Hash

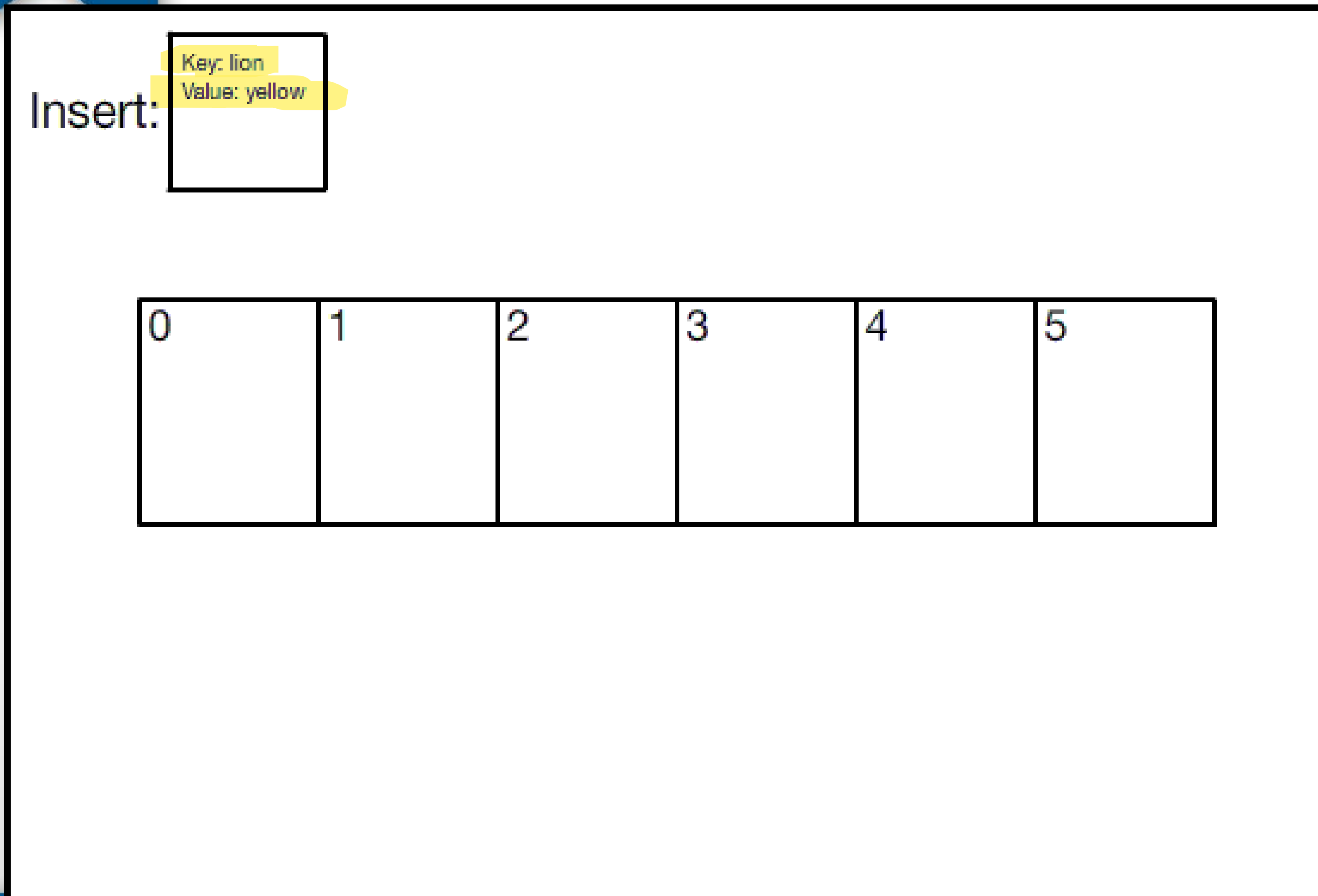
Dr. Abdallah Karakra

Computer Science Department

COMP242

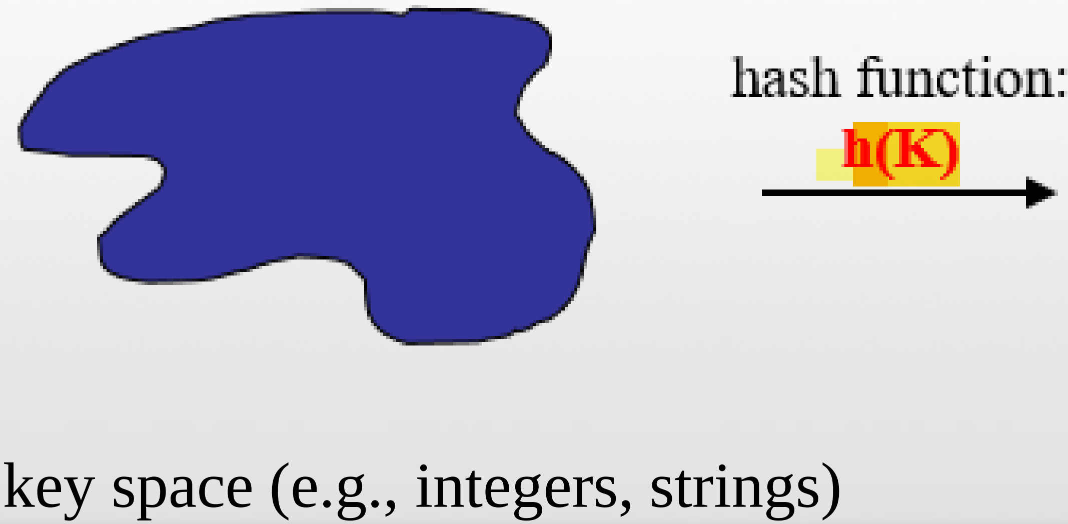
Monday, June 12, 2023

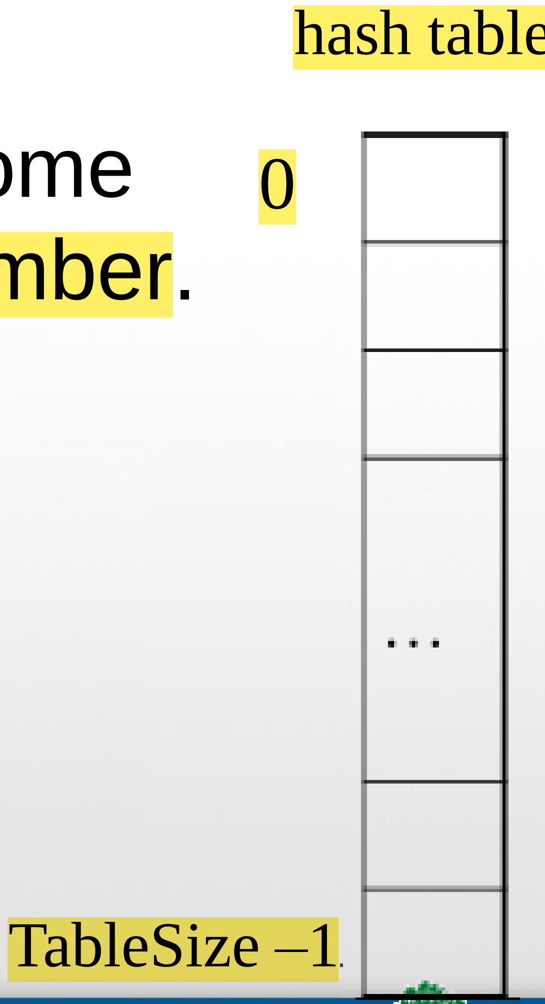
Motivation



Hash Tables

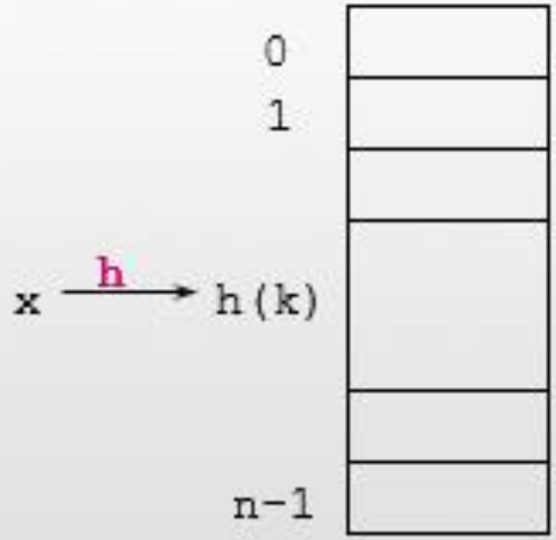
- Constant time accesses!
- A **hash table** is an array of some fixed size, usually a prime number.
- General idea:





Hash Functions

- Basic idea:
  - Don't use the data value directly.
  - Given an array of size n, use a *hash function*, h(k), which maps the given data record x to some (hopefully) unique index ("bucket") in the array.



