Tabele de dispersie

Hash tables

Hash

O funcție hash (hash function) reprezintă un algoritm matematic criptografic care generează un rezumat (checksum) unic pentru fiecare mesaj.

Funcțiile hash trebuie să asigure câteva proprietăți:

- două mesaje diferite generează două hash-uri diferite, adică fiecare mesaj generează un hash unic sau nu exista 2 mesaje diferite avand același hash; dimensiunea hash-ului este întotdeauna aceeași indiferent de mărimea datelor care generează hash-ul;
- nu sunt funcții inversabile(one-way functions): nu există posibilitatea practică ca din hash să fie recreat mesajul inițial;
- funcțiile de hash sunt extrem de sensibile la orice modificare oricât de mică (rezultatul este mult diferit);
- hash-ul unui mesaj este mereu același.

Exemple de utilizare: garantarea integrității unui fișier, semnarea digitală a unui mesaj
Exemple de algoritmi: SHA1, SHA2, SHA3, MD5,..

MD5

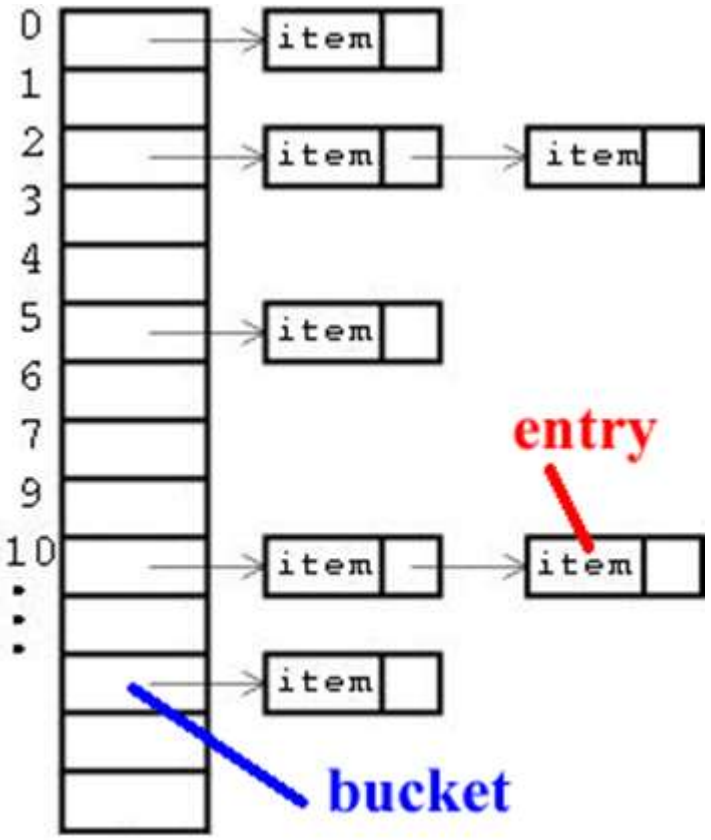
MD5 este o funcție criptografică de tip hash unidirecțional, care livrează ca rezultat o valoare fixă ca lungime de 128 Biți (32 car hex).

md5("parola") = `8287458823facb8ff918dbfabcd22ccb`

md5("Parola") = `0d1b8fa73ec062bf566e0a6beb1ba183`

https://www.md5hashgenerator.com/

Vector de pointeri vs. vector de liste



Dicționar (TAD)

- Un dicționar (dictionary, map, associative array - tabel asociativ) este o colecție de perechi **cheie - valoare**, in care cheile sunt *distincte* și sunt folosite pentru regăsirea rapidă a valorilor asociate.

*"**Map**"* Java, C++ (*each key can map to at most one value*)
*"**Dictionary**"* .Net, Python
*"**Associative array**"* Javascript, PHP

- Exemple: dicționar explicativ {expresie, explicația-conceptului}, dicționar englez român

- Un dicționar este o structură pentru căutare rapidă (ca și *mulțimea*) având diverse implementări: vector, listă de înregistrări, tabel de dispersie (hash), arbore binar echilibrat de căutare (pt mai multe chei -> micșorează timpul de acces).

