



Tema 6 Estructures no lineals: Hash

Maria Salamó Llorente
Estructura de Dades

Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona



Contingut

6.1 Conjunt

6.2 Map

6.3 Diccionari

6.4 Taules de Hash

6.4.1 Introducció

6.4.2 Funcions de hash

6.4.3 Organització del hash



6.1 Conjunt

Conjunt (set)

- Un conjunt és una col·lecció d'elements no repetits
- A diferència d'una llista o un array, un conjunt **no manté cap ordre** dels seus elements





TAD Conjunt

- **add(obj)**: afegeix un element en el conjunt, si encara no existeix
- **remove(obj)**: elimina un element del conjunt, si existeix
- **boolean contains(obj)**: comprova si un objecte existeix en el conjunt
- **int size()**: retorna el nombre d'elements que hi ha en el conjunt
- **boolean isEmpty()**: comprova si el conjunt està buit
- **list enumerate()**: retorna una llista amb tots els elements en algun ordre arbitrari



Implementació d'un conjunt

- Podem usar un array
 - **add**: afegir un element en l'array $O(1)$
 - **contains**: recórrer l'array $O(n)$
 - **remove**: trobar i eliminar $O(n)$
- Ho podem fer millor?



6.2 Map





Map



- Un mapa modela la cerca en una col·lecció d'entrades (clau-valor)
- Les principals operacions d'un mapa són: cercar, inserir i esborrar ítems
- Múltiples entrades amb la mateixa clau **no** es permeten
- Aplicacions:
 - Llibreta d'adreces
 - Registre d'estudiants



TAD Entry

- Una *entry* guarda un parell clau-valor (k,v)
- Mètodes:
 - `key()`: retorna la clau associada
 - `value()`: retorna el valor associat
 - `setKey(k)`: assigna la clau k
 - `setValue(v)`: assigna el valor v



