Tabele de dispersie

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. Această 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*. În cazul mulţimilor sortate, căutarea binară permite găsirea unuei anumite chei în complexitate logaritmică. Vom vedea în capitolele ce tratează arborii binari de căutare, cum poate fi rezolvată în complexitate logaritmică problema căutării în cazul mulţimilor dinamice de date, deci a mulţimilor care se modifică prin inserţii şi ştergeri.

În capitolul de faţă sunt prezentate tabele de dispersie - *hash tables, hash maps* - structuri de date care permit accesul prin cheie în timp constant. O astfel de structură de date este eficientă, atunci când nu sunt necesare operaţii de sortare şi păstrare ordonată a setului de date.

O tabelă de dispersie este de fapt o generalizare a noţiunii de vector - *array*. Operaţiile de bază sunt inserţia, căutarea şi eventual ştergerea. Tabelele de dispersie 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 dispersie operaţiile au în medie complexitatea $O(1)$. O tabelă de dispersie se memorează într-un *array*.

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 $T[k] = NIL$. O astfel de tabelă este reprezentată grafic în figura 1.

Algoritmii pentru operaţiile se căutare, inserţie şii ştergere sunt extrem de simpli.

Algoritm: AD-CAUT

Intrare: tabela T şi cheia k
RETURN $T[k]$

Algoritm: AD-INSEREAZA

Intrare: tabela T şi elementul x
$T[x.cheie] \leftarrow x$

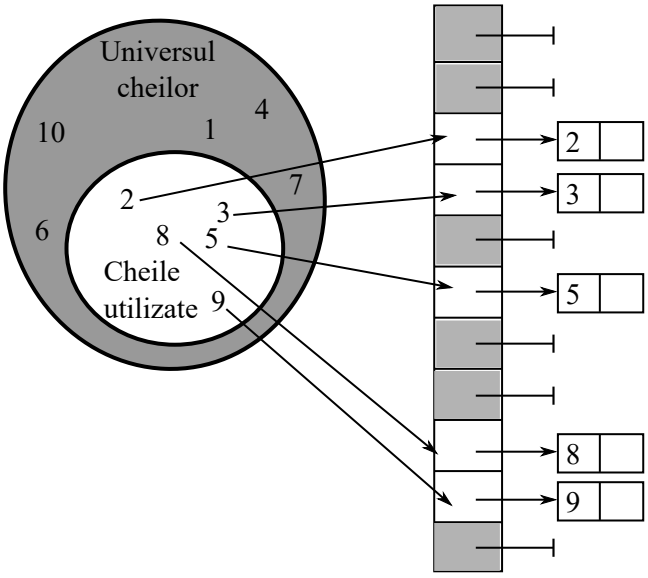

Figura 1: Tabelă cu adresare directă.

Algoritm: AD-STERGE

Intrare: tabela T şi elementul x

$T[x.cheie] \leftarrow NIL$

Complexitate: evident complexitatea este $O(1)$ pentru toate cele trei operaţii.

Observaţii:

- se presupune faptul că fiecare element x are o cheie unică, diferită de cheile tuturor celorlalte elemente din tabelă. Practic un element este identificat în mod unic prin cheie.
- o tabelă cu adresare directă este potrivită, atunci când universul U al cheilor este relativ redus şi numărul de elemente memorate în tabelă este comparabil cu $|U| = m$.
- de obicei elementele dintr-o tabelă cu adresare directă sau dintr-o tabelă de dispersie au un câmp pentru cheie, după care se face căutarea, şi un câmp pentru informaţia de interes.

2 Tabele de dispersie - *Hash Tables*

Pentru situaţii în care numărul de elemente stocate într-o tabelă este mult mai mic decât universul cheilor, de exemplu atunci când orice număr natural poate fi cheie, tabelele cu adresare directă devin practic inutilizabile, datorită cantităţii de memorie necesare, care poate fi astfel nelimitată. În această situaţie se înlocuieşte tabela cu adresare directă printr-o tabelă de dispersie.