Hashing

- Hash the key; this gives an index; use it to find the value stored in the table
- If this scheme worked, it would be O(1)
  - Great improvement over Log N.
- Main problems
  - Finding a good hash function
  - Collisions
  - Wasted space

Example

- key space = integers
- n = TableSize = 10

- **h**(K) = K mod 10

- **Insert**: 7, 18, 41, 94

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Example

- key space = integers
- TableSize = 6

- **h**(K) = K mod 6

- **Insert**: 7, 18, 41, 34

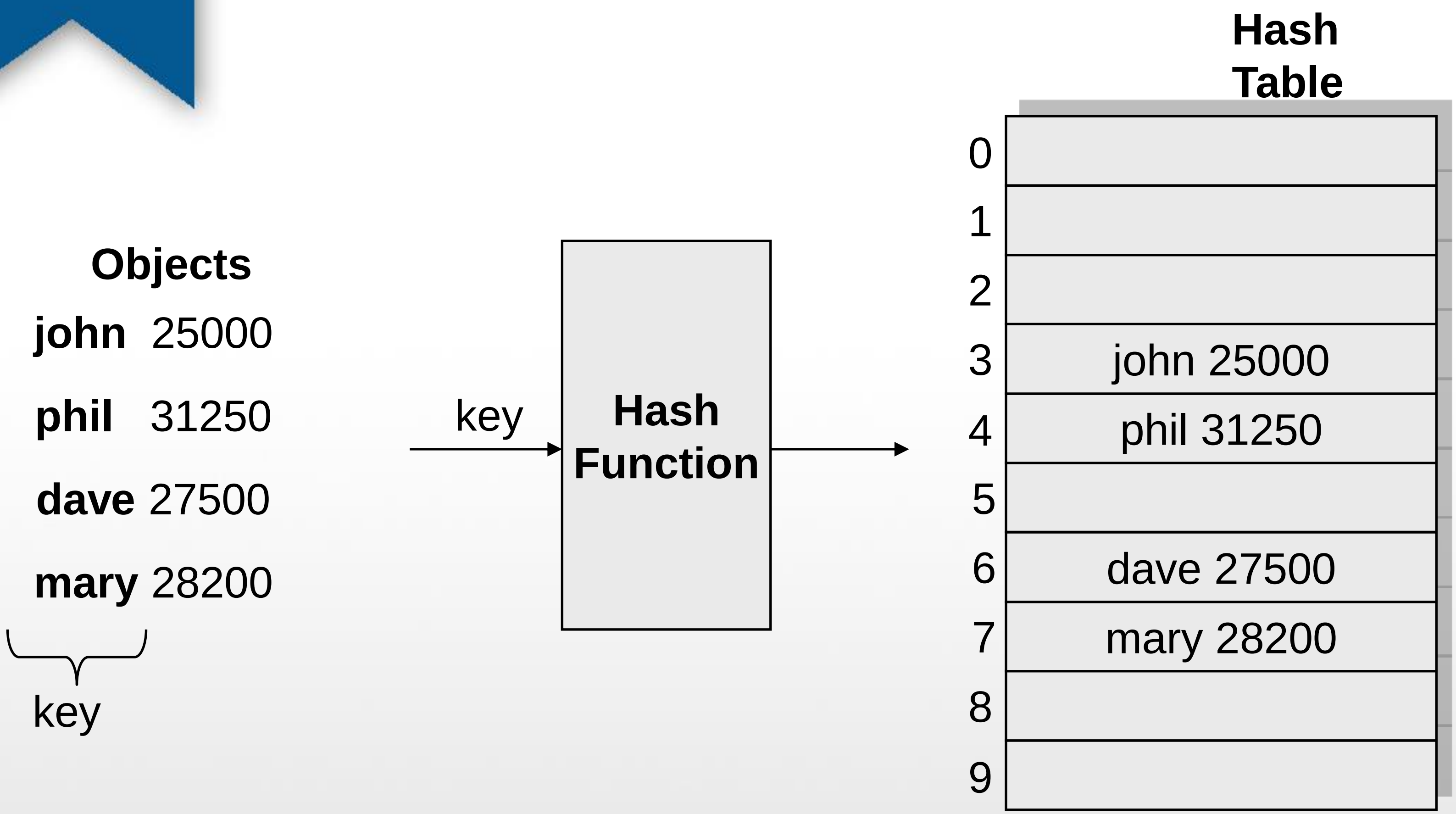
0	18
1	7
2	
3	
4	34
5	41

Hashing - hash function

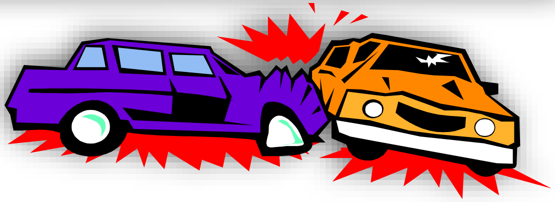
- Hash function
  - A mapping function that maps a key to a number in the range *0* to *TableSize -1*

```
int hashfunc(int integer_key)
{
    return integer_key % HASHTABLESIZE;
}
```

Recall


Hash Table
Objects
john 25000
phil 31250
dave 27500
mary 28200
key
key
Hash Function
0
1
2
3 john 25000
4 phil 31250
5
6 dave 27500
7 mary 28200
8
9

Collisions



