→ Hashing Intro

1) Subarray with Sum = 0

2) Longest Subarray with Sum = 0

3) longest Sequence

4) Implementation of Hashmap.

(18) 4 Years:

Roll No: 1507__

↳ If we have roll → All our details?

↳

All roll no are unique.

↓

(key → Value)

For every key value is assigned

↙

RollNo → Details

(key - Value)

↳

keys are unique

→ Insert <key, value>

→ Search <key> : If key there are

→ Delete <key> → (Delete <key, value>)

→ Access <key> → value

→ (keys are unique)

→ O(1) → avg

↳

Hashmap – Java

unordered_map – C++

Dict {} – Python

→ Only Roll No : (Only keys)

Insert (keys)

Search (key)

Delete (keys)

→ O(1) → avg

HashSet – Java

unordered_set – C++

set. – python

keys → (Distinct)

Q8) Given an array elements, if subarray with sum = 0

Continuous part of an array

↳ bool / True / False

	0	1	2	3	4	5	6	7	8	9
ar[] :	2	2	1	-3	4	3	1	-2	-3	2
pf[] :	2	4	5	2	6	9	10	8	5	7

obs:

pf[2] = 5

pf[8] = 5

↳ sum of all [0 - 8]

[0 - 8] = [0 - 2] + sum[3 - 8]

5 = 5

sum[3 - 8] = 0

obs:

pf[0] = 2

pf[3] = 2

↳ sum of all [0 - 3]

[0 - 3] = [0 - 0] + [1 - 3]

2 = 2 + [1 - 3]

LOS)

Check all subarray sums. == 0

→ TC: $O(N^2) * O(N)$

→ TC: $O(N^3)$

All subarray sums using a carry forward technique?

[3 7] 7

5

sum[3 7] = sum + ar[7]

TC: $O(N^2)$

SC: $O(1)$

obs: If Elements are repeating in pf[], There exists a subarray with sum = 0

Ex:

	0	1	2	3
ar[4] :	3	-2	0	1
pf[4] :	3	1	1	2

→ a single element is also a subarray.

↳ values are repeating, subarray sum = 0

Ex2:

	0	1	2	3
ar[4] :	1	2	-3	2
pf[4] :	1	3	0	2

→ subarray sum = 0

Failing

↳ values are not repeating,, (No subarray)

Edge Case: If there is a '0' in a pf[] array then there exists a subarray with sum = 0.

Idea:

→ There is a subarray with sum = 0

1) Calculate pf[] : O(N) ✓

2) If '0' is in pf[] : ✓

3) If elements in pf[] are repeating : O(N) ✓

TC: O(N) SC: O(N)

20) Given N Array check if repeation is there? [YES]

Ex: ar[4] : { 3, 1, 2, 5 } → {no repeation}

Ex2: ar[6] : { 3, 2, 1, 4, 5, 1 } → YES repeation

//

1) Sort & check : O(NlogN)

2) Set<Int> hs;

{ 3, 2, (1) }
{ 4, 5, }

hs.size() == 5 < 6 (N)

There is a repeation ✓

Pseudocode

```
HashSet<int> hs;
for(i=0; i<N; i++){
    hs.insert(ar[i])
}
if ( hs.size() != N){
    return True
}
else{
    return false
}
```

TC:
N * O(1)
TC: O(N)
SC: O(N)

Small modification

Before inserting ar[i] check if already present in HashSet

28) Given N Array elements, find length of longer subarray when sum = 0

	0	1	2	(3)	4	5	6	7	8	(9)	10	11	12
ar[] :	3	3	4	-5	-2	2	1	-3	3	-1	5	-4	-1
pf[] :	3	6	10	5	3	5	6	3	6	5	10	6	5

Obs:

pf[1] = 6
pf[6] = 6
len = 6 - 1 = 5

[0 - 6] = [0 - 1] + [2 - 6]

[0 - 6] : 6
[0 - 1] : 6

[2 - 6] = 0

length = (6 - 2 + 1) = 5

Idea: for every element pf[]

Store 1st occurrence index of

last occurrence index of

subarray & get max

idx1	idx2	len
<3, 0>	<3, 7>	7
<6, 1>	<6, 11>	10
<10, 2>	<10, 10>	8
<5, 3>	<5, 12>	9

ans = 10

	0	1	2	3	4	5	6	7	8	9	10	11	12
pf[] :	3	6	10	5	3	5	6	3	6	5	10	6	5

HM { first occurrence index }

{ <3, 0>, <6, 1>, <10, 2>, <5, 3> }

max of len

Element	index	first 0	len	ans
3	4	0	4	4
5	5	3	2	4
6	6	1	5	5
3	7	0	7	7
6	8	1	7	7
5	9	3	6	7
10	10	2	8	8
6	11	1	10	10
5	12	3	9	10

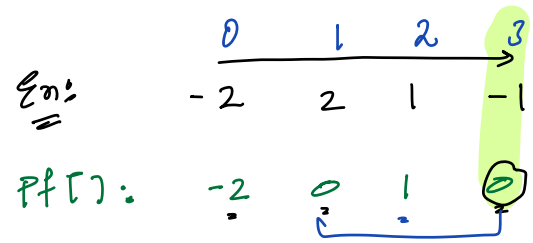
Ideas:

