Arrays & Strings Lecture 1

Saturday, 20 July 2024 2:56 PM

① Find the missing number in an array

https://leetcode.com/problems/missing-number/

0 → n

n+1

a = { 0, 3, 4, 2 } → ①

|a| = 4 [0-4]

|a| = n [0, n]

a = { 2, 0, 1, 5, 3 } → ④

|a| = 5 [0, 5]

(i) Brute force strategy:

```
for (i : 0 → n) {                    O(n)
    flag = false;
    for (a in arr) {                 O(n)
        if (a == i) {
            flag = True;
            break;
        }
        if (flag == False)
            return i;
    }
}
```

Time complexity :- $O(n^2)$

Space :- $O(1)$

(ii) Better approach.

Sort the array. 0, 1, 2, 4, 5 . . .

```
Find Missing Element ( arr , n) {
    sort (arr) ;  → nlog n
    { for (i : 0 → n) {
        if (arr[i] != i) return i;
      }
}
```

T = O(n log n)

S = O(1)

(iii) Better approach (Hashing):

0 → n

{ map → (K, V)
  Un-map
  dict() }

vector<bool> hash(n+1, false);

0 → T/F
i → T/F


```
for(a: arr) {
    hash[a] = True;
}
```

O(n)

```
for(i: 0→n) {
    if(hash[i] == False) return i;
}
```

O(n)

T = O(n)
S = O(n)

(iv) optimal

Sum of the numbers:

$[0, n] \rightarrow \frac{n(n+1)}{2} \leftarrow O(1)$

$\{0,1,2,3,4\} \rightarrow 10$

$\{3,4,1,0\} \rightarrow \frac{8}{2}$

```
Sum ← 0
for (a : arr) {
    sum += a;
}                                  } O(n)
return (n*(n+1))/2 - sum;
```

T = O(n)
S = O(1)

(V) Best Solution:

- $a \oplus a = 0$
- $a \oplus 0 = a$

$a \oplus a$

```
 1 1 0 1 1
 1 1 0 1 1
-----------
 0 0 0 0 0
```

XOR a b a^b

T T F
T F T
F T T
F F F

```
 1 1 0 1 1
⊕ 0 0 0 0 0
-----------
 1 1 0 1 1
```

{ 3, 2, 0, 5, 1 } |a| = 5, [0, 5]

~~3 ∧ 2 ∧ 0 ∧ 5 ∧ 1~~ ∧ 0 ∧ 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5

```
ans = 0
for (a : arr) {
    ans = ans ^ a;
}                          } O(n)
for (i : 0 → n) {
    ans = ans ^ i;
}                          } O(n)
return ans;
```

T = O(n)
S = O(1)

https://leetcode.com/problems/missing-number/

```cpp
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int ans = 0;
        for(auto n: nums)   ans^=n;
        for(int i=0; i<=nums.size(); i++) ans^=i;
        return ans;
    }
};
```

XOR of all elements in array

XOR of all elements in [0, n]

=x=

Subarrays & Subsequences

2, 3, 4, 1, 5, 6, 7, 8 ✓ ✓ ✓ ✓

2, 1, 7, 8 ✓

1, 3, 2 X

Contiguous

✗ Every subsequence is a subarray ? NO

• Every subarray is a subsequence ? Yes

arr = [1, 2, 3, 4] |arr| = 4

4, 4!, $2^4 = 16$,

Subarrays of arr = 10 (if we don't add empty subarray)

arr = [1, 2, 3, 4]

0, 1, 2, 3

i i

4 + 3 + 2 + 1 = 10

$\frac{n(n+1)}{2}$

i=0, j=0 [1]

i=0, j=1 [1, 2]

i=0, j=2, [1, 2, 3]

i=0, j=3, [1, 2, 3, 4]

Subarrays = $O(n^2)$

Subsequences for arr ? arr = [1, 2, 3, 4]

2^4 (if you are counting empty)

$\{1, 2, 3, 4\}$ → $2 \times 2 \times 2 \times 2 = 16$

$^4C_0 = 1$

$^4C_1 = 4$

$^4C_2 = 6$

$^4C_3 = 4$

$^4C_4 = 1$

$^4C_0 + {}^4C_1 + \dots + {}^4C_4 = 2^4 = 16$

Subseqn. $= 2^n$

Subsequences $= O(2^n)$

array of size n (arr) is given. How many pair of elements i, j exist such that $i, j \in arr$ and $i \neq j$