Pentru memorarea unei tabele de dispersie se utilizează de asemenea un *array* T de dimensiune m: $T[0, \ldots, m-1]$. Accesul la un element se face prin intermediul unei funcţii de dispersie (repartizare) - *hash function* - $h : U \to \{0, 1, \ldots, dim - 1\}$, $U >> m$.

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* $h(k)$). În figura 2 este reprezentată grafic o tabelă de dispersie.

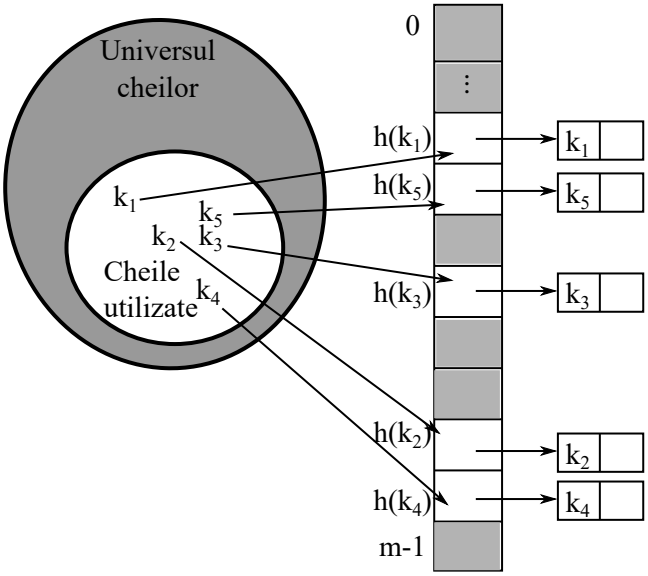

Figura 2: Tabelă de dispersie.

2.1 Problema coliziunilor

2.1.1 Exemplu de funcţie de repartizare:

O funcţie foarte simplă pentru repartizarea nunerelor naturale într-o tabelă de dispersie este dată prin funcţia *modulo* (mod), care reprezintă restul împărţirii la un număr natural.

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

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

Coliziunea

În cazul unei funcţii de dispersie 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 $h(28) = h(17) = 6$ (fig. 4).

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

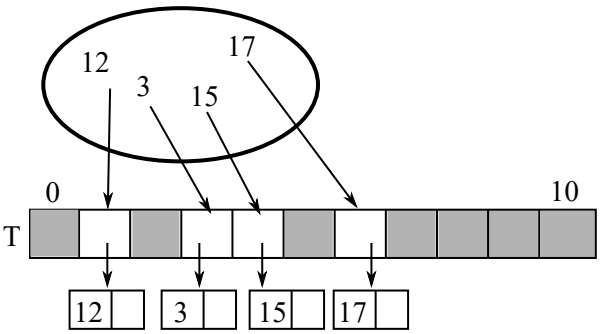

Figura 3: Tabela de repartizare cu funcţia de dispersie $h(k) = k \bmod 11$ în care au fost plasate cheile 3, 12, 15 şi 17.

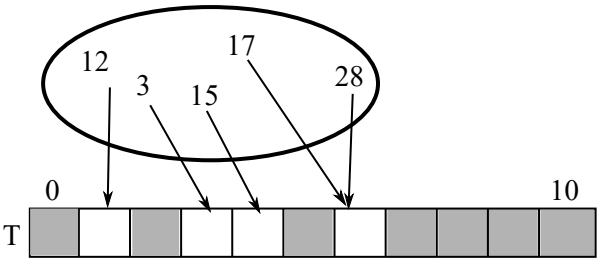

Figura 4: Coliziune între cheile 28 şi 17.

lucru nu este posibil. Totuşi, prin construirea atentă a funcţiei de dispersie, poate fi redusă semnificativ probabilitatea unei coliziuni. Vor fi discutate pe parcursul acestui curs câteva metode de construcţie a unei funcţii de dispersie, care să îndeplinească acest lucru.

În practică rezolvarea coliziunilor pentru tabele de dispersie poate fi făcută de exemplu cu ajutorul listelor înlănţuite. Astfel, în loc de a stoca într-o poziţie a tabelei de repartizare un singur element, se păstrează o listă de elemente (ideea de *bucket*). Lista de la poziţia k va conţine toate elementele a cărei cheie este repartizată pe poziţie.