TAD Map

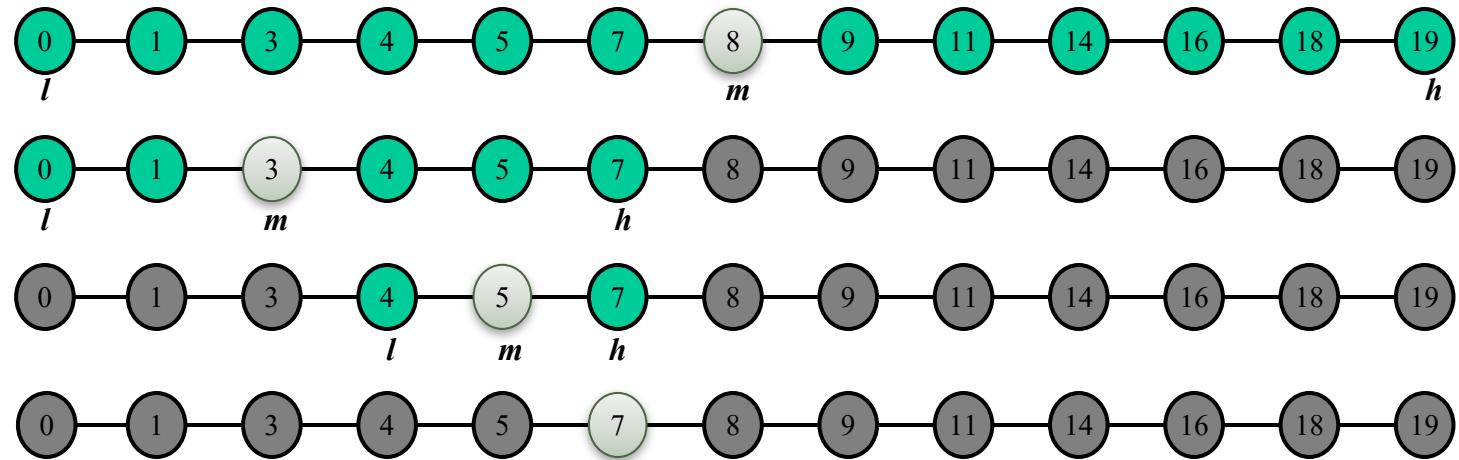


- `find(k)`: si el map M té una entry amb clau k , retorna un iterador a ell; sinó, retorna un iterador especial `end`
- `put(k, v)`: si no existeix una entry amb clau k , inserta la entry (k, v) ; sinó assigna el valor v . Retorna un iterador a la nova/modificada entry
- `erase(k)`: si el map M té una entry amb clau k , l'elimina de M ; si no la troba, dóna un error
- `size()`: retorna el nombre de entries en M
- `empty()`: retorna cert si M està buit, altrement fals
- `begin()`, `end()`: retornen iterators a la primera entry o a posició anterior a l'última entry de M

Exemple

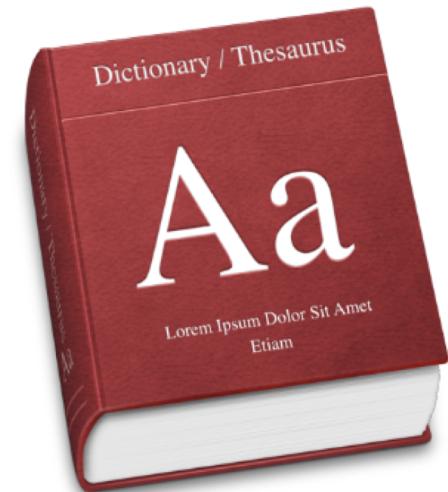
Operació	Sortida	Map
empty()	true	\emptyset
put(5,A)	[(5,A)]	(5,A)
put(7,B)	[(7,B)]	(5,A),(7,B)
put(2,C)	[(2,C)]	(5,A),(7,B),(2,C)
put(8,D)	[(8,D)]	(5,A),(7,B),(2,C),(8,D)
put(2,E)	[(2,E)]	(5,A),(7,B),(2,C),(8,D)
find(7)	[(7,B)]	(5,A),(7,B),(2,E),(8,D)
find(4)	end	(5,A),(7,B),(2,E),(8,D)
find(2)	[(2,E)]	(5,A),(7,B),(2,E),(8,D)
size()	4	(5,A),(7,B),(2,E),(8,D)
erase(5)	—	(7,B),(2,E),(8,D)
erase(2)	—	(7,B),(8,D)
find(2)	end	(7,B),(8,D)
empty()	false	(7,B),(8,D)

6.3 Diccionari



Diccionari

- Un diccionari està dissenyat per emmagatzemar ítems composts per (*key*, *value*), on la *key* és utilitzada per trobar els *values* de l'element corresponent
- És molt similar a un *map*
- Aplicacions:
 - Llibreta d'adreçess (name → address)
 - Diccionari (word → definition)



Diccionari



- Un mapa modela la cerca en una col·lecció d'entrades (clau-valor)
- Les principals operacions d'un mapa són: cercar, inserir i esborrar ítems
- Múltiples entrades amb la mateixa clau **sí** es permeten
- Aplicacions:
 - Parells de (paraula – definició)
 - Autoritzacions de targetes de crèdit
 - Mapeig de DNS dels noms de host (e.g., datastructures.net) a l'adreça IP (e.g., 128.148.34.101)



TAD Entry

- Una *entry* guarda un parell clau-valor (k,v)
- Mètodes:
 - `key()`: retorna la clau associada
 - `value()`: retorna el valor associat
 - `setKey(k)`: assigna la clau k
 - `setValue(v)`: assigna el valor v

TAD Diccionari

- `find(k)`: si existeix una entry amb clau `k`, retorna un iterador a ella; sinó, retorna un iterador especial `end`
- `findAll(k)`: retorna iteradors `b` i `e` de tal manera que totes les entries amb clau `k` estan en el rang de l'iterador `[b, e)` començant amb `b` i acabant just abans de `e`
- `put(k, o)`: inserta i retorna un iterador a ell
- `erase(k)`: elimina una entry amb clau `k`, error si no existeix
- `begin()`, `end()`: retorna iteradors a l'inici i al final del diccionari
- `size()`, `empty()`



