TABELA DE DISPERSIE
Hash Table

- Este o structură de date **eficientă** pentru implementarea dicționarelor (şi nu numai).
- Compilator – **tabelă de simboluri** (cheia = şir de caractere corespunzătoare unui identificator)
- TD poate fi folosită pentru implementarea containerelor pe care operațiile specifice sunt: **adăugare** element, **căutare** element, **ștergere** element. Ex: dicționare, colecții, mulțimi $\Rightarrow$ HashMap, HashSet.
- TD este o generalizare a noțiunii mai simple de **tabelă cu adresare directă**
- Notații
    - n – numărul de elemente din container
    - un element e din container este o pereche cheie (c) – valoare (v) ($TElement = TCheie \times TValoare$)
    - U – universul (domeniul) cheilor
    - K – domeniul actual al cheilor (mulțimea cheilor efective memorate)

Tabelă cu adresare directă

- funcționează bine dacă universul cheilor este mic
- complexitatea timp a operațiilor este $\phi(1)$
- spațiul de memorare este $\phi(|U|)$

PROBLEME

1. Sugerați cum se poate implementa o tabelă cu adresare directă în care cheile elementelor memorate nu sunt neapărat distincte și elementele pot avea date adiționale (operațiile ar trebui să se execute în $\phi(1)$)

Tabela de dispersie

- spațiul de memorare este $\phi(|K|)$
- complexitatea timp ***medie*** pentru toate operațiile pe TD (adăugare, căutare, ștergere) este $\phi(1)$.
    - **căutarea** unui element într-o TD poate necesita $\phi(n)$ în caz ***defavorabil*** (ca și căutarea în liste)
        - în practică, dispersia funcționează foarte bine

- timpul ***mediu*** preconizat pentru căutare este $\phi(1)$

- funcție de dispersie (*hash function*) $d: U \rightarrow \{0,1,...m-1\}$
- $d(c_1) = d(c_2)$ coliziune
- **adăugarea unui nou element** $e=(c, v)$
  - se calculează $i = d(c)$
  - dacă locația *i* este liberă, atunci se adaugă elementul
  - dacă la locația *i* mai e memorat un alt element $\Rightarrow$ **rezolvare coliziune**
- funcție de dispersie bună
  - este ușor de calculat (folosește operații aritmetice simple)
  - produce cât mai puține coliziuni.

Interpretarea cheilor ca numere naturale

- Majoritatea funcțiilor de dispersie presupun universul cheilor din mulțimea numerelor naturale
- În cazul în care cheile nu sunt numere naturale, trebuie găsită o modalitate de a le interpreta ca numere naturale – o funcție care asociază fiecărei chei un număr natural (implementare - $hashCode: TElement \rightarrow \{0,1,2...\}$)
  - Ex: identificatorul **pt** poate fi interpretat ca un număr în baza **128** – $(\mathbf{pt})_{128}=112\cdot128+116=14452$.

Funcții de dispersie

- O funcție de dispersie bună satisface (aproximativ) *ipoteza dispersiei uniforme simple* (**Simple Uniform Hashing**): fiecare cheie se poate dispersa cu aceeași probabilitate în oricare din cele *m* locații.
  - $P(d(c) = j) = \frac{1}{m}, \forall j = 0,...,m-1 \quad \forall c \in U$
  - Dacă această ipoteză este verificată, atunci se minimizează numărul de coliziuni
  - în practică se pot folosi tehnici euristice pentru a crea funcții de dispersie care să se comporte bine.

I. Metoda diviziunii

- Dispersia prin diviziune
- $d(c) = c \bmod m$
- Experimental: valori bune pentru *m* sunt numerele prime nu prea apropiate de puteri exacte ale lui 2 (ex: 13,...)

II. <u>Metoda înmulțirii</u>

- $d(c) = [m \cdot (c \cdot A \bmod 1)]$ unde "$c \cdot A \bmod 1$" reprezintă $c \cdot A - [c \cdot A]$
- Valoarea lui *m* nu e critică (în general este o putere a lui 2)
- Knuth: valoarea optimă pentru *A* este $\frac{\sqrt{5}-1}{2} \approx 0.6180339887$ - **golden ratio**

III. <u>Dispersia universală</u>

- $c = < c_1, c_2, ..., c_k >$
- $d(c) = (\sum_{i=1}^{k} c_i \cdot x_i) \bmod m$ unde $< x_1, x_2, ..., x_k >$ este o secvență de numere aleatoare fixate (selectate de-a lungul inițializării funcției de dispersie)
- Proprietate (foarte bună): oricare ar fi două chei distincte *a* și *b*, probabilitatea ca o funcție de dispersie aleatoare *d* să le mapeze în aceeași locație în tabela de dispersie este $\frac{1}{m}$.