- Cheia poate fi de orice tip.

http://en.wikipedia.org/wiki/Associative_array

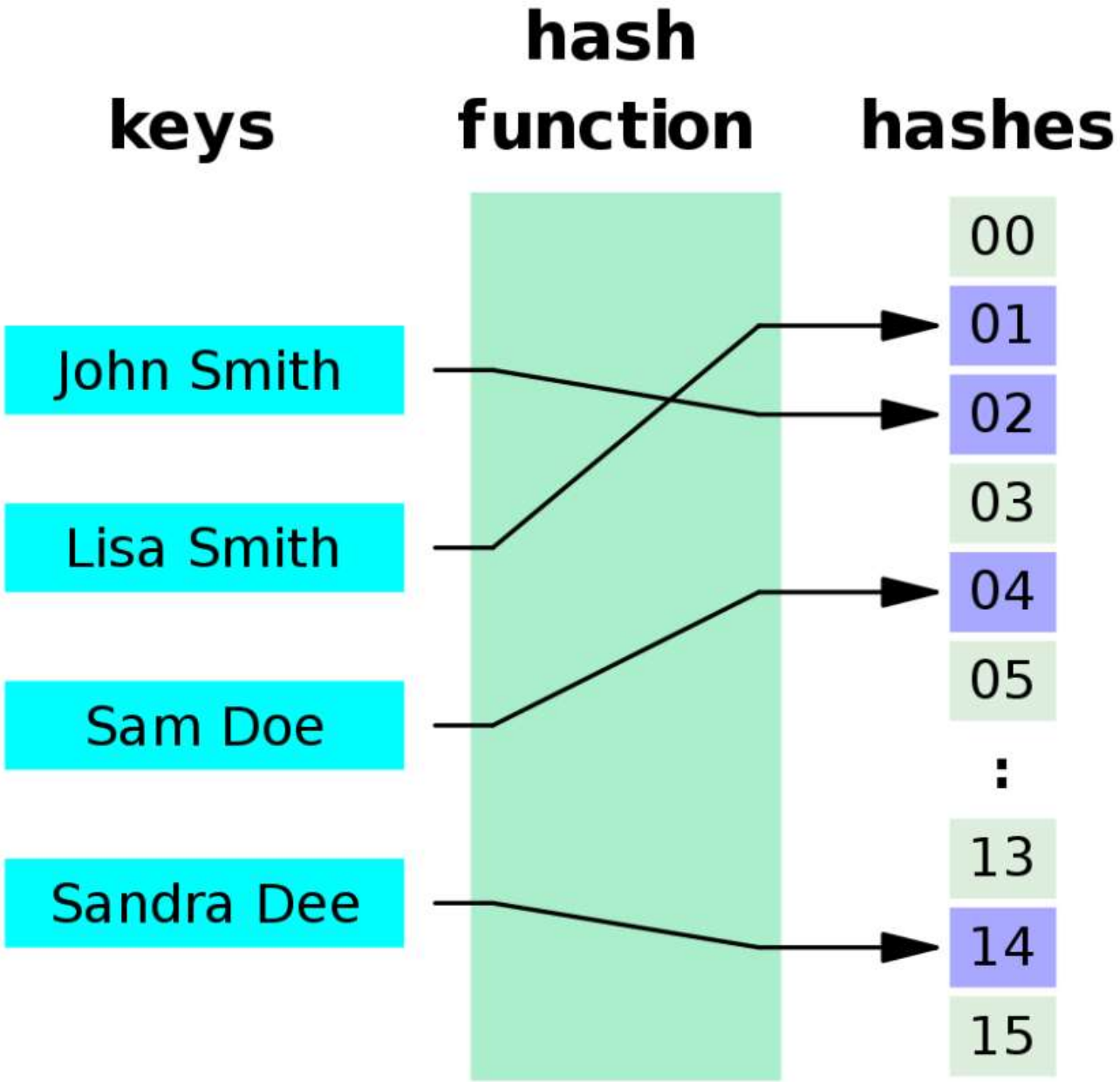
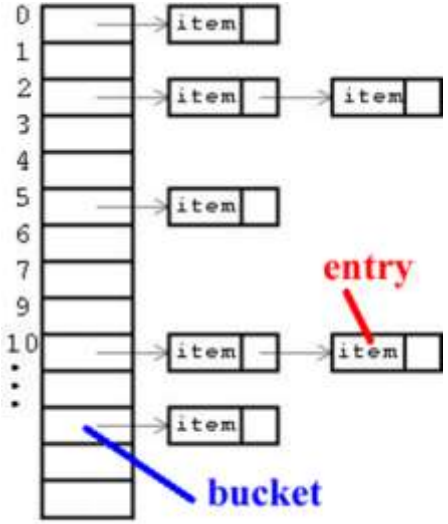

keys
hash function
hashes
John Smith
Lisa Smith
Sam Doe
Sandra Dee
00
01
02
03
04
05
:
13
14
15

