Hashing → Hashtheory

{ → Avoid flood fill

→ Emp and Manager }

Hashing

↳ enables searching in O(1)

{ Linear Search TC: O(N)

Binary Search TC: O(log N) }

{ Array }

↳ (data)

searching of data TC: O(1)

What if ?

{ All data is stored in HashMap and HashSet } data [ ] = { ✓5, ✓7, ✓3, ✓10, 12 }

Basic

present [ ] = {

0	1	2	3	4	5	6	7	8	9	10	11	12	…	500
F	F	F	T	F	T	F	T	F	F	T	F	T	false	T

}

search (10) → TC : O(1)

```
{ if (present[10] == T)
        print (yes)
  else
        print (No)
```

{ Memory (wasted) }

draw backs

data [ ] = { 5, 7, 3, 10, 12, 500 }

list of data

8

3

13

6

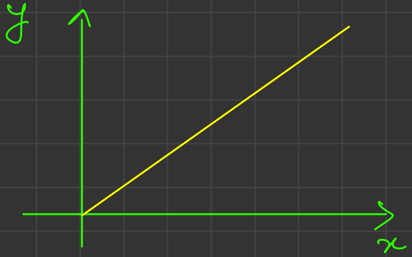
4

10

Key Space

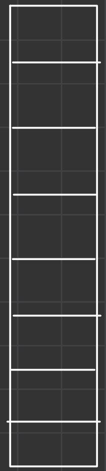
hashing fn

✓ $h(x) = x$ → one to one relation



drawback

{Memory wastage}



Hashtable

↳ linear data structure (similar to an array)

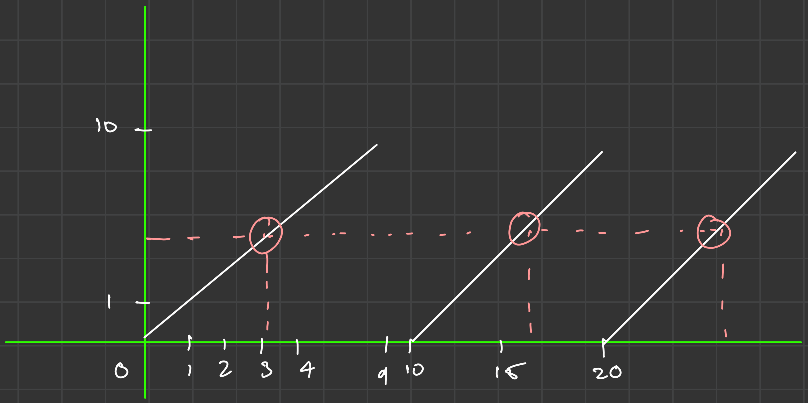
Many to One functions

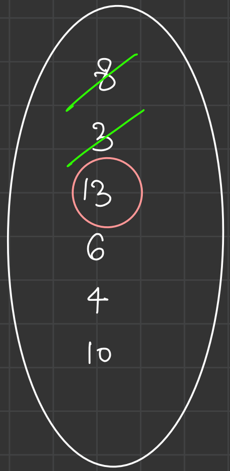
hash f^n

modulo

$$\boxed{h(x) = x \% 10}$$

$h(1) = 1 \% 10 = 1$





Key Space

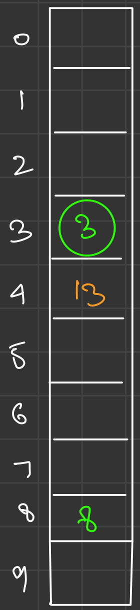
hashing fn

$$\boxed{h(x) = x \% 10}$$

$h(8) = 8\%10 = 8$

$h(3) = 3\%10 = 3$

$h(13) = 13\%10 = 3$



collision

$$\boxed{\text{load factor} = \text{Range} \times 0\ 75}$$ ✓

to decide size of hashfunction.