Exemplu: Se consideră o tabelă de dispersie T cu dimensiunea $m = 11$ care utilizează liste înlănţuite şi în care se introduc elementele cu cheile 33, 35, 55, 46, 12, 2, 7, 10, 11. Rezulttatul este este reprezentată în figura 5. Inserţia în liste s-a realizat la începutul listelor.

O altă metodă de rezolvare a coliziunilor este *adresare deschisă*, metodă prezentată la sfîrşitul capitolului.

2.2 Operaţiile uzuale

În continuare sunt prezentati algoritmii pentru operaţiile de căutare, inserare şi ştergere a unui element dintr-o tabelă de disprersie ce utilizează liste înlănţuite pentru rezolvarea coliziunilor. Se consideră că fiecare element x din tabelă este o structură cu

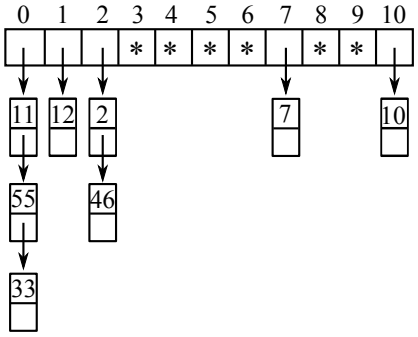

Figura 5: Tabela de dispersie de dimensiune $m = 11$ în care au fost plasate succesiv cheile 33, 35, 55, 46, 12, 2, 7, 10 şi 11. Au fost marcate cu * poziţiile neocupate din tabelă.

două câmpuri $x.cheie$ pentru cheie şi $x.val$ pentru informaţia de interes. În algoritmi se consideră notaţia TDL pentru Tabelă de Dispersie cu Liste înlănţuite.

Algoritm: TDL-INSEREAZA

Intrare: tabela T de dimensiune m cu funcţia de dispersie h şi elementul x
Insereaza x in capul listei $T[h(x.cheie)]$

Algoritm: TDL-CAUTA

Intrare: tabela T de dimensiune m cu funcţia de dispersie h şi cheia k
Iesire: elementul cu cheia x sau NIL în caz de eşec
Cauta elementul x cu cheia k in lista $T[h(k)]$
return x

Algoritm: TDL-STERGE

Intrare: tabela T de dimensiune m cu funcţia de dispersie h şi elementul x
Sterge x din lista $T[h(x.cheie)]$

Complexitate:

- funcţia de inserţie are evident complexitatea $O(1)$,
- la funcţia de căutare complexitatea depinde de lungimea listelor înlănţuite
- funcţia de ştergere, care are ca parametru un nod x deja identificat în lista înlănţuită, are complexitatea $O(1)$ dacă lista e dublu înlănţuită, altfel depinde de lungimea listei. Dacă întâi trebuie căuttat elementul cu cheia k, apoi şters, se adaugă complexitatea căutării.

2.3 Analiza complexităţii

Considerăm tabela de repartizare T cu m poziţii, în care rezolvarea coliziunilor se realizează prin înlănţuire. Presupunem că în T se stochează n elemente. Atunci numărul mediu de elemente stocate într-o listă este $\alpha = n/m$, unde α poate fi mai mic sau mai mare decât 1. α se numeşte *factor de încărcare* al tabelei.

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.

Notăm cu n_j lungimea listei de pe poziţia $T[j]$, $j = \overline{0, m-1}$. Observăm că $n_0 + n_1 + \ldots + n_{m-1} = n$, iar valoarea medie sau valoarea **estimată** (vezi Cormen) pentru lungimea oricărei liste n_j este dată prin: $E[n_j] = n/m = \alpha$.

Considerând o funcţie de dispersie 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 $\Theta(1+\alpha)$ cu ajutorul următoarelor două teoreme.