- Collisions occur when multiple items are mapped to same cell
    - h(idNumber) = idNumber % n
    - h(678921) = 21
    - h(354521) = 21
- Issues
    - number of buckets vs number of data items
    - Choice of hash function
- With a bad choice of hash function we can have lots of collisions
- Even with a good choice of hash functions there may be some collisions

Collision Resolution

Two ways to resolve collisions:

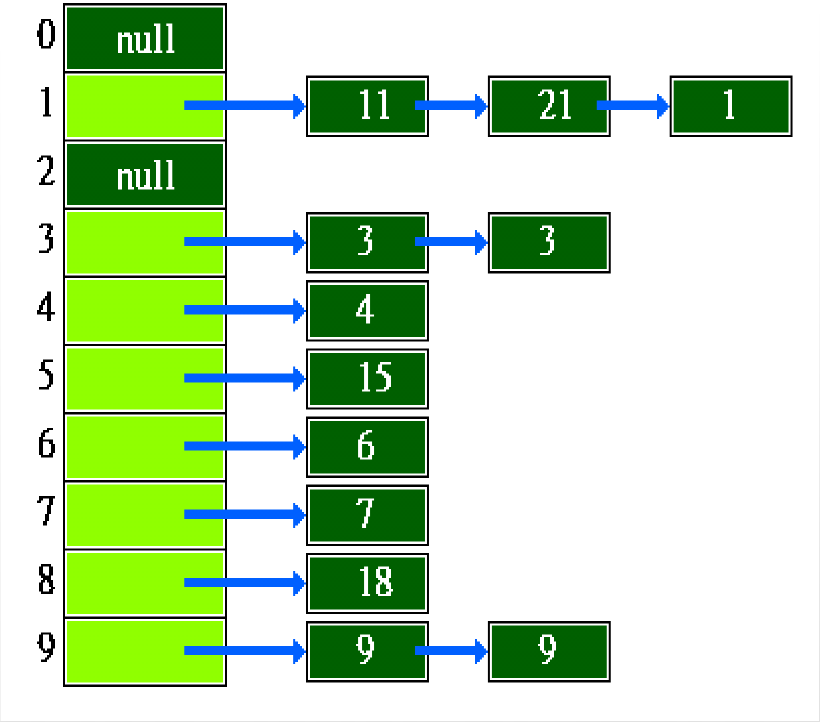
1. Open Hashing also called (Separate Chaining)
2. Closed Hashing also called Open Addressing (linear probing, quadratic probing, double hashing)

Open hashing (Separate Chaining)

- The idea is to keep a list of all elements that hash to the same value.
  - The array elements are pointers to the first nodes of the lists.
  - A new item is inserted to the end of the list.
- Advantages:
  - Better space utilization for large items.
  - Simple collision handling: searching linked list.
  - Overflow: we can store more items than the hash table size.

