HASHING

- implementation of the hash table ADT
- used for performing insertions, deletions and finds in constant average time.

Hash Table data structure

- array of some fixed size containing the keys
- typically a key is a string with some associated value
- let **size** denote the size of the hash table
- each key is mapped into some number in the range 0 to **size**-1 and placed in the appropriate cell
- the mapping is called a hash function
- the hash function should be simple enough to compute and should ensure than any two distinct keys get different cells.

Problems:

- choosing a function → most of the time **hash(key) = key % size**
- what to do when two keys hash to the same value → collision
- how to choose the correct table size (we normally choose a prime number).

Example: Suppose that size = 7 and our hash function is hash(key) = key % size

key 10 will be stored in slot 3
key 7 will be stored in slot 0
key 14 will be stored in a lot 0 causing a collision.

Strategies for solving collisions

- use open hashing technique
  - use a list to store all items that hash to the same value
- use closed hashing
  - try alternate cells in the array during collision

The Hash Table Operations

- add, remove, contains

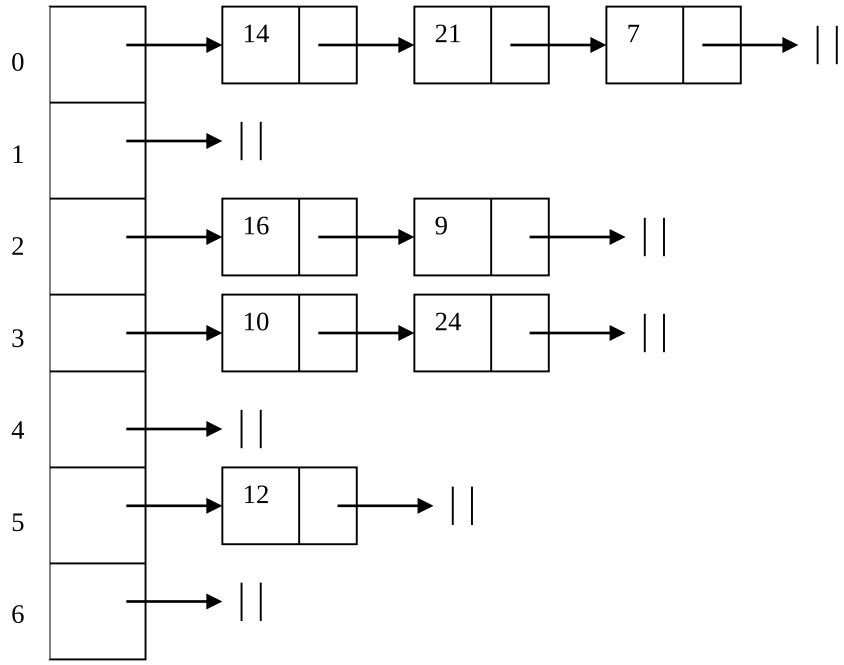
Closed Hashing Implementation of the Hash Table ADT

- in the implementation of closed hashing, keys that map to the same slot are stored in a linked-list.

To illustrate:

- Suppose that the hash table size is 7 and we use the function *hash(key) = key % size*.
- Suppose that we want to insert the following values in sequence

14 16 21 9 12 10 7 24



Closed Hashing

- if a collision occurs, alternate cells are tried until an empty cell is found
- formally, cells $h0(x)$, $h1(x)$, $h2(x)$ are tried where
  $hi(x) = (hash(x) + f(i)) \% size$ with $f(0) = 0$

Three common collision resolution strategies

Linear probing

- f is a linear function of i typically $f(i) = i$
- this amounts to trying cells sequentially with wrap-around in search of an empty cell.
- suffers from primary clustering

To illustrate:

- Suppose that the hash table size is 7 and we use the function *hash(key) = key % size*.
- Suppose that we want to insert the following values in sequence
- Suppose further that we want to use linear probing

14 16 21 9 7 13

$hi(x) = (hash(x) + i) \% size$

i=0	$((14 \% 7) + 0) \% 7 = 0$	okey, 14 is placed at slot 0
i=0	$((16 \% 7) + 0) \% 7 = 2$	okey, 16 is placed at slot 2
i=0	$((21 \% 7) + 0) \% 7 = 0$	collision
i=1	$((21 \% 7) + 1) \% 7 = 1$	okey, 21 is placed at slot 1
i=0	$((9 \% 7) + 0) \% 7 = 2$	collision
i=1	$((9 \% 7)+ 1) \% 7 = 3$	okey, 9 is placed at slot 3
i=0	$((7 \% 7) + 0) \% 7 = 0$	collision
i=1	$((7 \% 7) + 1) \% 7 = 1$	collision
i=2	$((7 \% 7) + 2) \% 7 = 2$	collision
i=3	$((7 \% 7) + 3) \% 7 = 3$	collision
i=4	$((7 \% 7) + 4) \% 7 = 4$	okey, 7 is placed in slot 4
i=0	$((2 \% 7) + 0) \% 7 = 2$	collision
i=1	$((2 \% 7) + 1) \% 7 = 3$	collision
i=2	$((2 \% 7) + 2) \% 7 = 4$	collision
i=3	$((2 \% 7) + 3) \% 7 = 5$	okey, 2 is placed in slot 5

Slot#	14	16	21	9	7	2
0	14	14	14	14	14	14
1			21	21	21	21
2		16	16	16	16	16
3				9	9	9
4					4	4
5						2
6						

Quadratic probing

- f is a quadratic function of i typically $f(i) = i^2$
- suffers from secondary clustering

To illustrate:

- Suppose that the hash table size is 7 and we use the function *hash(key) = key % size*.
- Suppose that we want to insert the following values in sequence
- Suppose further that we want to use linear probing

14 16 21 9 7 2

hi(x) = (hash(x) + i*i) % size

i=0	((14 % 7) + 0) % 7 = 0	okey, 14 is placed at slot 0
i=0	((16 % 7) + 0) % 7 = 2	okey, 16 is placed at slot 2
i=0	((21 % 7) + 0) % 7 = 0	collision
i=1	((21 % 7) + 1) % 7 = 1	okey, 21 is placed at slot 1
i=0	((9 % 7) + 0) % 7 = 2	collision
i=1	((9 % 7)+ 1) % 7 = 3	okey, 9 is placed at slot 3
i=0	((7 % 7) + 0) % 7 = 0	collision
i=1	((7 % 7) + 1) % 7 = 1	collision
i=2	((7 % 7) + 4) % 7 = 4	okey, 7 is placed in slot 4
i=0	((2 % 7) + 0) % 7 = 2	collision
i=1	((2 % 7) + 1) % 7 = 3	collision
i=2	((2 % 7) + 4) % 7 = 6	okey, 2 is placed in slot 6

Slot#	14	16	21	9	7	2
0	14	14	14	14	14	14
1			21	21	21	21
2		16	16	16	16	16
3				9	9	9
4					7	7
5						
6						2