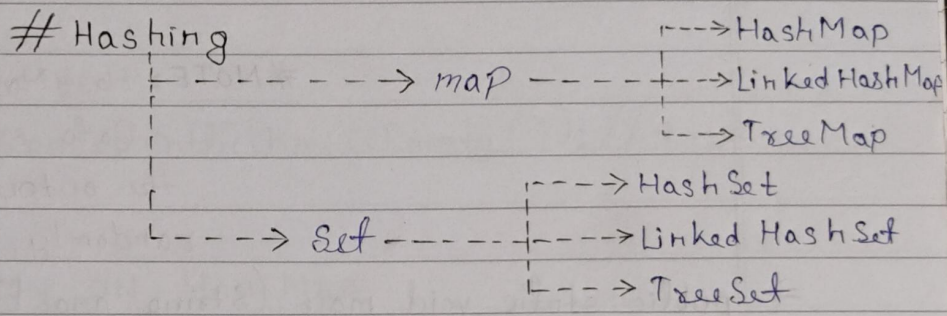
Enroll
Page No: 446
Date: 25/09/24

HASHING

* Hashing is like assigning an address to data so that we can quickly find it later.
* A Hash function gives you a unique number i.e., hash code, which is like the call number of the book.
* The Hash Table is the shelf where we store the data based on that number.



★ HASHMAP

It is used to store table of information.

* We store information in pairs → (key, value).
  key → if is unique.

Ex→ RESTAURANT MENU

Tea	10
Samosa	15
Pizza	50
Coffee	10

Key — Value → it can repeated & same.

* In HashMap, → insert() → O(1)
  → remove() → O(1)
  → search() → O(1)

★ HashMap Operations

* to import → import java.util.HashMap;
* to create → HashMap < Key data_type , Value data_type > name_of_HashMap = new HashMap<>();

i) put (key, value) → O(1) //insertion
ii) get (key) → O(1) //returns the value
iii) containsKey (key) → O(1) //returns true/false
iv) remove (key) → O(1) //deletion

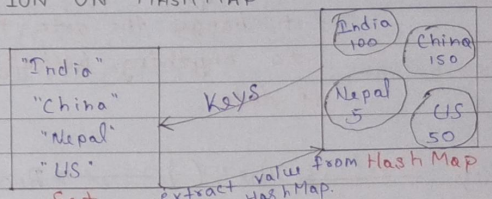
#NOTE: HashMap are unordered Data structure i.e. the output can be shown randomly.

⇒
```
public static void main (String args[]) {
  //create
  HashMap <String, Integer> hm = new HashMap<>();
  //Insert - O(1)
  hm.put ("India", 100);
  hm.put ("China", 150);
  hm.put ("US", 50);
  System.out.println (hm); //{China=150, US=50, India=100}
  //Get - O(1)
  int population = hm.get ("India");
  System.out.println (population); //100
  System.out.println (hm.get ("Indonesia")); //null
  //ContainsKey - O(1)
  System.out.println (hm.containsKey ("India")); //true
  System.out.println (hm.containsKey ("Ballia")); //false
  //remove key - O(1)
  System.out.println (hm.remove ("China")); //150
  System.out.println (hm); //{US=50, India=100}
}
```


★ SIZE(), CLEAR() & isEMPTY()

⇒
```
public static void main (String args[]) {
  HashMap <String, Integer> hm = new HashMap<>();
  hm.put ("India", 100);
  hm.put ("China", 150);
  hm.put ("US", 50);
  System.out.println (hm); //{China=150, US=50, India=100}
  //size
  System.out.println (hm.size()); //3
  //clear
  hm.clear();
  //Is Empty
  System.out.println (hm.isEmpty()); //true
}
```

★ ITERATION ON HashMap



⇒
```
public static void main (String args[]) {
  HashMap <String, Integer> hm = new HashMap <>();
  hm.put ("India", 100);   hm.put ("China", 150);
  hm.put ("US", 50);   hm.put ("Nepal", 5);
  //iterate
  Set <String> keys = hm.keySet(); → O(1)
  System.out.println (keys);
  for (String k : keys) {
    System.out.println ("keys = " +k+ ", value = " +hm.get (k));
  }
}
```

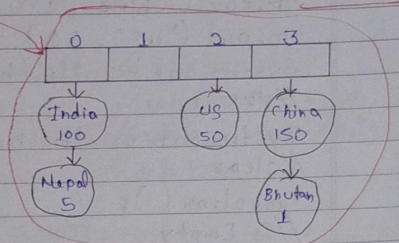
★ HashMap Implementation

put(), get(), containsKey(), remove(), size() → O(1)

map →

Key	Value
"India"	100
"China"	150
"US"	50
"Nepal"	5
"Bhutan"	1

* HashMap internally stores as Array of LinkedList.



