Lecture : Hashing - 2

Agenda

- Pair sum = k
- Distinct elements in every window of length = k.

class starts at 7:05 AM

Qu1 Given arr[n], check if there exists a pair (i,j) such

that arr[i] + arr[j] = k and i != j.

$$arr[\,] = \left[\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 9 & 1 & -2 & 4 & 5 & 11 & -6 & 7 & 8 \end{array}\right]$$

k = 11 : arr[4] + arr[8] = 11 true

k = 6 : arr[0] + arr[3] = 6 true

k = 22 : arr[6] + arr[6] [Invalid] false

i = = j

Brute force: Go to all pairs —

check sum == k

arr[i] + arr[j] = k.

arr[j] = k − arr[i] [for every i, check whether k − arr[i] exists or not]

$$arr[\,] = \left[\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 9 & 1 & -2 & 4 & 5 & 11 & -6 & 7 & 8 \end{array}\right]$$

k = 6

i = 2. , arr[i] = 1

arr[j] = k − arr[i]

⇒ 6 − 1 = 5. [true]

```
boolean checkPair( int[] arr, int k) {
    int n = arr.length;
    for(i=0; i<n; i++) {
        int a = arr[i];
        int b = k - arr[i];
        for(j=i+1; j<n; j++){
            if (arr[j] == b) {
                return true;
            }
        }
    }
    return false;
}
```

searching an el within the array

[hashmap. hashset]

TC: $O(n^2)$

SC: $O(1)$

Idea 2 Using hashset [failing for case when i == j]

$$arr[\,] = \left[\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 9 & 1 & -2 & 4 & 5 & 11 & -6 & 7 & 5 \end{array}\right]$$

Insert all el within the hashset.

set : [8 9 1 -2 4 5 11 -6 7]

case 1: k = 11

a	b = k − arr[i] k − a	b is present or not?
8	3	NO
9	2	NO
1	10	NO
−2	11 − (−2) = 13	NO
4	7	Yes [true]

case 2: k = −4

$$arr[\,] = \left[\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 9 & 1 & -2 & 4 & 5 & 11 & -6 & 7 & 5 \end{array}\right]$$

Insert all el within the hashset

set : [8 9 1 -2 4 5 11 -6 7]

a	b [k − a]	b is present or not?
8	−4 −8 = −12	No
9	−4 − 9 = −13	No
1	−4 − 1 = −5	No
−2	−4 − (−2) = −2	Yes [true] i == j

observation : freq of an array el does matter

Idea3: Hashmap

arr[] =

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

map =

key		value (freq)
8	:	1
9	:	1
1	:	1
-2	:	1
4	:	1
5	:	2
11	:	1
-6	:	1
7	:	1

case1: k = -4

a	b = k - a	is B present	freq of B	final ans
8	-4 - 8 = -12	NO		NO
9	-4 - 9 = -13	NO		NO
1	-4 - 1 = -5	NO		NO
-2	-4 - (-2) = -2	Yes	1	false

case2: k = 10

a	b	b present?	freq B	final ans
8	2	NO		NO
9	1	NO		NO
⋮	⋮	⋮		⋮
5	10 - 5 = 5	Yes	2	true

arr[] =

0	1	2	3	4	5	6	7	8	9
8	9	10	-2	4	5	11	-6	7	5

map =

key		value (freq)
8	:	1
9	:	1
1	:	1
-2	:	1
4	:	1
5	:	2
11	:	1
-6	:	1
7	:	1

```
          boolean  checkIfPairExists( int[] arr, int k) {

   o(n) — HashMap<Integer, Integer> map = new HashMap<>();

              // create freq map for every arr el.

 o(n)  —  for( int el : arr) {

         o(1) — if( map.containsKey (el)) {
            o(1)    1 int freq = map.get (el);
                    2 freq = freq +1;
            o(1)    3 map.put( el, freq);
                  } else {
            o(1)    map.put(el, 1);

                  }


   o(n)        for( i=0; i< arr.length; i++) {

                    int a = arr[i];

                    int b = k-arr[i];

 k = 6   why check  if( map.containsKey(b)  &&  a != b) {
 a = 2   freq ? o(1)
 b = 4.                   return true;
                    }

 k = -4  check    if( a == b  &&  map.get(b) > 1) {
 a = -2  freq   o(1)     return true;
 b = -2.
                   }
               }

            return false;
      }
                        TC: o(n)
                        SC: o(n).
```