Exemple

Operació	Sortida	Diccionari
put(5,A)	(5,A)	(5,A)
put(7,B)	(7,B)	(5,A),(7,B)
put(2,C)	(2,C)	(5,A),(7,B),(2,C)
put(8,D)	(8,D)	(5,A),(7,B),(2,C),(8,D)
put(2,E)	(2,E)	(5,A),(7,B),(2,C),(8,D),(2,E)
find(7)	(7,B)	(5,A),(7,B),(2,C),(8,D),(2,E)
find(4)	end	(5,A),(7,B),(2,C),(8,D),(2,E)
find(2)	(2,C)	(5,A),(7,B),(2,C),(8,D),(2,E)
findAll(2)	(2,C),(2,E)	(5,A),(7,B),(2,C),(8,D),(2,E)
size()	5	(5,A),(7,B),(2,C),(8,D),(2,E)
erase(5)	—	(7,B),(2,C),(8,D),(2,E)
find(5)	end	(7,B),(2,C),(8,D),(2,E)



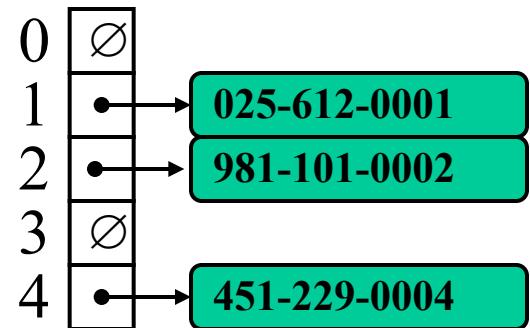
Objectiu

- **put(key, val)**: afegeix un ítem (*key, value*) al diccionari
- **V find(key)**: retorna la *value* mapejada per la *key* corresponent
- **erase(key)**: elimina l'ítem corresponent a la *key* del diccionari
- **int size()**: retorna el nombre d'ítems que conté el diccionari
- **boolean isEmpty()**: comprova si el diccionari està buit

Implementar un map o un diccionari on totes aquestes funcions tinguin complexitat O(1)

6.4 Taules de Hash

6.4.1 Introducció





Taules de Hash

- Una taula hash és la **implementació d'un map o d'un diccionari**
- Les taules hash estan construïdes mitjançant arrays
- Tenim una **funció de hash** $h(key)$ que obté una clau (key) i retorna un índex dins de l'array, on els valors (*values*) de la key corresponents estan emmagatzemats
- És possible que diferents keys vagin a parar al mateix índex

Com es pot guardar múltiples *values* de diferents *key* en un únic índex?