Tabela de dispersie (hash table)

- Tabela de dispersie (tabelă asociativă) este o structură de date care asociază chei cu valori
  - permite căutarea eficienta a unei valori pe baza cheii corespunzătoare:
  - in locul dispunerii intr-o singură listă liniară a tuturor perechilor cheie-valoare (entry-uri) din tabelă, acestea sunt grupate în mai multe subliste (bucket-uri), pe baza unui criteriu.

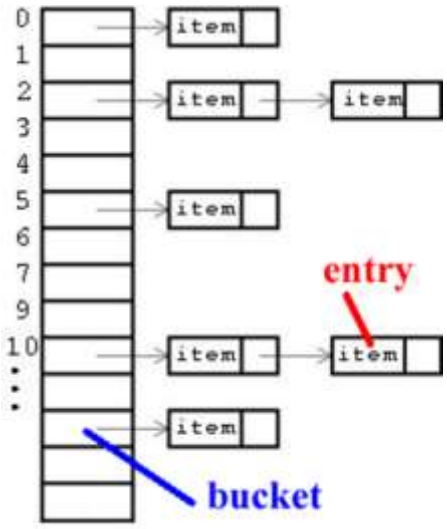


- Căutarea unei chei se va face doar in bucket-ul corespunzător, și nu în toată lista, ceea ce reduce timpul de căutare.

- Pentru stabilirea bucket-ului in care va fi plasată o valoare(entry), cheii ii este aplicată o funcție - numită funcție de dispersie (hash function) care furnizează un număr întreg, numit cod de dispersie (hash code). Pe baza acestuia se determina bucket-ul în care se va introduce valoarea.

Tabela de dispersie

- Dimensiunea tabelei este fixă
- Redimensionarea tabelei presupune rearanjarea conținutului (re-hashing!)
- Nu toate pozițiile din tabelă vor fi ocupate la un moment dat
- Mai multor chei (diferite) li se poate repartiza aceeași poziție în tabelă - situație de **coliziune**



Funcția de dispersie

• trebuie să returneze un intreg intre 0..N (N-dimensiunea tabloului); aceasta valoare mai este numită "*valoare hash*“

• intotdeauna să returneze aceeași valoare hash pentru o anumită cheie și o anumită dimensiune N a tabloului, fără a depinde de altceva.

• să încerce să *împrăștie* (dispersie) aceste valori în tablou cât mai eficient posibil, astfel incât să fie evitată alocarea aceleiași valori unor chei diferite.

• de preferat, cât mai rapidă

un exemplu:

```
unsigned int getHashValueForString(
                        const char* str,
                        unsigned int arraySize   )
{
        return str[0] % arraySize;
};
```

Exemple funcții hash

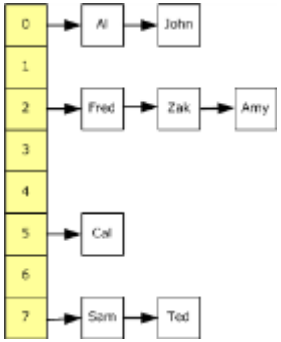
Exemple pentru șiruri (`char *s; len=strlen(s);` parcurs secvențial)

```
for (h=len; len--;) h = ((h<<7) ^ (h<<27)) ^ *s++;      /* Knuth */
for (h=5381; c=*s++; ) h += (h << 5) + c;            /* Bernstein */
for (h=0; c=*s++; ) h = (h<<6) + (h<<16) - h + c;       /* SDBM */
```

- e.g. key = string
  - $h(k) = (s_0a^{k-1} + s_1a^{k-2} + ...+s_{k-2}a + s_{k-1}) \%N$
  - for e.g. a = 33
  - Horner's method: $h(k) = ((((s_0a + s_1)* a + s_2)*a + s_3)*a + ...)*a + s_{k-1}$

```
int hash (char[] v, int N)  {
      int h = 0, a = 33;
      for (int i=0; i< v.length; i++)
            h = (a *h  + v[i])
      return h % N;
}
```

