


Hash Tables



ADT Dictionary

Has following operations:

- **INSERT**: inserts a new element, associated to unique value of a field (key)
- **SEARCH**: searches an element with a certain value of the key. If it exists, it returns it
- **DELETE**: cancels element with given key, if exists

2



Uses of dictionaries

- Symbol table in a compiler
 - Key: name of identifier
 - Values: types, context
- Citizens in a country
 - Key: social security number
 - Values: name, surname, age, address

3



Associative array

A dictionary would be easily implemented with an associative array (index of value = key instead of position)

Ex:

- Citizens = {{"jr50", "john", "red"}, {"bg40", "bill", "green"}, }
- Citizens["jr50"] = {"jr50", "john", "red"}

4



Goal

Complexity of insert/search/delete:

- $O(1)$ average case
- $\Theta(n)$ worst case

5



Hash tables

Implementation of associative arrays

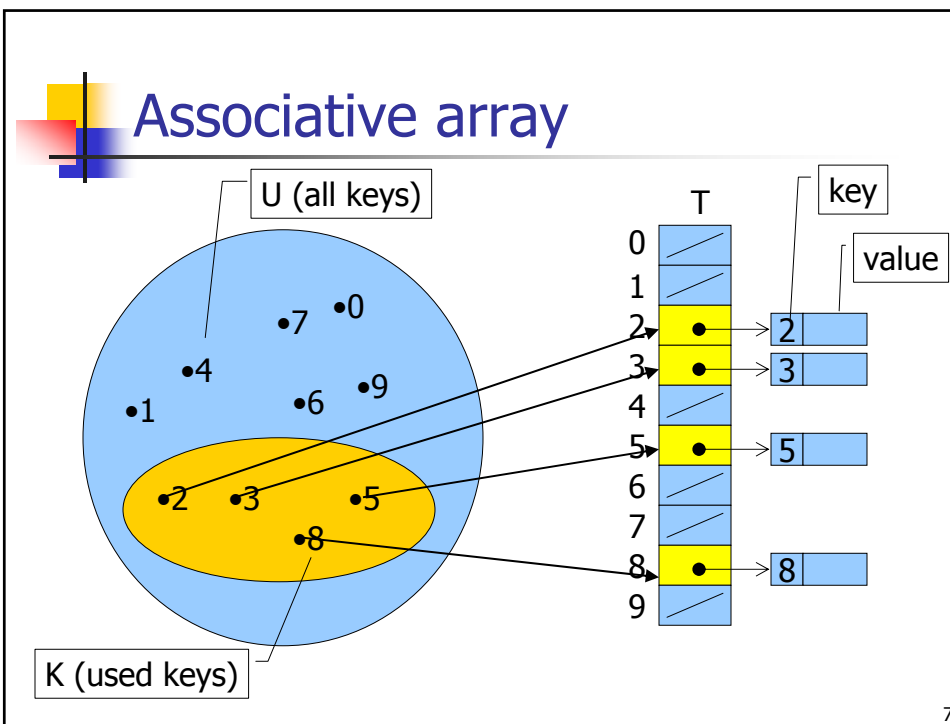
An array containing elements.

Address of element is computed by hash function, in time $O(1)$.

Ex:

- $\text{Hash}(\text{"jr50"}) = 117$: element john red is in position 117 of vector

6



Dictionary implemented w associative array

- T: associative array, key: key, x: value
- Search(T, key)
 - Return T[key]
- Insert(T, x)
 - $T[\text{key}[x]] \leftarrow x$
- Delete(T, x)
 - $T[\text{key}[x]] \leftarrow \text{NIL}$
- Complexity $O(1)$, memory $O(|U|)$

$O(|U|)$ number of different values of key

8



Assumptions

Two assumptions are needed:

- No two elements with same key (keys are unique)
- Size of T == size of max number of possible values of key, $|U|$.
 - This is critical, if $|U|$ is large, array unfeasible
 - Ex: key = SSN, 10chars, $|U| = 24^{10} \approx 10^{13}$
 - Assuming 24 values alfabet
 - But, the citizens of a country are in the order $10^7 - 10^9$
- It is essential that size of array be $O|K|$ and not $O|U|$
- A hash table is an array

9



Hash tables

- A kind of associative array with size $O|K|$ and not $O|U|$
- Insert/search/delete are $O(1)$ on average
- However, the way of computing index given key must be different: hash function