Hashing - separate chaining

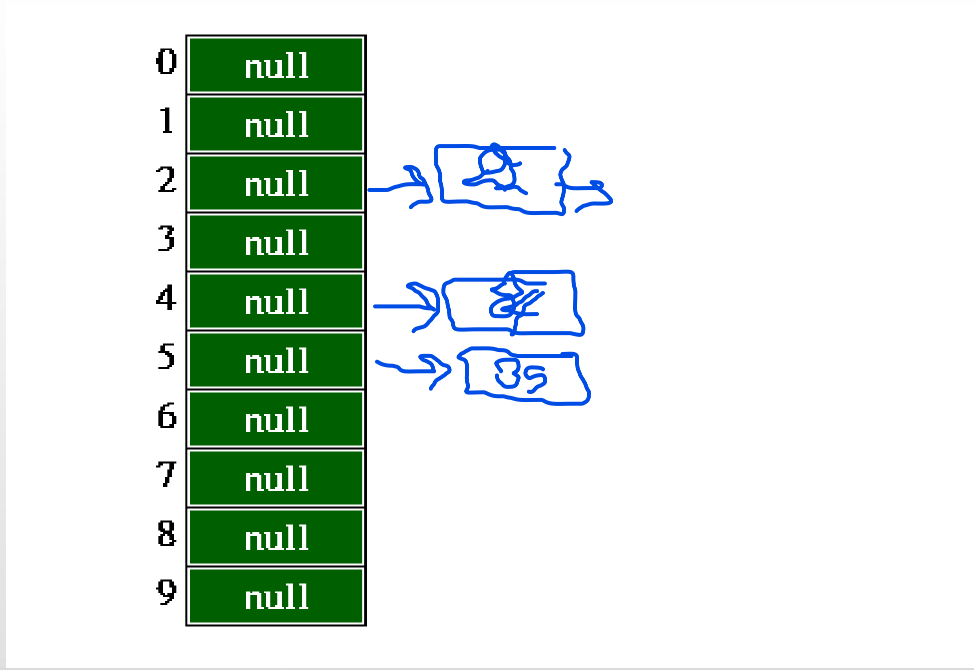
- If two keys map to same value, the elements are chained together.



