MODULUL 9 - TABELE DE REPARTIZARE

Aşa cum s-a menţionat în primul capitol al acestui curs, structurile de date au fost dezvoltate cu scopul memorării şi manipulării ***eficiente*** a unei mulţimi dinamice de date. Acesastă manipulare eficientă depinde desigur semnificativ de scopul urmărit şi de tipurile de operaţii care se doresc a fi efectuate.

Una dintre principalele operaţii pe un set de date este căutarea sau *accesul prin cheie*. Am văzut în capitolele precedente că problema căutării poate fi rezolvată în complexitate logaritmică prin utilizarea arborilor de căutare echilibraţi.

Există însă şi structuri de date care permit accesul prin cheie în timp constant, atunci când nu sunt necesare operaţii de sortare şi păstrare ordonată a setului de date. Aceste structuri sunt *tabelele de repartizare*.

O tabelă de repartizare - *hash table* - este de fapt o generalizare a noţiunii de vector - *array*. Operaţiile de bază sunt inserţia, căutarea şi eventual ştergerea. Tabelele de repartizare pot fi utilizate cu succes pentru implementarea de dicţionare, în care căutarea este cea mai frecventă operaţie. O altă utilizare a tabelelor de repartizare este în cadrul compilatoarelor pentru implementarea tabelei de identificatori sau pentru cuvintele cheie. Vom vedea pe parcurs că, într-o tabelă de repartizate operaţiile au în medie complexitatea $O(1)$. O tabelă de repartizare se memorează într-un tablou liniar.

1 Tabele cu adresare directă

Considerăm $U = \{0, 1, \ldots, m-1\}$ universul cheilor posibile, m relativ mic şi $T[0, \ldots, m-1]$ tabloul în care se memorează elementele din tabelă. Elementul cu cheia k se plasează în T pe poziţia $T[k]$. Dacă în tabelă nu există cheia k, atunci $T[k] = NULL$.

Observaţie: o astfel de tabelă de adresare directă este potrivită, atunci când universul U al cheilor este relativ redus şi numărul de elemente memorate în tabelă este comparabil cu m.

2 Tabele de repartizare - *Hash Tables*

Pentru situaţii în care numărul de elemente stocate este mult mai mic decât universul cheilor se recomandă înlocuirea tabelei cu adresare directă printr-o tabelă de repartizare.

În cazul unei tabele de repartizare $T[0,\ldots,m-1]$ accesul la un element se face prin intermediul unei funcţii de repartizare - *hash function* - $h : U \to \{0,1,\ldots,m-1\}$, în care de obicei $|U|$ este semnificativ mai mare decât m.

Elementul cu cheia k va fi plasat în tabelă pe poziţia $T[h(k)]$. Spunem că, elementul cu cheia k este repartizat pe poziţia $h(k)$ (*hashes to slot* $h(k)$).

Exemplu de funcţie de repartizare:

$$h : U \to \{0,1,\ldots,m-1\}, h(k) = k \bmod m$$

Considerând $m = 11$ şi cheile $\{3,12,15,17\}$ atunci: $h(3) = 3$, $h(12) = 1$, $h(15) = 4$, $h(17) = 6$ (fig. 2).

*	12	*	3	15	*	17	*	*	*	*
0	1	2	3	4	5	6	7	8	9	10

Figure 1: Tabela de repartizare cu funcţia de repartizare $h(k) = k \bmod 11$ în care au fost plasate cheile 3, 12, 15 şi 17. Au fost marcate cu * poziţiile neocupate din tabelă.

Coliziunea

În cazul unei funcţii de repartizare h este posibil ca pentru două chei diferite k_1 şi k_2 să se obţină $h(k_1) = h(k_2)$, adică ambele elemente ar fi repartizate pe aceeaşi poziţie. O astfel de situaţie se numeşte *coliziune*. În exemplul de mai sus $h(24) = h(35) = 2$.