(n) $^nC_2$ pair of elements. $= O(n^2)$

triplets $= {}^nC_3 = O(n^3)$

=x=

② Longest Subarray ✓ with Sum equals k

Subarrays ✓

Whether a subarray exists ✓

https://leetcode.com/problems/subarray-sum-equals-k

subarrays with sum = k.

arr = {1, 1, 2, 1, 3, 2, 1, 1, 1, 1, 4, 2}

K = 6

2, 1, 3
1, 3, 2
3, 2, 1
2, 1, 1, 1, 1
1, 1, 4
4, 2

} ⑥

(i) Brute force :-

Check every subarray.

Find the sum

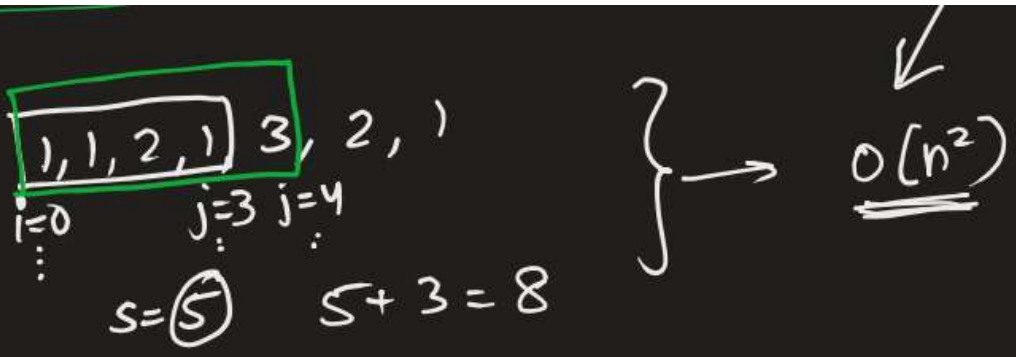
sum == k ans++;

```
ans=0;
for (i : 0 → n-1) {
    for (j : i → n-1) {
        sum=0;
        for (l : i → j) {
            sum += arr[l];
        }
        if (sum==k) ans++;
    }
}
return ans;
```

1, 1, 2, 1, 3, 2, 1

T= $O(n^3)$

S= $O(1)$



https://leetcode.com/problems/subarray-sum-equals-k/

```cpp
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        int ans = 0, n = nums.size();
        for(int i=0; i<n; i++) {
            int sum = 0;
            for(int j=i; j<n; j++) {
                sum += nums[j];
                if(sum == k) ans++;
            }
        }
        return ans;
    }
};
```

$T = O(n^2)$

$S = O(1)$

Optimal :-

* Subarray
  - prefix Sum / XOR
  - suffix Sum / XOR

	0	1	2	3	4	5	6	7	8	9	10	11
arr =	1	1	2	1	3	2	1	1	1	1	4	2
preSum =	1	2	4	5	8	10	11	12	13	14	18	20
suffSum =	20	19	18	16	15	12	10	9	8	7	6	2

$$pre[i] = \sum_{j=0}^{i} arr[j]$$

Sum[3, 8] =

pre[8] − pre[2] = 9

pre[8] → Sum(0 → 8) pre[2] → Sum(0 → 2)