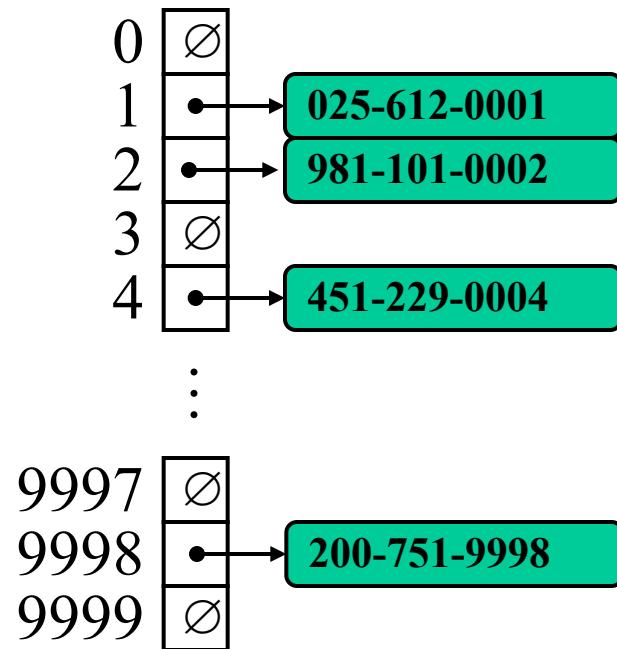
Funcions de hash i Taules de hash



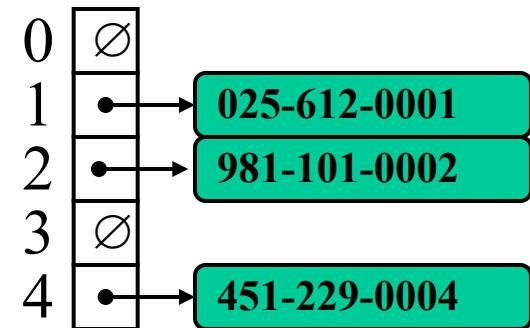
- Una **funció de hash** h mapeja claus de un tipus d'enters a un interval fixe $[0, N - 1]$
- Exemple: $h(x) = x \bmod N$ és una funció de hash per claus de tipus enter
- L'enter $h(x)$ s'anomena valor hash de la clau x
- Una taula de hash per a un tipus de clau consisteix en:
 - Una funció hash h
 - Un array (anomenat taula) de mida N
- Quan s'implementa un map amb una taula de hash, l'objectiu és guardar el ítem (k, o) en l'índex $i = h(k)$

Exemple

- Dissenyem una taula de hash per a un mapa que guarda les entrades com (SSN, Name), on SSN (número de la seguretat social) és un enter positiu de 9 díigits
- La nostra taula de hash usa un array de mida $N=10,000$ i la funció de hash és:
$$h(x) = \text{últims quatre díigits of } x$$



6.4.2 Funcions de Hash





Funcions de Hash



- Una funció de hash normalment s'especifica com la composició de dues funcions:

Codi Hash:

h_1 : keys \rightarrow integers

Funció de compressió:

h_2 : integers $\rightarrow [0, N - 1]$

- El codi hash s'aplica primer i la funció de compressió s'aplica després amb el resultat del codi hash, i.e., $h(x) = h_2(h_1(x))$
- L'objectiu de la funció de hash és “dispersar” les claus d'una manera aparentment random

Codis hash



□ Adreça de memòria:

- Es reinterpreta l'adreça de memòria de la clau com un enter
 - Bona en general, excepte per claus numèriques o strings

□ Conversió a enter:

- Es reinterpreten els bits de la clau com un enter
 - Adequada per claus de mida menor o igual al nombre de bits del tipus enter (e.g., byte, short, int i float en C++)

- Suma de components:
 - Es parteixen els bits en components de mida fixa (e.g., 16 or 32 bits) i es sumen els components (ignorant overflows)
 - Adequada per claus numèriques de mida fixa major o igual al nombre de bits del tipus enter (e.g., long i double en C++)
 - No és adequada per cadenes de caràcters o strings

Codis hash (cont.)

- Acumulació polinomial:
 - Es partitionen els bits de la clau en una seqüència de components de mida fixa (e.g., 8, 16 or 32 bits)
$$a_0 \ a_1 \ \dots \ a_{n-1}$$
 - S'avalua el polinomi
$$p(z) = a_0 + a_1 z + a_2 z^2 + \dots + a_{n-1} z^{n-1}$$
amb un valor fixe z ,
ignorant overflows
 - Especial adequada per strings (e.g., escollint $z = 33$ dóna com a molt 6 col·lisions en un conjunt de 50,000 paraules en Anglès)

- El polinomi $p(z)$ es pot avaluar en un temps $O(n)$ usant la **regla de Horner**:
 - Els polinomis es calculen successivament, cadascú a partir de l'anterior en un temps $O(1)$
- Tenim $p(z) = p_{n-1}(z)$

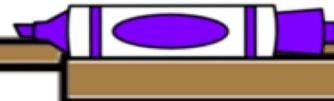
Funcions de compressió



- Divisió:
 - $h_2(y) = y \bmod N$
 - La mida N de la taula de hash normalment s'escull un nombre primer
- Multiplica, Afegeix i Divideix (MAD):
 - $h_2(y) = (ay + b) \bmod N$
 - a and b són enters no negatius tal que $a \bmod N \neq 0$
 - Altrament, cada enter mapejaria al mateixa valor b