Idea 4: Hashset [left | right side]

$$arr[\,] = \left[\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 9 & 1 & -2 & 4 & 5 & 11 & -6 & 7 & 5 \end{array}\right]$$

for every i —

checking k − arr[i] in whole array [0, 9] — hashset

i = 2 , j [0-9] 2 twice — failing

k = 10

i = 1 [9] , a = 9

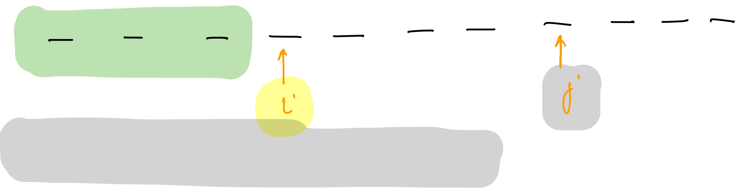
b = 10 − 9 = 1.

Search for 1 on left of 1st idx. [no]

i = 2 [1] a = 1

b = 10 − 1 = 9

Search for 9 on left of 2nd idx [yes]



arr[i] + arr[j] = k

$$arr[\,] = \begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 8 & 9 & 1 & -2 & 4 & 5 & 11 & -6 & 7 & 5 \end{array}$$

Example: $k = 10$

a	b ⇒ k − a	look towards left	
8 [0]	2	Nothing on left	
9 [1]	1	0th idx	
1 [2]	9	0th idx, 1st idx [9]	true $\overset{idx}{(2, 1)}$
−2 [3]	12	0th, 1st, 2nd	
4 [4]	6	0th, 1st, 2nd, 3rd	
5 [5]	5	0th, 1st, 2nd, 3rd, 4th	
⋮	⋮	⋮	
5 [9]	5	0th, 1st, 2nd ----- 8th [Yes]	

```
boolean ifPairExists (int[] arr, int k) {

    HashSet<Integer> set = new HashSet<>();

    for(i=0; i< arr.length; i++) {
        int a = arr[i];

        int b = k - arr[i];

        if( set.contains(b)) {

            return true;
        }

        set.add(arr[i]);
    }
    return false;
}
```

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

Dry run:

set[] = { }

i=0	[operations]	set = { arr[0] }
i=1	{ we have 0th idx in my set }	set = [arr[0], arr[1]]
i=2	[we have 0th, 1st idx in my set]	

Qu Given arr[n], calculate no of distinct elements in every subarray of len = k

arr[10] = [

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

]

k=4

subarray	distinct el
[0-3]	4
[1-4]	3
[2-5]	3
[3-6]	4
[4-7]	3
[5-8]	2
[6-9]	3

← printed

Hint: 1. Sliding window

2. Distinct elements [hashset]

Approach1: sliding window + Hashset.

arr[10] = [2 4 3 8 3 9 4 9 4 10] k = 4

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

subarray	removing	adding	hashet	count
[0-3]			[2 4 3 8]	4
[1-4]	arr[0]	arr[4]	[~~2~~ 4 3 8]	3
[2-5]	arr[1]	arr[5]	[~~4~~ 3 8 9]	3
[3-6]	arr[2]	arr[6]	[~~3~~ 8 9 4]	3 (wrong)

Approach2: **Hashmap** $arr[10] = \left[\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2 & 4 & 3 & 8 & 3 & 9 & 4 & 9 & 4 & 10 \end{array}\right]$ $k=4$