Se pune problema rezolvării coliziunilor. Ideal ar fi, evitarea completă a acestora. Din faptul că se presupune că $|U| > m$, prin tipul de repartizare descris mai sus, acest lucru nu este posibil. Totuşi, prin construirea atentă a funcţiei de repartizare, poate fi redusă semnificativ probabilitatea unei coliziuni. Vor fi discutate pe parcursul acestui curs câteva metode de construcţie a unei funcţii de repartizare, care să îndeplinească acest lucru.

Rezolvarea problemei coliziunilor se poate realiza dacă în loc de a stoca într-o poziţie a tabelei de repartizare un singur element, se păstrează o listă de elemente (*bucket*). Lista de la poziţia $h(k)$ va conţine toate elementele care au această valoarea de repartizare.

Exemplu: Considerăm $h(k) = k \bmod 11$ şi cheile $\{23,34,78,15,37,42,45\}$. $h(23) = 1$, $h(34) = 1$, $h(78) = 1$, $h(15) = 4$, $h(37) = 4$, $h(42) = 9$, $h(45) = 1$

Operaţiile uzuale:

```
TAB_INSERT(T,x)
   Insereaza x in capul listei T[h(x.cheie)]
RETURN
```

```
TAB_CAUT(T,k)
   Cauta elementul cu cheia k in lista T[h(k)]
RETURN

TAB_STERG(T,x)
   sterge x din lista T[h(x.cheie)]
RETURN
```

O tabelă de repartizare cu dimensiunea $m = 11$ care utilizează liste înlănţuite şi în care au fost inserate cheile 33, 35, 55, 46, 12, 2, 7, 10, 11 este reprezentată în figura 2.

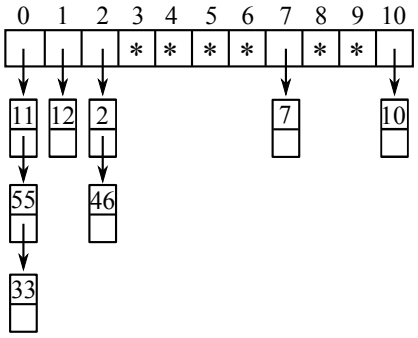

Figure 2: Tabela de repartizare cu funcţia de repartizare $h(k) = k \bmod 11$ şi utilizând liste înlănţuite în care au fost plasate succesiv cheile 33, 35, 55, 46, 12, 2, 7, 10 şi 11. Au fost marcate cu * poziţiile neocupate din tabelă.

Complexitate:

- funcţia de inserţie are evident complexitatea $O(1)$,
- la funcţia de căutare complexitatea depinde de lungimea listelor înlănţuite
- funcţia de ştergere are complexitatea $O(1)$ dacă lista e dublu înlănţuită, altfel depinde de lungimea listei.

Analiza complexităţii

Considerăm tabela de repartizare T cu m poziţii, în care se stochează n elemente, folosind liste înlănţuite pentru rezolvarea coliziunilor. Atunci numărul mediu de elemente stocate într-o listă este $\alpha = n/m$, unde α poate fi mai mic sau mai mare decât 1.

Cea mai defavorabilă situaţie este aceea când toate cele n elemente se repartizează prin h pe aceeaşi poziţie, adică sunt stocate în aceeaşi listă. În această situaţie complexitatea la căutare este $O(n)$. Pentru a evita astfel de situaţii este necesară alegerea unei funcţii de repartizare potrivită. Modul de construcţie a unor funcţii de repartizare care să permită o repartizare cât mai uniformă va fi discutată ulterior.

Repartizarea uniformă presupune ca probabilitatea ca un anumit element să fie de repartizat pe oricare dintre poziţiile din tabelă este aceeaşi.

Considerând o funcţie de repartizare care repartizează cheile uniform în tabela T şi pentru care calculul lui $h(k)$ este $O(1)$, se demonstrează faptul că funcţia de căutare e unei chei este $O(1+\alpha)$ (vezi bibliografia recomandată).

Dacă n este proproţional cu m, atunci $n = am$, deci complexitatea la căutare este $O(1 + am/m) = O(1 + a) = O(1)$.