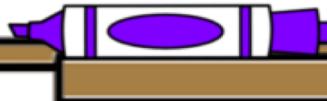


Hashing d'un enter

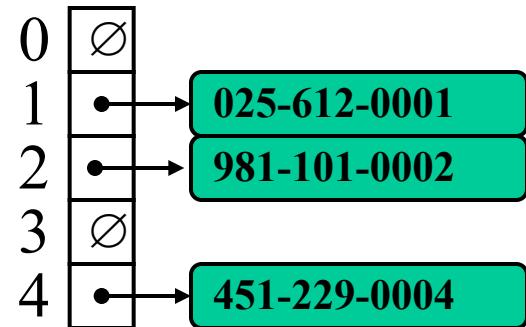




Hashing d'una cadena de caràcters



6.4.3 Organització del hash



Resolució de col·lisions



- Una col·lisió ocorre quan diferents elements es mapegen a la mateixa cel·la

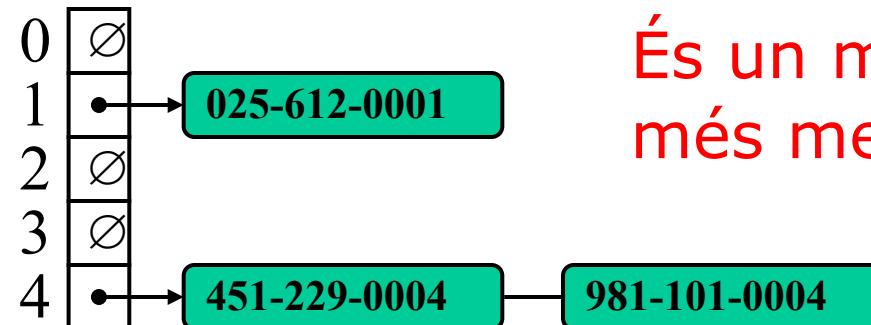
ESTRATÈGIES:

- **Hash obert** o encadenament separat: es dissenya una estructura de llista enllaçada que permet incloure varíes claus en una mateixa posició
- **Hash tancat** o direcccionament obert: Quan es produeix una col·lisió hi ha caselles lliures a la taula, es busca una nova posició per la clau que ha col·lisionat

Tipus de Taules Hash

• Hash Obert

- Consisteix en tenir a cada posició de la taula, una llista encadenada dels elements `<key,values>` (també anomenat array de “buckets”) que, d’acord a la funció de hash, corresponguin a dita posició
- En el pitjor cas, el hashing obert ens porta a una llista on totes les claus estan en única llista
 - El pitjor cas en una cerca és $O(n)$



