Hashing - Part 2

May 2, 2022

AGENDA

- Sets
- 2-3 very interesting problems

Sets

* Sets contain unique elements **
* Sets are unordered.
* Sets are mutable.

Dictionary.

(K) V pair

```
li = [3, 5, 3, 4, 2]
set(li)
```

→ {3, 5, 4, 2}

() → Tuple
[] → List
{} → Sets/Dictionary

{ - , - } // set
{ - : - , - : - }
// Dictio

* Initialise a empty set.

```
ar = []        // List
de = {}        // Dict

s = {}
type(s)        ← Dictionary.

s = {"hello"}
type(s)        ← set.
```

`s = set()` // Initialise a set.

s = {""} XX

Functions.

```
s = set()
s.add(5)          // {5}
s.add(6)          // {5, 6}
s.add(5)          // {5, 6}
s.remove(5)       // {6}
t = {6, 7}
s.union(t)        // {6} U {6, 7} = {6, 7}
s.intersection(t) // {6}
```

* s.add()
  s.remove()
  → TC: O(1)

* Given an array of N elements, check if an element exists.

  ↓ Q queries.

  ↳ O(N)

  Brute force O(N * Q)

Using set !

* Add all element to a set. → O(N)
* Answer Q queries. → Q * O(1) → O(Q)

TC: O(N+Q)

Q. Given N array elements, check if a pair exists with given sum 'k'.

i.e. $ar[i] + ar[j] = k$ & $i \neq j$

e.g.

0	1	2	3	4	5	6	7	8	9
8	9	1	-2	4	5	11	-6	7	5

	i	j	ar[i]	ar[j]	
K=11	4	8	4	7	
K=6	2	9	1	5	
K=22	6	6	11	11	← not valid.

Brute force solution.

* Iterate over all the pairs.
* Find if it sums up to k.

```
i → 0 to N-1
    j → i+1 to N-1
        If ar[j] + ar[i] == k:
            return True

return False.
```

a + b = K

i → 0 to N-1

a = arr[i] ; b = k - a

searching for b.

j → i+1 to N-1

If arr[j] + arr[i] == k :

return True

return False.

If arr[j] == b :

return True.

Using set

	0	1	2	3	4	5	6	7	8	9	
arr =	8	9	1	-2	4	5	11	-6	7	5	11

my_set = set(arr)

= { 8 9 1 -2 4 5 11 -6 7 }

Hashing

Dictionary Set

K = 11

a	b	
✗ 8	11-8=3	existence check → O(1)
✗ 9	2	
✗ 1	10	
-2	13	✗
4	7	✓

→ O(N)

K = 22.

a

b

11

22 - 11 = 11 → Yes.

Actually, it should have been No

If a == b:
return false

Using Dict

0	1	2	3	4	5	6	7	8	9
8	9	1	-2	4	5	11	-6	7	5

Value : count

freq = {

8 : 1	✓4 : 1	✓7 : 1
9 : 1	5 : 2	
1 : 1	11 : 1	
-2 : 1	-6 : 1	

}

K = 11

a	b	
8	3	×
:		
4	7	✓

K = 22.

a	b	
5	17	×
11	11	

if (a==b):
    freq[a] > 1

* If a==b && freq[a] > 1 :
    return True

Pseudo-code

// freq/hash is created.

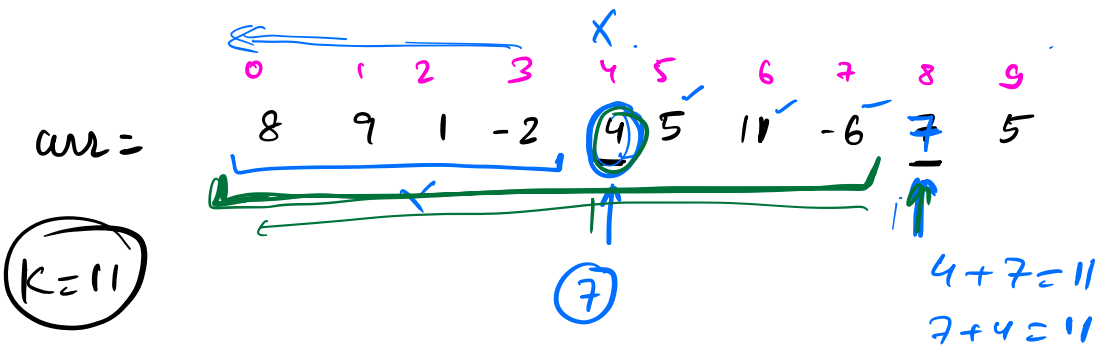
freq/hash → map { value : count }
↳ Dictionary.

```
i = 0 ⟶ N
    a = ar[i]
    b = K-a
    if a!=b :
        if b in freq:
            return True

    else:
        //a==b
        if freq[b] > 1:
            return True

return False
```

