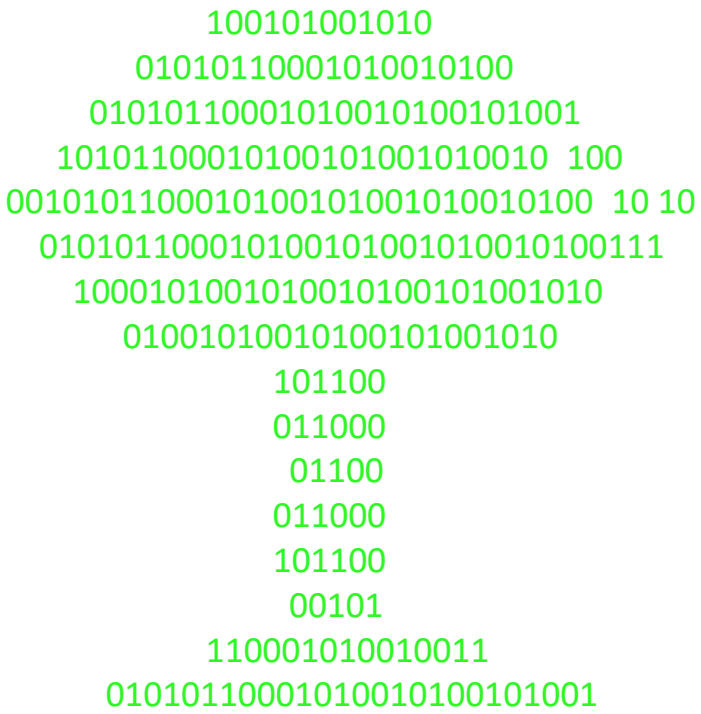
Structuri de date si algoritmi

Curs, IS – An II



Tabele de dispersie (Hash Tables)

Tabele de dispersie (Hash Tables)

Structuri de date ce implementeaza conceptul de dictionar

Elementele stocate intr-o tabela hash → *inregistrari*

Operatii elementare:

Insert

Search/Find

Delete

Tabele de dispersie (Hash Tables)

Tabele fixe (inchise)

- Numarul de elemente ce vor fi stocate este cunoscut in momentul crearii

Tabele dinamice (deschise)

- Numar variabil de inregistrari

Tabele de dispersie (Hash Tables)

Functii hash (functii de dispersie)

O functie ce transforma o cheie intr-un numar.

$f: K \rightarrow H$, K – multimea cheilor;

H – multimea valorilor hash (indici in tabel)

Pune la dispozitie modalitatea de a crea o mica amprenta digitala plecand de la orice tip de date (amprenta = valoarea hash)

Tabele de dispersie (Hash Tables)

Cautarea intr-o tabela hash

HT – tabela hash (vector cu M elemente) **O(1)**

f – functia hash

- Se cauta inregistrarea cu cheia key:

```
h = f(key) //se aplica functia de dispersie
```

- Inregistrarea key se afla pe pozitia h in tabela:

```
HT[h]
```

Tabele de dispersie (Hash Tables)

Functii hash (functii de dispersie)

Proprietati

1) Daca 2 valori hash sunt diferite, atunci si cele 2 chei corespunzatoare sunt diferite (functiile hash sunt deterministe)
2) O functie hash nu este injectiva (egalitatea a doua valori hash nu garanteaza egalitatea celor doua chei de intrare)
   - in practica $|K| >> |H|$
   - daca pt. $k_1 \neq k_2$ avem $f(k_1)=f(k_2)$, atunci k_1 si k_2 sunt *sinonime*

Se cauta o functie de dispersie care sa distribuie cat mai uniform cheile in intrarile tabelei.