És un mètode simple però requereix més memòria externa a la taula

Taula Hash Oberta

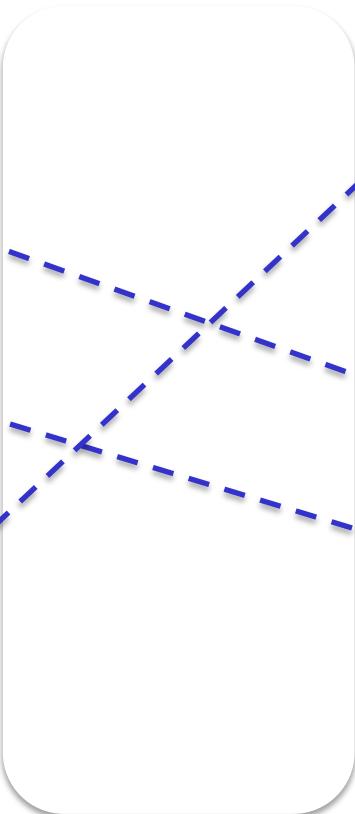
funció de hash:
 $h(key) = key \% 7$

keys:
Banner ID #

B00943855

B00238494

B00472885

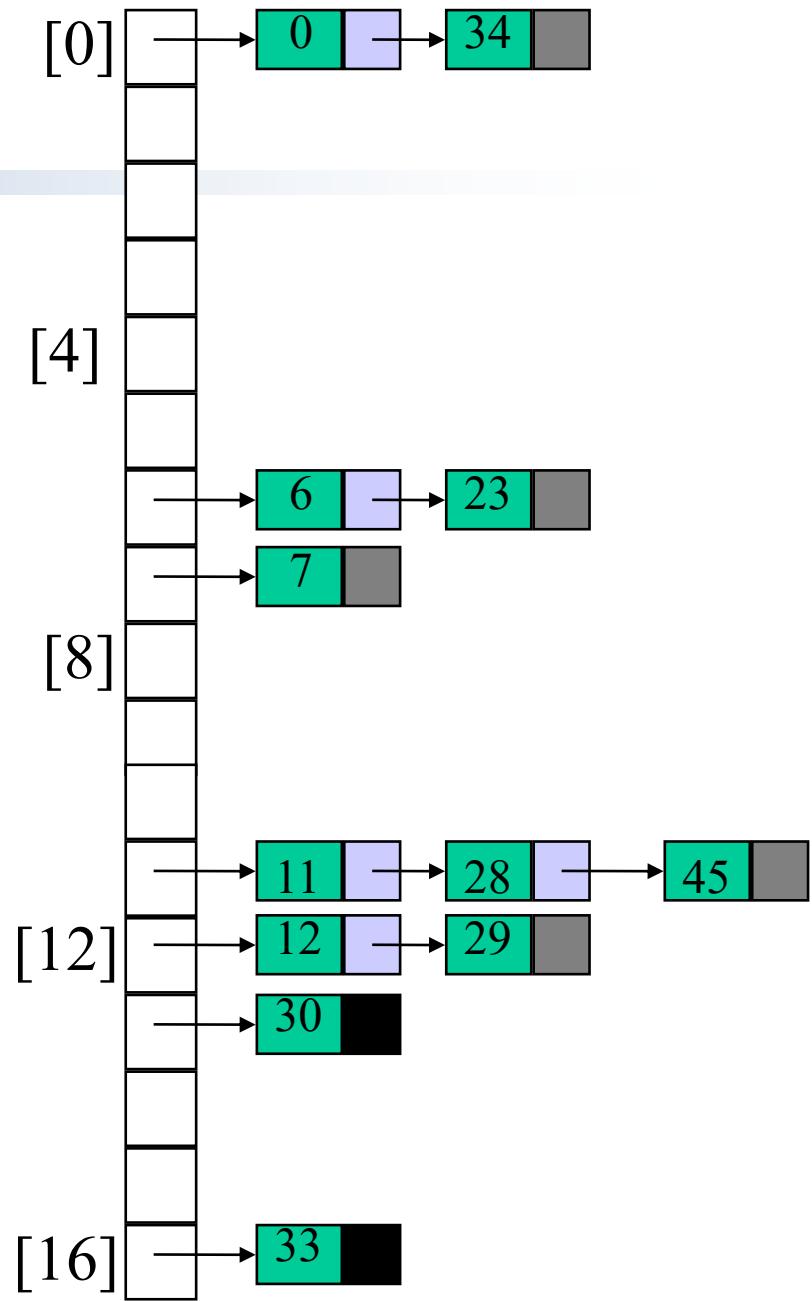


array de “buckets” amb parelles (key, value) :

0	B00472885 David Laidlaw	→	B00231924 Luke Fiorante
1	B00239625 Leah Steinberg		
2			
3	B00943855 Patrick Maiden		
4	B00238494 Sarah Parker		
5	B00745911 Marley Rafson	→	B00543163 Surbhi Madan
6			

Exemple: Taula Hash Oberta

- Claus a inserir en aquest ordre 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
- Bucket = key % 17



Tipus de Taules Hash

• Hash Tancat

- El hash tancat soluciona les col·lisions cercant cel·les alternatives fins a trobar una buida (dins de la mateixa taula). Es va buscant en les cel·les: $d_0(k)$, $d_1(k)$, $d_2(k)$, ..., on:

$$d_i(k) = (h(k) + f(i)) \bmod \text{MAX_TAULA}$$

$$h(k, i) = (h'(k) + f(i)) \bmod \text{MAX_TAULA}$$

- Quan es busca una clau, s'examinen varies cel·les de la taula fins que es troba la clau buscada, o està clar que aquesta no està emmagatzemada
- La inserció s'efectua provant la taula fins a trobar un espai buit