Using HashSet



K=11

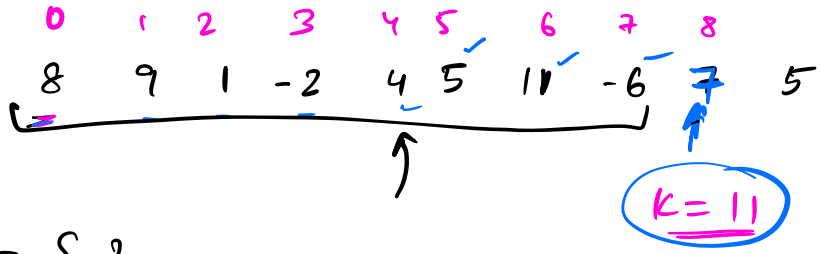
a=7
b=4

Ignoring → Proceed.
Add to the set and proceed.

Pseudo-code.

```
x=set()
i ⟶ 0 to N-1
        a = arr[i]
        b = k-a  // complement
        if b in x :
            return True ✓
        else:
            x.add(a)

return False
```



$X = \{\ \}$

// Keep on adding elements from the array.

$X = \{ 8, 9, 1, -2, 4, 5, 11, -6 \}$

Q. Calculate no. of i,j pairs such that arr[i] + arr[j] = k.

HashSet ✗

HW:)

{ 5, 5, 5, 5, 5 } k = 10

$5C_2$ =

Dictionary. ✓

Q. If there exists a pair i,j such that arr[i] - arr[j] = k

H.W

(K)

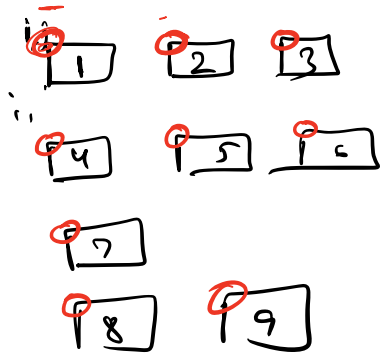
Break till 10:15

May 25

May 23 ✗

(K) May. 19/20. ✓✓

Doubts.



if factors(a) >
rt
< b

r

if a >= b

a < b.

my-compare (coord 1 , coord 2)
(i, j)
(i, j)

return 0

if coord 1[0] < coord 2[0]
return -1

elif [- - >
return

else coord[ ] <
-1

1

return 0.

Q. Given N array elements, calculate no. of distinct elements in every window of size K.

	0	1	2	3	4	5	6	7	8	9	10
arr:	2	4	3	8	3	9	4	9	4	1	0

K = 4

index	ans
[0-3]	4 ✓
[1-4]	3 ✓
[2-5]	3 ✓
[3-6]	4 ✓
[4-7]	3 ✓
[5-8]	2 ✓
[6-9]	3 ✓
[7-10]	4 ✓

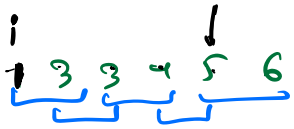
Brute force

* For every window, find no. of distinct elements.

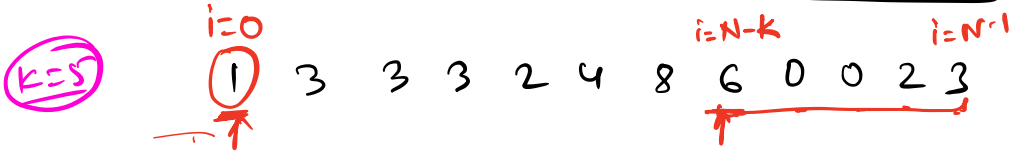
No. of subarrays of length

1 : N

1 3 3 4 5 6



2 : N-1
3 : N-2
⋮
K : N-(K-1)
= N-K+1



* Last subarray will start from index N-K.
arr[N-K : N]
N-K → N-1

Num of subarrays = N-K-0+1 = N-K+1

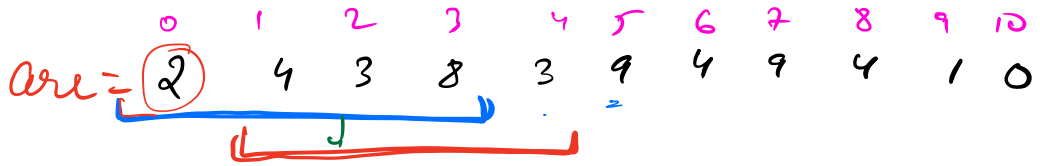
No. of subarrays = N-K+1
No. of distinct elem in a subarray = O(K)