⇒ Bucket of Linked List

* "India" ⟶ hashCode() ⟶ 1245

it changes the data form to anything, basically it do hashing of data.

* Key ⟶ Hash() ⟶ Bucket or array index

It works in constant time.

Ex⇒ N = size of Array = 4
n = number of nodes = 7
∴ λ (lambda) = $\frac{n}{N} = \frac{7}{4} = 1.75$

Since, Time Complexity → O(λ) where $\lambda = \frac{n}{N}$ is a constant value.

* Rehashing = new array = 2 * Old size( );

⇒
```
static class HashMap<K, V> {
  private class Node {
    K key;
    V value;
    public Node(K key, V value) {
      this.key = key;
      this.value = value;
    }
  }
  private int n;
  private int N;
  private LinkedList<Node> buckets[];
  @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 hc = key.hashCode();
    return Math.abs(hc) % buckets.length;
  }
  private int SearchInLL(K key, int bi) {
    LinkedList<Node> ll = buckets[bi];
    int di = 0;
    for (int i=0; i<ll.size(); i++)
      Node node = ll.get(i);
      if (node.key == key)
        return di;
      di++;
    } return -1;
```

```
public void put (K key, V value) {
    int bi = hashFunction (key);
    int di = SearchInLL (key, bi);
    if (di != -1)
        Node node = buckets [bi].get (di);
        node.value = value;
    } else {
        buckets [bi].add (new Node(key, value));
        n++;
    }
    double lambda = (double) n / N;
    if (lambda > 2.0) {
        rehash();
    }
}
private void rehash() {
    LinkedList < Node > oldBuck [] = buckets;
    buckets = new LinkedList (N*2);
    N = 2*N;
    for (int i=0; i < buckets.length; i++)
        buckets[i] = new LinkedList <>();
    }
    // nodes -> add in bucket
    for (int i=0; i<oldBuck.length; i++) {
        LinkedList <Node> ll = oldBuck [i];
        for (int j=0; j < ll.size(); j++) {
            Node node = ll.remove();
            put (node.key, node.value);
        }
    }
}
```



```
public boolean containsKey (K key) {
    int bi = hashFunction (key);
    int di = SearchInLL (key, bi);
    if (di != -1) { // valid
        return true;
    } else {
        return false;
    }
}
public V remove (K key) {
    int bi = hashFunction (key);
    int di = SearchInLL (key, bi);
    if (di != -1) {
        Node node = buckets [bi].remove (di);
        return node.value;
    } else {
        return null;
    }
}
public V get (K key) {
    int bi = hashFuction (key);
    int di = SearchInLL (key, bi);
    if (di != -1) {
        Node node = buckets [bi].get [di];
        return node.value;
    } else {
        return null;
    }
}
public boolean isEmpty () {
    return n == 0;
}
```

```
public ArrayList<K> keySet() {
    ArrayList<K> keys = new ArrayList<>();
    for(int i=0; i<buckets.length; i++) {
        LinkedList<Node> ll = buckets[i];
        for(Node node : ll) {
            keys.add(node.key);
        }
    }
    return keys;
  }
}

public static void main(String args[]) {
    HashMap<String, Integer> hm = new HashMap<>();
    hm.put("India", 100);
    hm.put("China", 150);
    hm.put("US", 50);
    hm.put("Nepal", 5);
    ArrayList<String> keys = hm.keySet();
    for(String key : keys) {
        System.out.println(key);
    }
    System.out.println(hm.get("India"));
    System.out.println(hm.remove("India"));
    System.out.println(hm.get("India"));
}
```

SPECIAL NOTE

In HashMap, Worst Case is O(n).

