

pviz - Descrierea soluției

Autor Adrian Diaconu

Soluție 1 (Adrian Diaconu)

Problema se poate rezolva folosind metoda programării dinamice. Se va construi o matrice H în care $H[i][j]$ = numărul permutărilor de lungime i care conțin primele j elemente vizibile. Relația de recurență este:

$$H[i][j] = H[i-1][j-1] + (X[j]-i+1) * H[i-1][j]$$

unde $X[i]$ reprezintă al i -lea element vizibil. Primul termen reprezintă cazul în care pe poziția i se pune cel de-al j -lea element vizibil. Al doilea termen reprezintă faptul că pe poziția i se plasează un element ce nu este vizibil, acesta putând fi ales din numerele mai mici decât $X[j]$, care nu au fost puse încă în permutare. Rezultatul final este memorat în $H[n][m]$.

Complexitate $O(n^2)$. Memorie $O(n)$ deoarece sunt necesare doar ultimele două linii ale matricii.

La aceeași soluție se poate ajunge și pornind de la o recurență de forma:

$$H[i][j] = \sum_{k=0}^i H[k][j-1] * A_{X[j-1]-k}^{i-k}$$

care poate fi prelucrată pentru a se ajunge la recurența prezentată anterior.

O soluție în $O(n^3)$ ar fi obținut aproximativ 60 de puncte.

Soluție 2 (Csaba Pătcas)

Se poate aplica metoda programării dinamice și în felul următor:

$H[i][j]$ = numărul de permutări de lungime j , în care pe ultima poziție avem fixat elementul i din lista elementelor vizibile pe poziția j , și primele $i-1$ sunt deja vizibile. Aici, formula de recurență pentru soluția în $O(n^3)$ se calculează fixând al $i-1$ -lea element vizibil pe poziția k și considerând toate elementele care pot apărea între poziția k și poziția j , obținându-se

$$H[i][j] = \sum_{k=1}^{j-1} H[i-1][k] * A_{X[i-1]-k}^{j-k-1}$$

Această soluție se poate reduce la complexitatea $O(n^2)$ prin aplicarea recurenței:

$$H[i][j] = H[i][j-1] * (X[i-1]-j+2) + H[i-1][j-1]$$

Dacă folosim această recurență, nu trebuie să uităm de posibilitățile de a alege elementele după ultima poziție vizibilă.

Așadar soluția finală se obține din suma: $\sum_{i=1}^n H[m][i] * A_{X[m]-i}^{n-i}$

albinuta - Descrierea soluției

Autor Emilian Miron

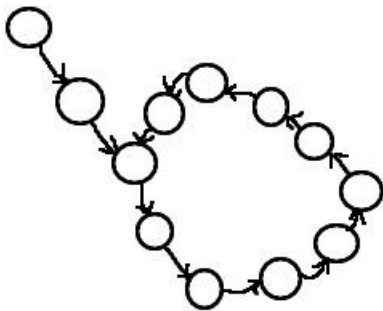
Putem caracteriza starea albinutei ca un tuplu (t, n) cu semnificatia la timpul t albinuta se afla in nodul n . Notand cu V_k vectorul de adiacenta al nodului k ($V_{k,i}$ – al i -lea vecin al nodului k , $1 \leq i \leq L_k$) obținem următoarea regula de tranziție:

Pentru (t, n) urmatorul nodul ales va avea pozitia $1 + (t - 1) \% L_n$:

$$(t, n) \rightarrow (t + 1, V_{n, (1 + (t - 1) \% L_n)})$$

Observam din relatia de mai sus ca pentru a afla succesorul nodului n la un moment t nu conteaza valoarea exacta a lui t ci doar restul impartirii acestuia la L_n sau la orice multiplu de L_n . Restul impartirii lui t la $\text{cmmmc}(L_k)$ este deci suficient pentru a caracteriza starea albinutei.

CMMMC-ul maxim pentru datele de intrare este $\text{MAX} = 4 \cdot 9 \cdot 5 \cdot 7 \cdot 11 \cdot 13$, deci spatiul starilor este o multime finita de maxim $N \cdot \text{MAX}$ elemente. De la un anumit timp starea albinutei va cicla.



Graful starilor va fi similar cu cel din stânga, format dintr-o coada de X elemente urmata de un ciclu de lungime Y .

Daca $T > X + Y$ putem reduce T la $X + 1 + (T - X - 1) \% Y$.

Memoria disponibila este suficienta pentru a pastra un vector de marcare pentru stari care ne permite sa determinam X si Y , iar pentru fiecare T_k putem sa recalculam drumul albinutei pana la T_k redus la maxim $X + Y$.

Complexitatea solutiei este $O(N \cdot \text{MAX} + Q \cdot N \cdot \text{MAX})$, cu memorie $O(N \cdot \text{MAX})$.

Se poate retine si vectorul de noduri parcurse pana la $X + Y$, raspunzand apoi la intrebari in $O(1)$ pentru fiecare T , insa trebuie avut grija la limita de memorie.

O solutie alternativa este presupunerea ca traseul albinutei va cicla. De la un moment de timp i (egal cu X din solutia de mai sus) sirul $T_i, T_{i+1}, \dots$ va fi un sir periodic. Se incearca succesiv valorile i pana cand se gaseste un sir periodic. Se poate determina in timp liniar daca un sir este periodic.