3 Metode de repartizare

Din discuţia complexităţii rezultă că, o funcţie de repartizare bună permite o repartizare aproape uniformă în tabelă. Pentru acest lucru poziţia obţinută la repartizare ar trebui să fie independentă de orice **pattern** care ar putea fi prezent în setul de date.

În plus, funcţiile de repartizare au ca mulţime a valorilor poziţii în tabela de repartizare, deci elemente din mulţimea numerelor naturale. În cazul în care cheile sunt de exemplu şiruri de caractere - de exemplu **hash tables** utilizate pentru tabela de simboluri din faza de analiză lexicală a unui compilator - este necesară stabilirea unei metode de interpretare a acestora ca numere naturale, trebuie deci determinată o funcţie de la universul cheilor către mulţimea numerelor naturale.

În continuare sunt prezentate câteva metode pentru construcţia funcţiilor de repartizare.

3.1 Metoda diviziunii

Funcţia de repartizare $h : U \to \{0, 1, \ldots, m-1\}$, $h(k) = k \bmod m$

Exemplu: $h(k) = k \bmod 11$.

Observaţie: o alegere bună pentru m este un număr prim nu prea apropiat de o putere a lui 2. În plus acest număr trebuie ales depinzând de numărul de elemente care se estimează că vor fi introduse în tabelă, precum şi de factorul de încărcare dorit.

Exemplu: dacă $n = 3000$ şi numărul mediu de elemente per poziţie este considerat 4 atunci, $\alpha = n/m \Rightarrow m = n/\alpha = 3000/4 = 750$. Pot considera $m = 751$, care este un număr prim nu prea apropiat de o putere a lui 2.

3.2 Metoda multiplicării

În acest caz funcţia de repartizare este de tipul:

$$h(k) = \lfloor m\,(kA - \lfloor kA \rfloor) \rfloor, 0 < A < 1$$

unde $\lfloor x \rfloor$ reprezintă partea întreagă a lui x.

În cazul acestor funcţii modul de alegere a lui m nu influenţează modul de repartizare. Se demonstrează (Knuth) faptul că, o alegere bună pentru A este

$$A = \left(\sqrt{5} - 1\right)/2 \approx 0.618033$$

3.3 Repartizarea universală

Dacă repartizarea în tabelă este realizată printr-o funcţie dată, există posibilitatea ca, pentru anumite seturi de date, toate cheile să fie repartizate pe aceeaşi poziţie, ajungându-se din nou la complexitatea $O(n)$ la căutare.

Acest lucru poate fi evitat, dacă în loc să se folosească o singură funcţie de repartizare, se utilizează o mulţime H de funcţii de repartizare. La fiecare execuţie se selectează în mod aleator din H una dintre funcţiie de repartizare, care va fi apoi utilizată.

Utilizare repartizării universale are ca efect faptul că, pentru acelaşi set de date de intrare, execuţii diferite ale programului produc în general tabele diferite, astfel putându-se evita în orice situaţie cel mai defavorabil caz, eventual prin reluare a repartizării - **rehashing**.

Definiţie: O mulţime finită de funcţii de repartizare H, care repartizează cheile dintr-un univers U într-o tabelă $T[0, \ldots, m-1]$ se numeşte universală, dacă pentru orice două chei k şi l, $k \neq l$, numărul de funcţii din H pentru care $h(k) = h(l)$ este cel mult $|H|/m$.

Acest lucru asigură faptul că, pentru o funcţie aleasă în mod aleator din H, probabilitatea unei coliziuni între k şi l este $1/m$. (Evident, probabilitatea alegerii oricărei funcţii din H este $1/|H|$. Probabilitatea alegerii unei funcţii care repartizează k şi l pe aceeaşi poziţie este($|H|/m) * 1/|H| = 1/m$).