(very rear case)

★ LINKED HASHMAP

* Keys are insertion ordered → मतलब जिस order में insertion है उसी order में LinkedHashMap भी हो।

Syntax ⇒ LinkedHashMap<Key datatype, Value datatype> hm = new LinkedHashMap<>();

* LinkedHashMap also performs all operations → O(1)

⇒

```
public static void main(String args[]) {
    LinkedHashMap<String, Integer> lhm = new LinkedHashMap<>();
    lhm.put("India", 100);
    lhm.put("China", 150);
    lhm.put("US", 50);
    //Hash Map Implementation
    HashMap<String, Integer> hm = new HashMap<>();
    hm.put("India", 100);
    hm.put("China", 150);
    hm.put("US", 50);
    //printing of LinkedHashMap & HashMap
    System.out.println(hm);
    System.out.println(lhm);
}
```

Output: {China=150, US=50, India=100}
{India=100, China=150, US=50}

#NOTE: HashMap is printed in Unordered.
LinkedHashMap is printed in Insertion Ordered.

★ TREE MAP

→ Keys are sorted (in alphabetical order).
→ put, get, remove are O(log n).

★ TreeMap <Key data-type, Value data-type> tm = new TreeMap <>();

★ TreeMap is implemented by Red Black Tree. (self-balancing tree).

⇒
```
public static void main (String args[]) {
    TreeMap <String, Integer> tm = new TreeMap <>();
    tm.put ("India", 100);
    tm.put ("China", 150);
    tm.put ("US", 50);
    tm.put ("Indonesia", 6);
    // HashMap
    HashMap <String, Integer> hm = new HashMap <>();
    hm.put ("India", 100);
    hm.put ("China", 150);
    hm.put ("US", 50);
    // Printing HashMap & TreeMap
    System.out.println (tm);
    System.out.println (hm);
}
```

Output: {China=150, India=100, Indonesia=6, US=50}
{China=150, India=100, US=50}

NOTE: TreeMap outputs are sorted in an alphabetical order & HashMap are unsorted or jumble output.

★ MAJORITY ELEMENT – Given an Integer array of size n, find all elements that appear more than [n/3] times.

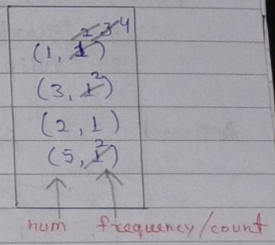
Input: nums [] = {1, 3, 2, 5, 1, 3, 1, 5, 1}
Output: 1 (n=9, n/3 = 9/3 = 3 times)

Input: {1, 2};
Output: 1, 2 (n=2, 2/3 = 0)

Ex → nums [] = {1, 3, 2, 5, 1, 3, 1, 5, 1}

∴ (1, 4) → 4 times

∴ Output → 1



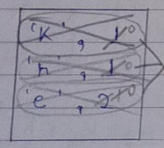
⇒
```
public static void main (String args[]) {
    int arr [] = {1, 3, 2, 5, 1, 3, 1, 5, 1};
    HashMap <Integer, Integer> map = new HashMap <>();
    // Time Complexity - O(n)
    for (int i=0; i<arr.length; i++) {
        if (map.containsKey (arr[i])) {
            map.put (arr[i], map.get (arr[i]) + 1);
        } else {
            map.put (arr[i], 1);
        }
    }
    Set <Integer> keySet = map.keySet();
    for (Integer key : keySet) {
        if (map.get (key) > arr.length (3) {
            System.out.println (key);
        }
    }
}
```

★ VALID ANAGRAM – Given two strings s & t, return true if 't' is an anagram of 's' & false otherwise.

Anagram: it is a word or phrase formed by rearranging the letters of different word or phrase typically using all the original letters exactly once.

Ex ⇒ s = "race", t = "care" → TRUE
s = "heart", t = "earth" → TRUE
s = "tulip", t = "lipid" → FALSE

Approach: s = "knee" t = "keen"



Now, we will traverse 't' then we will decrease the frequency whenever letter come.

HashMap<Character, Frequency>

⇒
```
public static boolean isAnagram(String s, String t){
  if(s.length() != t.length())
    return false;
  HashMap<Character, Integer> map = new HashMap<>();
  for(int i=0; i<s.length(); i++)
    char ch = s.charAt(i);
    map.put(ch, map.getOrDefault(ch, 0)+1);
  for(int i=0; i<t.length(); i++)
    char ch = t.charAt(i);
    if(map.get(ch) != null)
      if(map.get(ch) == 1)
        map.remove(ch);
      else
        map.put(ch, map.get(ch)-1);
    else
      return false;
  return map.isEmpty();
```