Dispersie perfecta = fiecare cheie este mapata pe un index unic

Tabele de dispersie (Hash Tables)

Functii hash (functii de dispersie)

Coliziuni

- 2 chei distincte carora functia de dispersie le asociaza o aceeasi valoare, produc o ***coliziune***

Tabele de dispersie (Hash Tables)

Functii hash (functii de dispersie)

Restrictii de alegere a functiei hash

1) pt. orice k din K, valoarea hash trebuie obtinuta cat mai rapid posibil
   - Unele functii hash pot fi costisitoare d.p.d.v. al calculelor => timpul pt. calcularea valorii hash poate fi semnificativ

2) functia hash trebuie sa minimizeze nr. de coliziuni
   - In general, nr. de operatii necesare pt. rezolvarea coliziunilor creste liniar cu nr. de chei mapate pe aceeasi valoare hash => excesul de coliziuni va degrada performanta la cautare a tabelei

Tabele de dispersie (Hash Tables)

Functii hash (functii de dispersie)

2 pasi: (1) transformarea cheii intr-un nr. intreg

(2) transformarea intregului intr-un index din tabel

M – nr. de intrari in tabela

Exemple:

1) Functie hash modulara

$h(k) = \gamma(k) \bmod M$

$\gamma(k)$ – asociaza cheia k cu un nr. natural

M – nr. de intrari in tabela (ideal nr. prim)

ex. chei numerice:

$\gamma(k) = k$ sau $\gamma(k) = k(k+3)$

chei alfanumerice:

$\gamma(k)$ = suma codurilor ASCII ale caracterelor cheii

Tabele de dispersie (Hash Tables)

Functii hash (functii de dispersie)

Exemple:

2) Dispersie dupa metoda inmultirii

h(k) = [M * {ɣ(k) * A}], 0 < A < 1

{ } – partea fractionara; [ ] – partea intreaga

0 ≤ {k * A} < 1 => 0 ≤ h(k) < M

Obs.: valoarea lui M nu mai are o importanta atat de mare

D. Knuth propune $A = \frac{\sqrt{5}-1}{2} \cong 0.618034$

Tabele de dispersie (Hash Tables)

Functii hash (functii de dispersie)

Exemple:

3) Metode cu deplasari pe biti

Bernstein: h = 5381

```
while (c=*key++)
        h += (h << 5) + c
```

CRC: (cheie = secventa de intregi fara semn)

```
h = 0
for i=0 to k
        high_order = h & 0xf8000000    |pastreaza cei
        h = h << 5                     |mai semnificativi
        h = h ^ (high_order >> 27)     |5 biti
        h = h ^ key[i]
h = h mod M
```

Tabele de dispersie (Hash Tables)

Functii hash (functii de dispersie)

Exemple:

Cheie	∑ASCII	h=∑ASCII mod M		h=[M*{∑ASCII * 0.618034}]
		M=7	M=11	M=7
Ionel	503	6	8	6
George	601	6	7	3
Maria	490	0	6	5
Ion	294	0	8	4
Andrei	595	0	1	5
Ioana	488	5	4	4
Ioan	391	6	6	4
Alex	394	2	9	3
Gabi	371	0	8	2
Daria	481	5	8	1
…				

$\gamma(k) = \sum ASCII$

Tabele de dispersie (Hash Tables)

Rezolvarea coliziunilor

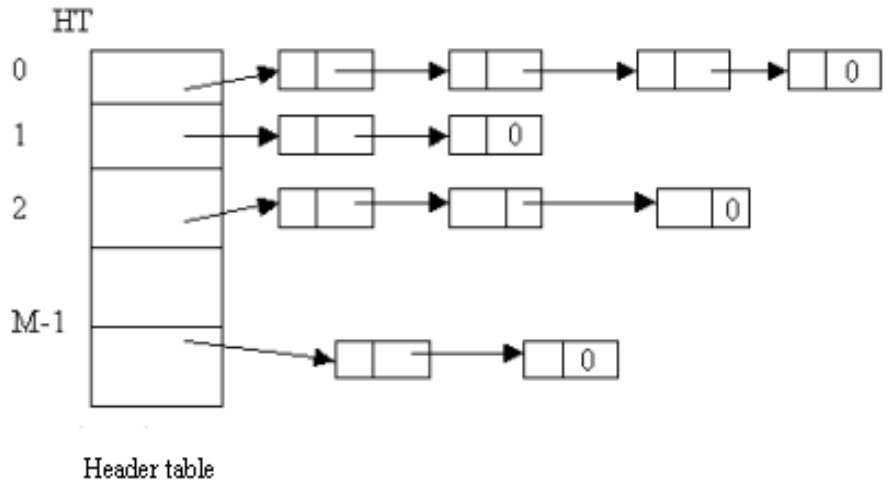
Tabele deschise

Inlantuire

- Pastrarea unei liste pentru fiecare intrare in tabela
- Intr-o astfel de lista se afla toate elementele care au aceeasi valoare hash

```
struct elem {
    char * key;
    …
    elem *leg;
};

elem *HT[M];
```