Construcţie a unei clase universale de funcţii de repartizare

Se alege un număr prim p relativ mare, $p >> m$ Pentru fiecare $a \in \{1, \ldots, p-1\}$ şi $b \in \{0, 1, \ldots, p-1\}$ se defineşte funcţia de repartizare:

$$h_{ab}(k) = ((ak + b) \bmod p) \bmod m)$$

Mulţimea de astfel de funcţii de repartizare este

$$H_{pm} = \{h_{ab} | a \in \{1, \ldots, p-1\} \text{ şi } b \in \{0, 1, \ldots, p-1\}\}$$

Se poate demonstra că mulţimea astfel definită este universală (vezi bibliografia recomandată).

3.4 Adresare deschisă

În cazul adresării deschise, fiecare poziţie a tabelei de repartizare T conţine un singur element. Pentru rezolvarea coliziunilor la inserţie nu se utilizează liste înlănţuite, ci se testează diferite poziţii, obţinute pe baza funcţiei de repartizare, până când se determină o poziţie liberă, pe care poate fi inserat elementul dorit.

La căutare de asemenea se testează diferite poziţii până când găseşte elementul căutat sau se ajunge la o poziţie neocupată.

Faptul că nu se utilizează liste înlănţuite, ci fiecare poziţie conţine cel mult un element, are ca urmare posibilitatea umplerii tabelei. În schimb se evită utilizarea pointerilor, iar memoria, care altfel ar fi fost utilizată pentru listele înlănţuite, poate fi utilizată pentru o tabelă de dimensiune mai mare.

Modul de testare a poziţiilor în tabelă în cazul adresării deschise nu este liniar ci depinde de cheia inserată şi de numărul de testări efectuate până la momentul curent.

O funcţie de repartizare pentru repartizare deschisă este de forma:

$$h : U \to \{0, 1, \ldots, m-1\} \times \{0, 1, \ldots, m-1\}$$

$h(k, t) =$ poziţia de repartizare a cheii k după t teste.

Inserţia unei chei într-o tabelă $T[0, \ldots, m-1]$

```
TAD_INSERT(T,k)
    i = 0
    repeta
        j = h(k, i)
        daca T[j] = NULL atunci
            T[j] = k
```

```
            RETURN j
        sfarsit daca
        i = i + 1
    pana cand i = m
    scrie("tabela este plina")
RETURN -1
```

Algoritmul de căutare testează pentru o anumită cheie k aceeaşi secvenţă de poziţii ca şi algoritmul de inserţie al cheii k.

```
TAD_CAUt(T,k)
    i = 0
    repeta
        j = h(k,i)
        daca T[j] = k atunci
            RETURN j
        i = i + 1
    pana cand i = m sau T[j] = NULL
RETURN -1
```

Observaţie: Operaţia de ştergere este problematică în cadrul adresării deschise. Dacă o cheie k este ştearsă prin marcarea poziţiei cu $NULL$ atunci o cheie p, inserată după k, dar care la inserţie/ştergere presupune testarea poziţiei pe care s-a aflat k, nu va mai fi găsită prin algoritmul de mai sus $\Rightarrow$ este nevoie de un algoritm de complexitate mai mare la căutare.

O soluţie a acestei probleme este marcarea poziţiilor şterse cu un marcaj special de poziţie ştearsă, dar tot presupune complexitate crescută pentru număr mare de poziţii şterse.

Atunci când este nevoie de operaţia de ştergere, se recomandă folosirea de tabele de repartizare cu liste înlănţuite.

În ceea priveşte alegerea unei funcţii de repartizare pentru repartizarea cu adresare deschisă, aceasta poate fi realizată în diferite moduri. Vor fi descrise mai jos trei metode de construcţie a unei astfel de funcţii.