Taula Hash Tancada

funció de hash
 $h(key) = key \% 7$

keys:
Banner ID #

B00943855

B00238494

B00472885

Array amb parelles (key, value):

B00231924
Luke Fiorante

B00239625
Leah Steinberg

B00472885
David Laidlaw

B00943855
Patrick Maiden

B00238494
Sarah Parker

B00745911
Marley Rafson

Taula Hash Tancada

- **Avantatge:** Elimina totalment els apuntadors. S'allibera així espai de memòria, el que pot ser usat en més entrades de la taula
- **Desavantatge:** Si l'aplicació realitza eliminacions freqüents, pot degradar-se el rendiment de la mateixa. Es requereix una taula més gran. Per garantir el funcionament correcte, es requereix que la taula de hash tingui, almenys, el 50% de l'espai disponible.



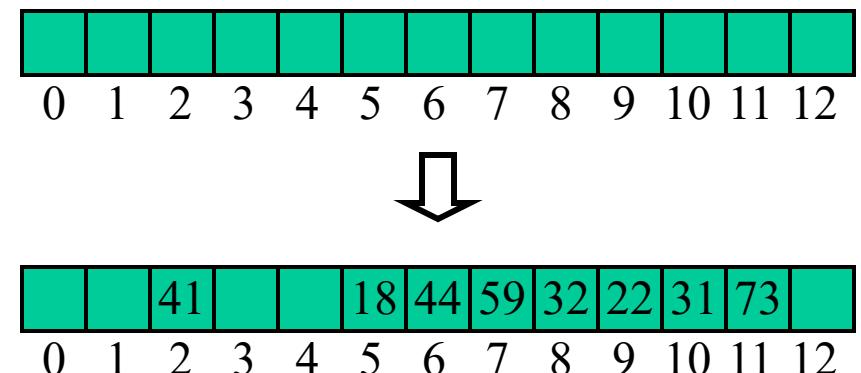
Estratègies en Hash tancat

- **Exploració Lineal**: compara una a una totes les posicions de la taula
- **Hashing doble**: consisteix en definir dues funcions de hash
 - La primera per obtenir la posició inicial
 - La segona en usar en cas de col·lisió

Exploració Lineal

- **Hash tancat:** el ítem que col·lisiona es situa en una posició diferent de la taula
- **Exploració lineal:** manega les col·lisions situant l'element que col·lisiona a la següent (circular) posició disponible a la taula
- Cada cel·la de la taula inspeccionada es coneix com a "prova"
- Els elements que col·lisionen s'ajunten, causant a les futures col·lisions una seqüència més llarga de proves

- **Exemple:**
 - $h(x) = x \bmod 13$
 - Inserta claus 18, 41, 22, 44, 59, 32, 31, 73, en aquest ordre



Cerca amb Exploració lineal



- Considerem una taula hash A que usa exploració lineal
- **find(k)**
 - Es comença a la cel·la $h(k)$
 - Es proven posicions consecutives fins a que una de les següents condicions ocorre
 - Un ítem amb clau k es troba, o
 - Una cel·la buida es troba, o
 - N cel·les han estat provades sense èxit

Algorithm $find(k)$

$i \leftarrow h(k)$

$p \leftarrow 0$

repeat

$c \leftarrow A[i]$

if $c = \emptyset$

return $null$

else if $c.key() = k$

return $c.value()$

else

$i \leftarrow (i + 1) \bmod N$

$p \leftarrow p + 1$

until $p = N$

return $null$



Modificació amb Exploració lineal

- Per manegar insercions i esborrats, introduïm un objecte especial, anomenat *AVAILABLE*, el qual reemplaça els elements esborrats
- $\text{erase}(k)$
 - Es busca una entry amb clau k
 - Si la entry (k, o) es troba, es reemplaça amb un ítem especial *AVAILABLE* i es retorna l'element o
 - Sinó, es retorna *null*
- $\text{put}(k, o)$
 - Es llença una excepció si la taula està plena
 - Es comença a la cel·la $h(k)$
 - Es proven cel·les fins a que ocorre una de les següents condicions
 - Una cel·la i es troba que està “buida” o guarda *AVAILABLE*, o
 - N cel·les han sigut provades sense èxit
 - Es guarda (k, o) a la cel·la i