Tabele de dispersie (Hash Tables)

Rezolvarea coliziunilor

Tabele deschise

Inlantuire

```
initHT(elem *HT[], int M)
   for(i=0; i < M; i++)
      HT[i] = null;
```

```
find(elem *HT[], int M, char *key)
   h = f(key);
   p = HT[h];
   while(p != null)
      if(strcmp(key, p->key) == 0)
         return p;
      p = p->leg;
   return null;
```

Tabele de dispersie (Hash Tables)

Rezolvarea coliziunilor

Tabele deschise

Inlantuire

```
insert(elem *HT[], int M, char * key)
   p = new elem;
   p->key = key;
   p->… = …
   h = f(key);       //calc. val. hash (index in tabela)
   if(HT[h] == null)//nu exista in tabela inregistrari cu val. h
      HT[h] = p;     //inserez p ca prim element al listei HT[h]
      p->leg = 0;
   else              //exista in tabela inregistrari cu val. h
      q = find(key);
      if(q == 0)     //nu exista cheia key
         p->leg = HT[h]; //inserez in fata listei HT[h]
         HT[h] = p;
      else           //cheia key este deja in tabela
         processRecord(p, q); //ex. update
```

Tabele de dispersie (Hash Tables)

Rezolvarea coliziunilor

Tabele deschise

Avantaje:

- Functii hash simple

Dezavantaje (cele ale listelor):

- Pentru chei si valori hash mici -> overhead de stocare (*leg)
- Traversarea nu permite exploatarea unei strategii de cache

Tabele de dispersie (Hash Tables)

Masurarea performantelor

HT[M]

N – nr. total de elemente din colectie

α=N/M -> factor de incarcare (nr. mediu de elemente per index)

Pentru tabele deschise:

- Lungimea asteptata a listei inlantuite = α
- Costul cautarii = O(1+α)
  - Daca $\alpha < \alpha_{max}$ (fix) => **O(1)**
  - In practica, vom obtine cea mai buna performanta cand $\alpha \in [0.5, 2]$

    daca α < 0.5 => HT are multe goluri (M trebuie sa fie mai mic)

    daca α > 2 => costul traversarii listelor este limitativ pentru cautare

➔ Valoarea M este foarte importanta!

Tabele de dispersie (Hash Tables)

Masurarea performantelor

Gradul de clusterizare

- In mod ideal, functia hash definita trebuie testata pentru a verifica daca se comporta bine cu date reale.
- O modalitate este de a masura **gradul de clusterizare** a elementelor:

$$C = \sum_{i=0}^{M-1} \frac{x_i^2}{N} - \alpha,$$

  - unde x_i reprezinta numarul de chei mapate pe valoarea hash (indexul) **i**.
- O functie hash uniforma (care disperseaza uniform inregistrarile in tabela hash) produce o clusterizare aproape de 1.0 cu probabilitate mare.
- Un factor C > 1 indica faptul ca tabela hash este afectata de clusterizare.

Tabele de dispersie (Hash Tables)

Masurarea performantelor

Gradul de clusterizare $C = \sum_{i=0}^{M-1} \frac{x_i^2}{N} - \alpha$

Cheie	∑ASCII	h=∑ASCII mod M		h=[M*{∑ASCII * 0.618034}]
		M=7	M=11	M=7
Ionel	503	6	8	6
George	601	6	7	3
Maria	490	0	6	5
Ion	294	0	8	4
Andrei	595	0	1	5
Ioana	488	5	4	4
Ioan	391	6	6	4
Alex	394	2	9	3
Gabi	371	0	8	2
Daria	481	5	8	1
…				

M = 7:

α = 10/7 = **1.42**

C = 2.7 – 1.42 = **1.28**

M = 11:

α = 10/11 = **0.9**

C = 2.4 – 0.9 = **1.5**

M = 7:

α = 10/7 = **1.42**

C = 2 - 1.42 = **0.58**

Tabele de dispersie inchise (closed hashing)

Toate elementele sunt memorate in interiorul tabelei. Prin urmare, fiecare intrare in tabela contine fie un element al multimii, fie 0.

