CS2700 : Programming and data structures.

Hashing : another dictionary

- What is a hash table ADT?
- Hashing functions
- How to handle collisions!

Why do we need another dictionary?

- How efficient are operations with known dictionaries?
  (do we know any?)

- Many applications cannot afford ____ time supported by known dictionaries

  ↳ want O(1) for insert, delete, search.

Ordering amongst keys doesn't matter } not required to support min, max, range query

Hash table : an efficient look up table

Hashing : mapping large keys to a smaller fixed length key

↳ achieved via a hash function

Hash Table : data structure (typically an array)
that uses the hash function to map keys.
This enables efficient insert and search.

Comparison with known dictionaries

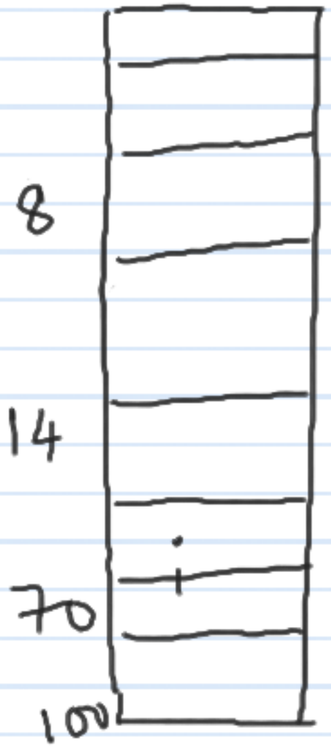
	Sorted Array	Balanced BST	Hash Tables
Insert	$O(n)$	$O(\log n)$	$O(1)^{**}$
Delete	$O(n)$		
Search	$O(\log n)$		

Hashing : a first example

Example : Assume student roll nos were

1, 2, 3 . . . 100

storing and retrieving quiz marks



Advantage : easy and efficient

Limitations :

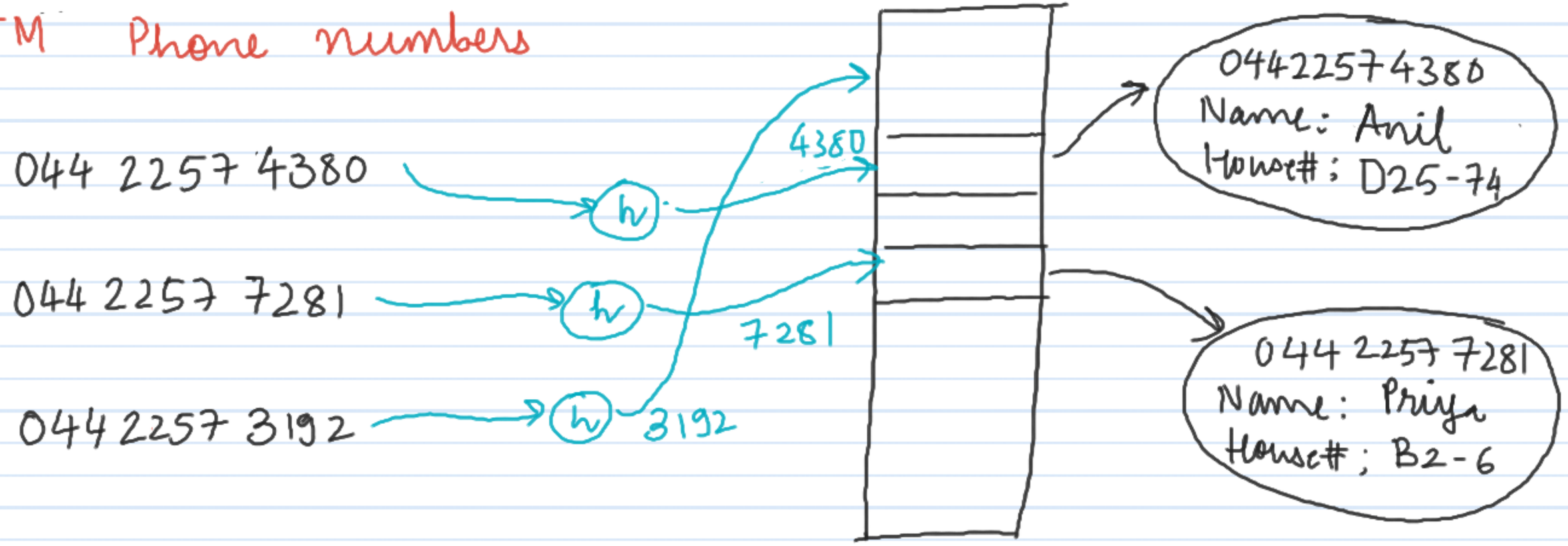
- non integer keys? CS20B013 ME20B013
- What if many roll #s are missing? (keys not dense)
- What if roll #s are 20001, 20002 . . . ?

