Problema menţinerii medianei

Problema menţinerii medianei cere identificarea unui mod de a păstra mereu valoarea mediană curentă a unei mulţimi dinamice de numere. Spre exemplu, ne putem imagina că sunt generate numere în mod aleator şi memorate într-un vector. La orice moment trebuie să cunoaştem valoarea mediană, cu alte cuvinte să o putem accesa în timp constant.

Menţionăm că în cazul unui număr par de elemente, mediana va fi media celor două elemente din mijloc (mijlocul fiind considerat pe vectorul sortat). Există variante ale problemei care permit alegerea oricăreia din cele două elemente din mijloc ca valoare mediană.

Practic, problema ne cere ca întotdeauna să avem în evidenţă cele două elemente din mijloc, care reprezintă candidaţi pentru valoarea mediană.

Cum procedăm?

Prezentăm întâi abordarea cea mai adecvată şi eficientă, urmând ca în secţiunea următoare să discutăm alte posibile soluţii.

Ansamblele (heap-uri) sunt organizate fie ca max-ansamble (orice element conţine o valoare mai mică sau egală decât cea asociată părintelui său), fie ca min-ansamble (orice element conţine o valoare mai mare sau egală decât cea asociată părintelui).

Astfel, dacă jumătatea superioară (valorile mai mari) ar fi păstrate într-o structură de min-ansamblu, cel mai mic element al acestei mulţimi, care este chiar unul din elementele de mijloc dorite, se va afla permanent în rădăcina ansamblului. Simetric, jumătatea inferioară (valorile mai mici) pot fi stocate într-un max-ansamblu, a cărui rădăcină va corespunde elementului cel mai mare din mulţime (al doilea element de mijloc căutat). În continuare, denumim max-ansamblul, respectiv min-ansamblul, descris mai sus *ansamblu inferior*, respectiv *ansamblu superior*.

Atunci când inserăm un element nou într-unul din cele două ansamble şi dimensiunea ansamblului respectiv depăşeşte jumătate din numărul total de elemente, putem să extragem elementul în plus şi să îl inserăm în celălalt ansamblu. Deci, aplicăm un mecanism de ,,echilibrare" celor două ansamble. Acesta este invariantul structurii de date formate din cuplul min-ansamblu, max-ansamblu: trebuie să ne asigurăm că dimensiunile lor nu diferă prin mai mult de un element.

Facem convenţia că mediana va fi rădăcina ansamblului inferior (deci cea mai mare valoare a elementelor din max-ansamblu).

Inserarea

Inserarea unui element nou urmează raţionamentul dat mai jos.

- Dacă valoarea de inserat este mai mică sau egală decât mediana, inserăm în ansamblul inferior.
- Altfel, inserăm în ansamblul superior.

Dacă numărul de elemente este par, dimensiunile celor două ansamble trebuie să fie egale, iar în cazul unui număr impar de elemente, drept consecinţă a convenţiei antemenţionate, ansamblul inferior conţine un element în plus faţă de ansamblul superior.

1. Dacă ansamblele au dimensiune egală
   (a) Dacă ansamblul superior conţine elemente şi valoarea de inserat este mai mare decât rădăcina sa
      - Extragem rădăcina ansamblului superior şi o inserăm în ansamblul inferior
      - Inserăm noua valoare în ansamblul superior
   (b) Altfel,
      - Inserăm noua valoare în ansamblul inferior
2. Altfel,
   (a) Dacă valoarea de inserat este mai mică decât rădăcina ansamblului inferior
      - Extragem rădăcina ansamblului inferior şi o inserăm în ansamblul superior
      - Inserăm noua valoare în ansamblul inferior
   (b) Altfel,
      - Inserăm noua valoare în ansamblul superior

Obţinerea medianei

1. Dacă ansamblul inferior nu conţine elemente
   - Mediana este rădăcina (singurul element al) ansamblului superior
2. Altfel, dacă ansamblul superior nu conţine elemente
   - Mediana este rădăcina (singurul element al) ansamblului inferior
3. Altfel, dacă dimensiunile ansamblelor sunt egale
   - Mediana este media aritmetica a rădăcinilor celor două ansamble
4. Altfel, dacă dimensiunea ansamblului inferior este mai mare decât cea a ansamblului superior
   - Mediana este rădăcina ansamblului inferior
5. Altfel,
   - Mediana este rădăcina ansamblului superior

Punctele cheie ale soluţiei

- Folosim un min-ansamblu şi un max-ansamblu pentru ordonarea numerelor
- Organizăm inserările astfel încât fiecare ansamblu să păstreze jumătate din elemente
- Dacă numărul de elemente este impar, mediana este rădăcina max-ansamblului
- Dacă numărul de elemente este par, mediana este media aritmetică a rădăcinilor celor două ansamble
- Ambele variante permit accesarea în timp $O(c)$ a valorii mediane, unde c este o constantă
- Actualizările ansamblelor după o inserare se fac în timp $O(\log n)$, unde n este numărul de elemente

Discuţie asupra modului de abordare a problemei

Menţinerea unui vector sau a unei liste sortat(e)

La prima impresie s-ar putea opta pentru utilizarea unei liste înlănţuite sau a unui vector care să păstreze elementele sortate. Dar sortarea elementelor s-ar putea face doar în timp $O(n \log n)$. Având în vedere că sortarea tuturor elementelor depăşeşte cerinţa problemei, fiind nevoie doar de menţinerea medianei, putem căuta o soluţie mai performantă.

