SEPTEMBER 2022

August
35th Wk • 238-127
26 Friday

HASHSET

Hashset is one of the most important data structures, in terms of time complexity. If compared to other data structures it provides a few better optimization.

	Hashset	Array	Sorted Array	BST
Insert/Add	O(1)	O(1)	O(n)	O(h)
Search/contains	O(1)	O(n)	O(log n)	O(h) h > log n in best case
Delete/Remove	O(1)	O(n)	O(n)	O(h)

→
```
import java.util.HashSet;
import java.util.Iterator;

Public class Hashing {
    public static void main (String args[]) {
        HashSet<Integer> set = new HashSet<>();

// Add
        set.add(1);
        set.add(2);
        set.add(3);
        set.add(1); // only unique element is stored.

// Size
System.out.println("Size of the set" + set.size());

// Search - contains
if (set.contains(1)) {          //Positive case
    System.out.println("present");
}
if (!set.contains(6)){          //negative case
    System.out.println("absent");
}
```

27 August
35th Wk • 239-126
Saturday

2022 AUGUST

M	T	W	T	F	S	S	M	T	W	T	F	S	S
1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28
29	30	31											

```
// Delete
set.remove(1);
if(!set.contains(1)){
        System.out.println("absent");
}

// print all elements
    System.out.println(set);
// Iteration - HashSet doesn't have an order
set.add(0);
Iterator it = set.iterator();

while(it.hasNext()){
        System.out.print(it.next() + ", ");
}
    System.out.println();
// isEmpty
    if(!set.isEmpty()){
        System.out.println("set is not empty");
    }
}
```

Note :- • HashSet only stores unique values. Even if we have duplicate values, it will only store unique values.

• hasNext() → checks whether there is any element next to it.

• HashSet doesn't iterate in any fixed order. It goes randomly.

HASHMAP

Scenarios where we have to deal with (Key : value) pairs are the ones where we shall use hashmap.

For. e.g.

roll no.	Student
35	Indranil
36	Debdina
37	Akash
38	Sumit
42	Sayan

Now, here we have a (Key : value) pair. (roll no : Student).

Here, there can be multiple students with the same name, but key should be unique. Same for our hashmaps also.

Key → unique
value → can be same or distinct.

```
→ public static void main (String args[]) {

// creation        → Key type   → value type
HashMap <String, Integer> map = new HashMap<>();

// Insertion
map.put ("India", 120);
map.put ("US", 30);
map.put ("china", 150);

System.out.println (map);

map.put ("china", 180);
System.out.println (map);
```

map.put()
- exist → update value
- doesn't exist → new pair is inserted

29 August
36th Wk • 241-124
Monday

2022 AUGUST
1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31
M T W T F S S M T W T F S S

```
// Searching or look up
if (map.containsKey ("Indonesia")) {
        System.out.println ("Key is present in the map");
    } else {
        System.out.println("Key is not present");
    }

System.out.println (map.get("china")); // Key exists
System.out.println(map.get("Indonesia")); // Key not exist
// Iteration 1
for ( Map.Entry <String, Integer> e : map.entrySet()){
    System.out.println (e.getKey());
    System.out.println (e.getValue());
}

// Iteration 2
Set <String> keys = map.keySet();
for (String key : keys) {
    System.out.println (key + " " + map.get(key));
}

// Removing
map.remove ("china");
System.out.println (map);
    }
}
```

Note :- HashMaps are unordered data structure. So, we may give the input in an order but the values of the maps may be printed in a random order

- containsKey()
  - ✓ (Present) → true
  - ✗ (not present) → false
- get()
  - key exists → value
  - key not exist → null

- put() → to insert a ~~value~~ pair (key : value)
- containsKey() → to check if a pair or key is present
- get() → to return value if key is found.