Tabelele inchise nu pot contine mai multe obiecte decat dimensiunea tabloului.

Daca se depaseste dimensiunea tabloului, se aloca un nou tabel (de obicei cu capacitate dubla) si se redistribuie cheile in noua tabela

Tabele de dispersie (Hash Tables)

Rezolvarea coliziunilor

Tabele inchise

- Probing
- Re-hashing
- Overflow area

Rezolvarea coliziunilor – tabele inchise

Metoda testarii intrarilor libere (probing)

In cazul aparitiei unei coliziuni se incearca pe rand celulele din tabela in ordinea

$h(x), h_1(x), h_2(x), \ldots$

pana cand se gaseste o celula libera.

$h_i(x) = (h(x) + c * f(i)) \% M$, $0 \le i < M$, $f(0) = 0$

f - functia de strategie; c – constanta {1,2,4}

$f(i) = i$ -> testare liniara (linear probing)
$f(i) = i^2$ -> testare patratica (quadratic probing)

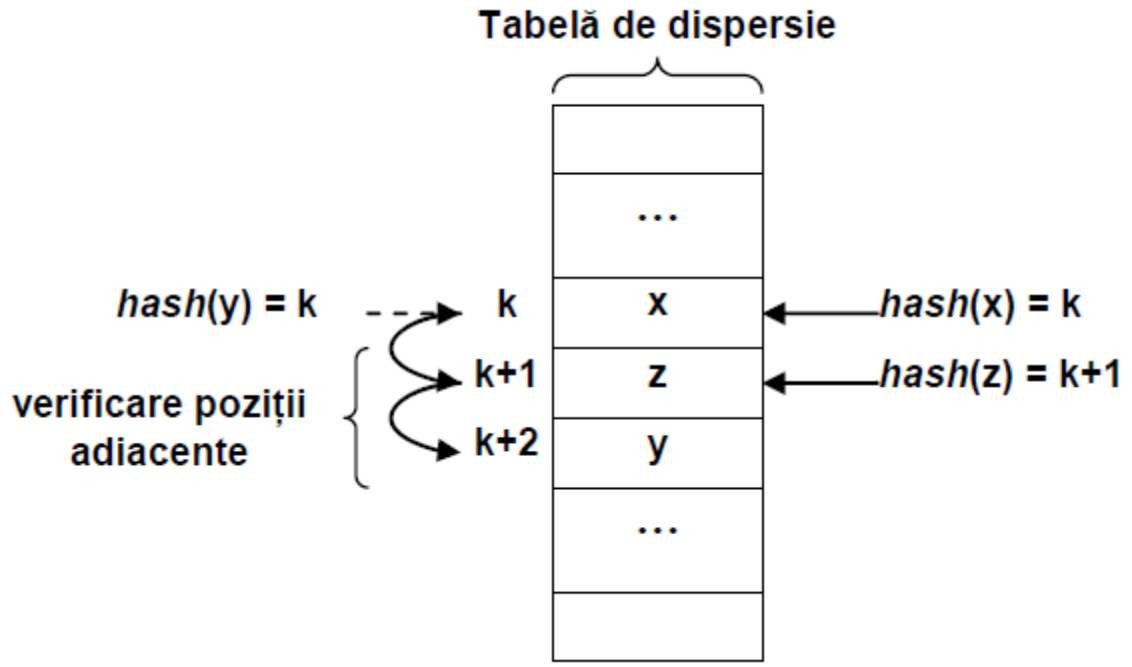
Rezolvarea coliziunilor – tabele inchise

Metoda testarii intrarilor libere (probing)

Linear probing

$h_i(x) = (h(x) + c * f(i)) \% M$

$f(i) = i$



- grad scăzut de complexitate
- are ca efecte secundare gruparea coliziunilor de acelaşi tip în aceeaşi zonă, *cluster*, fapt care conduce la creşterea probabilităţii de apariţie a coliziunilor pentru valorile *hash* adiacente.

