**LC#141 LINKED LIST CYCLES**

**1)Hashmap:**

Assign head to a temp variable of type listnode after that push into the map if it not pushed previously . Return true if detected or return false outside the loop.

/\*\*

\* class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }\*/

public class Solution {

public boolean hasCycle(ListNode head) {

Map <ListNode,Integer> hm = new HashMap();

ListNode temp = head;

while(temp!=null)

{

if(!hm.containsKey(temp))

{

hm.put(temp,1);

temp = temp.next;

}

else

return true; }

return false;

}}

**2)using hashset:**

**Prevents duplicates**

/\*\*

 \* class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) {

 \*         val = x;

 \*         next = null;

 \*     }

 \* }

 \*/

public class Solution {

    public boolean hasCycle(ListNode head) {

        Set <ListNode> hs = new HashSet();

        ListNode temp = head;

        while(temp!=null)

        {

            if(hs.contains(temp))

                return true;

            hs.add(temp);

            temp = temp.next;

        }

        return false;}}

**3)TORTOISE HARE METHOD(OPTIMAL)**

ListNode fast = head , slow = head;

While(fast!=null){

Fast = fast.next;

If(fast!=null){

Slow = slow.next;

Fast = fast.next;

}

If(slow==fast)//loop exist

Return true;

}

Return false;

**LC#287:FIND THE REPEATING ELEMENT**

->no array modification and op should be **inplace**

**1)so using hashset can solve the problem passes , but not allowed❌**

class Solution {

public int findDuplicate(int[] nums) {

Set <Integer> hs = new HashSet();

for(int n: nums)

{

if(hs.contains(n))

return n;

hs.add(n);

}

return 1;

}

}

**2)array modification**

class Solution {

public int findDuplicate(int[] nums) {

for(int i = 0 ;i < nums.length ;i++)

{

int val = Math.abs(nums[i]);

if(nums[val]<0)

{

return val;

}

nums[val]=-nums[val];

}

return 0; }}

**3)OPTIMAL(TORTOISE HARE LINKED LIST)**

class Solution {

public int findDuplicate(int[] nums) {

int fast = 0, slow = 0;

// Phase 1: Find intersection point

do {

fast = nums[nums[fast]];

slow = nums[slow];

} while (fast != slow);

// Phase 2: Find cycle entrance (duplicate number)

int slow2 = 0;

while (slow != slow2) {

slow = nums[slow];

slow2 = nums[slow2];

}

return slow;

}

}

**LC#202:HAPPY NUMBER**

**APPROACH 1: hashset**

class Solution {

public boolean isHappy(int n) {

Set<Integer> seen = new HashSet<>();

while (n != 1 && !seen.contains(n)) {

seen.add(n);

n = getNext(n);

}

return n == 1;

}

private int getNext(int n) {

int sum = 0;

while (n > 0) {

int d = n % 10;

sum += d \* d;

n /= 10;

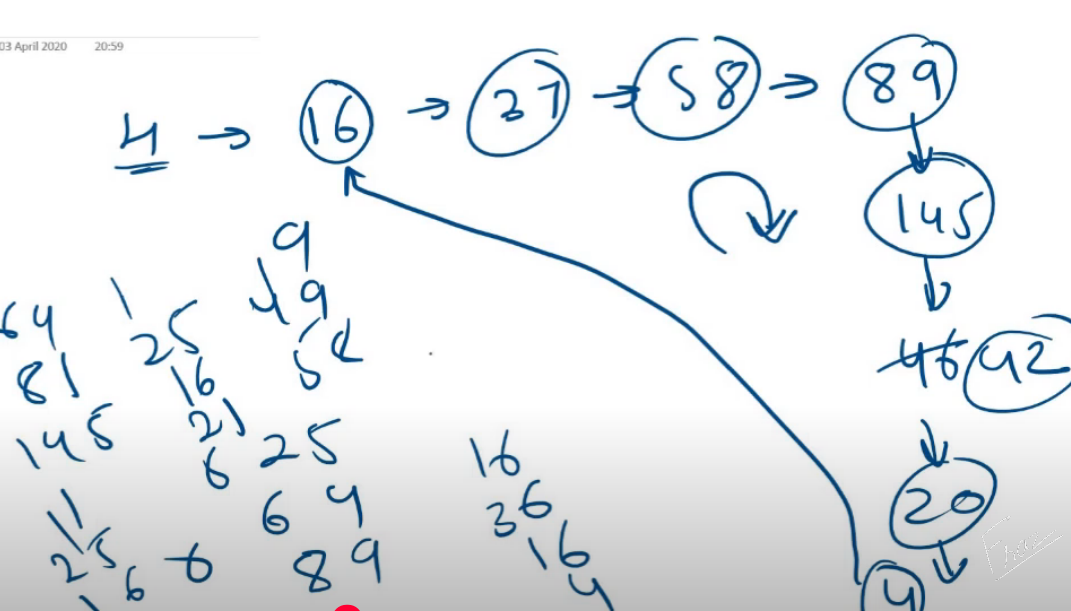
}

return sum;

}

}

**3)TORTOISE HARE DETECTION**



class Solution {

int compute(int n)

{

int res = 0;

while(n>0)

{

int d = n%10;

res = res +(d\*d);

n/=10;

}

return res;

}

public boolean isHappy(int n) {

int fast = n , slow = n;

do{

fast = compute(fast);

fast = compute(fast);

slow = compute(slow);

}while(slow!=fast);

return slow==1;

}}