Methods to Remove Collision

✓ (a) Open Hashing Technique → Chaining

✓ (b) Closed Hashing Technique → Linear Probing, Quadratic Probing

Linear Probing

$$h'(x) = [h(x) + f(i)] \% \text{ size}$$

$$\left\{ \begin{array}{l} h(x) = x \% \text{ size} \\ f(i) = i \quad , \quad i \to 0, 1, 2 \ldots\ldots\ldots \end{array} \right.$$

Quadric Probing

$$\left\{\begin{array}{l} h'(x) = [h(x) + f(i)] \% \text{ size} \\ h(x) = x \% \text{ size} \\ f(i) = i^2 \end{array}\right.$$

$i \rightarrow 0, 1, 2 - - \_\_\_\_$

4 data structures

① HashMap
② HashSet } Hashing

TC: O(1) {Searching}

data is stored in random order irrespective of i/p

③ TreeMap
④ TreeSet } Red-Black Tree

TC: O(logN) {Searching}

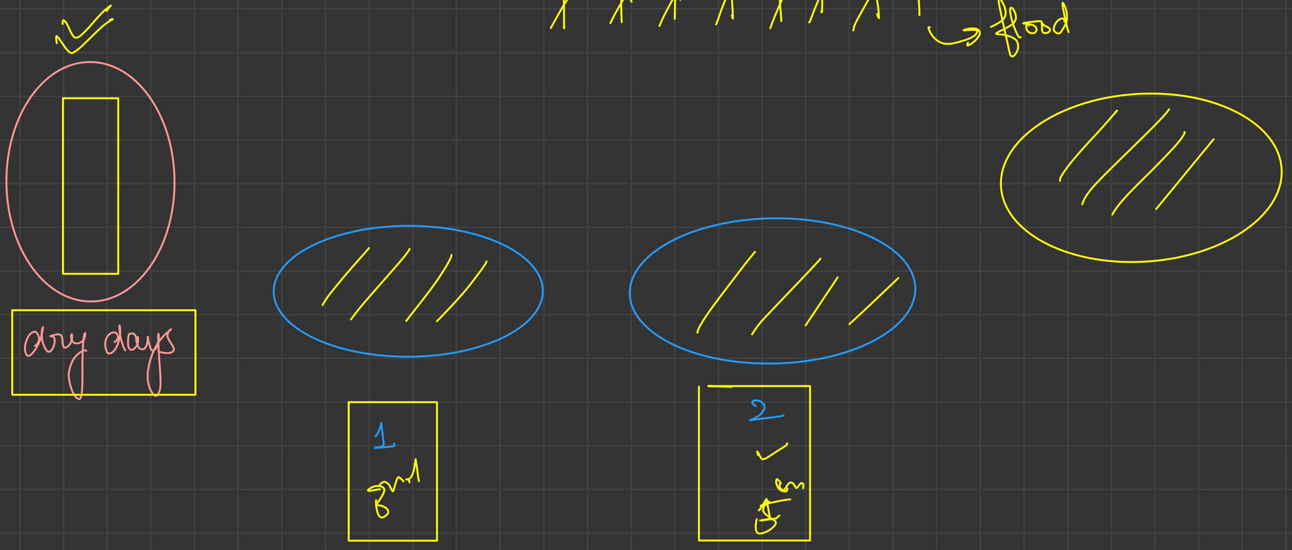
data is stored in an ordered way.
{increasing fashion}

Avoid flood fill

```
                  1       2
                0  1  2  3  4  5  6  7
int [] rains = {1, 2, 0, 1, 0, 2, 3, 3};
```

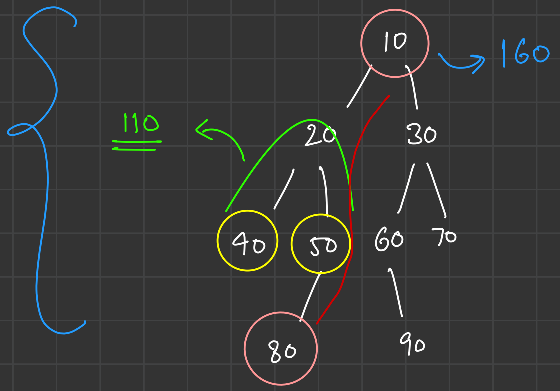
flood

[]



Max Path Sum

{ Path in a tree is defined as dist b/w any 2 Nodes of a tree



N - Nodes

{ $^NC_2$ paths in tree }

{ find that path, which has maxSum }

Brute force → get all path Sum
↳ and Maximize

```
class Pair {
    int maxPathSum;
    int Best Path;
}
```

$O(N)$