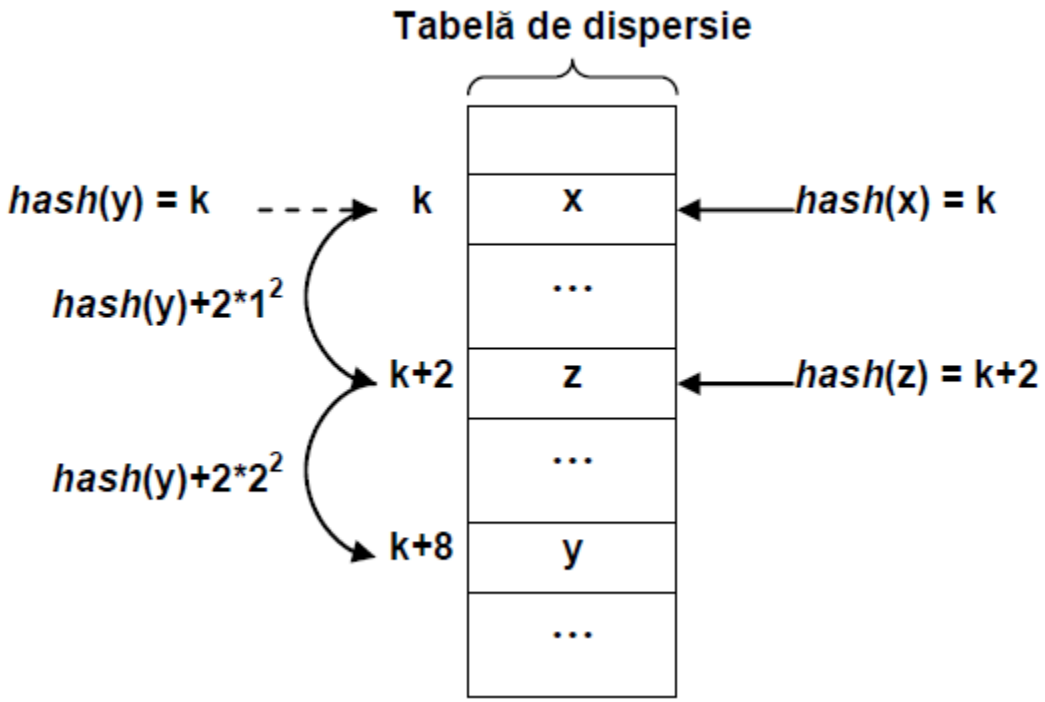
Rezolvarea coliziunilor – tabele inchise

Metoda testarii intrarilor libere (probing)

Quadratic probing

$h_i(x) = (h(x) + c * f(i)) \% M$

$f(i) = i^2$



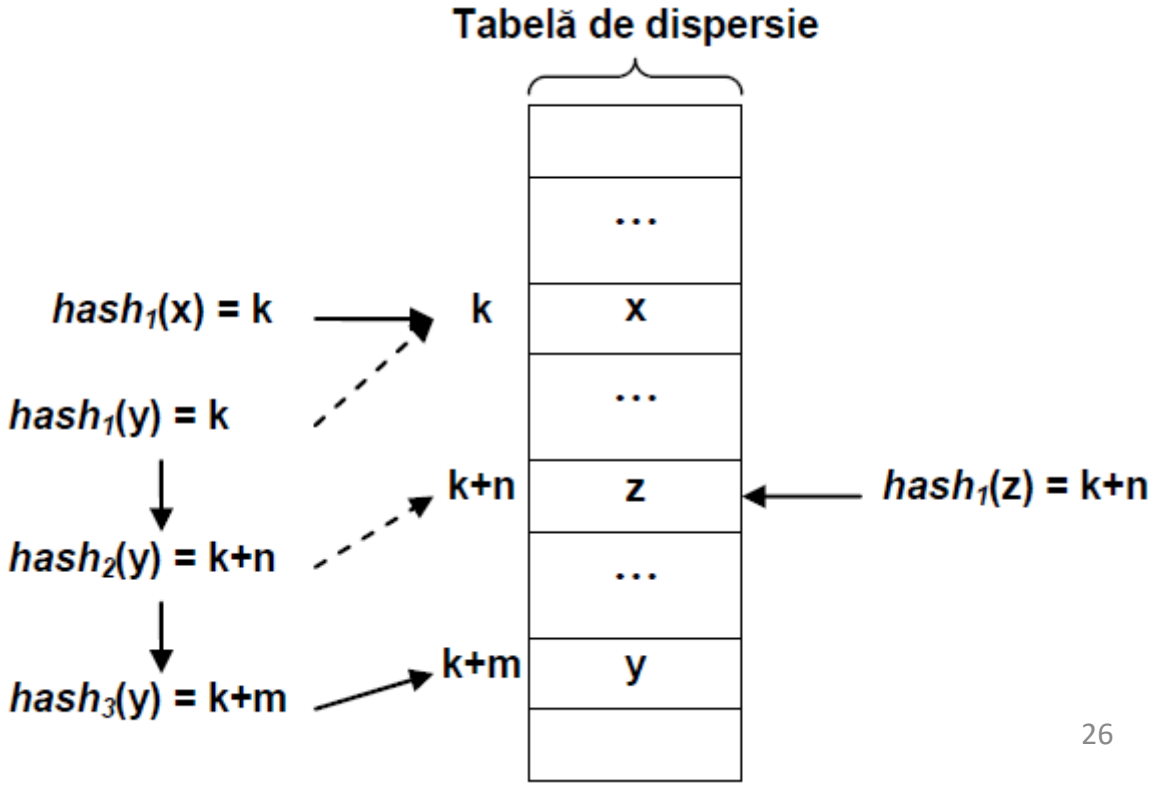
Rezolvarea coliziunilor – tabele inchise