PROPERTY	HASH MAP	LINKED HASH MAP	TREE MAP
Time Complexity of get, Put, Contains Key & Remove method	O(1)	O(1)	O(1)
Iteration Order	Random	Sorted according to Insertion Order.	Sorted according to natural order of key. not allowed if
Null Keys	allowed	allowed	keys uses natural ordering on null keys
Interface	Map	Map	Map, Sorted Map & Navigable Map.
Synchronisation	None, Use Collections. Synchronized Map()	None, Use Collections. Synchronized Map()	None. Use Collections. Synchronized Map()
Data Structure	List of buckets, if more than 8 entries then Java8 will switch to balanced trees.	Doubly Linked List of Buckets.	Red Black implement -ation of Binary trees.
Applications	General purpose, fast retrieval, Concurrent HashMap can be used where concurrency is involved.	Can be used for LRU cache, other places where insertion or access order matters.	Algorithms where Sorted or Navigable features are required. Ex⇒ find among the list of employees whose salary is next to employee
Requirements for Keys	Equals() & HashCode() needs to be overwritten.	Equals() & hashCode() needs to be overwritten.	Comparator needs to be supplied for key implementation, otherwise natural order.

★ HashSet

* HashSet is implemented by HashMap.
* Set is a collections of unique elements.
* HashSet are unique, has no duplicates.
* HashSet is an unordered.
* Null is allowed in HashSet.

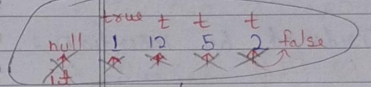
⇒ `HashSet<key data-type> hs = new HashSet<>();`

1. add (key) → O(1)
2. contains (key) → O(1)
3. remove (key) → O(1)

⇒
```
public static void main (String args[]) {
    HashSet<Integer> set = new HashSet<>();
    set.add(1); set.add(2); set.add(4);
    set.add(2); set.add(1);
    System.out.println(set); //[1, 2, 4]
    System.out.println(set.size()); //3
    if (set.contains(2))
        System.out.println("set contains 2"); //set contains 2
    set.remove(2); //removed element 2
    if (set.contains(2))
        System.out.println("set contains 2"); //Nothing will print
    System.out.println(set.isEmpty()); //false
    System.out.println(set); //[1, 4]
    set.clear(); //it will empty whole set
    System.out.println(set.size()); //0
    System.out.println(set.isEmpty()); //true
}
```


★ ITERATION ON HASHSETS

a) Using Iterators → `Iterator it = set.iterator()`
// iterator type object — it will call iterator




```
while (it.hasNext()) {
    print (it.next()); → point to next element
}
```

→ print → 1 12 5 2
it can be unordered

b) using Enhanced for loop → by using simply 'for' loop in code.

⇒
```
public static void main (String args[]) {
    HashSet<String> cities = new HashSet<>();
    cities.add("Delhi");
    cities.add("Mumbai");
    cities.add("Noida");
    cities.add("Bengaluru");
    Iterator it = cities.iterator();
    while (it.hasNext()) {
        System.out.println(it.next());
    }
```

↳ a) using iterators

```
    for (String city : cities) {
        System.out.println(city);
    }
}
```

↳ b) using enhanced for loop

Output: Delhi
Bengaluru
Noida
Mumbai

★ LINKED HASHSET

* Linked HashSet is implemented same as the Linked HashMap.
* Linked HashSet are Ordered by insertion. (using DLL)
* As Performance,
  LinkedHashMap < HashMap
  LinkedHashSet < HashSet
  but the time complexity is same – O(n)

⇒
```
public static void main (String args[]) {
    HashSet<String> cities = new HashSet<>();
    cities.add("Delhi");
    cities.add("Mumbai");
    cities.add("Ballia");
    cities.add("Bihar");
    System.out.println(cities);
    //Hash Set  LinkedHashSet
    LinkedHashSet<String> lhs = new LinkedHashSet<>();
    lhs.add("Delhi");
    lhs.add("Mumbai");
    lhs.add("Ballia");
    lhs.add("Bihar");
    System.out.println(lhs);
}
```

Output: [Delhi, Bengaluru, Noida, Mumbai]
[Delhi, Mumbai, Ballia, Bihar]

★ TREESET

* TreeSet is implemented by TreeMap.
* TreeMap is implemented by Red Black Tree. (self-balancing tree)
* TreeSet are Sorted in Ascending Order.
* In TreeSet, NULL values are NOT allowed.
* In TreeSet, insertion & remove ⟶ O(log n)