Hashing doble

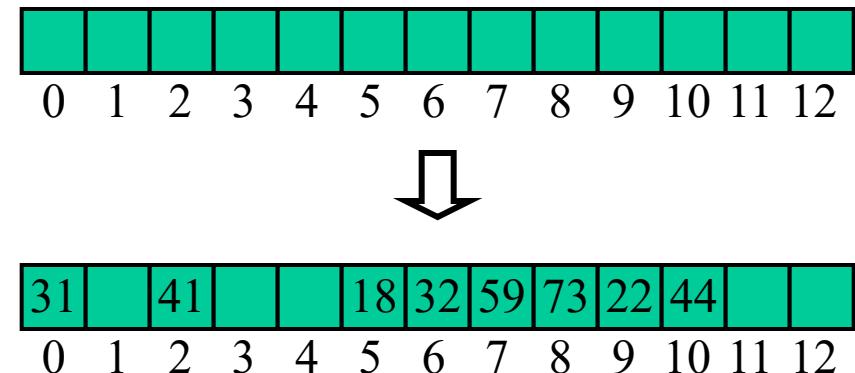


- El hashing doble usa una funció de hash secundària $d(k)$ i manega les col·lisions situant un ítem en la primera cel·la disponible de la sèrie $(i + jd(k)) \bmod N$
for $j = 0, 1, \dots, N - 1$
- La **funció de hash secundària** $d(k)$ no pot retornar un valor zero
- La mida de la taula de hash N ha de ser un nombre primer per permetre provar totes les cel·les
- S'escull la mateixa funció de compressió per la funció de hash secundària:
$$d_2(k) = q - k \bmod q$$
on
 - $q < N$
 - q és un nombre primer
- Els possibles valors per $d_2(k)$ són:
$$1, 2, \dots, q$$

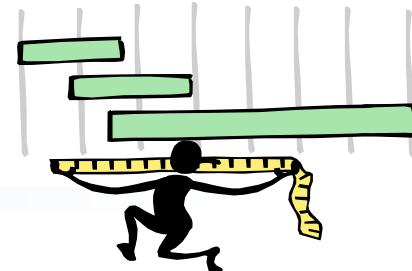
Exemple de Hashing doble

- Considera una taula de hash que guarda claus enteres que resol les col·lisions amb un doble hashing
 - $N = 13$
 - $h(k) = k \bmod 13$
 - $d(k) = 7 - k \bmod 7$
- Inserta les claus 18, 41, 22, 44, 59, 32, 31, 73, en aquest ordre

k	$h(k)$	$d(k)$	Proves
18	5	3	5
41	2	1	2
22	9	6	9
44	5	5	5 10
59	7	4	7
32	6	3	6
31	5	4	5 9 0
73	8	4	8



Rendiment del Hashing



- En el pitjor cas, busca, insereix i elimina en una taula de hash costa $O(n)$ en temps
- El pitjor cas ocorre quan totes les claus inserides en el map col·lisionen
- El factor de càrrega $\alpha = n/N$ afecta al rendiment de la taula de hash
- Assumint que els valors de hash són com nombres randoms, es pot demostrar que el nombre esperat de proves per una inserció amb hash obert és
$$1 / (1 - \alpha)$$
- El temps esperat d'execució per totes les operacions del TAD diccionari en una taula de hash és $O(1)$
- A la pràctica, el hashing és molt ràpid tenint en compte que el factor de càrrega no estigui proper al 100%
- Aplicacions de les taules hash:
 - Petites bases de dades
 - Compiladors
 - La caché dels browsers