(1) Funcţii de repartizare cu testare liniară

Se consideră o funcţie de repartizare obişnuită: $h_1 : U \to \{0, 1, \ldots, m-1\}$. Se defineşte: $h : U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$

$$h(k, i) = (h_1(k) + i) \bmod m$$

Exemplu: considerăm $m = 11$, $h1(k) = k \bmod 11$ şi

$$h(k, i) = ((k \bmod 11) + i) \bmod 11.$$

Atunci cheile 22, 33, 45, 59, 67, 13, 71 vor fi repartizate după cum urmează:
$h(22,0) = (0+0) \bmod 11 = 0$
$h(33,0) = (0+0) \bmod 11 = 0$, este deja ocupat, deci testarea continuă
$h(33,1) = (0+1) \bmod 11 = 1$
$h(45,0) = (1+0) \bmod 11 = 1$, este deja ocupat, deci testarea continuă
$h(45,1) = (1+1) \bmod 11 = 2$
$h(59,0) = (4+0) \bmod 11 = 4$
$h(67,0) = (1+0) \bmod 11 = 1$, este deja ocupat, deci testarea continuă
$h(67,1) = (1+1) \bmod 11 = 2$, este deja ocupat, deci testarea continuă
$h(67,2) = (1+2) \bmod 11 = 3$,
$h(13,0) = (2+0) \bmod 11 = 2$, este deja ocupat, deci testarea continuă
$h(13,1) = (2+1) \bmod 11 = 3$, este deja ocupat, deci testarea continuă
$h(13,2) = (2+2) \bmod 11 = 4$
$h(71,0) = (5+0) \bmod 11 = 5$

Observaţie: Dezavantajul testării liniare este acela că, pe măsură ce se adaugă elemente, cresc secvenţele de poziţii succesive ocupate - **primary clustering**, ceea ce duce la creşterea complexităţii de inserţie/căutare.

Un mod de îmbunătăţire a acestei situaţii este utilizarea funcţiilor de repartizare cu testare pătratică descrise în continuare.

(2) Funcţiile de repartizare cu testare pătratică

Se consideră o funcţie de repartizare obişnuită: $h_1 : U \to \{0, 1, \ldots, m-1\}$. Se defineşte: $h : U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$

$$h(k, i) = (h_1(k) + c_1 i + c_2 i^2) \bmod m$$

h_1, funcţie de repartizare obişnuită, c_1 şi c_2 constante întregi pozitive auxiliare.

Prin acest mod de repartizare, se îmbunătăţeşte uşor performanţa. Problema este că, pentru două elemente k_1 şi k_2 distincte, pentru care $h(k_1, 0) = h(k_2, 0)$, oricare ar fi i, $h(k_1, i) = h(k_2, i)$. Acest lucru are ca urmare faptul că, dacă se inserează numeroase elemente care au aceeaşi repartizare iniţială, creşte complexitatea inserţiei. Rezultă o formă oarecum atenuată de *clustering*, numită *secondary clustering* .

Una dintre cele mai bune metode de repartizare deschisă se realizează cu ajutorul **dublei repartizări**.

(3) Funcţii de repartiţie cu dublă repartizare

Se consideră două funcţii de repartizare obişnuite distincte $h_1(k)$ şi $h_2(k)$. Se construieşte funcţia cu dublă repartizare:

$$h(k, i) = (h_1(k) + i h_2(k)) \bmod m$$

Observaţie: Se recomandă ca funcţia $h_2(k)$ să producă valori care sunt prime faţă de m. Acest lucru se poate obţine dacă

- m putere a lui 2 şi $h_2(k)$ produce întotdeauna un număr impar
- m prim şi $h_2(k)$ produce numere naturale din $\{0, 1, \ldots, m-1\}$, de exemplu considerăm funcţiile de repartiţie

$$h_1(k) = k \bmod m$$

$$h_2(k) = 1 + (k \bmod m_1)$$

cu m prim şi m_1 astfel încât m_1 ceva mai mic decât m - de exemplu $m_1 = m - 1$.