Teorema 2.1 *Într-o tabelă de dispersie care tratează coliziunile prin înlănţuire, o căutare fără succes are complexitatea medie de timp $\Theta(1+\alpha)$. (citat Cormen)*

Demonstraţie: *Presupunem că se caută în T cheia k, care nu există. Datorită faptului că s-a presupus o repartizare uniformă cu h, probabilitatea repartizării cheii k pe oricare dintre poziţii este aceeaşi. Timpul necesar pentru a constata că această cheie nu este în T este timpul necesar de parcurgere a listei $T[h(k)]$. În medie acest timp depinde de lungimea medie a listei $h(k)$, deci de $E[n_h(k)] = \alpha$. Dat fiind că se ia ţine cont şi timpul necesar pentru calculul valorii $h(k)$ se obţine o complexitate de $\Theta(1+\alpha)$.*

Teorema 2.2 *Într-o tabelă de dispersie care tratează coliziunile prin înlănţuire, o căutare cu succes are complexitatea medie de timp $\Theta(1+\alpha)$. (citat Cormen)*

Demonstraţie *Situaţia căutării cu succes diferă de cea a căutării fără scces prin faptul că, timpul de căutare al elementului x cu cheia k din tabelă depinde de numărul de elemente aflate în lista $T[h(k)]$ inaintea lui x. Modul de calcul al acestui timp mediu presupune ceva cunoştinţe de probabilităţi şi depăşeşte cadrul acestui curs. O desriere detaliată poate fi găsită în bibliografie (Cormen)*

Dacă numărul de elemente stocate în tabelă, n, este proproţional cu dimensiunea m a tabelei, adică $n = am$, atunci complexitatea la căutare devine $\Theta(1 + am/m) = \Theta(1 + a) = \Theta(1)$. Cu alte cuvinte, dacă funcţia de dispersie este aleasă în mod potrivit, astfel încât să ofere o repartizare uniformă şi dacă dimensiunea tabelei este aleasă proporţional cu numărul estimat de elemente ce vor fi introduse în tabelă, timpul de căutare a unei chei devine în medie constant.

3 Metode de repartizare

Din discuţia complexităţii rezultă că, o funcţie de dispersie 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 dispersie au ca mulţime a valorilor poziţii în tabela de dispersie, deci elemente din mulţimea numerelor naturale. În cazul în care cheile sunt de exemplu şiruri de caractere - situaţie frecventă de altfel - 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. La finalul capitolului vor fi prezentate câteva funcţii de dispersie pentru *string*-uri.

În continuare sunt prezentate câteva metode pentru construcţia funcţiilor de dispersie pentru numere naturale.

3.1 Metoda diviziunii

Funcţia de dispersie în acest caz este dată prin:

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

Alegerea dimensiunii tabelei. O alegere bună pentru dimensiunea tabelei m este un număr prim nu prea apropiat de o putere a lui 2.
Justificare. Dacă s-ar alege $m = 2^p$ atunci de fapt împărţirea la m ar reprezenta o *shift*-are a biţilor cheii, ceea ce ar creea o dependenţă de *patter*-uri în structura cheilor, lucru ce trebuie evitat.

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, $\alpha = n/m \Rightarrow m = n/\alpha = 3000/4 = 750$. Se poate 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 dispersie este de tipul:

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

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 Dispersie 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 $\Theta(n)$ la căutare.

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

Utilizarea dispersiei 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: (Cormen) O mulţime finită de funcţii de dispersie 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 aleatoriu din H, probabilitatea unei coliziuni între k şi l este cel mult $1/m$. (Evident, probabilitatea alegerii oricărei funcţii din H este $1/|H|$. Probabilitatea alegerii unei funcţii care repartizează k şi l pe aceeaşi poziţie este cel mult ($|H|/m) * 1/|H| = 1/m$).

3.3.1 Construcţie a unei clase universale de funcţii de dispersie

Se alege un număr prim p suficient de mare, astfel încât să fie mai mare decât orice cheie posibilă. Evident $p >> m$. Pentru fiecare $a \in \{1, \ldots, p-1\}$ şi $b \in \{0, 1, \ldots, p-1\}$ se defineşte funcţia de dispersie:

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

Mulţimea de astfel de funcţii de dispersie este

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

cu $|H_{pm}| = p(p-1)$.