Metoda re-hashing

Presupune aplicarea în cascadă a aceleiaşi funcţii *hash* sau a altui model dintr-o mulţime de funcţii până când valoarea obţinută reprezintă o poziţie liberă din cadrul tabelei de dispersie. La fiecare pas al procesului de regăsire, valoarea cheii de căutare este introdusă într-o listă de funcţii *hash* până când se identifică elementul cu valoarea căutată sau nu mai există alte posibilităţi de a recalcula valoarea *hash*.

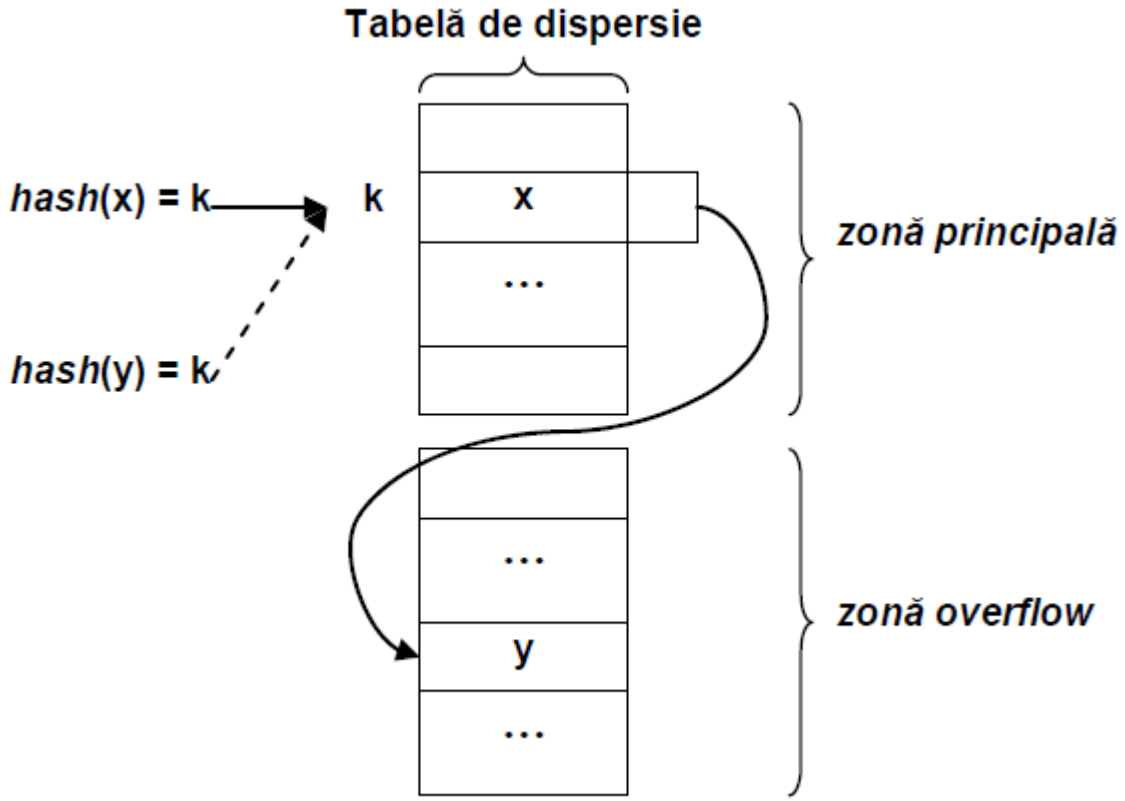
Tipul şi numărul de funcţii *hash* aplicate valorilor intermediare descriu o procedură bine definită de căutare, pentru a conduce de fiecare dată la aceleaşi rezultate:

