Hash tables

ESME AA1 - 2024

Naming convention

- Hash table
- Hash map
- Hash set
- Associative Array
- Dictionary

How to access a specific element in an array ?

What is the Complexity ?

Hash tables

- Access by a key instead of an index
- What is the complexity ?
  - O(1)

- General operations
  - Insert
  - Delete
  - Lookup
  - O(1)

Trade-off

- Not great at ordering

- Not always built-in the language (e.g. C)

What are hash table made of ?

- A Hash function
  - Returns an integer value aka hash code

- An Array capable of storing data

- First hash the data then store it in the array

What is a hash function ?

- Takes an arbitrary stream of bytes and returns a hash code
    - Hash value, digest, hashes

- All Molière
    - 11a3e229084349bc25d97e29393ced1d
- All Molière *
    - 0e8c8c427db2cb97f15a7371fe66c570
- Toto
    - 11a3e229084349bc25d97e29393ced1d

What makes a good hash function ?

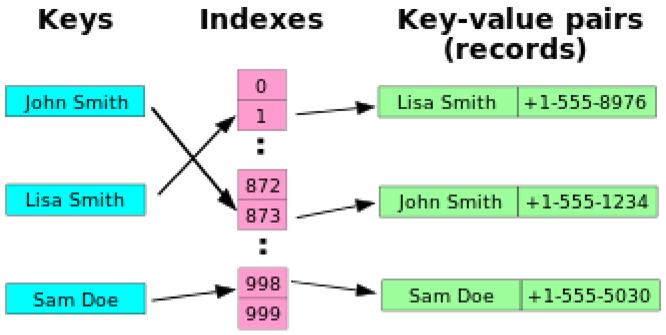
- Fast

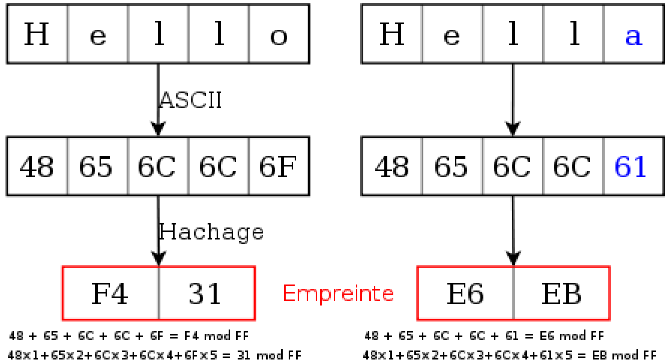
- No/Low collision

- Non-reversible

Sidenote : where are hash functions used ?

- EVERYWHERE !
- Hash table
- Integrity (Checksum)
- Store keys (`cat /etc/shadows`)
- `git`
- Bitcoin





H e l l o
ASCII
48 65 6C 6C 6F
Hachage
F4 31
Empreinte
H e l l a
48 65 6C 6C 61
E6 EB
48 + 65 + 6C + 6C + 6F = F4 mod FF
48x1+65x2+6Cx3+6Cx4+6Fx5 = 31 mod FF
48 + 65 + 6C + 6C + 61 = E6 mod FF
48x1+65x2+6Cx3+6Cx4+61x5 = EB mod FF

Sidenote : Popular hash algorithms

- SipHash
  - PEP 456
- CRC
- MD5
- SHA-1
- SHA-256
  - Recommended by National Institute of Standards and Technology
- Etc.

Implement my_hash function

- Input string
- Output integer
    - index of an array
- From any size to fix size

- Remember ASCII ?
- ord('A') #65

- modulo %
    - Fixed output
    - Fast to compute
    - Uniform function aka well distributed

Naïve implementation

Naïve implementation

```
def my_hash(s):
    hash_value = 0
    for char in s:
        hash_value += ord(char)
    return hash_value % 10
```

How does it work ?

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

```
hashtable = [""] * 10
```

How does it work ?

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

```
x = my_hash("John")
# x is now 9
```

How does it work ?