⇒
```
pubkic static void main (String args[]) {
    //Hash Set
    HashSet<String> hs = new HashSet<>();
    hs.add("Delhi");
    hs.add("Mumbai");
    hs.add("Noida");
    hs.add("Bengaluru");
    // Linked Hash Set
    LinkedHashSet<String> lhs = new LinkedHashSet<>();
    lhs.add("Delhi");
    lhs.add("Mumbai");
    lhs.add("Noida");
    lhs.add("Bengaluru");
    // Tree Set
    TreeSet<String> ts = new TreeSet<>();
    ts.add("Delhi");
    ts.add("Mumbai");
    ts.add("Noida");
    ts.add("Bengaluru");
    // Printing hs, lhs, ts
    Syso(hs); //[Delhi, Bengaluru, Noida, Mumbai]
    Syso(lhs); //[Delhi, Mumbai, Noida, Bengaluru]
    Syso(ts); //[Bengaluru, Delhi, Mumbai, Noida]
}
```

★ COUNT DISTINCT ELEMENTS

Input :- num = {4, 3, 3, 5, 6, 7, 3, 4, 2, 1}
Output :- 7 → we will count only one time a number.

* We know that inside set there are unique element so simply we put num into set, then count.

⇒
```
public static void main (String args[]) {
    int num[] = {4, 3, 2, 5, 6, 7, 3, 4, 2, 1};
    HashSet <Integer> set = new HashSet<>();
    for (int i=0; i<num.length; i++) {
        set.add (num[i]);
    }
    System.out.println ("Ans = " + set.size());
}
```

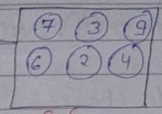
★ UNION & INTERSECTION OF 2 ARRAYS

→ arr1 = {7, 3, 9}
→ arr2 = {6, 3, 9, 2, 9, 4}
⇒ union = 6 (7, 3, 9, 6, 2, 4)
⇒ intersection = 2 (3, 9)

* Approach
- If there 2 sets : set1 , set2
- Union ——→ set1 ∪ set2 → unique elements.
- Intersection → set1 ∩ set2 → common elements

We know that inside set, there are only unique element.
* We put arr1 → set1, arr2 → set2 } set.size() will be Union.

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



→ set.size() ⇓ 6 ⇓ Union

Now, for intersections
① add all elements of arr1 // O(n + m)
② for (int i=0 to arr2.length)
check if exists in set
T → count ++
→ remove that element

⇒
```
public static void main (String args[]) {
    int arr1[] = {7, 3, 9};
    int arr2[] = {6, 3, 9, 2, 9, 4};
    HashSet <Integer> set = new HashSet<>();
    //union
    for (int i=0; i<arr1.length; i++) {
        set.add (arr1[i]);
    }
    for (int i=0; i<arr2.length; i++)
        set.add (arr2[i]);
    System.out.println ("Union = " + set.size());
    //intersection
    set.clear();
    for (int i=0; i<arr2.length; i++) {
        if (set.contains (arr2[i]))
            count++;
            set.remove (arr2[i]);
    }
    System.out.println ("Intersection = " + count);
}
```

★ FIND ITINERARY FROM TICKETS
↳ Journey

* We have 4 flight tickets that are :
  - 1st → "Chennai" → "Bengaluru"
  - 2nd → "Mumbai" → "Delhi"
  - 3rd → "Goa" → "Chennai"
  - 4th → "Delhi" → "Goa"

Output: "Mumbai" → "Delhi" → "Goa" → "Chennai" → "Bengaluru"

* We have taken Mumbai as starting point because Mumbai से हमलोग कहीं जा रहे हैं या कि वहाँ जा रहे हैं।

Approach : (from, to)

* starting point : only 'from' exist ✓ to ✗

from	to
Chennai	Bengaluru
Mumbai	Delhi
Goa	Chennai
Delhi	Goa

  ↓
  ऐसी कौन सी जगह है जो to के अंदर exist नहीं करती।

* We will take two maps that are :
  - <from, to> map (from = key)
  - <to, from> RevMap (to = key)

* We will compare both keys i.e., which one key (from) doesn't exist in key (to), so we will get starting point.