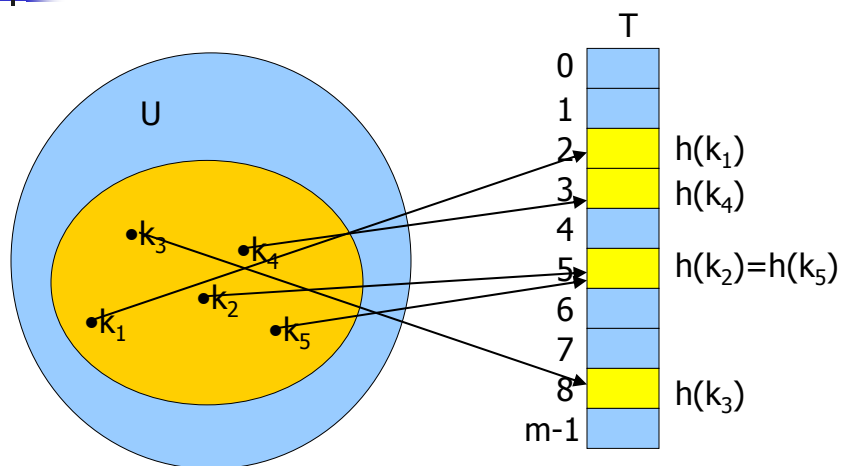
10

Hash function

- Hash table is array with size m ($m \ll |U|$)
- Hash function h , from key to position in array (index)
 - $h: U \rightarrow \{ 0, 1, \dots, m-1 \}$
- Element x is stored in
 - $T[h(\text{key}[x])]$

11

Hash function



12



Collision

- Collision
 - when $h(k_i) = h(k_j)$ and $k_i \neq k_j$,
- Essential to:
 - Minimize number of collisions
 - Depend on hash function
 - Manage collisions

13



Example

Key is a string of characters

Hash function

$$h(k) = \sum(c_i) \bmod m$$

with

- c_i Ascii code of i -th char of string k
- m number of elements (size) of array T

14



Ex (II)

$m = 15$.

- $h(\text{"pippo"}) = (112+105+112+112+111) \bmod 15 = 552 \bmod 15 = 12$
- $h(\text{"pluto"}) = (112+108+117+116+111) \bmod 15 = 564 \bmod 15 = 9$
- $h(\text{"paperino"}) = (112+97+112+101+114+105+110+111) \bmod 15 = 862 \bmod 15 = 7$
- $h(\text{"topolino"}) = (116+111+112+111+108+105+110+111) \bmod 15 = 884 \bmod 15 = 14$
- $h(\text{"paperoga"}) = (112+97+112+101+114+111+103+97) \bmod 15 = 847 \bmod 15 = 7$

Collision with strings
"paperino" and "paperoga"

15



How to reduce collisions

The best hash functions are capable of distributing as uniformly (randomly) as possible the $|K|$ elements among the m positions available

Typical choices

m is a prime number

Manipulate bits of k

16



How to manage collisions

- Chaining
- Open Addressing

17

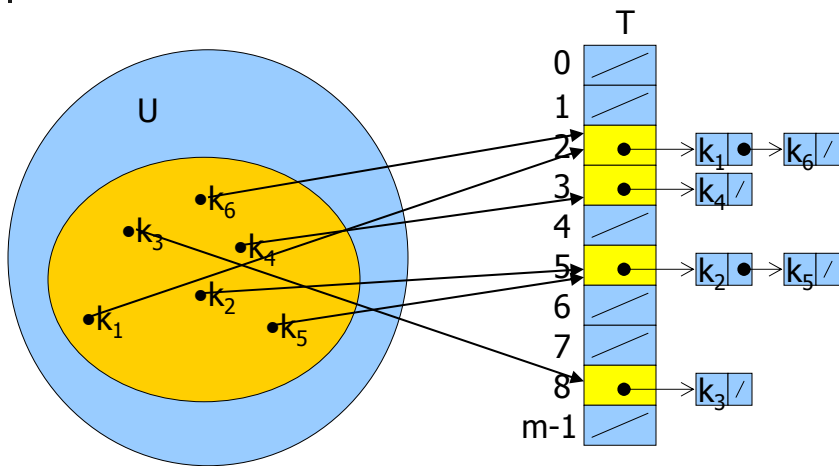


Chaining (I)

Position i can contain more than one element
This can be implemented through linked list

18

Chaining (II)



19

Chaining (III)

- $T[i]$ is a pointer to a list, initially NIL.
- CHAINED-HASH-INSERT(T, x)
 - insert x at head of list $T[h(\text{key}[x])]$
- CHAINED-HASH-SEARCH(T, k)
 - Search element with key k in list $T[h(k)]$
- CHAINED-HASH-DELETE(T, x)
 - Cancel x from list $T[h(\text{key}[x])]$


20



Chaining - Complexity

- Assumption: unordered list, single chaining
- Insert: $O(1)$
- Search: $O(\text{length of list})$
- Cancel: $O(\text{length of list})$
 - Requires a search


21



Search (hash + chaining) - complexity

- We have
 - n : number of elements in hash table T
 - m : size of hash table T
 - $\alpha = n/m$: load factor for hash table T
- Normally $\alpha > 1$
- What if $m, n \rightarrow \infty$ (with same α) ?