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	"John"

```
x = my_hash("John")
# x is now 9

hashtable[x] = "John";
```

How does it work ?

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	"John"

```
x = my_hash("Paul")
# x is now 4
```

How does it work ?

0	
1	
2	"Paul"
3	
4	
5	
6	
7	
8	
9	"John"

```
x = my_hash("Paul")
# x is now 2

hashtable[x] = "Paul";
```

How does it work ?

0	
1	"Ringo"
2	"Paul"
3	
4	
5	
6	
7	
8	
9	"John"

```
x = my_hash("Ringo")
# x is now 1

hashtable[x] = "Ringo";
```

How does it work ?

0	
1	"Ringo"
2	"Paul"
3	
4	
5	
6	
7	
8	
9	"John"

```
x = my_hash("George")
# x is now 1
# Collision !!!!
```

Resolve collisions with **Linear probing**

- if we have a collision
- place the data in the next index
- return to 0 if necessary
- until we find a free slot

- if we don’t find what we’re looking for in the first location
- at least 🤞 the element is somewhere nearby

Linear probing

0	
1	
2	
3	
4	
5	
6	“Bart”
7	
8	
9	

```
my_hash(“Bart”) # 6
```

Linear probing

0	
1	
2	
3	
4	
5	
6	“Bart”
7	“Lisa”
8	
9	

my_hash(“Lisa”) # 6

Linear probing

0	
1	
2	
3	
4	
5	
6	“Bart”
7	“Lisa”
8	“Homer”
9	

my_hash(“Homer”) # 7

Linear probing

0	
1	
2	
3	“Maggie”
4	
5	
6	“Bart”
7	“Lisa”
8	“Homer”
9	

my_hash(“Maggie”) # 3

Linear probing

0	
1	
2	
3	“Maggie”
4	
5	
6	“Bart”
7	“Lisa”
8	“Homer”
9	“Marge”

my_hash(“Marge”) # 3

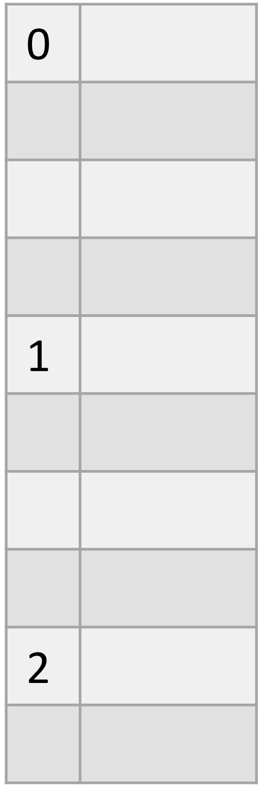
Problems with **Linear probing**

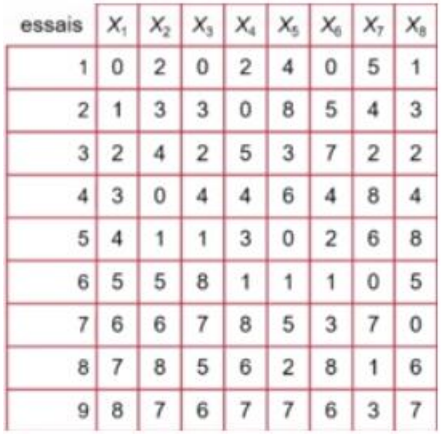
- Clustering
  - After a collision you augment the risk of collision and the “Cluster” will grow

- We can only store so much as location in the array

Mitigate the clustering

- Use other functions to calculate the next position
  - Quadratic probing (i^2)
  - Multiple calculation per match
- Pre-allocate more



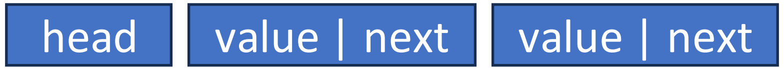


essais	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
1	0	2	0	2	4	0	5	1
2	1	3	3	0	8	5	4	3
3	2	4	2	5	3	7	2	2
4	3	0	4	4	6	4	8	4
5	4	1	1	3	0	2	6	8
6	5	5	8	1	1	1	0	5
7	6	6	7	8	5	3	7	0
8	7	8	5	6	2	8	1	6
9	8	7	6	7	7	6	3	7

Resolve collisions with **Chaining**

- What if we use the element of the array as a reference only ?

