Today's Content:

- { Band m head signifies my reaction after Doctor Strange }
- Treeset / Treemap ✓
- Family of Strings → { TODO }
- Basic idea of implementation of Hashmap / Hashset } *

Q1) Given an of Size N = 4oy → { ⟶ }

Given Q Queries

Type 1 → Given index, toggle 0 ⇄ 1 at that index

Type 2 → Given index, return closest index with 1 from that index : Note: If multiple answers return greater ind

Example:

N = 12 Q = 13

	0	1	2	3	4	5	6	7	8	9	10	11
ar[12] =	0	0	1	0	0	1	0	1	0	0	0	0
										↑		

Queries

type index:

2 4 : -1

1 3 :

1 10 :

2 6 : Le : 3, Ri : 10 : 3

2 2 : Le : _, Ri : 3 : 3

1 2 :

1 3 :

2 6 : Le : 2 Ri : 10 : 10

1 5 :

1 7 :

2 5 : ans = 5, at ar[5] = 1

1 10 :

2 9 : Le : 7, Ri : ⟶

ans = 7

Idea 1:

int ar[N] = {0}

Type: 1 → ind y

ar[y] = 1 - ar[y] }

Type: 2 → ind y

if (ar[y] == 1) {return y}

find closest index which contains 1 in left & Right

L & R → { ceil / floor } → { Binary Search }

↑

smallest index ≥ i

↓

largest index ≤ i

Return index closest to y

Note 1: If Both are at same distance: return R

Note 2: Edge Case If is possible, L, R might not be present for every y

Note 3: If Both L & R are not present

return -1

Constraints: ?

1 ≤ N ≤ 10^9 → 1MB, 10MB

1 ≤ Q ≤ 10^5

Idea 2: Instead of storing all indexes, store only those indexes which contain 1

Type 1: O(1)

Type 2: O(N)

Idea2: Maintain an array, which contain 1 in a Sorted Array ?

N = 20:

Type	Query
1	5
1	15
1	8
1	1
2	10 → to print * Elem ≤ 10 : 8 Elem > 10 : 15 } ans = 8
1	12
1	8 → Search 8 : → If not present : Insert 8 in Sorted arr[], & make entire arr[] Sorted
3	10 → to not print Elem ≤ 10 : 5 Elem > 10 : 12 } ans = 12 If present : Delete element 8 from Sorted arr[]
1	14 → Search 14 : * & Insert 14 in Sorted []
2	5 → Search 5 : 5 is ans

// ar[] : { index where data is 1, in sorted order }

↳ max ar[] size → Q

Type 1 : ele : → { A single type1 → O(Q) } → for single query

Search ele in ar[] : → TC : (log Q)

if present : Remove ——→ } Q

else : Insert ele in ar[], →

After insertion whole array again have to be sorted

Type 2 : ele : → { { A single type2 → log Q } }

Search ele is present ——→ { log Q }

→ first of ele in ar[] : { L } → { log Q }

→ Cell of col in ar[] : { R } → { log Q }

→ Return closest 1

overall TC : Q * { Type1 Q , Type2 $\log_2 Q$ }

TC : Q * Q

Data struc :

1) Keep Sorted order
2) Insert
3) Delete
4) Search
5) Floor
6) Ceil

All operations $\log_2^N$ → N is elements present

Java → TreeSet / TreeMap

C++ → set / map

Python → OrderedDict / ↳

JS → [redacted]

→ TreeSet → { BBST } → Balanced Binary Search Tree

↳ Height of Tree = $\log_2^N$

$\log_2^N$: Sea Del Ins Flr Cil

HashSet vs TreeSet (of Based on keys)

{ India, Austra, China, Canada, Japan } → { No order in order } → { Australia, Canada, China, India, Japan }

// given N & Q queries

TreeSet<int> Ts; → Ts.insert(-N), Ts.insert(2N)

for(i=0; i<Q; i++){

TC: Q * (log Q)

SC: O(Q)

// type, ind

Read type, ind

if(type==1){

if(Ts.search(ind)){ Ts.remove(ind) }

else{ Ts.insert(ind) }

}

else{

if(Ts.size()==2){return -1} // There is no 1's in Tree, because we add 2 distinct 1's

if(Ts.search(ind)){ print(ind) }

L = Ts.floor(ind) → L ind R available in TreeSet

R = Ts.ceil(ind)

if(ind-L <= R-ind){ print(L) } else{ print(R) } — lowerbound, upperbound

}

}

Note1: At max our ans can be N-1 ?

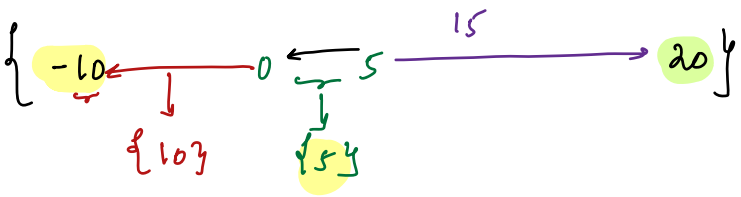
// for all the function please check actual syntax.