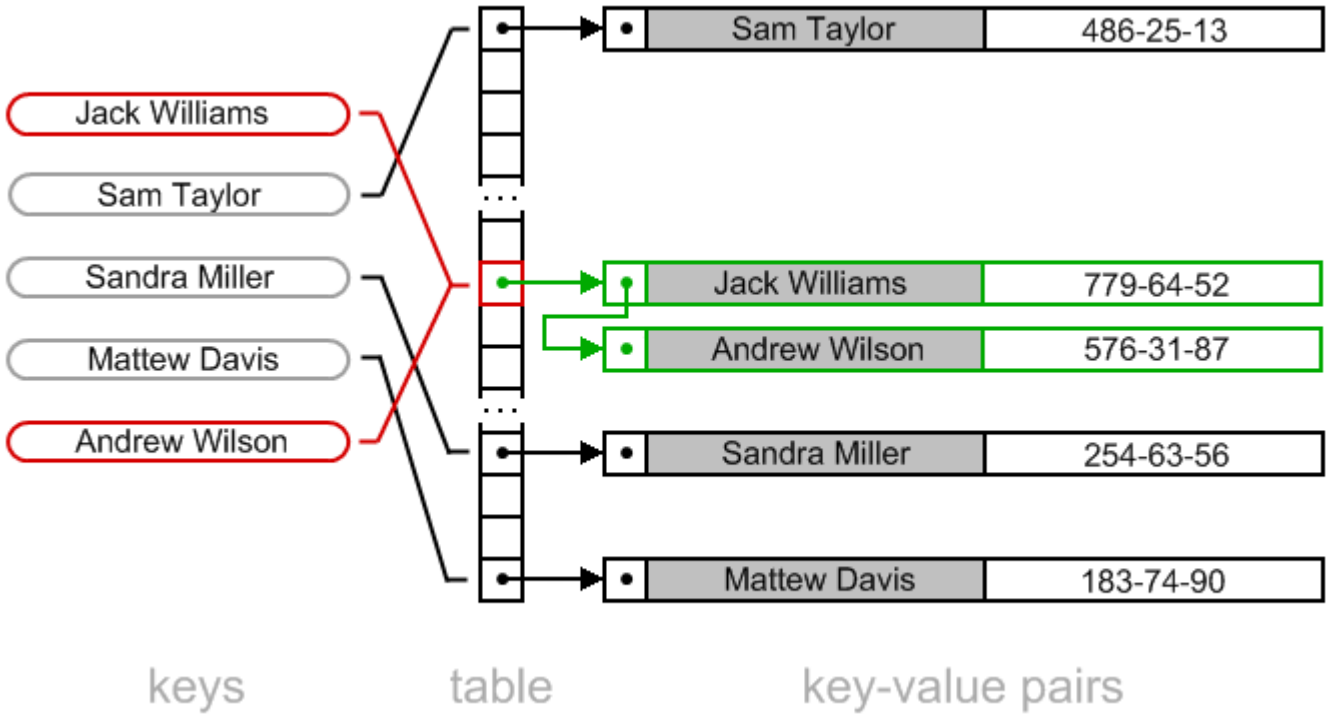
```
int hash (char[] v, int N)  {
      int h = 0, a = 33;
      for (int i=0; i< v.length; i++)
            h = (a *h  + v[i]) %N
      return h;
}
```



Coliziuni

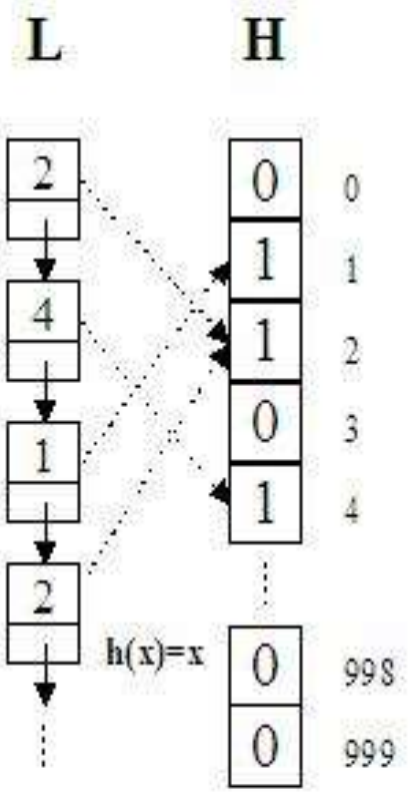
- It can be mathematically proven that the probability of collisions is less if we let the hash function perform % on a prime number rather than on a non-prime number.

