Todays Content:

a. Comparison
b. DAT
c. Hashing
d. Collison Resolution techniques
    1. Separate chaining
e. list Implementation.
f. Tips for preparation

Q) Say we want to store 10^5 Telecommunications / Airtel / Jio / ....

phoneno / name / details : Inf Retrival is done based on phone.no

: operations on data

a) Insert ✓

b) deletion ✓

c) update ✓

d) Search phone no ✓

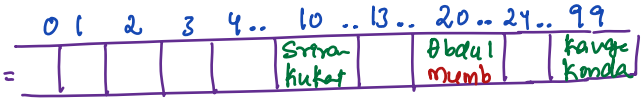
Take Multiple DS & Compare their Operations.

DS	insert()	delete()	search()	update()
unsorted list	O(1)	O(N)	O(N)	O(N)
sorted list	O(N)	O(N)	O(log N)	O(log N)
linked list	O(1) back/front	O(N)	O(N)	O(N)
BBST / Red black Tree / TreeMap / ordered_map	log N	log N	log N	log N

Assi: Assume all our phone numbers are at max 2 digits.

0, 1, 2, 3, .. 10, 11, 12, .. 99 ? *

Phone dat[100] =



Insert: 10, Sriram, kukat goto ind:10 : O(1)

Insert : 20, Abdul, Delhi goto ind:20 : O(1)

Insert : 99, Kavya, Konda goto ind : 99 : O(1)

update no: 20, mumbai : O(1)

Get det : 10 : Sriram & kukatpally : O(1)

*. Note: Our phone no: itself acts as index : DAT : Direct Acess Table

Q): Say we want to store 10^5 phone numbers

↓ A Single number = 10 digit = highest phone no: 9999999999

Say we want to use DAT to store numbers, phoneno ≈ index

Max no: 9999999999, Size of DAT $[10^{10}]$

If we use above DAT$[10^{10}]$?

Advantages:

Insert / update / access / delete : ≈ 1 = O(1) ← Keep TCs reduce space wastage?

Disadvantages:

huge space complexity
huge space wastage

Storing = 10^5 / Size = 10^{10}

% used = $[10^5/10^{10}] * 100 = \frac{10^7}{10^{10}} \approx$ 0.001% usage

Ass2: assume our phone no are at max 2 digits

DAT[10] :

Phone no: { 63 27 32 45 98 76 16 .. }

0	1	2	3	4	5	6	7	8	9
		32	63		45	76	27	98	
						16			

DAT :

phone no: 63 index 63 *

phone no(x) —map : x%10→ index: [0-9]

Map / hash function:

{ A function which maps (converts) data to smaller integer, which can be used as index in DAT }

Issues: phoneno { 0, 1, 2, .. 98, 99 } —map→ index {0, 1, .. 9}

1. When we map a higher range to a lower range, there is a chance that 2 different points can have same index / hashvalue : Collision

Collison Resolution Techniques:

a) Separte Chaining ✓

b) Open Addressing * {TODO}

a) linear probing b) Quadratic Probing c) Double Hashing

Separate Chaining: Java7

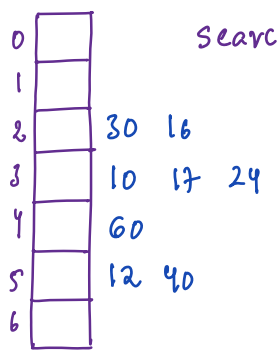
Case-I: Each ele in your DAT is a LL

Data: 12 10 30 16 17 24 40 60 DAT[7]

{2 digit phone no} ⟶ {0..6}

fun: n%7

DAT[7]



search(24): 24%7 = 3

Goto DAT[3]:

0	1	2
10	17	24

Iterate on DAT[3] & search:

Iterations: 3

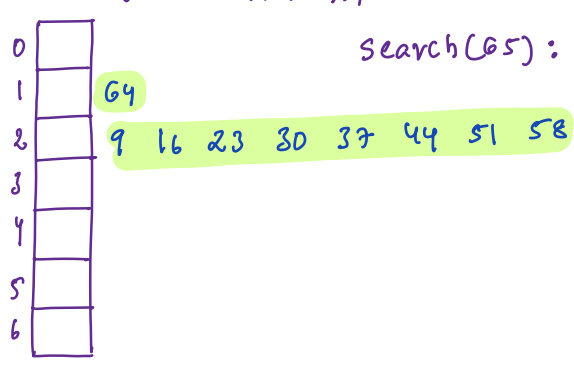
Issues in Above logic:

Data: 9 16 23 30 37 44 51 58 64 DAT[7]

Data ⟶ {0-6}

fun: n%7

DAT[7]



search(65): 65%7 = 2

Goto DAT[2]:

0	1	2	3	4	5	6	7
9	16	23	30	37	44	51	58

Iterate on DAT[2] & search:

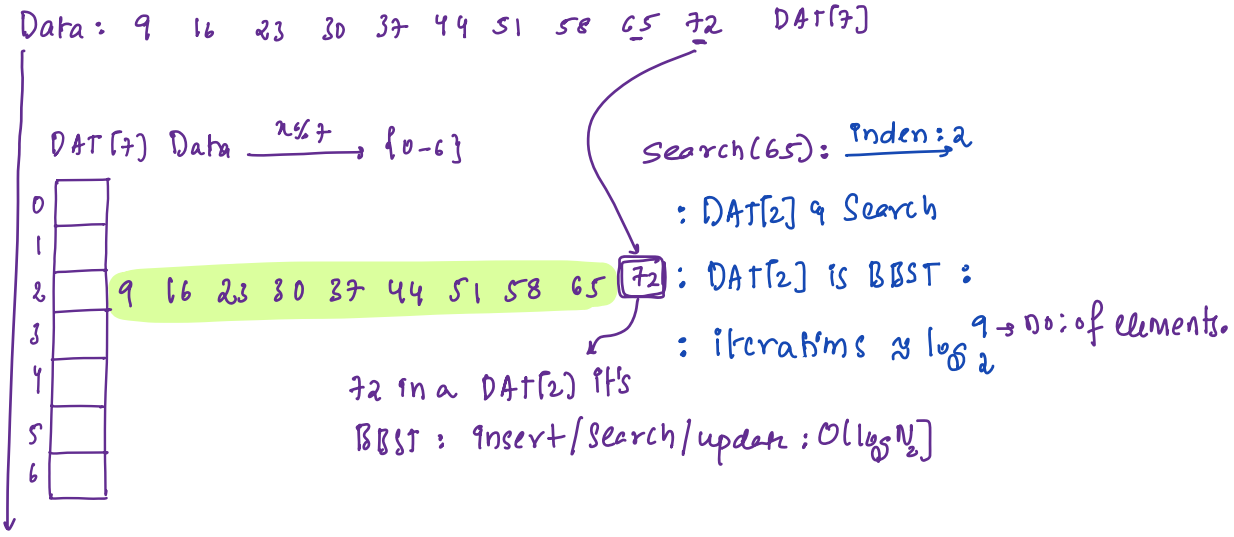
Iterations: 8

Worst Case: Insert all N phone no, all N elements can have same hashvalue / goto same index

insert(): O(1) search(): O(N) delete(): O(N) update(): O(N)

Separate Chaining:

Case: 2 Each ele in your DAT is a BBST / Treemap → Java 8

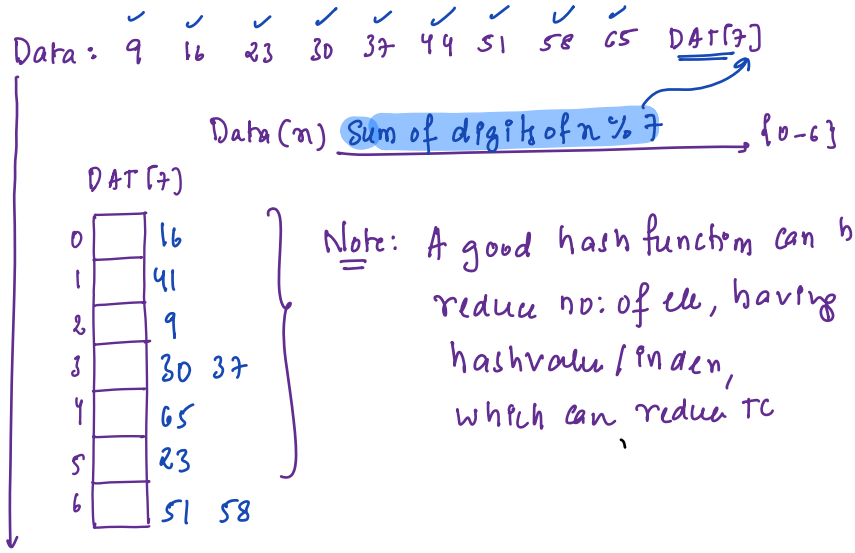


Worst Can: Insert N phonenos all N elements can have hashvalue & goto same index.