Exemplu: $h_1(k) = k \bmod 11$, $h_2(k) = 1 + (k \bmod 10)$. Atunci cheile 22, 33, 45, 59, 67, 13, 71 vor fi repartizate după cum urmează:
$h(22, 0) = (22 \bmod 11) \bmod 11 = 0$
$h(33, 0) = (33 \bmod 11) \bmod 11 = 0$, este deja ocupat, deci continuă testarea
$h(33, 1) = ((33 \bmod 11) + (1 + 33 \bmod 10)) \bmod 11 = (0 + 4) \bmod 11 = 4$
$h(45, 0) = (45 \bmod 11) \bmod 11 = 1$
$h(59, 0) = (59 \bmod 11) \bmod 11 = 4$, este deja ocupat, deci continuă testarea
$h(59, 1) = (59 \bmod 11 + 1 + 59 \bmod 10) \bmod 11 = (4 + 10) \bmod 11 = 3$ $h(67, 0) = (67 \bmod 11) \bmod 11 = 1$, este deja ocupat, deci continuă testarea
$h(67, 1) = ((67 \bmod 11) + (1 + 67 \bmod 10)) \bmod 11 = 9$
$h(13, 0) = (13 \bmod 11) \bmod 11 = 2$
$h(71, 0) = (71 \bmod 11) \bmod 11 = 6$

Se observă că sunt necesare mai puţine încercări decât în cazul repartizării cu testare liniară.

Complexitate: considerând factorul de încărcare al tabelei de repartizare $\alpha = n/m < 1$, unde $n =$ numărul de poziţii ocupate şi $m =$ dimensiunea tabelei atunci:

- numărul de testări în cazul unei căutări fără succes este în cel mai defavorabil caz $1/(1-\alpha)$ (presupunând repartizare uniformă) $\Rightarrow$ căutarea fără succes are complexitate $O(1)$ pentru α constant;
- numărul de testări în cazul unei căutări cu succes este în cel mai defavorabil caz $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ (presupunând repartizare uniformă) $\Rightarrow$ căutarea cu succes are complexitate $O(1)$ pentru α constant

Aplicaţii:

- Compilatoare - pentru păstrarea tabelei de identificatori se recomandă utilizarea unei tabele de repartizare, deoarece este necesar acces rapid la informaţie.
- Criptografie - pentru securitatea parolelor se utilizează *hashing* criptografic. Acest tip de repartizare este mai dificil de realizat şi presupune îndeplinirea anumitor criterii de securitate, care depăşesc însă limitele acestui curs.

4 Repartizarea cheilor de tip şir de caractere

O funcţie de repartizare ideală va produce pentru orice două chei inserate poziţii de inserţie diferite. Există situaţii, atunci când sunt cunoscute toate cheile posibile (de exemplu în cazul unui dicţionar) să fie construite funcţii de repatizare ideale în care să se evite orice coliziune. Acest lucru se numeşte *perfect hashing* şi există documentaţie în internet. În cazul general acest lucru este practic imposibil, astfel încât se recomandă construirea de funcţii de repartizare care să producă o repartizare uniformă a cheilor în tabelă, rezultând o probabilitate mică de coliziune. Pentru chei numere întregi au fost prezentate câteva criterii de construcţie a acestor funcţii. În plus, o funcţie de repartizare bună trebuie să utilizeze toate părţile unei chei pentru generarea valorii de repartizare. De exemplu, dacă se utilizează *string*-uri pentru chei, nu se vor considera doar primele 5 caractere pentru generarea valorii de repartizare sau dacă se folosesc chei numere întregi cu mai multe cifre, nu se vor utiliza doar anumite cifre ale acestora.

Se spune despre o funcţie de repartizare că obţine o avalanşă - *achieves avalanche* - dacă şi doar prin modificarea unui singur bit al unei chei se obţine o valoare de repartizare complet diferită de cea anterioară. Acest efect ajută la o distribuire uniformă a cheilor în tabelă şi la o reducere a probabilităţii coliziunilor. Conceptul de *avalanche* este derivat de la *hashing*-ul criptografic.

Este extrem de dificil de elaborat o funcţie de repartizare bună. De aceea se recomandă utilizarea unor funcţii deja existente, despre care se ştie că produc rezultate bune. În continuare sunt prezentate câteva funcţii de repartizare cunoscute împreună cu anumite proprietăţi ale lor. Sunt considerate chei de tip *string*, dar pot fi utilizate şi pentru chei de tip întreg, în loc de câte un caracter considerând câte un byte al cheii.