index = (index + 1 * indexH) % hashTableSize;
index = (index + 2 * indexH) % hashTableSize;



Rezolvarea coliziunilor – tabele inchise

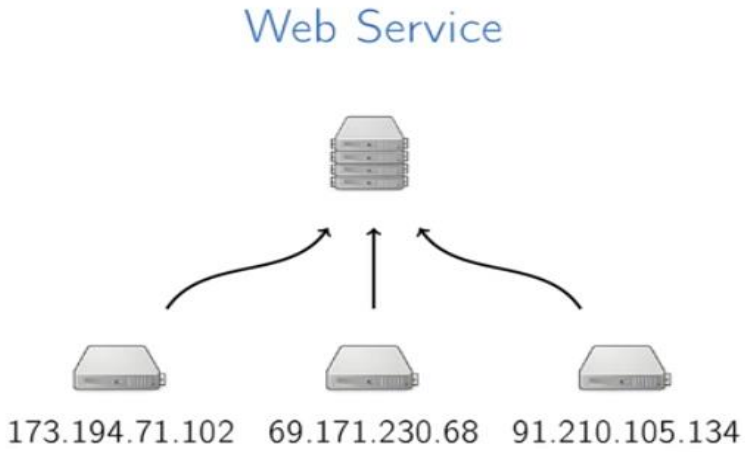
Metoda overflow area - împarte tabela de dispersie în două zone: cea primară pentru reţinerea elementelor iniţiale şi cea secundară pentru elementele ce generează coliziuni. Accesul la zona secundară se realizează prin intermediul unui pointer din zona primară, funcţia hash negenerând poziţii în acest interval de valori Regăsirea informației este mai rapidă decât în cazul metodei *chaining* care presupune parcurgerea de liste.

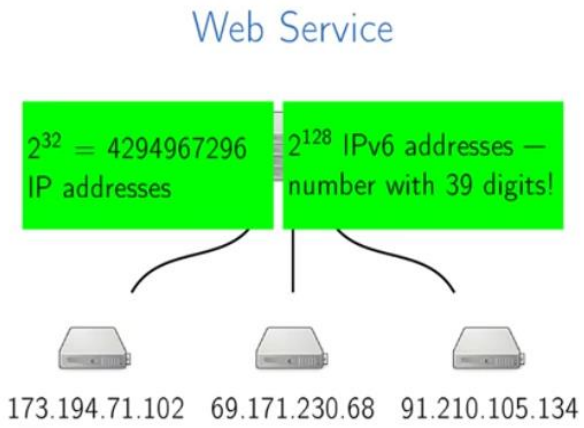


Aplicații

- Aplicații tip dictionar
- Stocare keywords într-un mediu de programare (Python de ex.)
- File System (nume fisier -> locatia fizică a înregistrării)
- Parole acces
- Optimizare spatiu de stocare (Dropbox, Google Drive)
- Codare IP-uri acces la un server

Analiza log-urilor la un server





Access Log

Date	Time	IP address
09 Dec 2015	00:45:13	173.194.71.102
09 Dec 2015	00:45:15	69.171.230.68
...	...	...
...	...	...
09 Dec 2015	01:45:13	91.210.105.134