- Each element of the array is a linked list



head

value | next

value | next

- Therefore can hold multiple values

Resolve collisions with **Chaining**

- Eliminate clustering

- Insert in a linked-list is O(1)

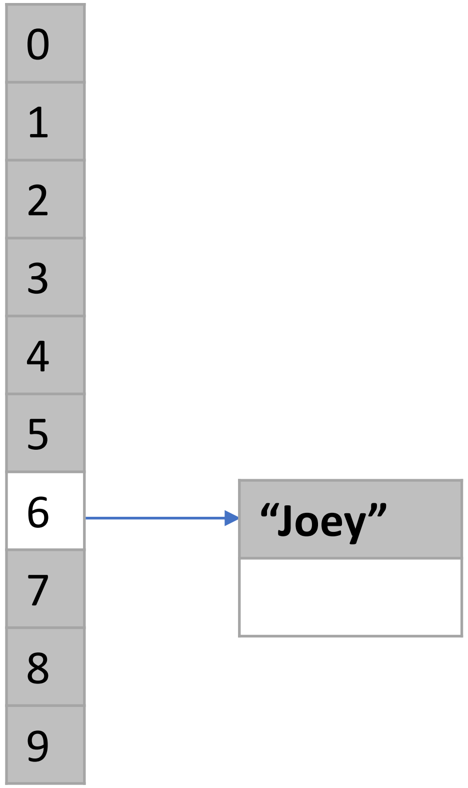
- Upon lookup we have to search through a small list 🤞

How does it work ?

0
1
2
3
4
5
6
7
8
9

```
node* hashtable[10]
```

How does it work ?

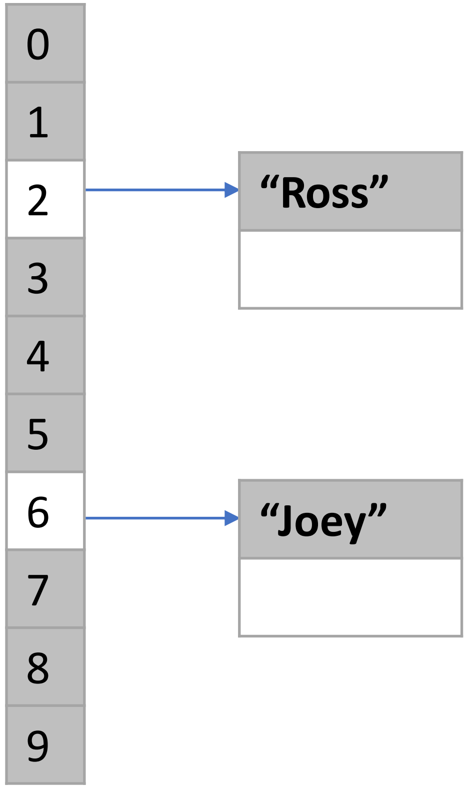


0
1
2
3
4
5
6
7
8
9

“Joey”