Este posibila rezolvarea cu $O(1)$ memorie aditionala cu ajutorul unui algoritm de detectie a ciclurilor similar cu ideea de la algoritmul de factorizare Pollard-Rho.

http://en.wikipedia.org/wiki/Cycle_detection

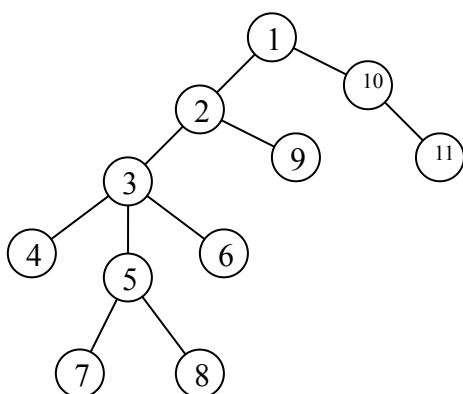
curent - Descrierea soluției

Autor Ionuț Fechete

Problema se poate rezolva facand o parcurgere in adancime pentru fiecare zi de interes pentru a afla numarul de noduri accesibile din nodul 1 la acel moment. Astfel se raspunde la o intrebare in complexitatea $O(N)$ si actualizarea se face in $O(1)$ marcand nodul ca fiind cu siguranta arsa. O astfel de rezolvare obtine 25-30 de puncte.

O alta abordare presupune stocarea in memorie a numarului de noduri accesibile pentru fiecare nod al arborelui, initial acest numar este egal cu numarul de noduri din subarboarele a carui radacina este nodul. Pentru a se actualiza acest vector trebuie parcurs arborele plecand de la nodul care trebuie actualizat, mergand din tata in tata pana se ajunge la un nod cu siguranta arsa, deci complexitatea este $O(N)$ in cel mai rau caz, iar de raspuns se raspunde in $O(1)$. Aceasta rezolvare obtine 35-40 de puncte.

Solutia oficiala foloseste arbori de interval si parcurgerea Euler a arborelui pentru a face o actualizare in $O(\log N)$ si pentru a raspunde in $O(1)$. Folosind parcurgerea Euler a arborelui obtinem nodurile intr-o anumita ordine astfel incat toate nodurile dintr-un anumit subarboare se afla intr-un interval compact din secventa.



O parcurgere Euler in care scriem un nod doar o data pentru arborele din figura genereaza secventa:

1 2 3 4 5 7 8 6 9 10 11

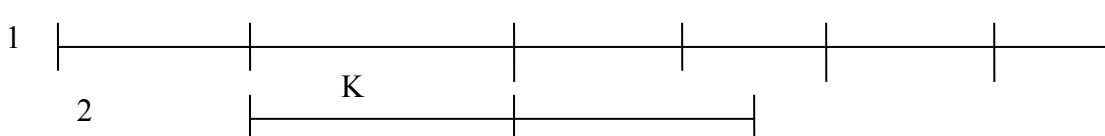
subsecventa determinata de pozitiile a 2-a si a 9-a din aceasta secventa contine toate nodurile din subarboarele al carui radacina este 2

subsecventa determinata de pozitiile a 5-a si a 7-a din aceasta secventa contine toate nodurile din subarboarele al carui radacina este 5

O alta proprietate a acestei secvente este ca oricare ar fi doua intervalele pentru care se fac actualizari acestea fie sunt disjuncte, fie unul dintre ele este continut de celalalt.

Dupa ce am obtinut aceasta secventa putem folosi arbori de intervale pentru a rezolva problema in felul urmator: pentru fiecare interval retinem numarul de noduri accesibile din nodul radacina corespunzator intervalului **S[nod]**(dupa cum am precizat mai sus un interval din secventa este echivalent cu un subarboare

din arborele dat la intrare) și numărul de stramosi cu sigurantele arse ale nodului **Arsuri[nod]**. Initial toate nodurile au sigurantele bune deci pentru fiecare interval numărul de noduri accesibile este egal cu lungimea intervalului. Pentru o actualizare mergem initial în jos în arborele de intervale ca la o actualizare normală, obținem o împărțire a intervalului ce trebuie initializat în k intervale $A_1 A_2 A_3 \dots A_K$, pentru fiecare din aceste intervale incrementăm **Arsuri[A_i]** dacă actualizarea este o arsură sau decrementăm dacă actualizarea este o înlocuire. Apoi mergem în sus pe arbore actualizând **S[nod]** pentru nodurile stramosi ca la o actualizare normală doar că **S[nod]=S[left]+S[right]** numai atunci când **Arsuri[nod]** este 0.



De exemplu dacă avem de actualizat intervalul de sus (1) vom updata **Arsuri[x]** pentru fiecare interval din împărțirea făcută de arborele de intervale (pentru a evita actualizarea pentru fiecare nod). Când vom actualiza intervalul de jos (2), intervalul **Arsuri[K]** va fi incrementat a doua oară, nodurile din acel interval având doi stramosi cu sigurantele arse.

Pentru a răspunde la o întrebare folosim **S[1]**, numărul de noduri accesibile din 1.

O descriere a structurii de date “arbore de intervale” poate fi găsită la următorul link:
<http://infoarena.ro/arbori-de-intervale>

iar o descriere a parcurgerii euler a unui arbore aici:

http://www.ginfo.ro/revista/11_4/probleme.pdf