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 dispersie 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 dispersie, 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 pointerii, iar memoria, care altfel ar fi fost necesară 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.

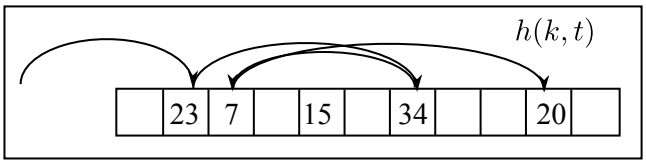

Figura 6: Căutarea elementului cu cheia 20. Acesta este găsit după 3 teste.

Forma generală a unei funcţie de dispersie în cazul adresării deschisă este de forma:

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

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

În mod evident, pentru a permite testarea tuturor poziţiilor posibile şi a nu reveni periodic la aceleaşi poziţii, testate anterior, funcţia $f(k, t)$ trebuie să producă pentru valorile lui $t \in \{0, \ldots, m-1\}$ o permutare a mulţimii poziţiilor $\{0, \ldots, m-1\}$. Căutarea unei cheil într-o astfel de tabelă este ilustrată în figura 6.

În continuare sunt prezentaţi algoritmii pentru inserarea şi căutare într-o tabelă cu adresare deschisă.

Algoritm: TAD-INSEREAZA

Intrare: tabela T cum m elemente, elementul cu x cheia k
$t \leftarrow 0$//numarul de teste deja efectuate
repeta
  $j \leftarrow h(k, t)$//determina pozitia pentru testare
  daca $T[j] = NIL$ **atunci**
    $T[j] \leftarrow x$
    return j
  sfarsit_daca
  $t \leftarrow t + 1$ //incrementarea numarului de teste
pana_cand $t = m$;
return -1 //terminare cu esec

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

Algoritm: TAD-CAUTA

Intrare: tabela T cum m elemente, elementul cu x cheia k
$t \leftarrow 0$
repeta
  $j \leftarrow h(k, t)$
  daca $T[j].cheie = k$ **atunci**
    return j
  sfarsit_daca
  $t \leftarrow t + 1$
pana_cand $t \neq m$ *şi* $T[j] \neq NIL$;
return -1

Operaţia de ştergere este problematică în cadrul adresării deschise. Dacă o cheie k este şterasă prin marcarea poziţiei cu NIL 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.

O soluţie a acestei probleme este marcarea poziţiilor şterse cu un marcaj special de poziţie ştearsă, acest lucru presupune modificarea algoritmilor de căutare şi inserare şi în plus duce la o complexitate crescută a operaţiei de căutare pentru număr mare de poziţii şterse. Pe măsură ce creşte numărul de poziţii marcate ca şterse, creşte timpul de căutare şi scade performanţa. Acest lucru se numeşte *contaminare*. Singura posibilitate de rezolvare a acestei probleme este prin *rehashing*.

În ceea priveşte alegerea unei funcţii de dispersie pentru adresarea deschisă, aceasta poate fi realizată în diferite moduri. Vor fi descrise mai jos trei metode de construcţie a unei astfel de funcţii. Toate cele trei metode garantează faptul că secvenţa $\{h(k,0), h(k,1), \ldots, h(k,m-1)\}$ este o permutare a mulţimii $\{0,1,\ldots,m-1\}$ pentru orice cheie k.

3.4.1 Funcţii de dispersie cu testare liniară

Se consideră o funcţie de dispersie auxiliară, de tipul celor descrise la începutul capitolului: $h_1 : U \to \{0,1,\ldots,m-1\}$. Se defineşte: $h : U \times \{0,1,\ldots,m-1\} \to \{0,1,\ldots,m-1\}$

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

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

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

Atunci cheile 22, 33, 45, 59, 67, 13 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
$h(13,2) = (2+2) \bmod 11 = 4$ este deja ocupat, deci testarea continuă
$h(71,0) = (2+3) \bmod 11 = 5$

Inserearea cheilor din acest exemplu este ilustrată în figura 7.

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.