HASHMAP

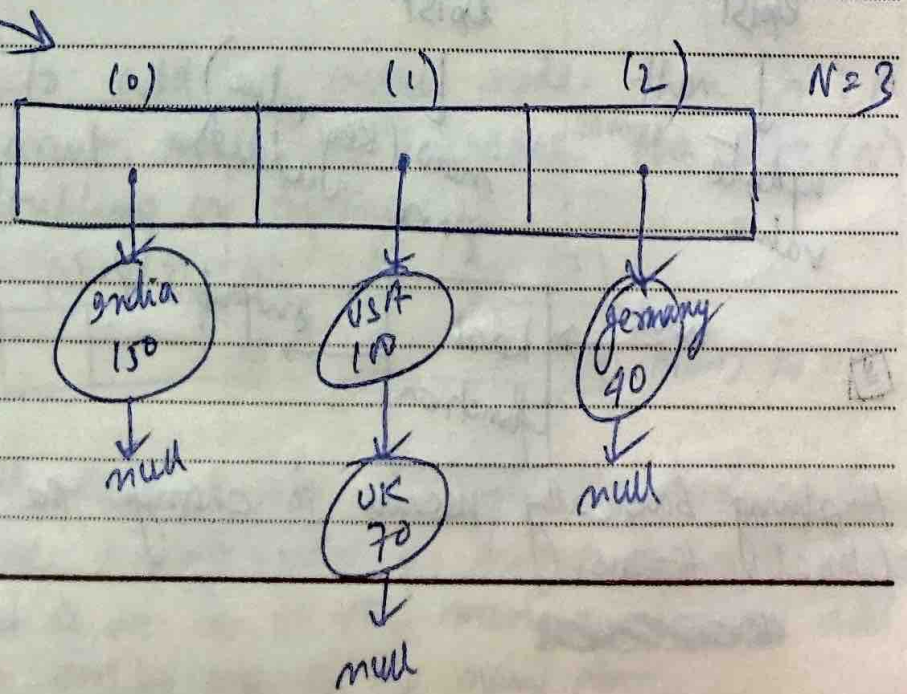
HashMap Implementation in Java

- remove() → to remove any key : value pair
- size() → to find out the total no. of pairs in the hashmap.
- keySet() → to give the set of all keys only.

✓ HashMap is stored as an Array of Linked Lists.

Eg.

India	150
USA	100
UK	70
Germany	40



Functions of HashMap

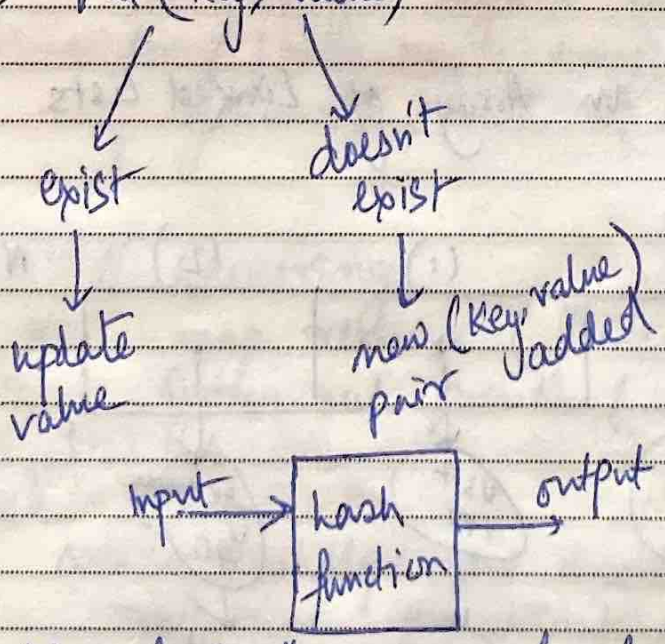
1. Put()
2. get()
3. containsKey()
4. remove()
5. size()
6. KeySet()

conventionally, every index of the array is called a bucket.

n(nodes) = 4
N(size) = 3

So, bucket 0 → ["India", 150]
bucket 1 → ["USA", 100], ["UK", 70]
bucket 2 → ["germany", 40].

① Put(key, value)



Hashing basically means to change the data format, in literal terms.

Here, among 'n' number of nodes are getting distributed 'N' buckets. So, $\frac{n}{N} = \lambda$ (lambda).

and, $\lambda \leq K$ (where K is a constant value)

So, basically when we put( ) or add the nodes of the linked list, we have to fit 'n' number of nodes in N buckets,

and hence time complexity = $O(\lambda)$. λ is a constant.