1) Create pf[]

2) Hashmap < Int, Int > hm

```
for(i=0; i<N; i++){
    if( pf[i] == 0) { ans = max(ans, i+1); continue; }   // resolve edge case
    if( pf[i] is in hm){
        // pf[i] getting repeated - (first occurrence)
        ans = max( i - hm[pf[i]], ans)
    }
    else{ // insert guys
        hm.insert( pf[i], i)
    }
}
return ans;
```

{ 10 is final ans }



Hm:

{ <-2, 0>
<0, 2>
<1, 2> }

Element	Index	first occ	len	ans
0	3	1	2	2

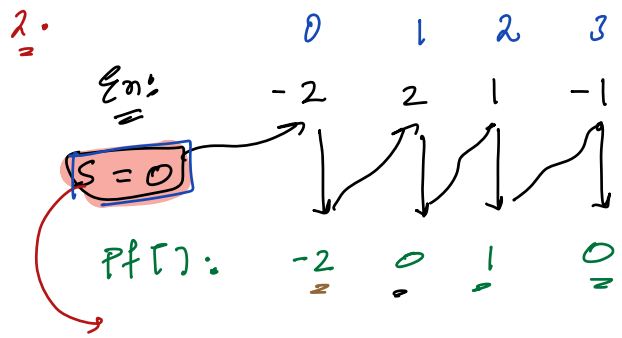
1. How to resolve this case

1) When Pf[i] = 0,

len

Subarray sum[0 i] = 0 = (Pf+1)

2.



first occurrence of 0 or -1?

Hm:

{ <0, -1>
<-2, 0>
<1, 2> }

Element	Index	first occ	len	ans
0	1	-1	2	2
0	3	-1	4	4

- ans = 0

1) Create pf[]

2) HashMap<Int, Int> hm

hm.insert({0, -1})

Topic: Calculate pf[] value on the flyo

```
for(i=0; i<N; i++){
    if( pf[i] is in hm ){
        // pf is getting repeated - (first occurrence)
        ans = max( i - hm[pf[i]], ans)
    }
    else{ // insert guys
        hm.insert( pf[i], i)
    }
}
return ans;
```

TC: O(N) SC: O(N)

Eg:

II: OS PPT : Implementation of HashMap

ar[] = 0

pf[] = 0

hm:

{0, -1} {0 - (-1) = 1}

Implementation of Hashmap } → { 4 Students → Arun, GS, Sargc } Implementation of Hashmap ?

→ Say we have to N Numbers [$1 <= ar[i] <= 10^7$]

Insert / delete / Search

	Insert	delete	Search
Unsorted array :	O(1)	O(N)	O(N)
sorted array :	O(N)	O(N)	O(logN)
Linked list :	O(1)	O(N)	O(N)
Balanced Binary Search Tree (BBST) :	O(logN)	O(logN)	O(logN)

sorted array → sorted

BBST → 1) AVL Trees 2) Red - Black Tree

: arr [10^7] : O(1) O(1) O(1)

arr [10^7] → DAT: Direct Access Table

→ Issue

1) There is a ton of memory wastage
2) We cannot always declare huge space.