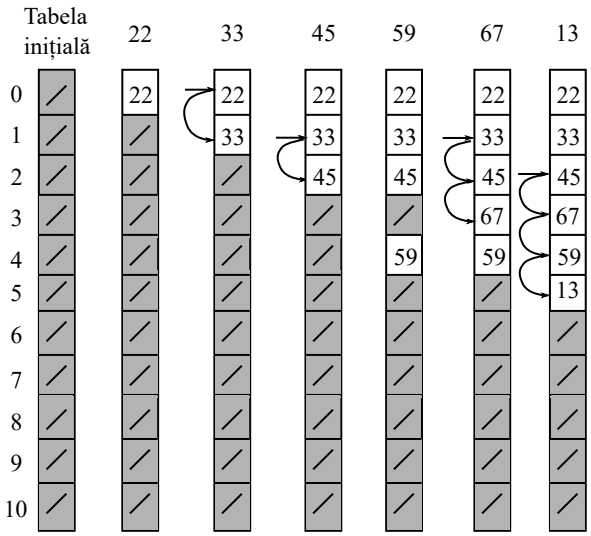

Figura 7: Inserarea cheilor 22, 33, 45, 59, 67, 13 într-o tabelă de dispersie de dimensiune $m = 11$ folosind testarea liniară. Poziţiile testate pentru fiecare element sunt marcate cu săgeţi.

3.4.2 Funcţiile de dispersie cu testare pătratică

Se consideră o funcţie de dispersie auxiliară de tipul celor descrise la începutul capitolului: $h_1 : U \to \{0, 1, \ldots, m-1\}$. Se defineşte: $h : U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$

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

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 t, $h(k_1, t) = h(k_2, t)$. 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*. (citat)

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

3.4.3 Adresarea deschisă cu dublă repartizare

Se consideră două funcţii de dispersie auxiliare distincte $h_1(k)$ şi $h_2(k)$. Se construieşte funcţiade dispersie cu dublă repartizare:

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

Observaţii:

- poziţia iniţială testată este $h(k, 0) = h_1(k)$, deci nu depinde de h_2, pentru orice cheie k
- poziţia $h(k, t)$ este la distanţa $h_2(k)$, faţă de poziţia $h(k, t-1)$, pentru orice $t > 0$

- pentru ca secvenţa de poziţii $\{h(k,0), h(k,1), \ldots, h(k,m-1)\}$ să fie o permutare a poziţiilor $\{0,1,\ldots,m-1\}$ trebuie ca funcţia $h_2(k)$ să producă valori care sunt prime faţă de m. Dacă există o cheie k, pentru care $h_2(k)$ şi m au un divizor comun $1 < d < m$, atunci cel mult după d teste se repetă din nou testarea poziţiei iniţiale. Adică vor rămâne în tabelă poziţii netestate!!

  Satisfacerea condiţiei discutate mai sus, 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: Considerând funcţiile de dispersie $h_1(k) = k \bmod 11$ c si $h_2(k) = 1 + (k \bmod 10)$, 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
$h(67,1) = ((67 \bmod 11) + (1 + 67 \bmod 10)) \bmod 11 = 9$
$h(13,0) = (13 \bmod 11) \bmod 11 = 2$
$h(71,0) = (71 \bmod 11) \bmod 11 = 6$

Procesul de inserţie a acestor chei este ilustrat în figura 8. Se observă că sunt necesare semnificativ mai puţine încercări decât în cazul repartizării cu testare liniară.

3.4.4 Analiza complexitatăţii

Considerăm o tabelă de dispersie cu factorul de încărcare $\alpha = n/m < 1$, unde $n =$ numărul de poziţii ocupate şi $m =$ dimensiunea tabelei atunci. De asemenea presupunem că funcţia de dispersie asigură repartizarea uniformă. Acest lucru presupune că probabilitatea ca secvenţa de poziţii testate pentru o anumită cheie k să fie oricare dintre cele $m!$ permutări ale mulţimii $\{0,1,\ldots,m-1\}$ este aceeaşi. Atunci complexitatea căutării unei chei este determinată pe baza următoarelor două teoreme.