E.g.

Let's say,

$n = 7$, $N = 4$ and $K = 2$.

Now, $\frac{n}{N} = \frac{7}{4} < 2$

If we add 1 more node, then

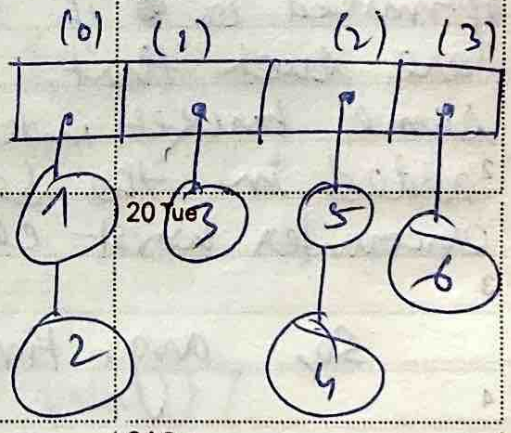
$\frac{n}{N} = \frac{8}{4} = 2$. Still $\lambda = K$.

But, if we try to add 1 more node, then $\lambda > K$, that we don't want. Now, we increase the size (N) of the array by doubling or tripling it.

Now,

Now, N = 8

now, we shall each node through the hashing function, and it shall give 'bucket index' as output. So, node (1) may get index = 3, and so on. So, in this manner we shall add all nodes again one by one in the new Array.



N = 4

01 September
36th Wk • 244-121
Thursday

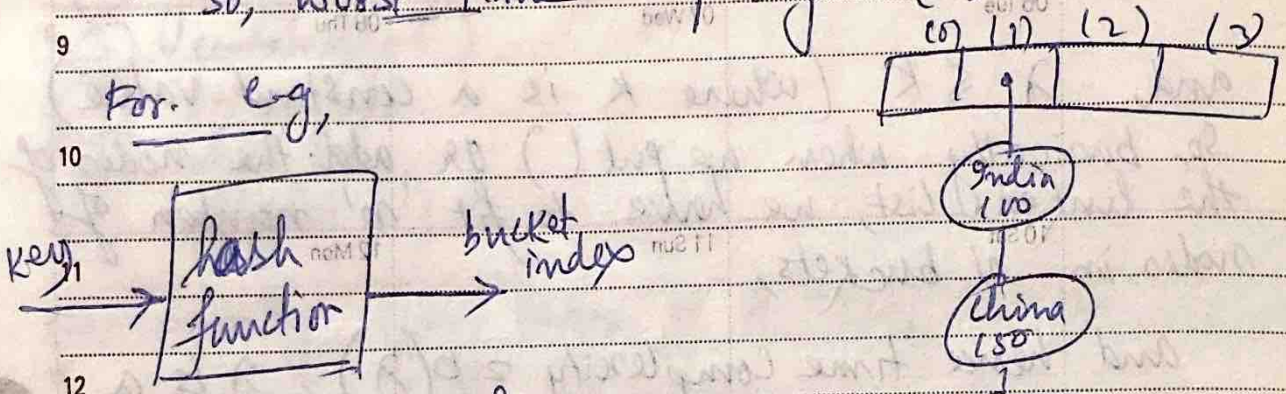
2022 SEPTEMBER

M	T	W	T	F	S	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

Avg. time complexity is $O(\lambda)$. However, there may be some functions, which may yield bad value for the different keys.

So, worst time complexity = $O(n)$.

For. eg,



Let's say, if hash function is not optimised or if the data sets are such that, for different key it's returning the same bucket index. Then all the new nodes will be added in the same index. In that case, we may encounter worst-case time complexity.

So, avg. time complexity = $O(\lambda)$, $\lambda = \frac{n}{N}$.

worst case time complexity = $O(n)$.

(2) get()

key exists (value)

Key doesn't exist → null

So, first search bucket index of the key.

Then, get value by searching the linked list with bucket index.

