Hashing

Theory 1-4 Date 18/01/2020

1. HashSet
2. HashMap
3. LinkedHashSet
4. LinkedHashMap

(3, 4) } Preserves Order of insertion.

Self balancing BST vs. Hashing

[TreeSet, TreeMap] [HS, HM, LHS, LHM]

Self balancing BST	Hashing
1. Search, insert, delete → O(log n)	1. Search, Insert, delete → O(1)
2. Maintains SORTED order.	2. Can maintain insertion order. Can't maintain sorted order.
3. Has many more functions like ≥, ≤, etc.	3. Simple hashing libraries will NOT have these functions.

HashSet:
1) Add
2) remove
3) contains
4) size

How hashing works?

If the dataset were simple - say characters in alphabet, we could so something like the following to do operations in O(1) time.

bool set[26];
Set[x-'a'] = true or false gives present or not
97 99
a b c d ....

Now, what if the data is BIG?

Let's say ~~hashing~~ doing the above approach on a 10 digit number. Then, we have 10^{10} combinations. Such a big array isn't just economical.

So, use modular arithmatic.

(94923...44)% (prime-number) → Result is a small numb that can be stored at a particular known index to retrieve later. → A HASH FUNCTION DOES THIS.

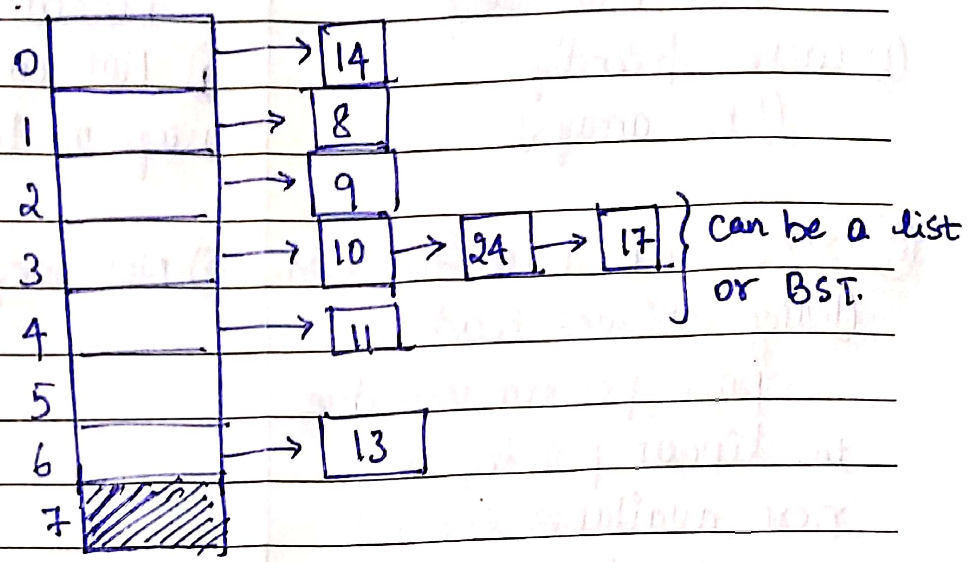
Handling hash collisions:-

1. Open Addressing
2. Linear chaining

Let's say hash-func(Key) = Key % 7
values: 10, 11, 9, 13, 8, 14, 24, 17

Date ....

Linear Chaining:



If the keys are non-uniformly distributed, then chain at an index grows in size and can lead to O(n) traversal.

If the chain of list is replaced by a BST, then it reduces to O(log n) from O(n).

Open Addressing: Initial length = length of i/p data

0	14
1	8
2	9
3	10
4	11
5	24
6	13
7	17

When a hash collision occurs, insert at the next onoccupied position.

Search: 24 % 7 = 3. Occupied by some other element. Start linear search till next empty space or the whole list is iterated.

Aha! Now if you delete 24, how'll you search for 13 & 17.

So, deleted elements are marked 'deleted'. They're not empt

Date ....

Open Addressing	Linear Chaining
① Cache friendly (like arrays)	① Not cache friendly. (Usage of linked/tree DS)
② Sensitive to hash function. (Many collisions lead to poor performance due to linear probing of next available space). → Quadratic probing.	② Not very sensitive
③ Space wastage (Deleted elements)	③ None such wastage
④ Used when frequency and number of key is known.	④ Useful when it's unknown how many and how frequently keys may be inserted or deleted.