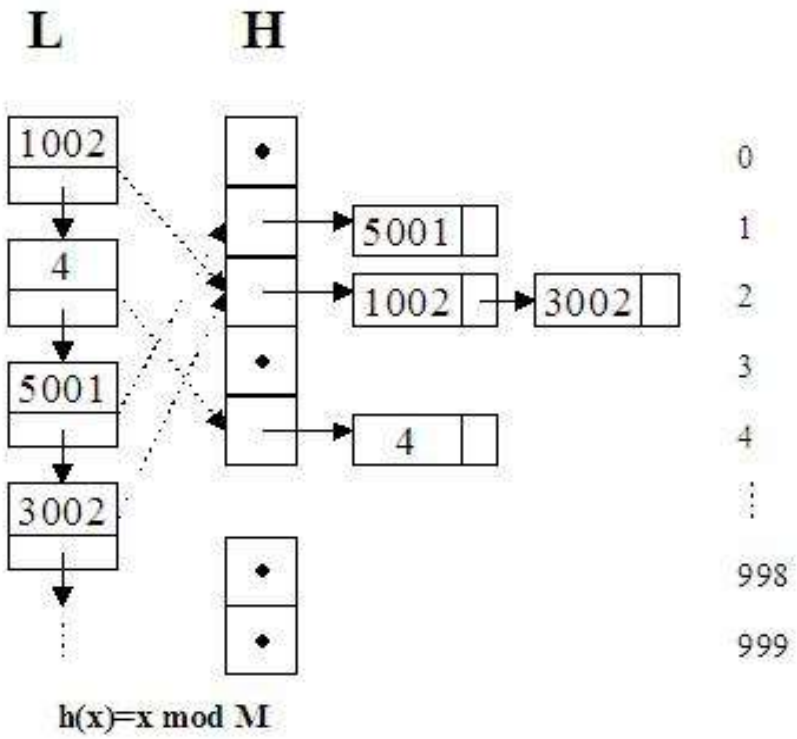
- Hash table [i] -> linked list



Problemă: frecvența de apariție

a unei valori/string într-o mulțime





exemplu generic: h(x)=x

```
// demo hash table
#include <cstdlib>
#include <iostream>

using namespace std;

const int M = 17;
typedef int DataType;
typedef DataType Hash[M];
Hash H;

void InitHash1(Hash H) {
for (int i = 0; i < M; H[i++] = 0);
}

// functia hash
inline int h(DataType K) {
return K;
}

// cauta in tabela hash
int Search1(Hash H, DataType K) {
// Intoarce -1 daca elementul nu exista in hash
// sau indicele in hash daca el exista
return H[h(K)] ? h(K) : -1;
}

// adauga in tabela hash
void Add1(Hash H, DataType K) {
 H[h(K)] = 1;
}

// sterge din tabela hash
void Delete1(Hash H, DataType K) {
 H[h(K)] = 0;
}
```

```
int main(int argc, char *argv[])
{
  InitHash1(H);
  Add1(H,15);
  Add1(H,11);
  Add1(H,12);

  if(Search1(H,12)) printf("\nvaloarea 12 exista\n");
  else printf("\nvaloarea 12 nu exista\n");

  Delete1(H,12);

  printf("\ntabela hash:");
  for(int i=0;i<M;i++) printf("%3d", H[i]);
  printf("\n\n");
  system("PAUSE");
  return 0;
}
```

Re-hashing

```
for (unsigned int i=0;i<oldArray.size();i++) {

        if (oldArray[i]!=NULL) {
                unsigned int newHashValue = theHashFunction(
                        oldArray[i]-> GetKey(), newArray.size()  );

                newArray[newHashValue] = oldArray[i]

        }

}
```

studiu tabel de dispersie