(3) remove()
- doesn't exist → null
- exist → remove & return value

(i) find the bucket index (bi)
(ii) find di (data index) from the LL.
(iii) check if di is valid.
If valid, then
node = LL.remove(di),
return node.value

** Implementation in Java **

```
import java.util.*;

Public class HashMapCode {
    static class HashMap<K, V> { // generics
        Private class Node {
            K key;
            V value;

            Public Node(K key, V value) {
                this.key = key;
                this.value = value;
            }
        }

        Private int n; // n - nodes
        private int N; // N - buckets

        Private LinkedList<Node> buckets[]; // N = buckets.length

        @Suppresswarnings("unchecked")
        public HashMap() {
            this.N = 4;
            this.buckets = new LinkedList[4];
```

```
        for (int i=0; i<4; i++) {
            this.buckets[i] = new LinkedList<>();
        }
    }

    private int hashFunction(K key) {
        int bi = key.hashCode();
        return Math.abs(bi) % N;
    }

    Private int searchInLL(K key, int bi) {
        LinkedList<Node> ll = buckets[bi];

        for (int i=0; i<ll.size(); i++) {
            if (ll.get(i).key == key) {
                return i; //di
            }
        }
        return -1;
    }

    @SuppressWarnings("unchecked")
    private void rehash() {
        LinkedList<Node> oldBucket[] = buckets;
        buckets = new LinkedList[N*2];

        for (int i=0; i<N*2; i++) {
            buckets[i] = new LinkedList<>();
        }

        for (int i=0; i<oldBucket.length; i++) {
            LinkedList<Node> ll = oldBucket[i];
            for (int j=0; j<ll.size(); j++) {
                Node node = ll.get(j);
                put(node.key, node.value); 33}
```

```
Public void put ( K Key, V value) {
    int bi = hashFunction (Key);
    int di = searchInLL (Key, bi); // di = -1

    if (di == -1) {   // Key doesn't exist
        buckets[bi].add (new Node (Key, value));
        n++;
    } else {  // Key exists
        Node node = buckets[bi].get (di);
        node.value = value;
    }
    double lambda = (double) n/N;
    if (lambda > 2.0) {
        rehash();
    }
}

Public boolean containsKey ( K Key) {
    int bi = hashFunction (Key);
    int di = searchInLL (Key, bi); // di = -1

    if (di == -1) {
        return false;
    } else {  // Key exists
        return true;
    }
}

Public V remove (K key) {
    int bi = hashFunction (Key);
    int di = searchInLL (Key, bi); // di = -1
```

```
        if (di == -1) { // key doesn't exist
            return null;
        } else { // key exist
        Node node = buckets[bi].remove(di);
        n--;
        return node.value;
    }
}

Public V get (K key) {
    int bi = hashFunction (key);
    int di = searchInLL (key, bi); // di = -1

    if (di == -1) { // key doesn't exist
        return null;
    } else { // key exists
        Node node = buckets[bi].get[di];
        return node.value;
    }
}

Public ArrayList<K> keySet() {
ArrayList<K> keys = new ArrayList<>();

for (int i=0; i<buckets.length; i++) { // bi
    LinkedList<Node> ll = buckets[i];

    for (int j=0; j<ll.size(); j++) { // di
        Node node = ll.get(j);
        keys.add(node.key);
    }
}

return keys;
}
```

```
Public boolean isEmpty() {
    return n==0;
}
}
```

```
Public static void main (String args[]) {
HashMap <String, Integer> map = new HashMap<>();
map.put("India", 190);
map.put("China", 200);
map.put("US", 50);

ArrayList <String> Keys = map.keySet();

for(int i=0; i< Key.size(); i++) {
    System.out.println(Keys.get(i)+" "+map.get
                        (Keys.get(i)));
}

map.remove("India");
System.out.println(map.get("India"));
}
}
```

** Questions on HashMap & HashSet **

1) Majority Element

Given an integer array of size n, find all elements that appear more than [n/3] times.

nums[] = {1, 3, 2, 5, 1, 3, 1, 5, 1}; // 1

nums[] = {1, 2} // 1, 2

→ To solve the problem, we can use HashMap. without any DS, the time complexity would have been $O(n^2)$.

we can choose number & frequency as the (Key : value) pair in HashMap and optimize time complexity to $O(n)$.