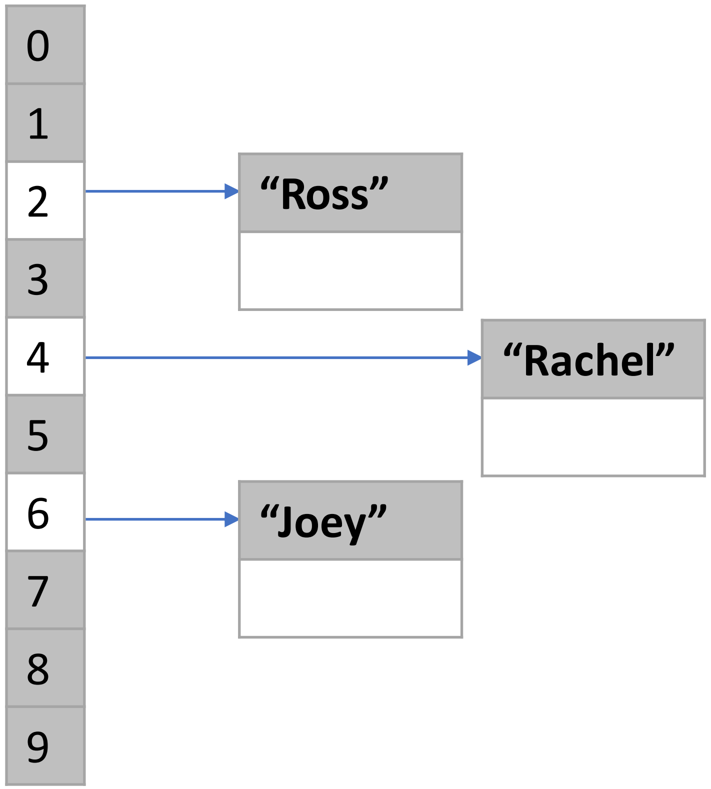
hash(“Joey”) # 6

How does it work ?



hash(“Ross”) # 2

How does it work ?



0

1

2

3

4

5

6

7

8

9

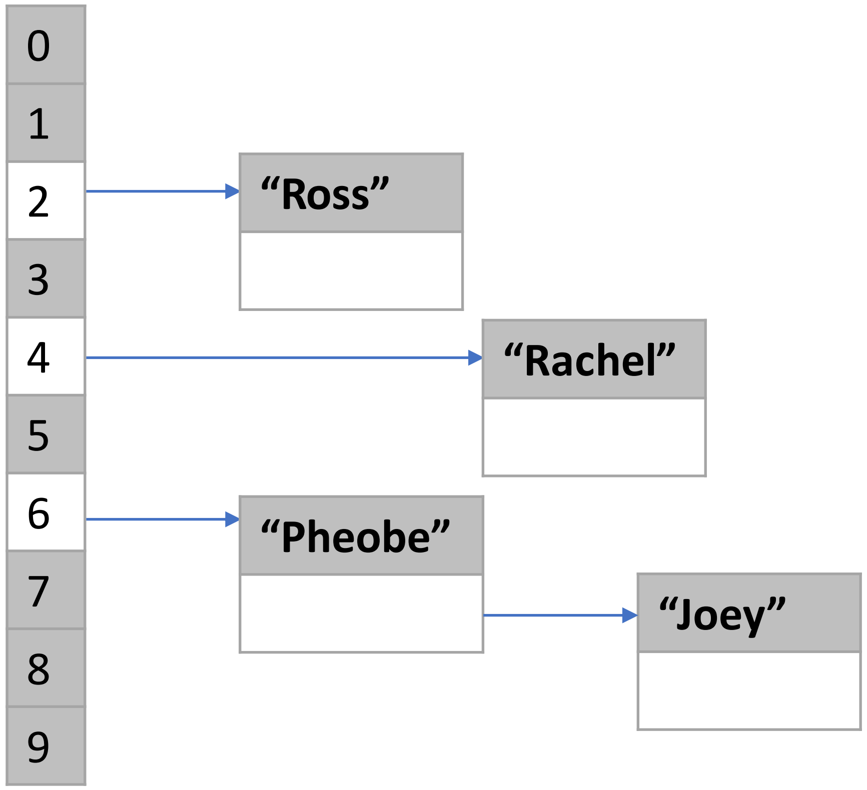
“Ross”

“Rachel”

“Joey”

hash(“Rachel”) # 4

How does it work ?



hash(“Phoebe”) # 6

Load Factor

- Load factor = number of key / size

- Closer to 1 means
  - Fuller table
  - Longer execution time

- 4/10 = 0.4

0	"George"
1	"Ringo"
2	"Paul"
3	
4	
5	
6	
7	
8	
9	"John"