subarray	remove	add	hashmap	count
[0-3]			2 : 1 4 : 1 3 : 1 8 : 1	4
[1-4]	arr[0]	arr[4]	2 : ~~1~~ 0 4 : 1 3 : ~~1~~ 2 8 : 1	3
[2-5]	arr[1]	arr[5]	4 : ~~1~~ 0 3 : 2 8 : 1 9 : 1	3
[3-6]	arr[2]	arr[6]	3 : ~~2~~ 1 8 : 1 9 : 1 4 : 1	4

```
void  distinctWindowElement( int[] arr , int k) {

    HashMap< Integer, Integer> map = new HashMap<>();

    // Handle first window alone : [ 0, k-1]

    for(i=0;  i<k;  i++) {
           int  el = arr[i];
O(1) — if( map.containsKey(el)) {
     O(1)    1 int freq = map.get (el);
             2 freq = freq +1;
     O(1)   3 map.put( el , freq);
          } else {
     O(1)    map.put(el , 1);
   }

   print( map.size())   // Ans for first window


   int  s = 1;

   int  e = k;

   while( e < arr.length) {
          // Add  arr[ e ]   in map

          if( map.containskey(arr[e])) {
              1 int freq = map.get (arr[e])
              2 freq = freq +1;
             3 map.put (arr[e], freq);
          } else {
             map.put(arr[e], 1);
          }
```

```
        // Remove arr[s-1] from map
        int freq = map.get(arr[s-1]);
         if ( freq == 1) {
             map.remove( arr[s-1]);
         } else{
        freq = freq -1;
        map.put(arr[s-1], freq);
      }
        print(map.size());

        s++;

        e++;
    }
}
```

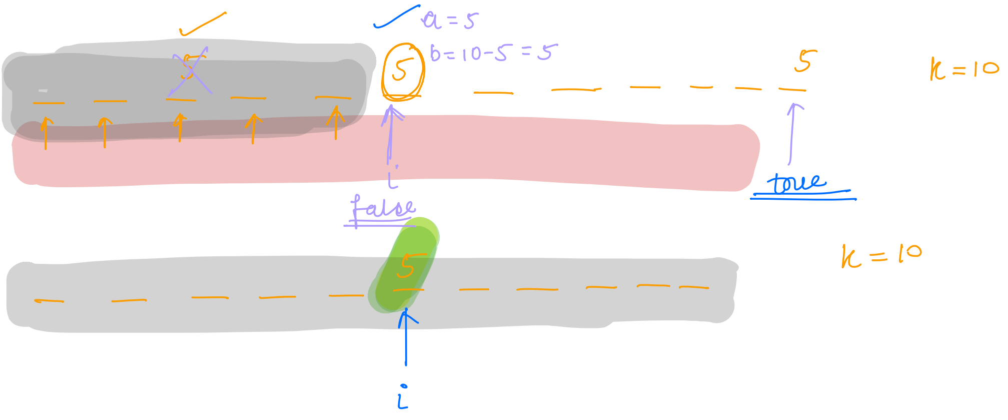
TC: O(n)

SC: O(n)

optional content: Count no of subarrays with sum==k

Thankyou ☺

Doubts



a = 5

b = 10 − 5 = 5

map = [

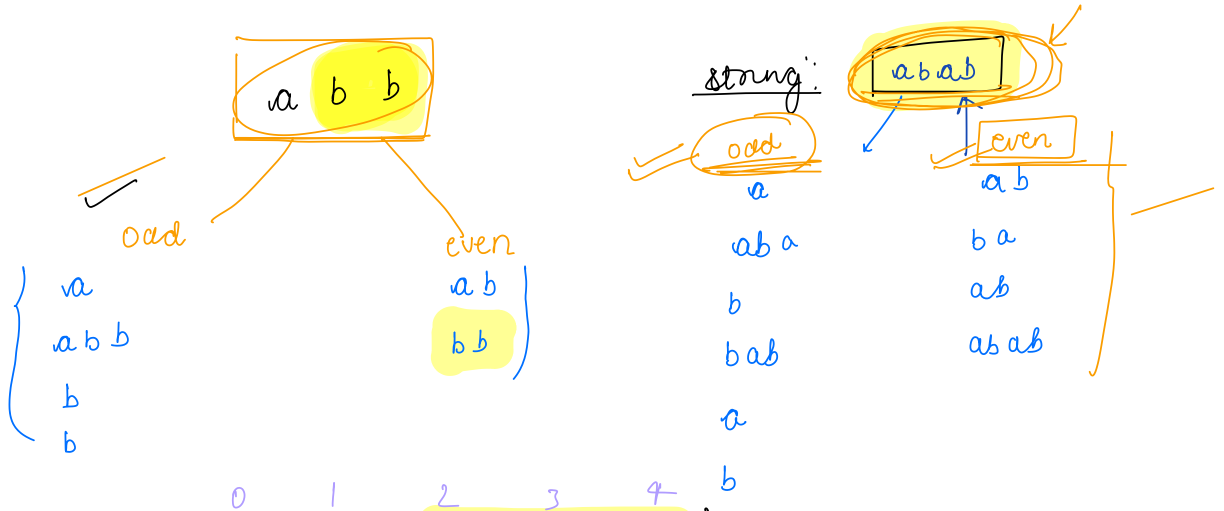
1 : 2

2 : 4

⋮

1000 : 8

]