22




Search (hash + chaining) – complexity (II)

■ Search

- Worst case: a linked list, not ordered
 - Time to compute $h(k)$ +
 - Time to transverse the list, $\Theta(n)$
- Best case: depends on how uniformly $h(k)$ distributes the elements
- Let's assume $h(k)$ is capable of *simple uniform hashing* (distributes in perfect uniform way) (this requires that the table grows with the elements, so that α remains constant)

23



Search (hash + chaining) – complexity (II)

Search

Time to compute $h(k) = O(1)$.

Time to trasverse the list,

depends on length of list $T[h(k)]$

depends on element found/not found

In both cases complexity is $\Theta(1+\alpha)$.

summing up $O(1) + \Theta(1+\alpha) = O(1)$

24



Open Addressing

$T[i]$ can contain only one element
In case of collision another free cell is searched for
 next one, after next, etc
Must be $\alpha < 1$.

25



Hash-Insert

```
HASH-INSERT( $T, k$ )
1    $i \leftarrow 0$ 
2   repeat  $j \leftarrow h(k, i)$ 
3       if  $T[j] = \text{NIL}$ 
4           then  $T[j] \leftarrow k$ 
5           return
6       else  $i \leftarrow i + 1$ 
7   until  $i = m$ 
8   error "hash table overflow"
```

26



Hash-Search

```
HASH-SEARCH( $T, k$ )
1    $i \leftarrow 0$ 
2   repeat  $j \leftarrow h(k, i)$ 
3       if  $T[j] = k$ 
4           then return  $j$ 
5        $i \leftarrow i + 1$ 
6   until  $T[j] = \text{NIL}$  or  $i = m$ 
7   return NIL
```

27



Re hash functions

- *Linear probing*
 - $h(k, i) = (h'(k) + i) \bmod m$
- *Quadratic probing*
 - $h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$
- *Double hashing*
 - $h(k, i) = (h_1(k) + i h_2(k)) \bmod m$

28

Ex - insert

- $m = 10$
- open addressing con linear probing.

Hash values sequence:

- $h(A)=5, h(B)=4, h(C)=9, h(D)=4, h(E)=8,$
 $h(F)=8, h(G)=10$

29

Ex - insert (II)

			A							5
		B	A							4
		B	A					C		9
		B	A	D				C		4
		B	A	D		E	C			8
		B	A	D		E	C	F		8
G		B	A	D		E	C	F		10

30



Ex - search (III)

search:

- D: ($h(D)=4$)
 - Read 4
 - Read 5
 - Read 6 \Rightarrow found
- G: ($h(G)=10$)
 - Read 10
 - Read 1 \Rightarrow found
- M: ($h(M)=4$)
 - Read 4,
 - Read 5,
 - Read 6,
 - Read 7, \Rightarrow not found

31



Delete

Very complex, because changes the rehash/collision sequence

In practice open hashing is used only if no delete

32



Complexity

With uniform hashing and linear probing:

- The number of probing trials is $1/(1-\alpha)$, and complexity is the same as for insert
- Complexity of search is

$$\frac{1}{\alpha} \ln \frac{1}{1-\alpha} + \frac{1}{\alpha}$$

33



Hash functions

34



Uniform hashing

Best hash functions do a uniform hashing: if keys have the same probability, also $h(k)$ should have equal probability

$$\sum_{k:h(k)=j} P(k) = \frac{1}{m}, \quad j = 0, 1, \dots, m-1$$

35



Keys are not uniform

However, keys often are not equally distributed (ex words in a language, ex names and surnames)

use all characters

amplify the differences

36



Keys as numbers

Usually keys are strings of characters

Easiest thing is to treat them as integers

- Ex: "abc" becomes
 $'a' * 256^2 + 'b' * 256 + 'c'$

However, with very long strings this is impractical, variants have to be used

In the following the key is an integer

37



Hash function = mod m

- k is an integer :
 - $h(k) = k \bmod m$
- Requires $m \geq n/\alpha$.
 - m size, n number of elements

38



Choice of m

- Avoid
 - Powers of 2
 - Division by m loses high bits of k
 - Powers of 10
 - Same as above, if k is decimal number
- Use
 - A prime number
 - Far from powers of 2

39



Ex

- $n = 2000$
- On average 3 comparisons in searches
- $m = 701$ is a prime, close to $2000/3$ but far from powers of 2
- $h(k) = k \bmod 701$

40



Hash function = multiply

- K integer:
 - A constant $0 < A < 1$
 - $\text{Frac}(x) = x - \lfloor x \rfloor$
 - $h(k) = \lfloor m \cdot \text{frac}(k \cdot A) \rfloor$
- $k \cdot A$ "shuffles" bits of k ,
- Multiplying by m expands $[0,1]$ in $[0,m]$

41



Choice of m and A

- M is not critical. Using a power of 2 simplifies the multiplication
- Best A depends on how keys are statistically distributed
- $A = (\sqrt{5} - 1) / 2 = 0.6180339887\dots$ Is a good choice

42