07 September
37th Wk • 250-115
Wednesday

2022 SEPTEMBER

we shall take the following steps —

(1) create map.

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

(2) find freq. of all nums & store in map.

E.g., 1, 3 → key doesn't exist → add new pair (key → number, value → freq.)

1 → key exists → update value freq → freq+1

(pseudo code)

```
for (int i=0 to n)
  if ( map. containsKey (nums[i]))
    map. put (nums[i], map. get[nums[i] + 1);
  else
    map. put (nums[i], 1);
```

(3) map traversal (all elements)
check which value > (n/3).

```
if(map. get(key) > n/3)
    print (key);
```

map. keySet() → to get all the key value, then traversal,

~~for(int key : map. keySet())~~

=

Code :-

```
import java.util.*;

Public class Classroom {
    public static void majorityElement (int nums[]){

HashMap <Integer, Integer> map = new HashMap<>();
int n = nums.length;
for (int i=0; i<n; i++) {
    if ( map.containsKey (nums[i]) { //true case
        map.Put ( nums[i], map.get(nums[i] +1);
    } else {
        map.Put (nums[i], value:1);
    }
}
for(int key : map.keySet()) {
    if( map.get(key) > n/3) {
        System.out.println (key);
    }
}
}
Public static void main ( String args[]) {
int nums[] = { 1, 3, 2, 5, 1, 3, 1, 5, 1};
majorityElement (nums);
}
}
```

09 September
37th Wk • 252-113
Friday

2022 SEPTEMBER

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

Q2) Union of 2 arrays

arr1 = {7, 3, 9}
arr2 = {6, 3, 9, 2, 9, 4}

union = 6 (7, 3, 9, 6, 2, 4)

E.g., Set A = (7, 3, 8, 5)
Set B = (1, 4, 7, 3, 9)

Set A ∪ Set B = (7, 1, 3, 4, 8, 5, 9)

✓ Every element is included if it occurs in one set or both.
✓ If an elements occurs multiple times in one or both sets, it shall be included once only.
✓ If we solve the problem using nested loops, then time complexity goes to $O(n^2)$.
If we solve the problem using sorting, then time complexity goes to $O(n \log n)$.
These are the brute force ways to solve.

→ So, step by step that shall be followed -

(I) create the Hashset
(II) Add the elements to the hashset. Unique elements to be added.
(III) print the set (set.size()). size
(IV) if elements need to be printed, just create a loop and print the elements

Code :-

```
import java.util.*;

Public class Classroom {
    public static int union (int arr1[], int arr2[]){
        HashSet<Integer> set = new HashSet<>();
    for (int i=0; i< arr1.length. i++) {
        set.add (arr1 [i]);
    }
    for (int j=0; j< arr2.length; j++) {
            set.add (arr2[j]);
    }
    Return set.size(); // returns the size in the set.
}
Public static void main ( String args[]) {
    int arr1[] = {7, 3, 9};
    int arr2[] = {6, 3, 9, 2, 9, 4};
    System.out.println (union ( arr1, arr2));
}
}
```

Time complexity :- O(n). So, to solve the question using HashSet() was a better approach to solve rather using the brute force approaches.

11 September
37th Wk • 254-111
Sunday

2022 SEPTEMBER

M	T	W	T	F	S	S	M	T	W	T	F	S	S
			1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23	24	25
26	27	28	29	30									

Q3) Intersection of 2 arrays

arr1 = {7, 3, 9}

arr2 = {6, 3, 9, 2, 9, 4}

intersection = 2 (3, 9)

Steps :-

(1) Pick any one of the arrays and create a HashSet from it to store only the unique elements

(2) check every element of the other array and compare it with the elements of HashSet. If it matches, increase the counter variable or else keep it same.