Hashing : Ideas.

(1) Map large keys to small keys.

(2) Map non-integer keys to integer keys.

IITM Phone numbers



Hash Table operations

h: hash function | A: array

- insert (key, data)

```
A[h(key)] = data;
```

- delete (key)

```
A[h(key)] = NULL;
```

- find (key)

```
return A[h(key)]
```

Hash Table operations

h: hash function | A: array

- insert (key, data)

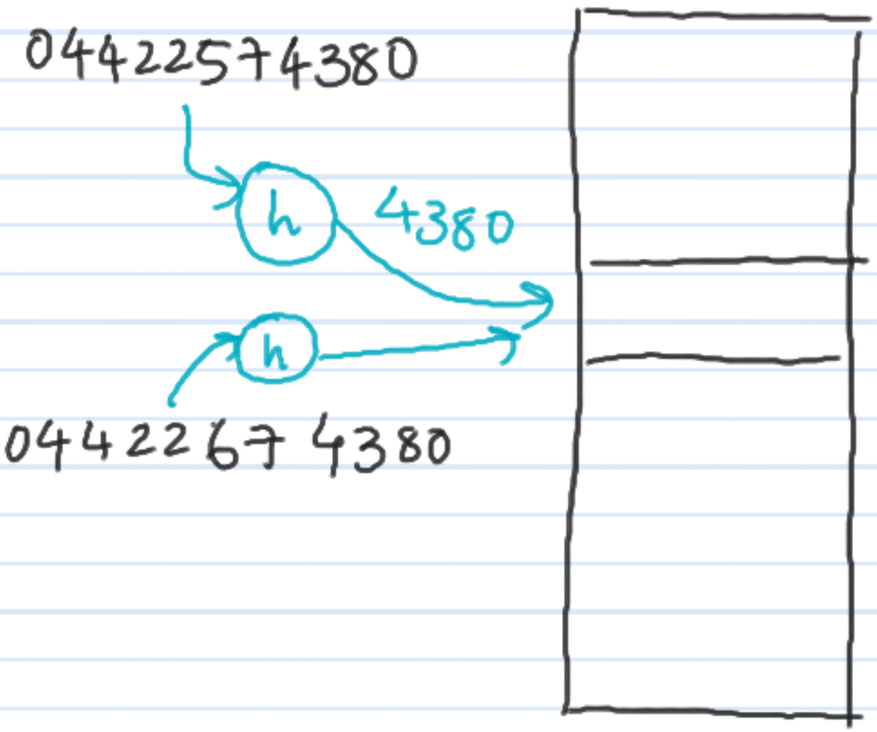
  A[h(key)] = data;

- delete (key)

  A[h(key)] = NULL;

- find (key)

  return A[h(key)]



This is a collision

2 or more keys have same hash value

Hashing : Two important aspects.

(1) Selecting a hash function

(2) Resolving collisions

Hash function : desired properties

(a) Fast to compute : nice to have O(1)

(b) Distributes keys evenly :

(c) Has less collisions

Bad hash functions : examples.

(1) IITM phone numbers

044 2257 4380 , 0442257 7281 , 04422573192

2257 2257 2257

---

(2) Say keys are integers

$h(\text{key}) = \text{key} \bmod \text{Tablesize}$

Suppose Table size is 10

There are 10 keys

10, 20, 30 . . 100

Hash Function : another (bad) example

Say Keys are strings containing 8 char

Table size is a prime number say 10007

$$h(\text{Key}) = \left[\sum_{i=1}^{8} \text{ASCII-val}(\text{Key}[i]) \right] \% \text{Tablesize}$$

Hash Function : another ~~(bad)~~ example

not so good

Say keys are strings containing 8 char

Table size is a prime number say 10007

$$h(\text{Key}) = [\text{ASCII val}[\text{key}[1]] + \text{ASCII Val}[\text{Key}[2]] \times c + \text{ASCII val}[\text{key}[3]] * c^2] \% \text{Table size}$$

- depends only on first 3 chars of key
- need not spread out.

Hash Function design : learnings so far

- select tablesize as a prime number
- good to have value dep on all char in string [makes is non const time computation]
- $\sum_{i=1}^{8} key[8-i-1] * c^i$ } a possible way to spread values
  - Use Horner's rule for faster computation
  - Select c to be a power of 2.