Teorema 3.1 *Pentru o tabelă de dispersie cu adresare deschisă, cu fatorul de încărcare $\alpha < 1$, numărul mediu teste necesare pentru căutarea fără succest este cel mult $1/(1-\alpha)$, presupunând o distribuţie uniformă.*

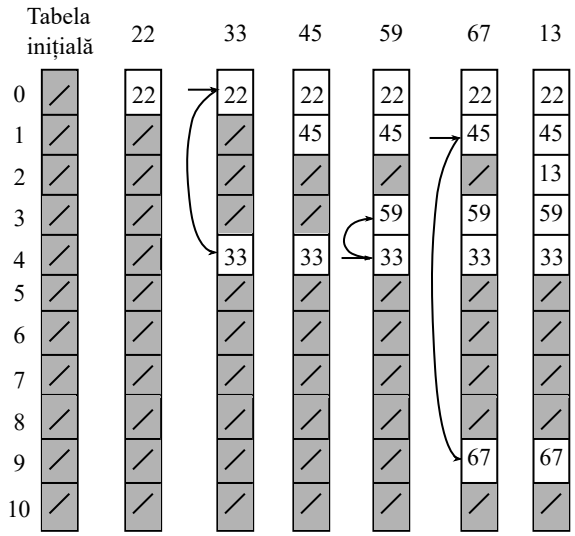

Figura 8: Inserarea cheilor 22, 33, 45, 59, 67, 13 într-o tabelă de dispersie de dimensiune $m = 11$ folosind testarea dubla repartizare. Poziţiile testate pentru fiecare element sunt marcate cu săgeţi.

Teorema 3.2 *Pentru o tabelă de dispersie cu adresare deschisă, cu fatorul de încărcare* $\alpha < 1$*, numărul mediu teste necesare pentru căutarea cu succest este cel mult* $\frac{1}{\alpha} \log \frac{1}{1-\alpha}$*, presupunând o repartizare uniformă.*

Ambele teoreme sunt demonstrate cu ajutorul relaţiilor probabilistice în (Cormen).

Observaţii

- Testarea liniară şi testarea pătratică nu garantează distribuţie uniformă datorită fenomenului de *clustering*.
- Dubla repratizare se apropie cel mai mult de o distribuţie uniformă.
- În practică, dacă factorul de încărcare devine foarte mare eficienţa inserţiei, care presupune căutara unei poziţii neocupate, deci de fapt căutarea fără succes a cheii elementului ce se inserează, scade semnificativ. Astfel încât, atunci când α se apropie de 80% se recomandă redimensionarea tabelei şi *rehashing*. De exemplu, în limbajul **Python**, care implementează dicţionarele folosind adresarea deschisă, se porneşte cu o tabelă de dimensiune 8 şi atunci când s-a umplut 75% din tabelă se redimensionează. ($https://adamgold.github.io/posts/python-hash-tables-under-the-hood/\#dictionary-resize$)

3.5 *Perfect hashing*

O funcţie de distribuţie 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 dispersie ideale în care să se evite

orice coliziune. Acest lucru se numeşte *perfect hashing*. În (Cormen) este propus un algoritm de dispersie perfect, care are la bază tot ideea de *bucket*-uri similar cu rezolvarea coliziunilor prin înlănţuire, doar că, în loc de liste înlănţuite se utilizează tabele de dispersie secundare.

În cazul general acest lucru este practic imposibil, astfel încât se recomandă construirea de funcţii de dispersie 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 dispersie bună trebuie să utilizeze toate părţile unei chei pentru generarea valorii de dispersie. De exemplu, dacă se utilizează *string*-uri pentru chei, nu se vor considera doar primele 5 caractere pentru generarea valorii de dispersie sau dacă se folosesc chei numere întregi cu mai multe cifre, nu se vor utiliza doar anumite cifre ale acestora.

4 Tratarea cheilor de tip şir de caractere

În partea teoretică prezentată până acum s-a considerat doar distribuirea de chei numere naturale. În practică există chei de diferite tipuri, fecvent caractere sau şiruri de caractere. Acestor chei trebuie mai întâi să li se asocieze un număr natural.