(3) one thing to remember is if a match is found, the matched element in the HashSet is to be removed, to avoid any re-match of the same element. eg: set.remove()

Pseudo-code

(1) HashSet create; count = 0

(2) for (int i = 0 to n1) {
set.add(arr1[i])

(3) for (int j = 0 to n2) {
if (set.contains(arr2[j])) // match
count++; set.remove(arr2[j]);

(4) return count;

Code :-

```
import java.util.*;

Public class Classroom {

    Public static int intersection (int arr1[], int arr2[])
    {
        HashSet <Integer> set = new HashSet <>();
        int count = 0;

        for (int i=0 ; i < arr1.length ; i++) {
            set.add (arr1[i]);
        }

        for ( int j=0; j< arr2.length ; j++){
            if (set.contains (arr2[j])) {
                count ++;
                set.remove (arr2[j]) ;
            }
        }
        return count ;
    }
    Public static void main (String args[]) {

        int arr1[] = { 7, 3, 9};
        int arr2[] = { 6, 3, 9, 2, 9, 4};

        System.out.println ( intersection ( arr1, arr2) );
    }
}
```

13 September
38th Wk • 256-109
Tuesday

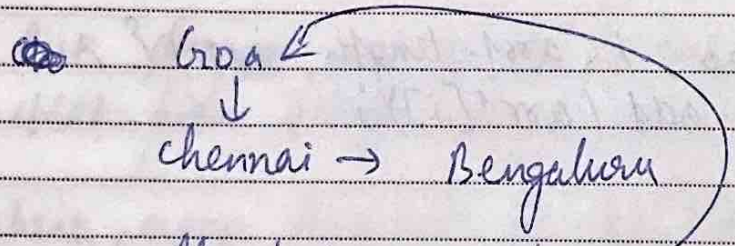
2022 SEPTEMBER

M	T	W	T	F	S	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

Q4) Find Itinerary from Tickets

So, the user shall be given a bunch of tickets. From the bunch, we need to find out the itinerary.

E.g → Ticket 1 : Chennai → Bengaluru
Ticket 2 : Mumbai → Delhi
Ticket 3 : Goa → Chennai
Ticket 4 : Delhi → Goa.

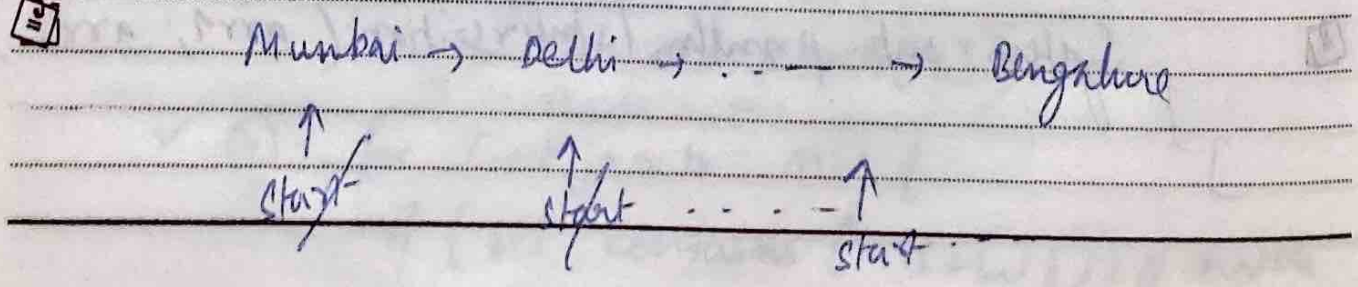


So, itinerary :- Mumbai → Delhi → Goa → Chennai → Bengaluru.

