Tabăra de pregătire a Lotului Naţional de Informatică
Alba Iulia, 11-18 iunie, 2010
Baraj 2 Seniori

Problema Carray, Autor: Mugurel Ionuţ Andreica
Descrierea soluţiei

O soluţie trivială constă în construcţia explicită a fiecărui vector. În cazul unei operaţii de concatenare se alocă spaţiu în memorie pentru numărul total de elemente din cei doi vectori concatenaţi, iar elementele respective sunt copiate în spaţiul de memorie alocat. În acest fel, răspunsul la fiecare întrebare poate fi dat în timp `O(1)`, prin accesarea locaţiei de memorie din vector. Această soluţie, însă, va depăşi limitele de timp şi de memorie specificate. În primul rând, trebuie să observăm că numărul de elemente dintr-un vector poate creşte exponenţial. De exemplu, pornind de la un vector cu un singur element şi efectuând `K` operaţii de concatenare cu el însuşi, putem obţine un vector având 2^K elemente.

O soluţie mult îmbunătăţită este următoarea. Vom construi un graf orientat aciclic ce va conţine câte un nod pentru fiecare vector. Iniţial, graful va conţine câte un nod pentru cei `N` vectori iniţiali. Fiecare nod `i` al grafului va avea un vecin stâng şi un vecin drept: `left(i)` şi `right(i)`. În momentul în care creăm vectorul `i` (`N+1≤i≤N+M`), setăm `left(i)=a(i)` şi `right(i)=b(i)`. De asemenea, pentru fiecare vector `i` vom menţine lungimea sa, `len(i)` (adică numărul său de elemente), Avem `len(i)=1` pentru `1≤i≤N`. Pentru `N+1≤i≤N+M` vom seta `len(i)=len(left(i))+len(right(i))`. În felul acesta, fiecare operaţie de concatenare este efectuată în timp `O(1)`. Pentru a răspunde la o întrebare `(i,p)` vom proceda în felul următor. Vom iniţializa o valoare `x=i`. Cât timp `len(x)>1` efectuăm următorii paşi: dacă `left(x)≥p` unci setăm `x=left(x)` altfel setăm `p=p-len(left(x))` şi apoi `x=right(x)`. Acest algoritm simplu este similar cu cel folosit pentru a găsi al `K`-lea element într-un arbore binar de căutare. Complexitatea unei interogări este proporţională cu lungimea drumului de la nodul `i` la nodul corespunzător vectorului de lungime `1` la care se ajunge. În cel mai rău caz, acest drum poate avea o lungime de ordinul `O(M)`.

În continuare vom îmbunătăţi modul de tratare al unei interogări din soluţia descrisă anterior. Pentru început vom aplica o metodă similară celei cunoscute sub numele de *"heavy path decomposition"* pentru arbori. Mai exact, pentru fiecare nod `i` vom alege un *"părinte"* `parent(i)` dintre vecinii săi `left(i)` şi `right(i)`. Dacă `len(left(i))>len(right(i))` atunci `parent(i)=left(i)` altfel `parent(i)=right(i)`.

Legăturile `(i,parent(i))` formează o mulţime de arbori (în mod similar, dacă graful ar fi un arbore, legăturile `(i,parent(i))` ar forma o mulţime de drumuri). Să revenim acum la algoritmul de tratare al unei interogări `(i,p)`. Drumul parcurs în graf de la nodul `i` la nodul final de lungime `1` va trece prin unul sau mai mulţi dintre arborii formaţi de legăturile `(i,parent(i))`. Considerând `LMAX` ca fiind lungimea maximă a unui vector, acest drum poate conţine cel mult `log(LMAX)` muchii care nu fac parte din niciun arbore. Motivul este foarte simplu: în momentul în care trecem de la un nod `x` la un vecin al său `y`, iar muchia `(x,y)` nu face parte din niciun arbore, ştim în mod cert că `len(y)≤len(x)/2`. Astfel, după trecerea prin `log(LMAX)` a astfel de muchii am ajunge în mod cert la un nod de lungime `1`. Aşadar, ceea ce vom realiza în continuare este să parcurgem în mod eficient muchiile din cadrul unui arbore.