În primul rând putem considera pentru fiecare caracter ASCII codul său numeric. Problema devine, cum combinăm codurile unei secvenţe de caractere într-un număr natral, astfel încât să nu depăşim valoarea maximă adimisă şi să evităm coliziunile care se produc, dacă pentru două şiruri de caractere se generează aceeaşi valoare numerică.

4.1 Funcţii elementare de transformare a unui *string* într-un număr natural

Cele mai simple metode de a transforma o cheie de tip *string* într-o cheie numerică sunt *hashing*-ul aditiv şi XOR -*hash*, descrise mai jos.

***Hashing* - aditiv** - presupune adunarea codurilor numerice corespunzătoare tutror caracterelor din şir. Acest tip de calcul al unei chei numerice este extrem de prost, deoarece generează coloziuni pentru toate permutările unui şir de caractere. În practică este deci inutilizabil.

XOR -*hash* - presupune aplicarea iterativă a operaţiei de XOR asupra caracterelor din şir. Practic se porneşte cu valoarea $h\_val = 0$, care pentru fiecare caracter $sir[i]$ din şir se modifică prin instrucţiunea

$$h\_val \leftarrow h\_val \text{ XOR } sir[i]$$

Nici această metodă nu dă rezultate bune în ceea ce priveşte coliziunile, dar operaţia XOR poate fi utilizată în cazul unor funcţii de repartizare mai sofisticate.

4.2 Funcţii polinomiale de dispersie

Un tip funcţie de dispersie frecvent utilizată pentru a asocia unui *string* un număr natural este un polinom de grad $L-1$ unde L este lungimea şirului considerat:

$$h(sir) = sir[0] + sir[1] * p + sir[2] * p^2 + \ldots + sir[L-1] * p^{L-1} = \sum_{i=0}^{L-1} sir[i] * p^i$$

În cazul unor şiruri de caractere lungi există riscul de a obţine valori extrem de mari, care să exceadă maximul admis pentru valorile numerice. Din acest motiv, adesea valoarea obţinută se consideră modulo M, unde M este suficient de mare, ca să poată permite obţinerea tuturor numerelor naturale din intervalul posibil ([0, MAX_INT]). Formula devine astfel:

$$h(sir) = \left(\sum_{i=0}^{L-1} sir[i] * p^i \right) \mod M$$

În [**?**] se folosesc metoda lui Horner şi regulile pentru restul împărţirii la un număr natural pentru a calcula iterativ formulă, astfel

$$h_n = 0, h_i = (h_{i+1} * p + sir[i]) \mod M, i = n-1, \ldots, 0$$

Pentru a avea o probabilitate cât mai redusă de coliziuni, este esenţială alegerea parametrilor p şi M. În [**?**] este discutat modul de alegere al acestora. În diferite exemple practice se utilizează pentru p un număr prim, de exemplu 31 sau 33 (Bernstein hash), iar pentru M poate fi considerat un număr prim mai mic decât 2^{64}, o valoare întâlnită în literatura online fiind $10^9 + 7$.

4.3 Algoritmul FNV

Unul dintre cei mai cunoscuţi algoritmi de asociere a unei valori naturale pentru un *string*, algoritmul FNV, utilizează o metodă similară cu cea a polinoamelor, doar că în loc de adunare între puteri se foloseşte operaţia de XOR. Algoritmul general este dat de:

Algoritm: FNV-Hash

Intrare: cheia k
$hash \leftarrow FNV\_Basis$
pentru *fiecare byte b din şir* **executa**
  $hash \leftarrow hash \times FNV\_prime$
  $hash \leftarrow hash$ XOR b
sfarsit_for
return hash

Constantele FNV_Prime, şi FNV_Basis depind de numărul de biţi pe care este reprezentată valoare de dispersie rezultată. În [**?**] este explicat modul de alegere al acestor constante, iar pentru valori pe 32 de biţi se consideră:

$$FNV\_Prime = 16777619, FNV\_Basis = 2166136261$$

.

Observaţie Evident, în cazul tuturor acestor funcţii care calculează o valoare naturală asociată unui **string**, pentru inserarea într-o tabelă de repartizare valoarea obţinută trebuie considerată modulo dimensiunea tabelei.