pe intervalul $\left[i, i + 2^j - 1\right]$, doar trebuie extras elemental de pe pozitia (i,j) din matricea creeata în preprocesare.
Imaginea de mai jos arată un exemplu concret de aplicare a unui sparse table pe un vector prestabilit.



Complexitate pe query: $O(1)$
Complexitate pe procesare: $O(n * log(n))$
Complexitate totală: $O(q + n(n))$
Memorie folosită: $O(n * \log(n))$

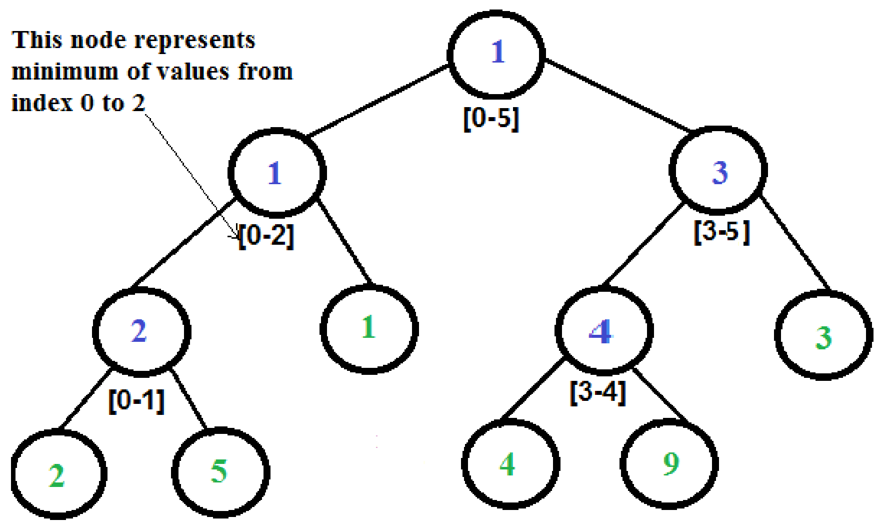
Avantaje : complexitate buna pe interval, structură de date simpla
Dezvantaje : vectorul nu poate fi modificat fara a relua preprocesarea

Segment tree

Un “segment tree” este în general o structură flexibilă, cu care se pot rezolva o mulțime de probleme. De exemplu, cu un “segment tree” bidimensional putem afla suma elementeor unei submatrice a unei matrice date.

Fiind dat un interval, se determină o reuniune de noduri care conțin acel interval pentru a determina minimul cu o eficiența mai buna.

Într-un segment tree sunt in total 2n-1 noduri (n = numarul de elemente al vectorului), fiecare valoare a nodului fiind determinată o singură dată.



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

Complexitate pe query: $O(log(n))$

Complexitate pe procesare: $O(n)$

Complexitate totală: $O(q * \log(n) + n)$

Memorie folosită: $O(n)$

Complexitate de update: $O(log(n))$

Avantaj: complexitate bună, admite actualizări pe vector

Dezavantaj: implementare dificilă

- **Sqrt-decomposition**

  In preprocesare se împarte vectorul in bucati de lungime $\sqrt{n}$, iar pentru fiecare subvector se reține minimul într-un alt vector creat de noi.

  Aproape toate bucațile au lungimea $\sqrt{n}$ cu excepția ultimei. Ultima bucata are mai putine elemente daca numarul de elemente al vectorului (n) nu se imparte exact la $\sqrt{n}$.

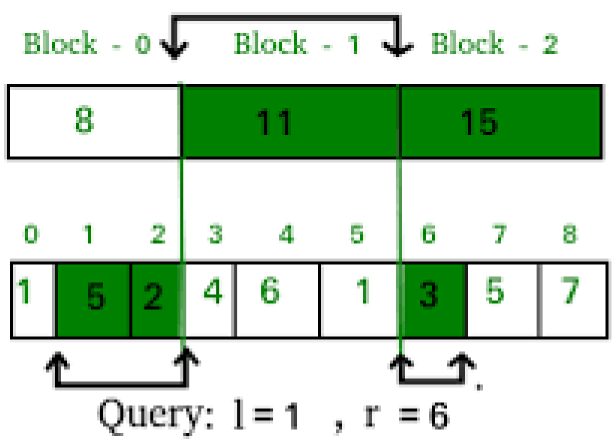
  Pentru aflarea valorii minime pe intreval [i, j], se imparte acest interval in blocuri de lungime $\sqrt{n}$. Minimul unui astfel de bloc se afla in $O(1)$ având deja rezultatul precalculat. Rămân de calculat doar eventualele capete care nu au fost acoperite în întregime (dacă există rămase astfel de intervale).