insert( ) : $(\log_2^N)$ search( ): $(\log_2^N)$ delete( ) : $O(\log_2^N)$ update( ) : $\log_2^N$

Why is above worst can Scenario Occurs?

a) Poor hash Function



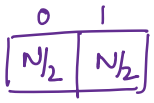
Note: A good hash function can help to reduce no: of ele, having same hashvalue / index, which can reduce TC

Below Calulate is a Rough estimate : TC depends Hash function ↑ as DAT size ↑

Store | DAT[] : Every Cell in a DAT is BBST

N Ele

DAT[2] : data(n) $\xrightarrow{map}$ {0..1}



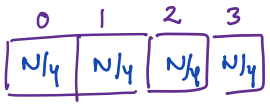
Ass: Hash Function distribute all numbers to all DAT, similarly / Good Hash Function

insert() : $\log_2^{N/2}$ search() : $\log_2^{N/2}$ delete() : $\log_2^{N/2}$ update() : $\log_2^{N/2}$

TC: $\log_2 \binom{N/2}{}$ ≈ $\log_2^N$ // $\log_c^{a/b} = \log_c^a - \log_c^b$ // $\log_2^{N/2} = \log_2^N - \log_2^2 = \log_2^N - 1$

N Ele

DAT[4] : data(n) $\xrightarrow{map}$ {0..3}



Ass: Hash Function distribute all numbers to all DAT, similarly / Good Hash Function

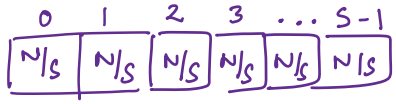
insert() : $\log_2^{N/4}$ search() : $\log_2^{N/4}$ delete() : $\log_2^{N/4}$ update() : $\log_2^{N/4}$

TC: $\log_2 \binom{N/4}{}$ ≈ $\log_2^N$ // $\log_c^{a/b} = \log_c^a - \log_c^b$ // $\log_2^{N/2} = \log_2^N - \log_2^4 = \log_2^N - 2$

N Ele

↳ // S is some random DAT size

DAT[S] : data(n) $\xrightarrow{map}$ {0..3}



Ass: Hash Function distribute all numbers to all DAT, similarly / Good Hash Function

insert() : $\log_2^{N/S}$ search() : $\log_2^{N/S}$ delete() : $\log_2^{N/S}$ update() : $\log_2^{N/S}$

2

Insert	DAT Size S	Size BBST : N/s	TC for each insert / search / delete() / update
N Elements	16	n/16	TC : $\log_2^{n/16} = \log_2^n - \log_2^{16} = \log_2^n - 4$
	N/1000	N/(N/1000) = 1,000	TC : $\log_2^{1000}$ ≈ O(10) ≈ O(1)
	N/100	N/(N/100) = 100	TC : $\log_2^{100}$ ≈ O(6) ≈ O(1)

avg TC for hashmap : Size ↑ ⇒ Hash Fun ↑

Storing String in Hashmap :

Data insert Hashmap : { Data $\xrightarrow[\text{function}]{\text{Hash}}$ index = i, goto Dat[i] & insert data }

For Number HashFunction

$x \longrightarrow h_1(x) = x \,\%\, \{\text{Size of DAT}\}$ = TC : O(1)

$\hookrightarrow h_2(x) = \text{sum of digits}(x) \,\%\, \{\text{Size of DAT}\}$: TC : O(1)

For String HashFunction

S = a b c d a (0 1 2 3 4)

$\hookrightarrow h_1(S) = \{\text{Sum of chars}\} \,\%\, \text{Size of DAT}$ = TC : O(l) : length of String.

$= \{a + b + c + d + a\} \,\%\, \text{Size of DAT}$

TC to calulate hash function for String is O(l)

For That reason TC to insert a String in Hash fun : O(l)

Dynamic Array : Size not fixed / Java : ArrayList<int> / Python : list<int> /

Implementation:

	insert()_front	insert()_back()	delete()	Access i^{th} ele
linked list :	O(1)	O(1) end pointer	O(N)	O(i) ≈ O(N) Iterate to get it
list DA :	O(N)	O(1)	O(N)	O(1)

Note: Dynamic array not implemented using linked

Implementation fixed sized arrays?