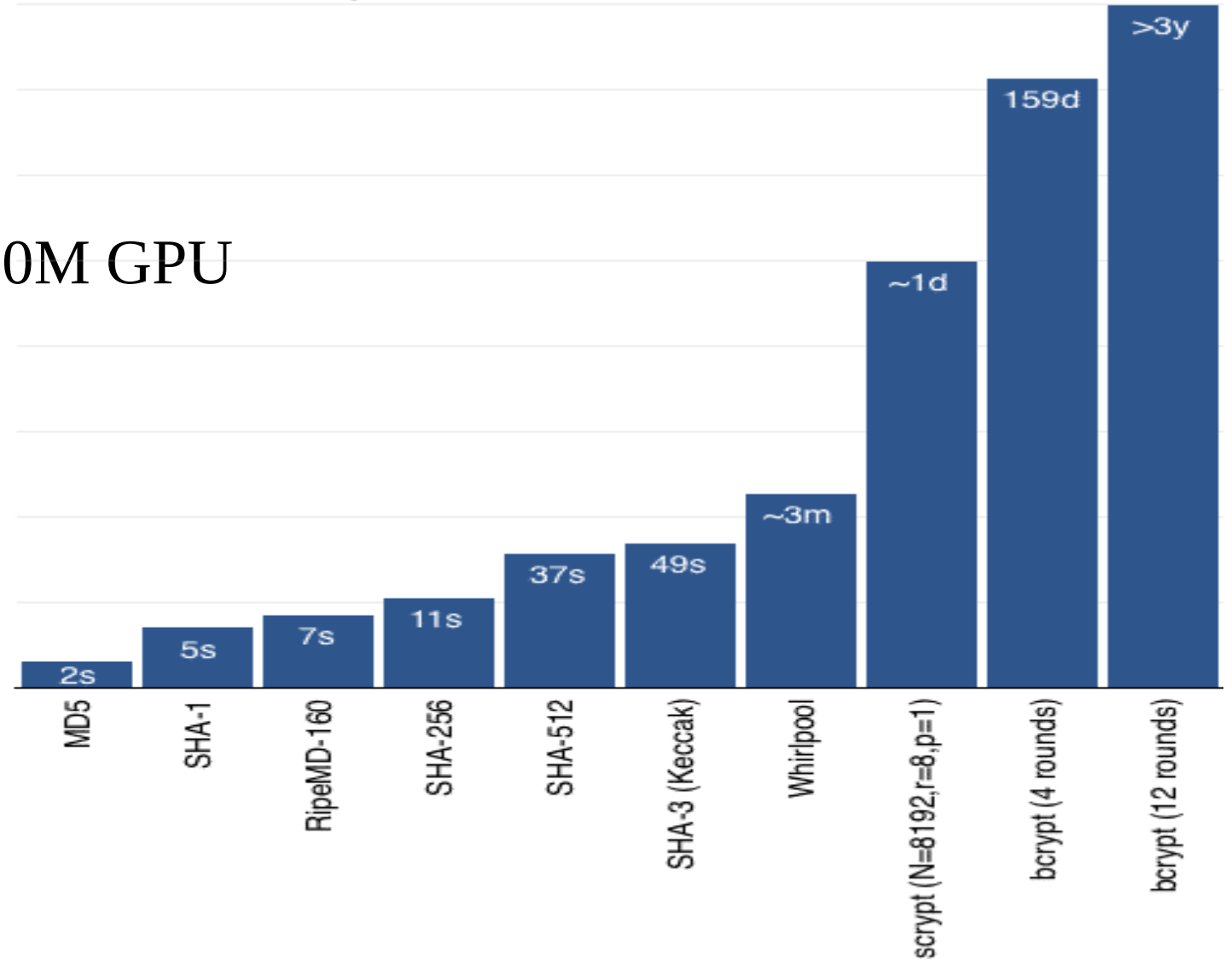
- Numarul de accesari de la un anumit IP într-un interval de timp
- De la cate IP-uri au avut loc accesarile?
- Milioane de linii pentru logarile dintr-o ora
- Trebuie blocat un IP?
- Nu se poate procesa pentru fiecare cerere de acces

Imagini preluate: https://www.coursera.org/learn/data-structures

Comparatie între algoritmi de hashing

Brute force pentru parola **Pw#1!**

Rulat pe NVIDIA Quadro M2000M GPU



Preluat: https://www.novatec-gmbh.de/en/blog/choosing-right-hashing-algorithm-slowness/