TC : O((N-K+1) * K) ✓

↓

(**) O(N²) O(NK) ✓
K = N/2

* Sliding window approach.



k = 4

x = set = {2, 4, 3, 8}

set
{2, 4, 3, 8}
↓
O(1) time.
4

set

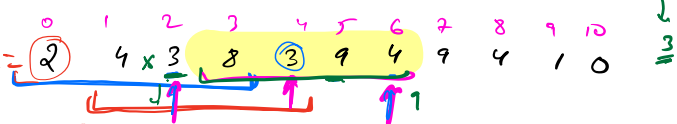
x.remove(arr[0]) ← O(1)
x.add(arr[4]) ← O(1)

{4, 3, 8}

x.remove(arr[1])
x.add(arr[5])

2-5
{3 8 9} ← {8 9 4}
↓
3

→ {3, 8, 9}



x.remove(arr[2])
x.add(arr[6])

→ {8, 9, 4} ← 3

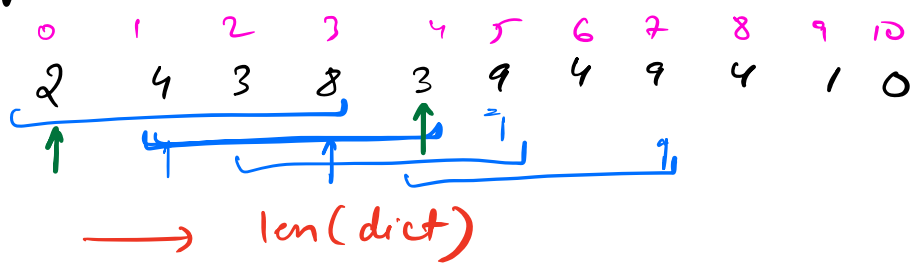
Ans should have been 4.

* No. that was repeating got removed completely when we removed element from the set.

* You need frequency in the picture

* Dictionary

K=4



hash = { 2: 1
4: 1
3: 1
8: 1 }

Once I slide:,

{ ~~2: 0~~
4: 1
3: 2
8: 1 }

len(dict) → 4

len(dict) → 3

{ ~~4: 0~~
3: 2
8: 1
9: 1 }

len(dict) → 3

{ 3: 1
8: 1
9: 1
4: 1 }

len(dict) → 4

{ 3 : 2
~~8 : 0~~
9 : 2 } len(dict) → 2

Pseudo-code.

```
// Populate the first window.
hm = {}
i = 0 ⟶ k-1
    if arr[i] in hm:
        hm[arr[i]] += 1
    else
        hm[arr[i]] = 1

print ( len(hm))

i = 1 , j = k :
while (i<=N-k) :
    // Subarray is i⟺j
    // Remove i-1th element.
    hm[arr[i-1]] -= 1
    if hm[arr[i-1]] == 0:
        del hm[arr[i-1]]

    // Add jth element.
    if arr[j] in hm:
        hm[arr[j]] += 1
    else
        hm[arr[j]] = 1

    print ( len(hm))
    i += 1
    j += 1
```

1st window:
0 ⟺ k-1
2nd window:
1 ⟺ k
3rd window
2 ⟺ k+1
⋮
Last window:
⟶ N-k ⟺ N-1

{ j's value - insert to dict
(i-1)'s value - pop out

Q. Given an array, find the length of longest subsequence which can be rearranged to form a sequence of consecutive numbers.

HW

e.g. [100, 4, 200, 1, 3, 2, 101]

100, 101 → (2)

4, 1, 3, 2 → (4)

1+1=2

2+1=3

3+1=4.

ans = 4

key - value

count

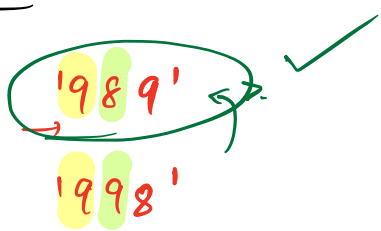
a

k, v pair

Sort - (N log N)

Expected : O(N)

Doubt session



{4 1 1 3 2 7 7}

11234 ✗