

Algoritmi si Structuri de Date

- seminar VI -

Hash.

In romana: tabel de dispersie.

Hash-urile sau dictionarele sunt structuri de date abstracte care va permit stocarea datelor si care au complexitati bune pentru operatiile de inserare, cautare si stergere ($O(1)$).

Problema: Se da o multime care contine N numere intregi si M operatii de trei tipuri:

- operatie de tipul 1 a: inserati elementul a in multime; daca exista nu faceti nimic.
- operatie de tipul 2 a: stergeti elementul a daca se afla in multime.
- operatie de tipul 3 a: afisati 1 daca elementul a se afla in multime, 0 altfel.

Solutie brute-force.

O solutie evidenta pentru aceasta problema este de a salva toate numerele intr-un vector, si cand primim o :

- operatie de tipul 1:
 - parcurgem vectorul
 - daca elementul a nu exista il inseram la final
 - altfel nu facem nimic
- operatie de tipul 2:
 - parcurgem vectorul
 - daca elementul a exista il inlocuim cu -1 (presupunem ca folosim doar nr. nat.)
 - altfel nu facem nimic
- operatie de tipul 3:
 - parcurgem vectorul
 - daca elementul a exista afisam 1 pe ecran
 - altfel afisam 0

Problema cu aceasta solutie este complexitatea ei (pentru fiecare operatie efectuam N pasi – care se datoreaza parcurgerii vectorului).

Solutie numere mici.

Daca ni se garanteaza faptul ca numerele pe care trebuie sa le salvam se afla in intervalul $[0, 100]$, putem sa alocam un vector de 101 de elemente si sa il folosim cu urmatoarea semnificatie:

$H[i] = 1$, daca elementul de pe pozitia i se afla in multime.

$H[i] = 0$, altfel

Astfel, daca trebuie sa inseram un element a in multime executam operatia $H[a] = 1$, daca trebuie sa il stergem facem operatia $H[a] = 0$ si daca trebuie sa verificam daca elementul a este in multime, verificam daca $H[a] == 1$.

Complexitatea acestei solutii este $O(1)$, intrucat facem o singura operatie la fiecare pas.

Solutie arbori.

Putem sa folosim arbori pentru a stoca aceasta multime. Daca folosim arbori binari de cautare obtinem o complexitate mai buna decat cea cu vectori (dar in cel mai rau caz arborele poate fi o lista inlantuita, obtinandu-se o complexitate de $O(N)$ per operatie).

Daca, in schimb, folosim arbori binari de cautare echilibrati (AVL) putem obtine o complexitate de $O(\log N)$ per operatie pentru ca toate operatiile folosite de un arbore binar de cautare echilibrat sunt proportionale cu inaltimea arborelui.

Complexitate arbori: $O(\log N)$.

Solutie liste inlantuite.

Pentru a obtine o solutie mai buna putem sa folosim listele simplu inlantuite si o functie de hash care nu este injectiva. Astfel incercam sa partitionam spatiul si sa salvam fiecare element intr-o lista.

De exemplu, presupunem ca trebuie sa construim un hash pentru elementele:

15 20 5 3 29 10 25 13 2

Alegem o functie de hash convenabila (care sa disperseze elementele cat mai bine).

$\text{hash}(x) = x \% 10$ (exemplu didactic, alegem ultima cifra a numarului)

Initial toate listele simplu inlantuite sunt vide.

0:	
1:	
2:	
3:	
4:	
5:	
6:	
7:	
8:	
9:	

Inseram primul element: 15, il trecem prin functia hash: $\text{hash}(15) = 15 \% 10 = 5$. Il salvam pe pozitia 5: (daca este liber il salvam direct, daca nu mai adaugam un nod).


0:	
1:	
2:	
3:	
4:	
5:	15
6:	
7:	
8:	
9:	

Continuam cu urmatorul element: 20; $\text{hash}(20) = 20 \% 10 = 0$, il salvam pe poz. 0.

0:	20
1:	
2:	
3:	
4:	
5:	15
6:	
7:	
8:	
9:	

Urmatorul element este 5; $\text{hash}(5) = 5 \% 10 = 5$, deja avem un element pe acea pozitie, deci vom adauga un nod nou.

0:	20
1:	
2:	
3:	
4:	
5:	15
6:	
7:	
8:	
9:	

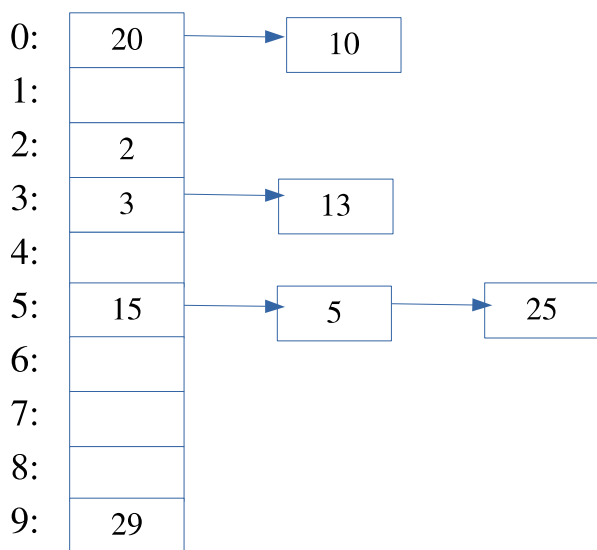


Efectuam aceleasi operatii si la final obtinem tabelul de mai jos.