Approach 1 –

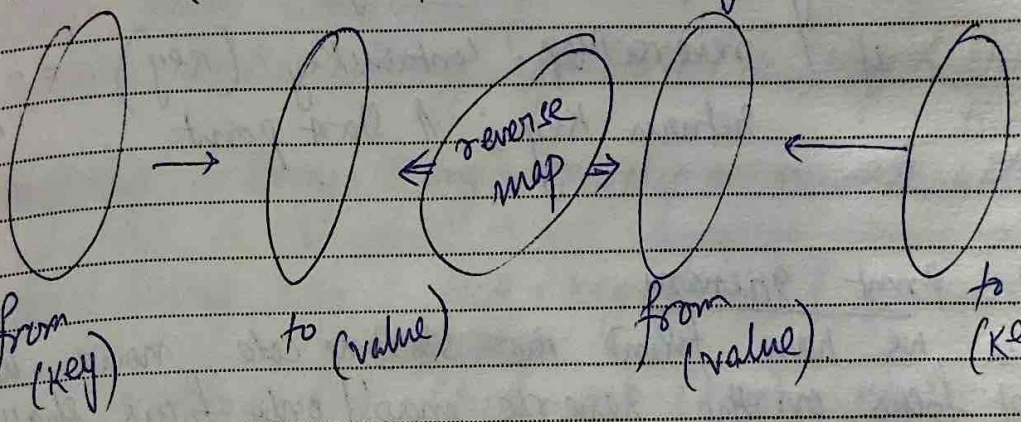
① Find the start. So, if we create 2 sets – 'from' & 'to' and then match every city, we can find start.
We need to find which has 'from' and not 'to'. That shall be our starting point.

② once we find the 'start', then simply find the next destination.



Pseudo code

1. We shall make 2 maps with (key : value) pair.



This shall be done so that we can utilize the containsKey() function.

map.containsKey() and directly find our intended key from any of the maps. The concept of 'reverse map' is possible because 'from' & 'to' contains unique set of value.

In this question, A → B, B → C is possible, but not A → C, B → C. So, starting from 2 locations, we cannot outreach 1 destination.

So, our 2 sets of map look this —

(Key)	(value)
Chennai	Bangalore
Mumbai	Delhi
Goa	Chennai
Delhi	Goa

ticket

(Key)	(value)
Bangalore	Chennai
Delhi	Mumbai
Chennai	Goa
Goa	Delhi

reverse map

So, now Starting Point is the city which doesn't exist in the Key of reverse map.

① Start

```
for (String key : ticket.keySet()) {
    if (reverseMap.containsKey(key) == false)
        return key; // start point
}
```

② Print Itinerary

once we have found the starting code, now we shall not focus on the reverse map, only focus shall be on the ticket map.

```
Start (String)
while (ticket.containsKey(start)) {
    print (start); // key
    start = ticket.get(start);
}
Print (start); // last value who has no 'to' value
```

So, basically prints the key of ticket map and then updated the start (key) with the next steps.

The last value 'bangalore' will not be in the 'key' list, so get out of the loop and print the last value.

Note:- ① No '2' locations can be traversed from 2 different locations.

② Every location should be at least connected either with 'from' or with 'to' or both. There shall be no discontinuity.

Code :-

```
import java.util.*;

Public class Classroom {
    public static String getStart(HashMap<String, String> tick) {
        HashMap<String, String> revMap = new HashMap<>();
        for(String key : tick.keySet()) {
            revMap.put(tick.get(key), key);
        }
        for(String key : tick.keySet()) {
            if(!revMap.containsKey(key)) {
                return key;
            }
        }
        return null;
    }
    Public static void main(String args[]) {
        HashMap<String, String> tickets = new HashMap<>();
        tickets.put("Chennai", "Bangalore");
        tickets.put("Mumbai", "Delhi");
        tickets.put("Goa", "Chennai");
        tickets.put("Delhi", "Goa");
        String start = getStart(tickets);
        while(tickets.containsKey(start)) {
            System.out.print(start + "→");
            start = tickets.get(start);
        }
        System.out.println(start);
    }
}
```

Q5) Subarray Sum equal to K

arr = {1, 2, 3} K=3 Return no. of such sub-arrays

ans = 2 (1,2) (3)

There are many approaches to solve the problem. Now, we shall, of course try to solve it in the least time complexity approach.

(1) Brute force

nested loop.

```
for(i=0; i<. . . .)
    for(j=0, j<i; j . . . .)
```

→ time complexity: $O(n^3)$, if K=3.

(2) Prefix Sum

arr: | 5 | 3 | -1 | 4 |

Prefix: | 5 | 8 | 7 | 11 |

i → j

Prefix(j) - Prefix(i-1)

we need to fix i and j index and call Prefix Sum.

So, let's say K is the index of the prefix sum.