// Time Complexity : O(n)

```
public static String getStart (HashMap<String, String> tickets){
    HashMap<String, String> revMap = new HashMap<>();
    for (String key : tickets.keySet()) {
        revMap.put (tickets.get (key), key);
    }
    for (String key : tickets.keySet()) {
        if (!revMap.containsKey (key)) {
            return key; //starting point
        }
    }
    return null;
}
public static void main (String args[]) {
    HashMap <String, String> tickets = new HashMap<>();
                    (from, to)
    tickets.put ("Chennai", "Bengaluru");
    tickets.put ("Mumbai", "Delhi");
    tickets.put ("Goa", "Chennai");
    tickets.put ("Delhi", "Goa");
                  from      to
    String start = getStart (tickets);
    System.out.print (start);
    for (String key : tickets.keySet()) {
        System.out.print ("->" + tickets.get (start));
        start = tickets.get (start);
    }
    System.out.println ();
}
```

Output : Mumbai -> Delhi -> Goa -> Chennai -> Bengaluru

★ LARGEST SUBARRAY WITH 0 SUM

* arr = {15, -2, 2, -8, 1, 7, 10, 23} ⇒ ans = 5

  -2+2-8+1+7

  -8+8

  0.

* arr = {3, 4, 5} ⇒ ans = 0.

Approach: arr = {15, -2, 2, -8, 1, 7, 10} (-7)

(0, i) i=2, j=1

(0, j)

→ Sum [0, i]

→ sum [0, j]

Ex⇒ sum [0, i] = 15

sum [0, j] = 8

⇓

→ sum [i+1, j] = sum [0, j] - sum [0, i] → // 8-15 = -7

↓

(i+1, j)

→ 0 = sum [j] - sum [i]

→ sum (j) = sum (i)

subarray = j - i

subarray ⇒ 2-0 = 2

S1: int sum = 0 → arr = {15, -2, 2, -8, 1, 7, 10}

int len = 0

S2: for (int j = 0 to n)

if (sum exist in map)

len → len, j-i (j-i = 5-0 = 5)

else

map.put (sum, j)

(sum, idx) (i)
(15, 0)
(13, 1)
(7, 3)
(8, 4)
(25, 5)

S3: print len

ans var

// Time Complexity → O(n)

⇒

```
public static void main (String args[]) {
  int arr [] = {15, -2, 2, -8, 1, 7, 10, 23};
  HashMap <Integer, Integer> map = new HashMap<>();
  //(sum, idx)
  int sum = 0;
  int ~~idx~~ len = 0;
  for (int j=0; j < arr.length; j++) {
    sum += arr [j];
    if (map.containsKey (sum)) {
      len = Math.max (len, j - map.get (sum));
    } else {                          (i)
      map.put (sum, j);
    }
  }
  System.out.println ("Largest subarray with sum as 0 => "+len);
}
```

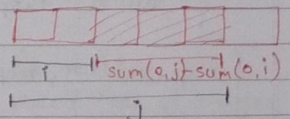
Output: Largest subarray with sum as 0 ⇒ 5

★ SUBARRAY SUM EQUAL TO K

⇒ arr = {1, 2, 3}, K = 3 → return no. of such subarrays

Output: ans = 2 (1,2) (3)

Approach: We know that, ⇒ sum (0, j) - sum (0, i) = sum (i+1, j)



* By above property, we can write another one that we will use in the approach i.e.,

⇒ sum (0, j) - sum (0, i-1) = sum (i, j)

$\Rightarrow$ sum(j) − sum(i) = k
sum(j) = sum(i) + k

* We have to check this sum(j) − k = sum(i) in HashMap

→ arr = {10, 2, -2, -20, 10}, k = -10

(0, 1) by default
(10, 1)
(12, 1)
(-10, 1)

(sum, count)
key, value

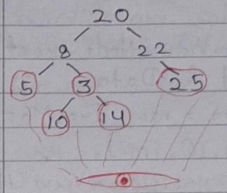
S1: sum = 0 +10 10 +2 12 +(-2) 10 +(-20) -10 +10 0
count = 0 1 2 3

S2: for (j = 0 to n)
if (sum(j) − k) exists in map
ans += map.get(sum-k)
map.put(sum, freq+1).

S3: print count.
answer

```
⇒ public static void main (String args[]) {
    int arr[] = {10, 2, -2, -10, 10};
    int k = -10;
    HashMap<Integer, Integer> map = new HashMap<>();
                // (sum, count)
    int sum = 0;
    int ans = 0;
    for (int j=0; j<arr.length; j++) {
      sum += arr[i];
      if (map.containsKey (sum-k)) {
        ans += map.get (sum-k);
      }
      map.put(sum, map.getOrDefault (sum, 0)+1);
    }
    System.out.println (ans); // 3
}
```


Ques 1. Bottom View of a Binary Tree — The bottom of a binary tree is the set of nodes visible when the tree is viewed from bottom. Given a binary tree, print the bottom view in any order.