{remove disadvantage}

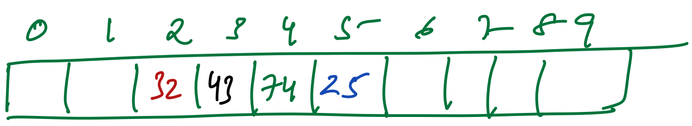
Data = { 25, 32, 43, 74, 12, 77, 19 }

⇒ data ⇒ { 0 - 9 }

⇒ ar[i] % 5 ⇒ { data ⇒ map it to a smaller index } → hash table size

Hash function

int ar[10] =

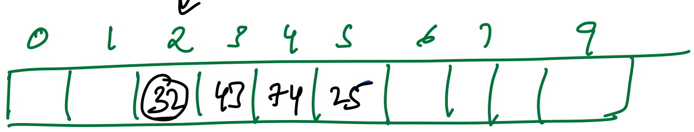


Hash table

HashTable size

Data = { 25, 32, 43, 74, 12, 77, 19 }

int ar[10] =



D_1

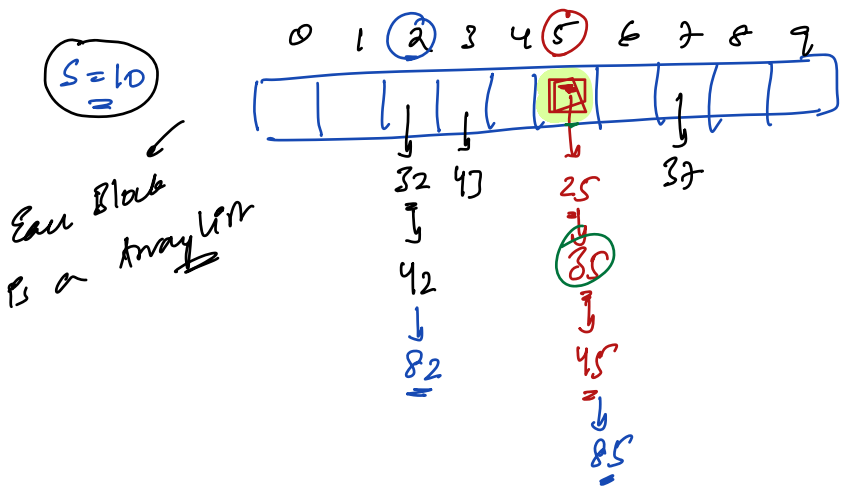
D_2

(Hash Function) : Same Index Value ⇒ Called Collision

Collision Resolving Techniques

- Separate chaining
  - → LinkedList
  - → BBST (Balanced Binary Search Tree)
- open addressing
  - → Linear probing
  - → Quadratic probing
  - → Double hashing

data = { 25, 32, 47, 35, 45, 37, 42, 85, 823 }



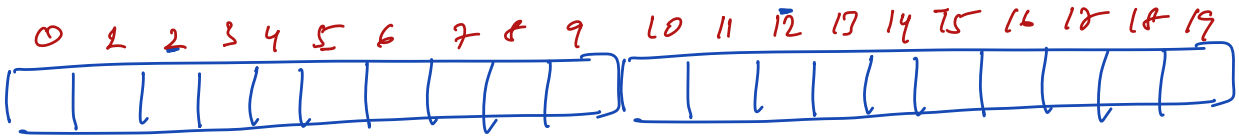
Insert	Search	Deletion
O(1)	O(P)	O(P)

Load factor increases → if S ↑ → P decreases

P = max Elements in a chain

data = {25, 32, 43, 35, 45, 37, 42, 85, 823}

S = 20 → data % (20) 35 % 20 = 15



Every Block is BST: S = 20

Max Elements in a chain → P

→ If S ↑, P ↓

If S = 10^4, P ↓

Time for explanation

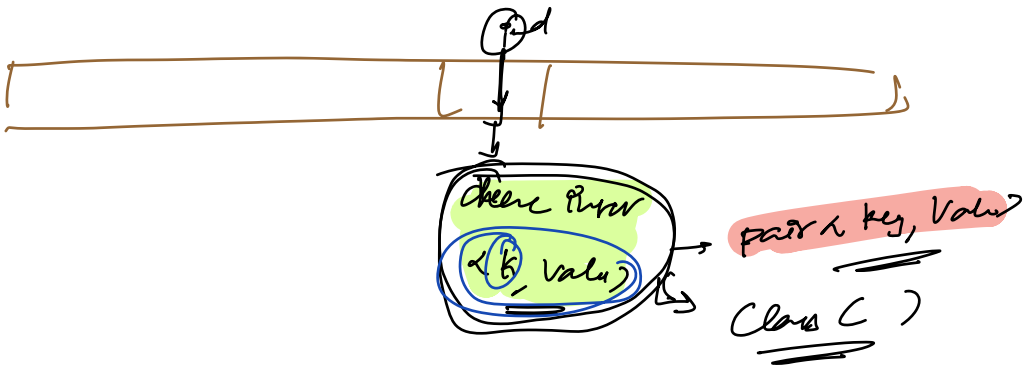
Initially → B[10^7] = —

B[104] → link

Insert	Delete	Search
log P	log P	log P
≈ O(1)	≈ O(1)	O(1)
avg	avg	avg

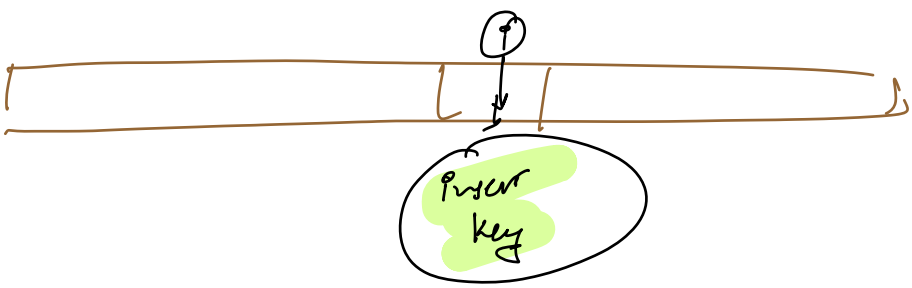
<key, value>

→ <hash function> = index



<key>

→ <hash function> = index



SC → (No hashtable Size)

Hash Table Logic :

Dynamic ar[2]

ar[4]

ar[8]

TC: O(n) SC: O(N)

S = 10

S = 20

S = 40

S = 80 → (Data of N)

all data re-align to 20

re-arrange

TC: O(1) SC: O(N)

→ Open Addressing

key

n

keep iterating until you get next open Address

BBST →

Library BBST

→ Tree Map

→ Tree Set

Map

Set

log N → Search

log N → Insert

log N → Deletion