~~So, (for K=0), Prefix(0) - Prefix(0-1)~~

~~for(K=1), Prefix(1) - Prefix(0)~~

Prefix(K) = arr[0] + arr[1] + arr[2] + . . . + arr[K]

So, K=0, Prefix[K] = arr[0] = 5

K=1, Prefix[K] = arr[0] + arr[1] = 8

. . . and so on.

once, the prefix array is filled, then, we can calculate the sum from $(i \leftrightarrow j)$ by,

$$Prefix(j) - Prefix(i-1)$$

Time complexity = $O(n^2)$.

3rd Now, we shall talk about the approach by which we are going to solve the problem, with a better time complexity.

$$Sub(i,j) = Sub(j) - Sub(i-1)$$

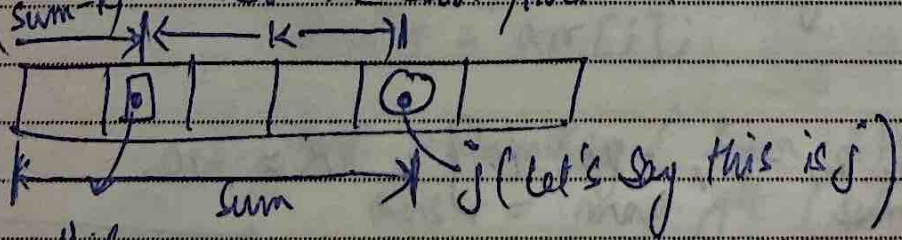
$$K = Sub(j) - Sub(i-1)$$

So, if we know K and Sub(j), of course we can find out Sub(i-1).

$$\Rightarrow \boxed{Sub(i-1) = Sub(j) - K}$$

let's say, $b = a - K$

So, if we can fix a and we already know K, then we can find out b.



Suppose this is the point where (sum-K) exists.

If we find out (Sum) till jth point, and we find that there is a point where (sum - K) exists, then that point is the (i-1) point.

So, then we can get all the values of subarrays, sum of which is equal to K.

19 September
39th Wk • 262-103
Monday

(5, 1) (5, 1, -1, 1)

2022 SEPTEMBER

M	T	W	T	F	S	S	M	T	W	T	F	S	S
						1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30						

Eg, | 1 | -1 | 1 | 5 | k = 6.

for, j=0, sum = 1
j=1, sum = 0 → sum - K = 6 - 6 = 0
j=2, sum = 1
j=3, sum = 6.

① create a hashMap <Int, Int>.
sum ↙ ↘ freq.
By default, we shall store (0, 1) because this is for empty subarray.

② for (j=0 to n)
if (sum - K → exists in map),
then ans = ans + map.get(sum - K)
else
(add sum) → if sum → map exist, (freq+1)
→ if sum → not exists in map, (freq = 1)

Eg,

arr = {10, 2, -2, -20, 10} K = -10.

① j=0, sum = 10, sum - K = 20.
check (sum - K) exists in map.
if yes → add freq,
else, check (sum exists in map)

K (sum)	V (freq)
0	1 → 2
10	1 → 2
12	1

② j=1, sum = 12, sum - K = 22.
③ j=2, sum = 10, sum - K = 20.
④ j=3, sum = 10, sum - K = 0.
⑤ j=4, sum = 0, sum - K = 10.

So, ans = 3

Basically, either (sum-K) or (sum) has to be there in the map.

If (sum-K) is there, then add freq by +1.
If not, then check (sum)
↳ if sum → ✓, then freq +1
if sum → X, then add value.

! code :-

```
import java.util.*;
Public class Classroom {
    Public static void main (String args[]) {
        int arr[] = {10, 2, -2, -20, 10};
        int K = -10;
        HashMap<Integer, Integer> map = new HashMap<>();
        map.put(Key: 0, Value: 1);
        int ans = 0;
        int sum = 0;
        for (int j=0; j<arr.length; j++) {
            sum += arr[j];

            if (map.containsKey(sum-K)) {
                ans += map.get(sum-K);
            }
            if (map.containsKey(sum)) {
                map.put(sum, map.get(sum)+1);
            } else {
                map.put(sum, Value: 1);
            }
        System.out.println(ans);
    }
```