Hashing - example

- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10

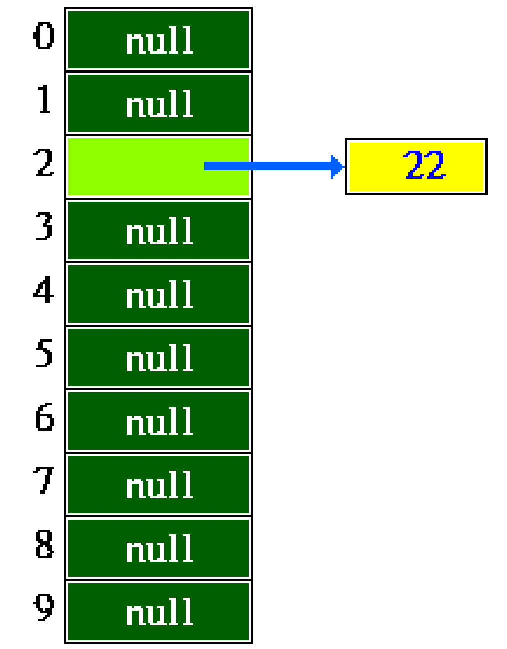
Initial hash table



Hashing - example

- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10

22 % 10 = 2

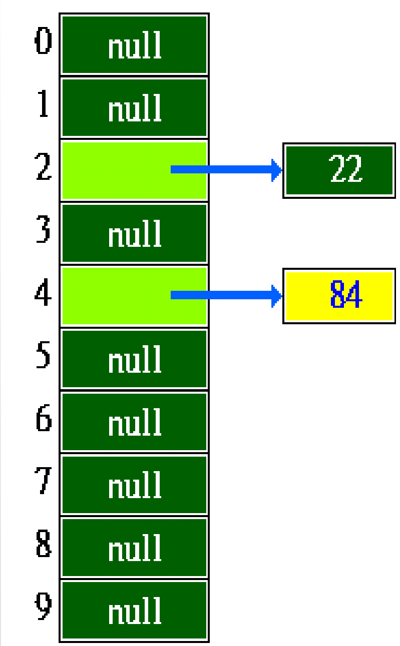


After insert 22

Hashing - example

- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10

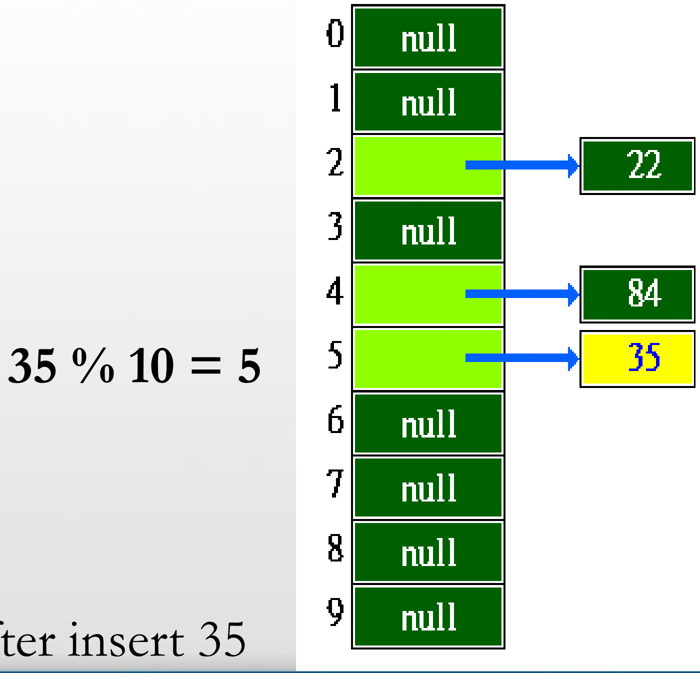
84 % 10 = 4



After insert 84

Hashing - example

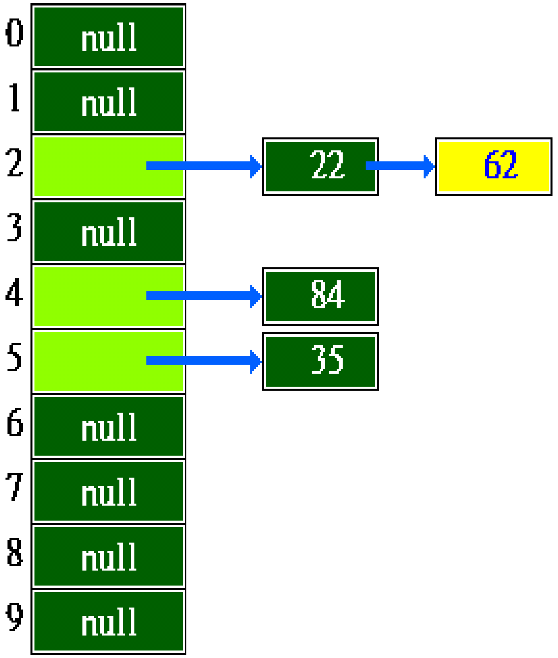
- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10



Hashing - example

- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10

62 % 10 = 2



After insert 62

Open Addressing

There are three common collision resolution strategies:

1. Linear Probing
2. Quadratic probing
3. Double hashing

Linear Probing

- In linear probing, collisions are resolved by sequentially scanning an array (with wraparound) until an empty cell is found.
  - i.e. *f* is a linear function of *i*, typically $f(i)= i$.
- Example:
  - Insert items with keys: 89, 18, 49, 58, 9 into an empty hash table.
  - Table size is 10.
  - Hash function is hash(x) = x mod 10.
    - $f(i) = i$;

Linear Probing

Linear probing hash table after each insertion

If there is a collision, the next available cell is used

h(k) = k mod 10

f(k)=(h(k)+i) **mod** table size, Where i>=0

```
hash ( 89, 10 ) = 9
hash ( 18, 10 ) = 8
hash ( 49, 10 ) = 9
hash ( 58, 10 ) = 8
hash (  9, 10 ) = 9
```

	After insert 89	After insert 18	After insert 49	After insert 58	After insert 9
0			49	49	49
1				58	58
2					9
3					
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

Hashing - Open addressing

- Linear probing

$$f_0(k) = (h(k) + 0) \bmod \mathit{TableSize}$$
$$f_1(k) = (h(k) + 1) \bmod \mathit{TableSize}$$
$$f_2(k) = (h(k) + 2) \bmod \mathit{TableSize}$$

...