Utilizarea arborilor binari de căutare echilibraţi

Putem considera şi o abordare folosind arbori binari de căutare, eventual echilibraţi.

Discutie.

Găsirea medianei...
cu partitia de la QuickSort

Discutăm şi problema găsirii medianei, dacă aceasta ar avea o soluţie în timp constant, atunci pentru accesarea medianei am putea folosi algoritmul de găsire a medianei. Această variantă nu este satisfăcătoare, în cel mai bun caz obţinem un timp liniar.

Se poate găsi o soluţie recursivă a problemei folosind varianta generalizată a acesteia, aceea în care căutăm a $\lfloor \frac{n}{2} \rfloor$-a statistică de ordine sau media aritmetică între a $\lfloor \frac{n}{2} \rfloor$-a şi a $\lfloor \frac{n}{2} \rfloor + 1$-a statistică de ordine. Putem folosi algoritmul de selecţie al celui de-al k-lea minim, pentru $k = \lfloor \frac{n}{2} \rfloor$, şi eventual $k = \lfloor \frac{n}{2} \rfloor + 1$, în funcţie de n par sau impar. Mai exact după k iteraţii ale algoritmului de sortare prin selecţie obţinem mediana.

Pentru un rezultat mai bun se poate folosi varianta aleatoare a unui algoritm divide şi impera care să găsească al k-lea minim.

Se poate împărţi (operaţia *split*) lista de numere L, pentru orice număr v din ea, în trei subliste: L_l elemente mai mari decât v, L_e elemente egale cu v şi L_g elemente mai mici decât v. Căutarea se restrânge după fiecare *split* pe una din liste. De exemplu, dacă k este mai mare decât $|L_l|$, dar mai mic decât $|L_l|+|L_e|$, căutarea se continuă pe lista L_e, actualizând şi k la valoarea $k-|L_l|$[1].

Operaţia *split* se poate face ,,pe loc" dintr-o singură parcurgere. Importantă este însă şi alegerea rapidă a lui v, elementul care determină cele trei subliste. Ideal acesta ar trebui să producă două liste L_l şi L_g aproape egale ca număr de elemente. Dacă această partiţionare ar avea loc atunci timpul de execuţie ar fi dat de recurenţa

$$T(n) = T(\frac{n}{2}) + O(n).$$

Dar aceasta ar însemna că v ar trebui să fie **chiar mediana**. Cum scopul alegerii lui v este acelaşi ca şi în cazul alegerii pivotului în algoritmul de sortare rapidă (quicksort), putem utiliza o procedură de partiţionare din cele utilizate de quicksort. Mai mult, putem să îl alegem pe v aleator. Pentru această variantă se poate arăta că timpul de execuţie este liniar în cazul mediu.

Putem adapta algoritmul quicksort (varianta recursivă, implementată cu tehnica divide şi impera, folosind un pivot ales aleator) pentru găsirea medianei în acest mod într-un vector $A[1..n]$.

Caută–Minimul–k(p, u)

1. Alegem poz_v folosind partiţia aleatoare din subvectorul $A[p..u]$.
2. Dacă poz_v este egal cu k, atunci algoritmul se termină şi returnăm $A[v]$.
3. Altfel, dacă poz_v este mai mic decât k, continuăm căutarea pe subvectorul $A[poz_v + 1..u]$, printr-un apel recursiv cu $p = poz_v + 1$ şi $u = u$.
4. Altfel, continuăm căutarea pe subvectorul $A[p..poz_v - 1]$, printr-un apel recursiv cu $p = p$ şi $u = poz_v - 1$.

Algoritmul are ca date de intrare indicii de început p şi de sfârşit u, care indică segmentul de vector pe care se efectuează căutarea. Iniţial apelul se va face pentru întreg vectorul A: $p = 1$, $u = n$.

Referinţe

1. G. Laakmann – *Cracking the Coding Interview* (4^{th}ed.), CareerCup, LLC. 2008
2. S. Dasgupta, C.H. Papadimitriou, U.V. Vazirani – *Algorithms.* 2006

[1]Notăm cu $|\cdot|$ numărul de elemente din listă.