A. <u>Rezolvare coliziuni prin *liste independente* (*înlănțuire*) – SEPARATE CHAINING</u>

<u>Observații</u>

- Este posibil ca listele independente să fie memorate ordonat după cheie sau valoare
- Funcția de dispersie este considerată *bună* dacă listele au aproximativ aceeași lungime
- Dacă apar multe liste de vide sau liste prea lungi ⇒ redispersare (**rehashing**)

<u>Analiza dispersiei cu înlănțuire</u>

Notații și presupuneri

- $\alpha = \frac{n}{m}$ *factorul de încărcare* al tabelei (numărul mediu de elemente memorate într-o înlănțuire)
- Pp. că timpul de calcul al funcției de dispersie este $\theta(1)$ (!! la timpul de căutare se adaugă și timpul de calcul al funcției de dispersie

– La **căutare** apar 2 cazuri
  - Căutare **cu succes**
  - Căutare **fără succes**

Teorema 1. Într-o TD în care coliziunile sunt rezolvate prin înlănțuire, în *ipoteza dispersiei uniforme simple* (SUH), o căutare **fără succes**, necesită, în ***medie***, un timp $\theta(1+\alpha)$.

Teorema 2. Într-o TD în care coliziunile sunt rezolvate prin înlănțuire, în *ipoteza dispersiei uniforme simple* (SUH), o căutare **cu succes**, necesită, în ***medie***, un timp $\theta(1+\alpha)$.

CONCLUZII

- Dacă $n = O(m) \Rightarrow \alpha = \frac{O(m)}{m} = O(1) \Rightarrow$ **căutarea** necesită, în ***medie***, timp constant $\theta(1)$
- Adăugarea necesită $\theta(1)$
- Dacă listele sunt dublu înlănțuite atunci și ștergerea se poate face în $\theta(1)$

$\Rightarrow$ **TOATE OPERAȚIILE (adăugare, căutare, ștergere) POT FI EXECUTATE ÎN *MEDIE* ÎN** $\theta(1)$

PROBLEME

2. Presupunem că folosim o funcție de dispersie aleatoare **d** pentru a dispersa *n* chei distincte într-o tabelă *T* de dimensiune *m*. Care este numărul mediu de coliziuni? (cardinalul probabil al mulțimii $\{(x, y) \in TCheie \times TCheie : d(x) = d(y)\}$)
3. Presupunem că folosim o TD în care coliziunile sunt rezolvate prin înlănțuire (liste independente), dar fiecare listă este ordonată după cheie. Care va fi timpul de execuție pentru **căutare** (cu succes, fără succes), **adăugare** și **ștergere**?
4. Arătați că dacă $|U| > n \cdot m$, atunci există o submulțime a lui U de mărime *n* ce conține chei care se dispersează toate în aceeași locatie, astfel încât timpul de căutare pentru dispersia cu înlănțuire, în cazul cel mai defavorabil, este $\phi(n)$.

B. Rezolvare coliziuni prin *liste întrepătrunse* (*întrepătrunderea listelor*) – COALESCED CHAINING

- Toate listele înlănțuite (care memorează coliziuni) se memorează în tabelă, nu sunt liste înafara tabelei (vezi lista înlănțuită cu înlănțuiri reprezentate pe tablou)
- Nu se folosesc pointeri pentru memorarea înlănțuirilor

- Factorul de încărcare este subunitar $\alpha < 1$, altfel tabela este plină
- Gestiunea spațiului liber în tabelă poate fi făcută ca la lista înlănțuită cu înlănțuiri reprezentate pe tablou (folosind o listă înlănțuită a spațiului liber)
- Dezavantaj: tabela se poate umple ($\alpha = 1$). Soluție: se crește *m*, ceea ce presupune redispersarea elementelor.
- Experimental: funcția de dispersie se consideră bună dacă spațiul de memorie e ocupat mai puțin de 75% ($\alpha < 0.75$)

Teorema. Într-o TD în care coliziunile sunt rezolvate prin liste întrepătrunse, în *ipoteza dispersiei uniforme simple* (SUH), o **TOATE** operațiile (**adăugare, căutare, ștergere**), necesită, în ***medie***, un timp $\theta(1)$.

!!! IMPLEMENTAREA OPERAȚIILOR LA SEMINARUL 6