	0	1	2	3	4
arr() = [	2	3	-2	-2	4];
⇒ pf()	2	5	3	1	5

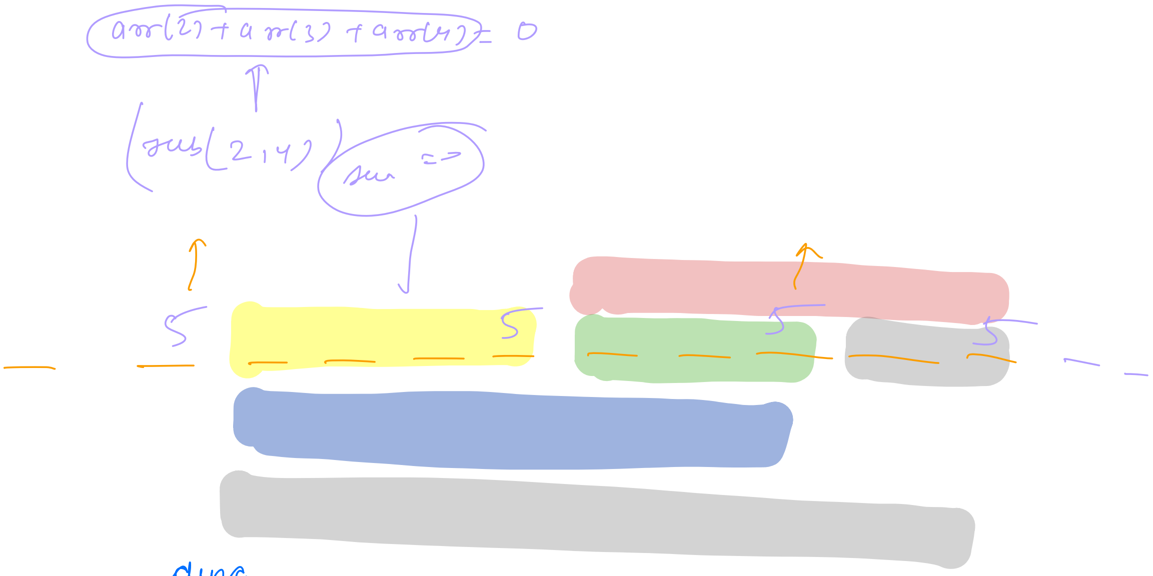
pf(1) = 5

arr(0) + arr(1) = 5 → ①

pf(4) = 5

arr(0) + arr(1) + arr(2) + arr(3) + arr(4) = 5 → ①

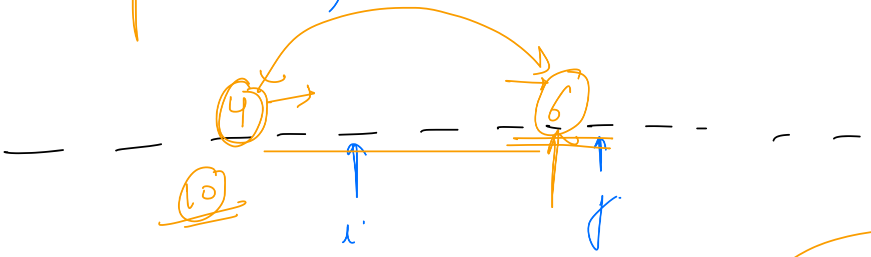
① — ①



dung

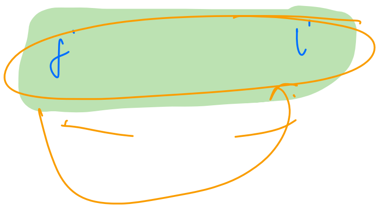
arr[10] = [2 4 3 8 3 9 4 9 4 10]   k = 4

3 — 9



i < j

(6,4)



arr(i) + arr(j) = k