⇒ Output: 5 10 3 14 25

We will use the concept that we have used in Top View ques.

```
⇒ static class Node {
    int data;
    int hd;
    Node left, right;
    public Node (int key) {
      this.data = key;
      this.hd = Integer.MAX_VALUE;
      this.left = this.right = null;
    }
}
public static void bottomViewHelper (Node root, int TreeMap<Integer, int[]> m){
    if (root == null)
      return;
//if node for a particular Horizontal Distance (HD) is not present, add to the map.
    if (!m.containsKey (hd))
      m.put (hd, new int[] (root.data, curr));
//compare height for already present node at similar HD.
    else
      int[] p = m.get (hd);
      if (p[1] <= curr)
        p[1] = curr;   p[0] = root.data;
      m.put (hd, p);
```

```
//call for left subtree
bottomViewHelper(root.left, curr+1, hd-1, m);
//call for right subtree
bottomViewHelper(root.right, curr+1, hd+1, m);
}

public static void printBottomView(Node root) {
//map to store HD, Height & Data.
TreeMap<Integer, int[]> m = new TreeMap<>();
bottomViewHelper(root, 0, 0, m);
//prints values stored by printBottomViewUtil()
for(int val[] : m.values()) {
    System.out.print(val[0] + " ");
}
}

public static void main(String args[]) {
Node root = new Node(20);
root.left = new Node(8);
root.right = new Node(22);
root.left.left = new Node(5);
root.left.right = new Node(3);
root.right.left = new Node(4);
root.right.right = new Node(25);
root.left.right.left = new Node(10);
root.left.right.right = new Node(14);

System.out.println("Bottom view of Binary Tree:");
printBottomView(root); //5,10,3,14,4,22,25
}
```


Ques 2. Two Sum - Given an array of integers arr[] & an integer target, return indices of the two numbers such that they add upto target.

* We may assume that each input would have exactly one solution, & we may not use the same element twice.
* We can return the answer in any order.

Input: arr = [2, 7, 11, 15], target = 9
Output: [0, 1] (As arr[0] + arr[1] == 9, we return [0, 1])

⇒

```
public int[] twoSum(int arr[], int target) {
Map<Integer, Integer> visited = new HashMap<>();
for(int i=0; i<arr.length; i++) {
  //diff = given target - number given at ith index.
  int diff = target - arr[i];
  //check if found difference is present in MAP list.
  if(visited.containsKey(diff)) {
    //if difference in map matches with ith index element in arr
    return new int[] {i, visited.get(diff)};
  }
  //add arr element in map to match with future element if
  visited.put(arr[i], i);          forms a pair.
}
//if no matches are found.
return new int[] {0, 0};
}

//write main() by your own.
```

Enroll
Page No: 473
Date: / /

Ques 3 : Sort by Frequency – Given a string s, sort it in decreasing order based on the frequency of the characters. The frequency of a character is the number of times it appears in the string. Return the sorted string. If there are multiple answers, return any of them.

Input: s = "cccaaa" Output: "aaaccc"

⇒ Both 'c' & 'a' appear 3 times, so both "cccaaa" & "aaaccc" are valid answers. Note that "cacaca" is incorrect, as the same characters must be together.

Input: s = "tree" Output: "eert"

⇒ 'e' appears twice while 'r' & 't' both appear once. So, 'e' must appear before both 'r' & 't'. Therefore, "eert" is also a valid answer.

⇒
```
public String frequencySort (String s)
    HashMap<Character, Integer> map = new HashMap<>();
    for (int i=0; i< s.length(); ++i)
        map.put(s.charAt(i), map.getOrDefault(s.charAt(i),0)+1);
    PriorityQueue<Map.Entry<Character, Integer>> pq =
        new PriorityQueue<>((a, b) -> a.getValue()
        == b.getValue? a.getKey() - b.getKey():
        b.getValue() - a.getValue());

    For(Map.Entry<Character, Integer> e : map.entrySet())
        StringBuilder res = new StringBuilder(); pq.add(e);
        while(pq.size() != 0) {
            char ch = pq.poll().getKey();
            int val = map.get(ch);
        while (val != 0){ res.append(ch); val--;}
    return res.toString();
}
```