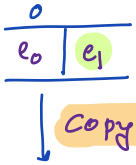
list<int> l {

int a[1] = de-allocate memory

: Insert 1 ele in O(1) : 1

copy 1 ele

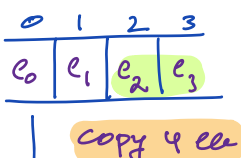
int b[2] = de allocate memory



: Insert 2^{nd} ele in O(1) : 1

copy 2 ele

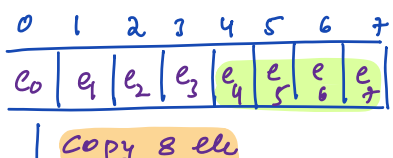
int c[4] = dellocate memory



Insert 3^{rd}/4^{th} ele in O(1) : 2

copy 4 ele

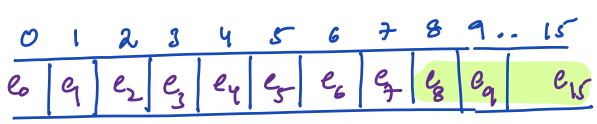
int d[8] = dellocate memory



Insert 5^{th} 6^{th} 7^{th} 8^{th} ele in O(1) : 4

copy 8 ele

int f[16] =



Insert 8 ele : O(1) : 8

}

Total TC for all 16 insertions = 31 = avg = O(2) ≈ O(1)

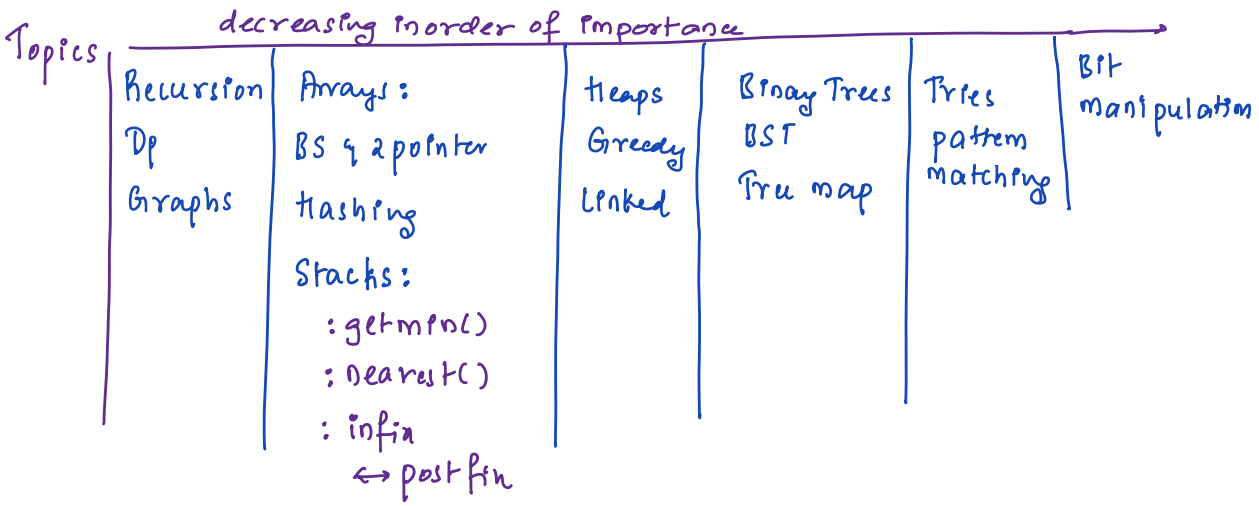
a) Copy : 15

b) Inserting : 16

Total iterations = 31

For N insertions ≈ Total iterations = 2N , avg TC for single insertion ≈ O(1)

Preparation:



How to prepare a topic?

a) Revise Notes

b) Solve Scaler ass/ hw

c) In a paper: make a list keywords & touch points req for that topic

How to prepare for a Company

a) Revise Entire Notes / Touch point for each topic

b) Leet-code: Previously asked Interview Questions.

↳ Use 3/4 weeks.

- Floyd Warshall : 45min ✓
- Graphs: 6 ✓
- Greedy ✓

Raise
Batch: Academy Jun 22 Intermediate Morning 1