10:55pm break

Eg:

N = 10 : → max possible ans

Q = 2



Type:

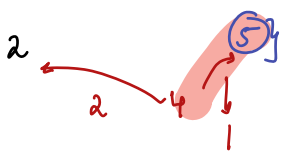
1 : 0

1 : 5

2 : 5 : ans = 5

2 : 0 : ans = 0

N = 5 : Trick : {-1,



1 : 2

2 : 4 : ans = 5

Q) Family of Strings : {TODO}

$S_0 = 0$

$S_1 = 0\ 0\ 1$

$S_2 = 0\ 0\ 1\ 0\ 1\ 1\ 0$

$S_3 = 0\ 0\ 1\ 0\ 1\ 1\ 0\ \ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1$

$S_i = S_{i-1} + 1 + \sim(S_{i-1})$

Given N, k find k^{th} character in S_N

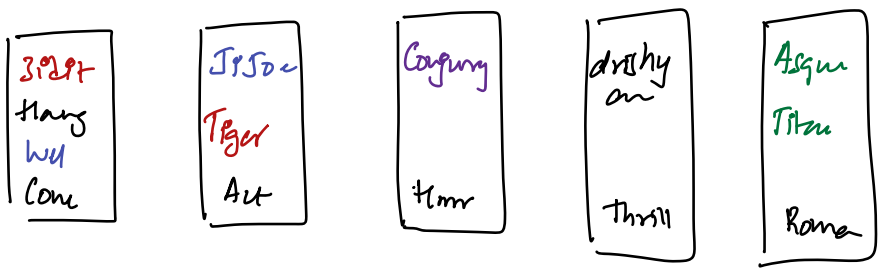
308) Implementation of Hashmap:

↳ {Bucket} → {Searching becomes slightly easier}

Eg: movies: → {3 Idiots} / Gajini / Akbar / Titanic / Jigsaw / Tiger

→ {Hagara} / Welcome / drishyam

geyser:

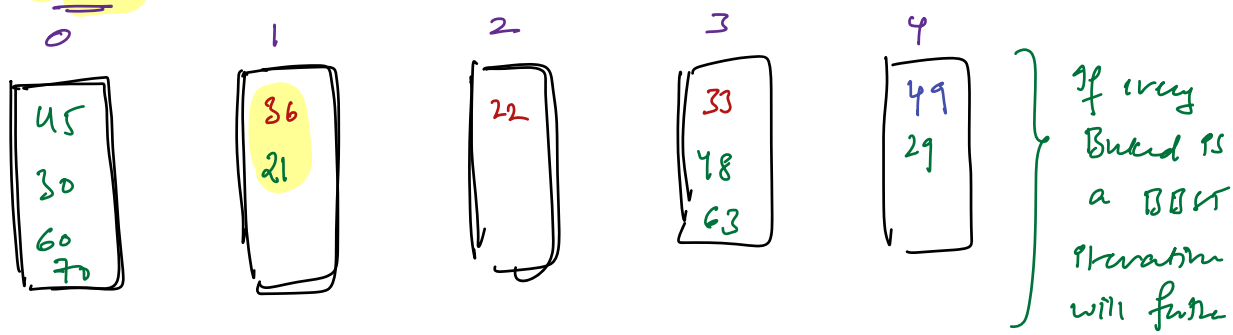


→ (Arabella) geyser → Hmm → X

Data:

36 49 33 21 29 45 30 48 60 63 70 22

⇒ {n % 5}



If every Bucket is a BST Operation will faster Reduce

→ {30}

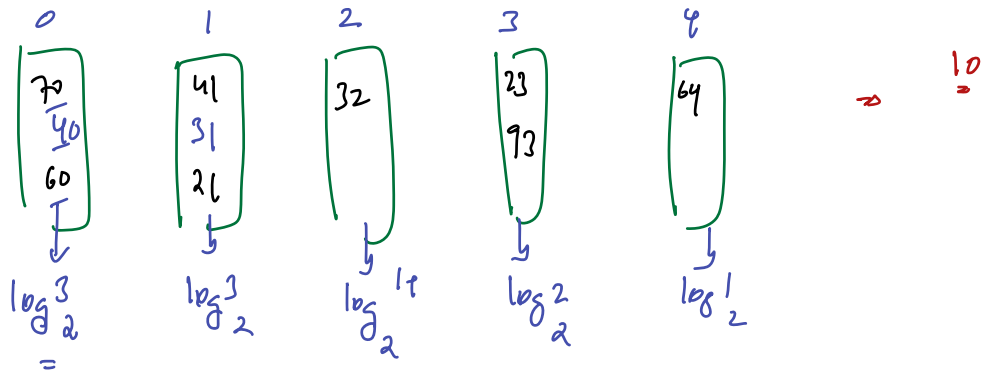
Data → {hashing} → Bucket no

⇒ {24}*

//

arr[10]: 31 40 60 21 23 70 64 32 41 73

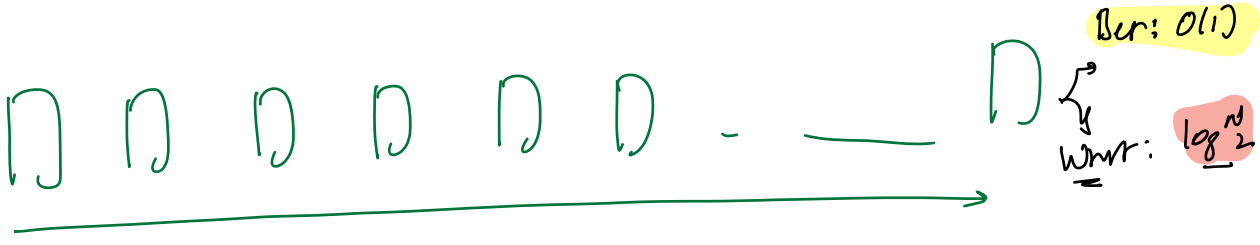
(x) % 5



// 100 Elements

100 Buckets:

N



Best → O(1)

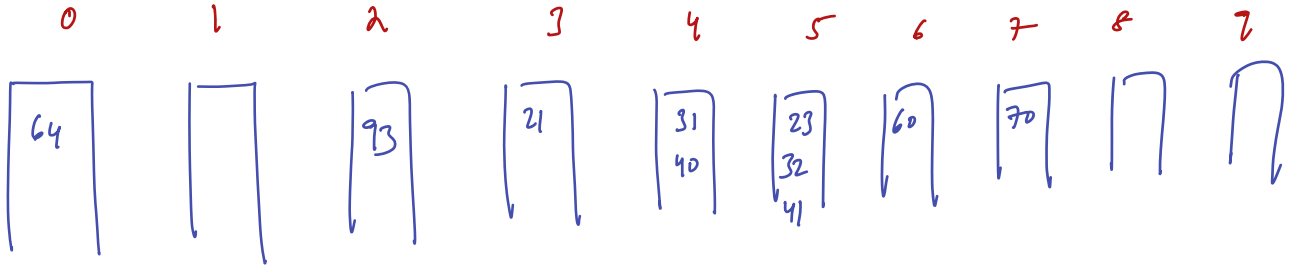
Avg → O(1)?

ar[10] : 31 40 60 21 23 70 64 32 41 93

→ (sum of digits) % 10 → { more randomness in function TC is reduced }

→ { more buckets less TC }

↳ if long function more spread out



//function → { main function }

data → { main function } → { To which bucket should data go }

// Data: → {strings}

Data: ana maa aan

Hash fun: (s[0]) % 5 → {weak}

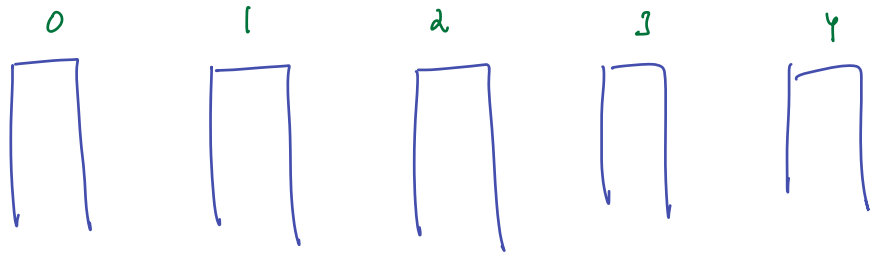
Hash fn: s[rand()] % 5 → amcat / amOat

Hash fun: $\left(\sum_{i=0}^{i<N} s[i]\right) \% 5$ → {Weak Hash Function}

Hash fn: $\left(\sum_{i=0}^{i<N} i * s[i]\right) \% 5$

$s_1 = a\ b\ c$ → $(a + 2b + 3c) \% 5$

$s_2 = b\ c\ b$ → $(b + 2c + 3b) \% c$

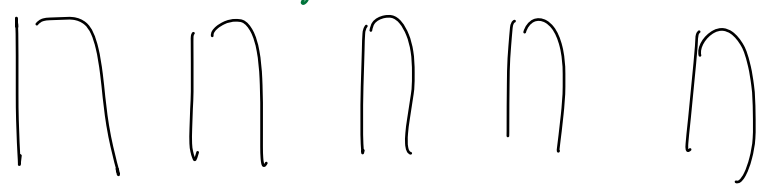


Hash fun c: $\left(\sum_{i=0}^{i<N} s[i] * p^i\right) \%$ {Number of Buckets}, p is a prime number of our choice, $p \geq 3$

Hf(amcat) → $(a3^0 + m3^1 + c3^2 + a3^3 + t3^4) \%$ {Number of Buc

String → {fun} → Bucket Number → O(N)

To enter in bucket: ≈ O(1)



string → {fun} → Bucket Number → goto that bucket & enter → O(1) ≈ O(N)

Hash value → O(M)

M length of string

put → {fun} → { O(1) } → ≈ O(1) } ≈ O(1)

// { // coming days?