$$f_i(k) = (h(k) + i\ ) \bmod \mathit{TableSize}$$

Hashing - Open addressing

- Linear probing

```
/* The h function */
int h(int i, int input)
{
  return (hash(input) + i)% HASHTABLESIZE;
}
```

Quadratic Probing

```
hash ( 89, 10 ) = 9
hash ( 18, 10 ) = 8
hash ( 49, 10 ) = 9
hash ( 58, 10 ) = 8
hash (  9, 10 ) = 9
```

A quadratic probing hash table after each insertion

	After insert 89	After insert 18	After insert 49	After insert 58	After insert 9
0			49	49	49
1					
2				58	58
3					9
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

$f(k)=(h(k)+ \boldsymbol{i}^2) \bmod \text{table size}$,
Where $i>=0$

Hashing - Open addressing

- Quadratic probing

$$f_0(k) = (h(k) + 0^2) \bmod \textit{TableSize},$$
$$f_1(k) = (h(k) + 1^2) \bmod \textit{TableSize},$$
$$f_2(k) = (h(k) + 2^2) \bmod \textit{TableSize},$$

.....

$$f_i(k) = (h(k) + i^2\ ) \bmod \textit{TableSize}$$

Hashing - Open addressing

- Quadratic probing

```
/* The h function */
int h(int i, int input)
{
   return (hash(input) + i * i) % HASHTABLESIZE;
}
```

Double Hashing

- Probe sequence:

$f_0(k) = (h1(k) + 0 * h2(k)) \bmod TableSize$
$f_1(k) = (h1(k) + 1 * h2(k)) \bmod TableSize$
$f_2(k) = (h1(k) + 2 * h2(k)) \bmod TableSize$
$f_3(k) = (h1(k) + 3 * h2(k)) \bmod TableSize$
....

$$f_i(k) = (h_1(k) + i * h_2(k)) \bmod TableSize$$

– A good choice for h2 is to choose a **prime R < TableSIze** and let $h_2(k) = R – (k \bmod R)$.

Double Hashing Example

h1(k) = k mod 7 and h2(k) = 5 – (k mod 5)

	76	93	40	47	10	55
0						
1				47	47	47
2		93	93	93	93	93
3					10	10
4						55
5			40	40	40	40
6	76	76	76	76	76	76

Rehashing

Idea:

Build a bigger hash table of approximately twice the size when λ (Load factor) exceeds a particular value

Why?

If ג gets large, number of probes increases. Running time of operations starts taking too long and insertions might fail

Solution

Rehashing with larger TableSize usually the first prime twice as large as the old hash table ➔ **NewhashTableSize= first prime>(2*OldTableSize)**

When to Rehash?

- When first insertion failed
- The table is half full → load factor 50%
- Load factor = 75%

Load factor

Let N = number of items to be stored

Load factor λ = N/TableSize

TableSize = 101 and N = 10, then λ = 0.1

Rehashing Example

- Elements 13, 15, 24 and 6 is inserted into an open addressing hash table of size 7
- h(k) = k mod 7
- Linear probing is used to resolve collisions

0	6
1	15
2	
3	24
4	
5	
6	13

Rehashing Example

- If 23 is inserted, the table is over 70 percent full.

0	6
1	15
2	23
3	24
4	
5	
6	13

A new table is created

17 is the first prime twice as large as the old one; so

$H_{new}(k) = k \bmod 17$

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

Hash Functions for String Keys

Option 1: If keys are strings, can get an integer by adding up ASCII values of characters in *key*

```
// Returns the index of
// the key in the table
int HashFunction(String key){
  int hashCode = 0;
  int len = key.length();

  for (int i=0; i<len; i++){
    hashCode += key.charAt(i);
  } //end-for

  return hashCode % tableSize;
} //end-HashFunction
```

$$h(k) = \left(\sum_{i=0}^{k-1} k_i \right) \bmod tableSize$$

Problems?

- Will map "abc" and "bac" to the same slot!
- Will map "eat" and "tea" to the same slot!

Hash Functions for String Keys

Option 2: Use first three letters of a key & multiplier

$$h(k) = \left(\sum_{i=0}^{2} k_i * 27^i \right) \bmod tableSize$$

Problem: Team && Tea

Option 3:

$$h(k) = \left(\sum_{i=0}^{k-1} k_i * 32^i \right) \bmod tableSize$$

Question?

“Success is the sum of small efforts, repeated day in and day out.”
Robert Collier