Daca dorim sa cautam un element a , vom calcula mai intai valoarea $\text{hash}(a)$ si vom parcurge liniar lista respectiva. Daca elementul nu se afla in acea lista, inseamna ca acel element nu se afla in multime, altfel afisam 1.

Daca vrem sa stergem un element, il vom cauta in lista $\text{hash}(a)$ si daca exista vom aplica metoda uzuala de stergere pentru liste simplu inlantuite.

Elemente: 15 20 5 3 29 10 25 13 2



Complexitate operatii: $O(\text{lungimea celei mai mari liste})$. Daca functia hash folosita este destul de buna, numerele vor fi dispersate si complexitatea operatiilor va fi $O(1)$.

Solutie open adressing.

Alta solutie de a rezolva aceasta problema este prin adresare directa. Vom defini la fel o functie de hash, dar vom salva elementele direct in vector. Daca pozitia pe care trebuie sa salvam un element este deja ocupata, vom merge liniar pana cand gasim o noua pozitie libera. Daca ajungem la final, continuam de la inceput.

Elemente: 15 20 5 3 29 10 25 13 2

Functie: $\text{hash}(x) = x \% 10$

0	1	2	3	4	5	6	7	8	9
20	10	13	3	2	15	5	25		29

Daca dorim sa stergem un element, nu putem sa o facem direct (intrucat atunci cand am salvat o alta valoare am considerat ca pozitia era ocupata si am mers mai departe). De aceea vom marca pozitia respectiva cu X. (altfel daca am sterge elementul 3 si l-am cautat pe 2 nu l-am mai gasi).

Daca vectorul este plin (nu mai exista pozitii libere) vom dubla dimensiunea vectorului, vom alege alta functie hash si vom reconstrui vectorul.

Solutie open adressing – quadratic probing.

In loc sa mergem liniar, putem sa folosim alta functie (patraticea – $ai^2 + bi + c$) si deci sa sarim peste mai multe elemente. Practic, pentru fiecare element vom calcula $\text{hash}(a, 0)$, $\text{hash}(a, 1)$, $\text{hash}(a, 2)$... si vom salva elementul pe prima pozitie care este libera.

Elemente: 15 20 5 3 29 10 25 13 2

Functie: $\text{hash}(x, i) = (x + 3 * i^2) \% 11$

0	1	2	3	4	5	6	7	8	9	10
2		13	3	15	5	25	29		20	10

$\text{Hash}(15, 0) = (15 + 3 * 0) \% 11 = 4$

$\text{Hash}(20, 0) = (20 + 3 * 0) \% 11 = 9$

$\text{Hash}(5, 0) = (5 + 3 * 0) \% 11 = 5$

$\text{Hash}(3, 0) = (3 + 3 * 0) \% 11 = 3$

$\text{Hash}(29, 0) = (29 + 3 * 0) \% 11 = 7$

$\text{Hash}(10, 0) = (10 + 3 * 0) \% 11 = 10$

$\text{Hash}(25, 0) = (25 + 3 * 0) \% 11 = 3$ (ocupat)

$\text{Hash}(25, 1) = (25 + 3 * 1) \% 11 = 6$ (liber)

$\text{Hash}(13, 0) = (13 + 3 * 0) \% 11 = 2$

$\text{Hash}(2, 0) = (2 + 3 * 0) \% 11 = 2$ (ocupat)

$\text{Hash}(2, 1) = (2 + 3 * 1) \% 11 = 5$ (ocupat)

$\text{Hash}(2, 2) = (2 + 3 * 4) \% 11 = 3$ (ocupat)

$\text{Hash}(2, 3) = (2 + 3 * 9) \% 11 = 7$ (ocupat)

$\text{Hash}(2, 4) = (2 + 3 * 16) \% 11 = 6$ (ocupat)

$\text{Hash}(2, 5) = (2 + 3 * 25) \% 11 = 0$ (liber)

Solutie open adressing – Double hashing.

Aceasta solutie se obtine folosind doua functii hash pe care le combinam:

$$\text{hash}(x, i) = (h_1(x) + i * h_2(x)) \% N.$$

Elemente: 15 20 5 3 29 10 25 13 2

Functie: $\text{hash}(x, i) = (x \% 12 + i * (x \% 10 + 1)) \% 11$