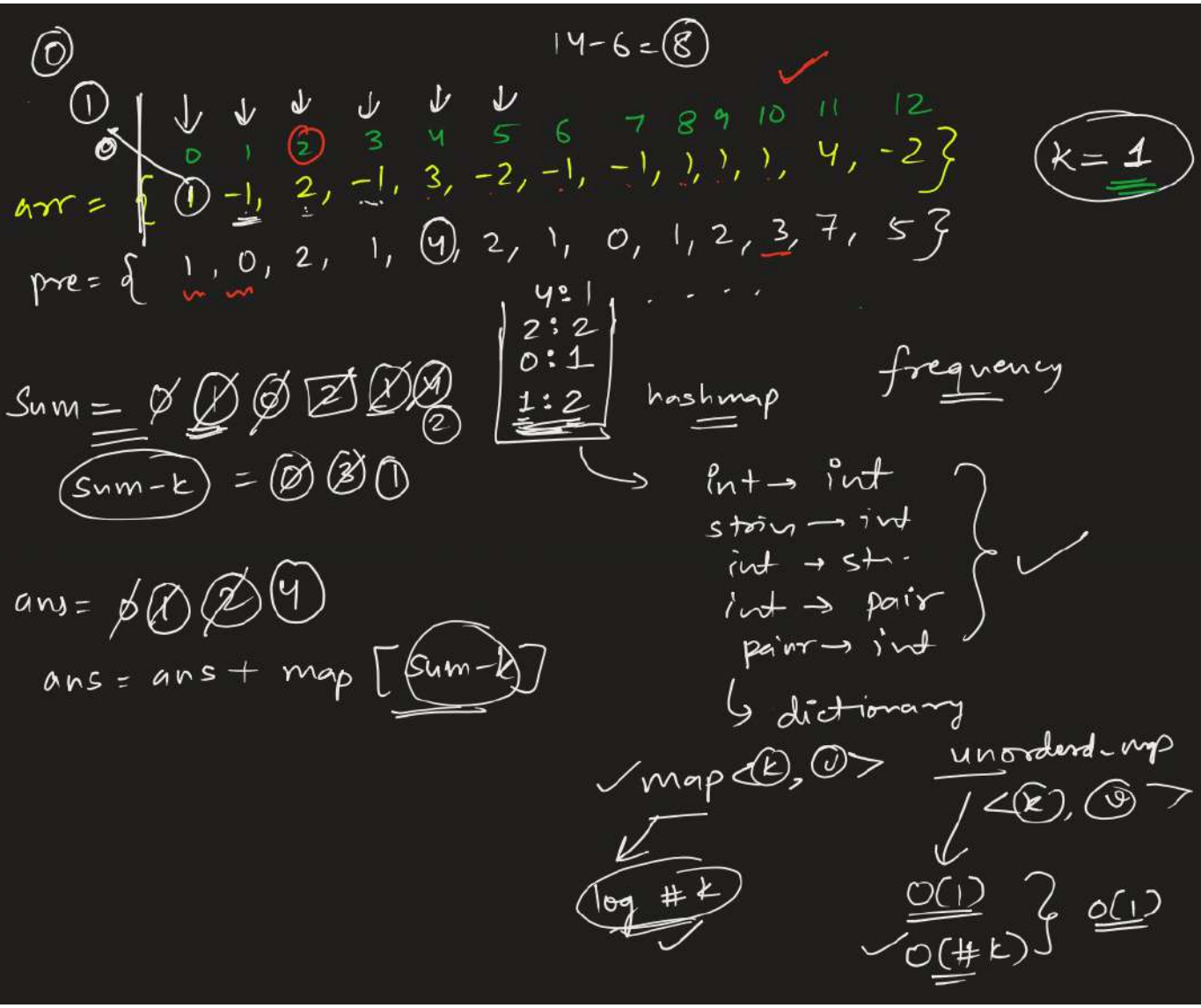
Sum[3,8] = Suffix[3] − Suffix[9] = 16 − 7 = 9

Suffix[3] → Sum(3 → n−1) Suffix[9] → Sum(9 → n−1)

	0	1	2	3	4	5	6	7	8	9	10	11
arr =	1	1	2	1	3	2	1	1	1	1	4	2
pre =	1	2	4	5	8	10	11	12	13	14	18	20

k = 6

Sum till 9 = 14


14-6=8
k=1
arr = { 1, -1, 2, -1, 3, -2, -1, -1, 1, 1, 1, 4, -2 }
pre = { 1, 0, 2, 1, 4, 2, 1, 0, 1, 2, 3, 7, 5 }
4:1
2:2
0:1
1:2
hashmap
frequency
Sum = ∅ 1 ∅ 2 1 4 2
Sum-k = ∅ 3 1
ans = ∅ 1 2 4
ans = ans + map [Sum-k]
int → int
string → int
int → string
int → pair
pair → int
dictionary
map<k, v>
unordered_map
<k, v>
log #k
O(1)
O(#k)
O(1)

```cpp
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        unordered_map<int, int> m;
        int s=0, ans=0;
        m[0] = 1;
        for(auto n: nums) {
            s += n;
            ans += m[s-k];
            m[s]++;
        }
        return ans;
    }
};
```

T= O(n) ✓

S= O(n) ✓