În momentul în care am ajuns la un nod `x` cu o valoare a lui `p`, drumul nostru va urma legăturile `(x,parent(x))`, `(parent(x), parent(parent(x)))`, `...`, ş.a.m.d., până când va ajunge ori la un nod de lungime `1`, ori până când va ajunge la o muchie care nu face parte din arbore şi pe care va trebui să o urmeze. Vom determina acel nod `y` până la care toate muchiile urmate de drumul nostru

începând de la x fac parte din arbore. Pentru a determina acest nod y în mod eficient, vom proceda după cum urmează.

Vom asocia fiecărei muchii $(i,parent(i))$ o valoare: dacă $parent(i)=left(i)$ atunci $val(i,parent(i))=0$ altfel $val(i,parent(i))=len(left(i))$. Să notăm prin $sum(u,v)$ suma valorilor muchiilor de pe drumul de la u la v într-un arbore (v este un strămoş al lui u). Putem obţine orice valoare $sum(u,v)$ în timp $O(1)$ dacă calculăm în prealabil nişte sume prefix $psum(i)$=suma valorilor muchiilor de pe drumul de la rădăcina arborelui la nodul i.

Nodul y căutat este cel mai depărtat strămoş al lui x pentru care $p-sum(x,y) \geq 1$ şi $p-sum(x,y) \leq len(y)$ (sau chiar nodul de lungime 1 ce reprezintă rădăcina arborelui). Trebuie observat că o dată ce condiţia ($p-sum(x,y) \leq 0$ sau $p-sum(x,y)>len(y)$) este nesatisfăcută pentru un nod y, ea nu va mai fi satisfăcută pentru niciun strămoş al lui y din arbore (demonstraţia este destul de simplă şi se bazează pe inducţie matematică : se demonstrează că dacă condiţia e falsă pentru un nod z, atunci ea e falsă şi pentru $parent(z)$).

Pentru a determina cel mai apropiat strămoş y pentru care condiţia este aevărată vom calcula în prealabil, pentru fiecare nod i, valorile $Anc(i,j)$=strămoşul lui i situat cu 2^j nivele mai sus în arbore. $Anc(i,0)=parent(i)$ şi $Anc(i,j \geq 1)=Anc(Anc(i,j-1), j-1)$. Având aceste valori, vom iniţializa o valoare $b=log(LMAX)$ şi $y=x$. Cât timp $b \geq 0$ execută: (1) dacă condiţia este adevărată pentru $Anc(y,b)$ atunci setează $y=Anc(y,b)$; (2) $b=b-1$. La final vom seta $p=p-sum(x,y)$.

Determinarea fiecărui nod y se realizează, astfel, într-o complexitate $O(log(M))$. Răspunsul la o interogare este dat în timp $O(log(M) \cdot log(LMAX))$.

Complexitatea întregului algoritm este $O(N+M+M \cdot log(M)+Q \cdot log(M) \cdot log(LMAX))$.

Algoritmul prezentat utilizează $O(M \cdot log(M))$ memorie. El poate fi modificat pentru a utiliza doar $O(M)$ memorie (şi păstrând complexitatea de timp), în felul următor. În loc să memorăm $O(log(M))$ strămoşi pentru fiecare nod, vom descompune fiecare arbore obţinut folosind metoda “heavy-path decomposition” – astfel, fiecare arbore este descompus în mai multe căi. Pentru a găsi nodul y căutat pornind de la un nod x, vom trece prin $O(log(M))$ căi în cadrul arborelui, iar la ultima cale s-ar putea să trebuiască să aplicăm algoritmul prezentat (să trebuiască să cunoaştem al 2^j-lea strămoş ; însă, dacă memorăm fiecare cale în câte un vector, putem obţine al 2^j-lea strămoş pur şi simplu accesând componenta corespunzătoare a vectorului).