Hashing 1

Date: 25-Aug-2021 Time: 10:00

1(a) Given N array elements check if there exists a pair (a, b) such that a-b=k, k≠0

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

k = 5 : ∃ 2

a - b = k

3 - -2 = 5 exists & in ar[] ✓

6 - 1 = 5 exists & in ar[] ✓

-1 - -6 ✗

Solutions:

1) All pairs: O(N²)

2) Sort + operator: O(N log N) + O(N) — TC: O(N log N), SC: O(1)

→ Pseudocode

// Hashset ← ts; // Insert all ar[] elements in hs

for (int i=0; i<N; i++) { → [Iterating on array]

b = a - k

// Search for b in ar[]

The above piece can be optimized, we will store all array elements in hashmap & search for b in map

}

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

→ frequency of ar[] — Hashmap / Hashset — Unordered map / Unordered set

→ ar[] = 14 10 7 3 (a = 14, b = 10); hs = {14, 10, 7, 3}

K = 4

1(b) Given N array elements, count all unique (a, b) pairs such that (a-b) = k (k ≠ 0)

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

k = 4

(3,3) (3,3)

(a, b) → (a-b = k)

{(6, 2), (9, 5), (3, -1), (11, 7)}

k = pairs

→ unique pairs

→ Insert all elements in hashset hs ← ts

a) int cnt = 0

for (int i=0; i<N; i++) { — If I only iterate on array[], indirectly nothing iterating on your hs right

int a = ar[i]; // for a

int b = a - k;

if (hs.search(b)) { — Searching for b in hashset — If b is present cnt++

cnt++

}

}

for (iterate on hashset/element by element)

// for a value

int b = k - a;

if (hs.search(b)) {

cnt++

}

}

→ Dry run for above problem

hs → [3, 6, -1, 2, 9, 3, 11], k = 4

a = 3, b = -1 | a = -1, b = -5 | a = 2, b = -2 | a = 9, b = 5 | a = 7, b = 3 | a = 11, b = 7

cnt = 1 | cnt = 1

(7, 3) (11, 7)

3(a) Given N array elements, check if there exists a subarray with sum = 0

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

Hint: If a elements repeat in pf[] can I say

subarray = 0

→ If number of distinct elements in your prefix array < N

→ [We have a subarray]

→ Insert all pf[] values in hashset hs] O(N)

→ if (hs.size() < N) — Meaning, In your pf[] few elements are repeating — return true — That means, there is a subarray with sum=0

if (hs.search(0)) return true

return false

Ex: ar[] : 1 2 -3

pf[] : 1 3 0

If your pf[] contains 0 even in that case simply return true

4(a) Given an array of N elements, find [length of longest subarray] where 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

// If elements repeat in pf[], then subarray sum = 0

3 : 0 4 7

We will consider 0 & 7 to get some subarray sum

// To solve the problem we simply need to get first occurrence & last occurrence

above problem using simple hm

hm: <Key, Value> — pf[] — first occurrence

<3, 0> ans = 4 - 0 = 4

<6, 1> <5, 5> 5 - 3 = 2 ✗

<10, 2> <6, 6> 6 - 1 = 5 ✓ ans = 5

<5, 3> <3, 7> 7 - 0 = 7 ✓ ans = 7

<6, 8> 8 - 1 = 7 ✗

<5, 9> 9 - 3 = 6 ✗

<10, 10> 10 - 2 = 8 ✓ ans = 8 ✓

<6, 11> 11 - 1 = 10 ✓ ans = 10

<5, 12> 12 - 3 = 9 ✗

Ex: ar[] : 0 1 2

0 1 3 0

Insert 0 at beginning:

First occurrence of 0 = -1

hm hm insert <0, -1> ✓✓

Note:

5(a) Given an array contains only 0s & 1's find max subarray which contains equal 0s & 1s

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ex: ar[] :	0	0	1	0	1	0	1	1	1	0	1	0	0	1	0	0	1	1	1	0

// Hint: Some how relate this problem to previous problem.

Hint: (Replace 0 with -1) & take pf[] // pf[] is slightly wrong, leave it as it is

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
pf[] :	-1	-1	-2	-1	-2	-1	-2	-1	0	1	0	1	-1	-2	-1	-2	-3	-4	-3	-2	-3

→ // Get length of longest subarray with sum = 0

Exactly same as previous

→ Logic: Replace 0s with -1s, If a subarray sum = 0, That means can I say it will have 0s & 1s, which is nothing nothing but above problem.

6(a) Given N array elements find length of longest sequence which can be re-arranged to a strictly increasing consecutive order

	0	1	2	3	4	5	6	7	8
ar[] :	6	7	10	8	14	9	12	13	15

→ [6 7 10 8 9]

s → {6 7 8 9 10}

[12 13 14 15]

ar[]: 3 1 2 3 4 11 8 7

{1, 2, 3, 4} → 4

Solutions:

1) Sort & check adjacent elements

TC: N log N + N (iterating)

SC: O(N) → ms

O(1) → QS

→ optimize

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ar[] :	9	-1	-3	7	0	6	-2	11	12	14	15	1	6	8	13

Solution:

Assume 9 is starting of sequence

-1 → 0 → 1 ✓

-3 → -2 → -1 → 0 → 1 → 5

Hint: with every ar[i] have start search for next set continuous elements has long as they are present in array.

TC: N [N time we search in hashset]

First Insert all data in hashset

Ex: 1 2 3 4 5 6 7

Small optimization

8 → 9 → Since 8 is there, we don't start sequence from 9

-1 → Don't start

-3 → -2 → -1 → 0 → 1 : 5 — This is start sequence

6 → 7 : ✗

-6 : ✗

6 → 7 → 8 → 9 : 4

-2 : ✗

11 → 12 → 13 → 14 → 15 : 5

✗ 12

✗ 14

✗ 15

✗ 1

6 → 7 → 8 → 9 : 4 ✗

✗ 8

✗ 13

Can we have a condition to skip this.

Entire Array

→ TC: O(N)

SC: O(N)

Entire Array Iterating

Note: Once we iterate on a sequence simply delete that sequence

Idea:

Insert all elements in hs

Iterate on hs element by ele

int n =

If n is start element of sequence only then we iterate on n.

if (hs.search(n-1)) { → If n-1 is present you don't start here.

continue.

}

for (q = 0; q < ; q++) { — n+1, n+2

if (hs.search(n+q)) {

}

else {

break

}

}

ans = max(ans, q)

}

return ans