  Un exemplu de imparțire al vectorului este următorul, unde

  $s = \sqrt{n}$, iar n reprezinta numarul de elemente al vectorului:

$$\underbrace{a[0], a[1], \ldots, a[s-1]}_{\text{b[0]}}, \underbrace{a[s], \ldots, a[2s-1]}_{\text{b[1]}}, \ldots, \underbrace{a[(s-1) \cdot s], \ldots, a[n-1]}_{\text{b[s-1]}}$$

  Pentru o intelegere mai bună a algoritmului voi explica pe imaginea de mai jos.



Se observă că vectorul de elemente a fost împărțit în trei bucăți egale de lungime $\sqrt{9}$. Dacă dorim aflarea valorii minime din intervalul [1, 6], vom străbate elementele de pe pozitiile 1 si 2 într-un loop si vom determina minimul acestora (deoarece nu sunt acoperite in intregime), urmând să luăm minimul precalculat din blocul 1 si sa il comparam cu minimul obtinut precedent astfel determinând un eventual nou minim, pentru ca la final sa incheiem prin a compara minimul curent cu elementul de pe pozitia 6 (fiind singurul ramas).

Complexitate pe query: $O\left(\sqrt{n}\right)$

Complexitate pe procesare: $O(n)$

Complexitate totală: $O\left(q * \sqrt{n} + n\right)$

Memorie folosită: $O\left(\sqrt{n}\right)$

Complexitate de update: $O\left(\sqrt{n}\right)$

Avantaje: simplitate, implementare ușoara, admite operație de update

Dezavantaje: complexitate proastă

- **Evaluare**

  Fiecare algoritm in parte este testat cu nouă fisiere de input.

  Pentru toate cele trei cazuri precizate in enunț: N>M, N=M, N<M (N=numarul de elemente al vectorului, M= numarul de intervale), avem alte 3 cazuri separate, cand elementele din vector sunt asezate crescatoar, descrescător și respectiv aleator.

  Am urmatoarele cazuri:

  - Test0 N = 50, M = 5, elementele vectorului generate la intamplare
  - Test1 N = 100, M = 100, elementele vectorului generate la intamplare
  - Test2 N = 100, M = 200, elementele vectorului generate la intamplare
  - Test3 N = 100, M = 5, elementele vectorului generate in ordine crescatoare
  - Test4 N = 100, M = 100, elementele vectorului generate in ordine crescatoare
  - Test5 N = 100, M = 200, elementele vectorului generate in ordine crescatoare
  - Test6 N = 100, M = 5, elementele vectorului generate in ordine descrescatoare
  - Test7 N = 100, M = 100, elementele vectorului generate in ordine descrescatoare
  - Test8 N = 100, M = 200, elementele vectorului generate in ordine descrescatoare

    Sistemul pe care au fost rulate testele are următoarele specificații: Windows 10 Pro, 64-bit, Processor AMD Ryzen 5 3600, 16GB DDR4.

    Outputurile algoritmilor au iesit in regulă, fară erori. Nu exista diferente intre fișierele de out ale celor trei algoritmi studiați(avand in vedere ca am avut aceleasi set de teste pt toti 3. Am afisat pentru fiecare intelval in parte, pe cate o linie separata, minimul acestuia. Am facut o testare manuala pentru a vedea dacă rezultatele obținute sunt bune și am ajuns la concluzia ca nu ar exista erori.

- **Concluzii**

  În practică aș utiliza algoritmul "Segment tree" pentru a rezolva aceasta problema de "Range minimum query" „ chiar dacă are o implementare mult mai grea decât a celorlalți doi algoritmi studiați, deoarece are o eficiență mai bună si un randament mai bun pentru rularea testelor mari. De asemenea in comparație cu "sqrt decomposition" , are beneficiul de a putea modifica vectorul fară a relua preprocesarea și are un timp de execuție mai bun decât al algoritmului "sparse table" .

- **Bibliografie**

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

https://cp-algorithms.com/data_structures/sqrt_decomposition.html

https://cp-algorithms.com/data_structures/sparse-table.html

https://cp-algorithms.com/data_structures/sqrt-tree.html

https://www.geeksforgeeks.org/sparse-table/

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

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

Documentele lăsate drept model pe moodle