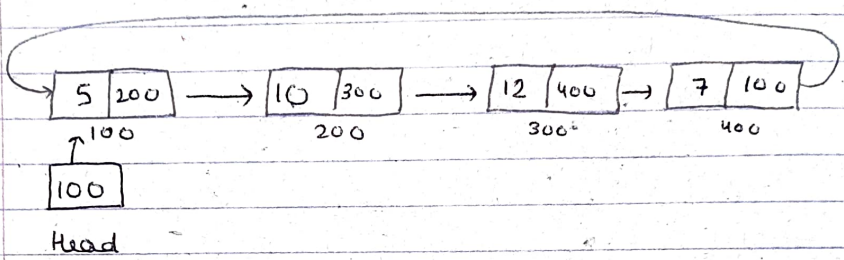
Date: / /
Page:

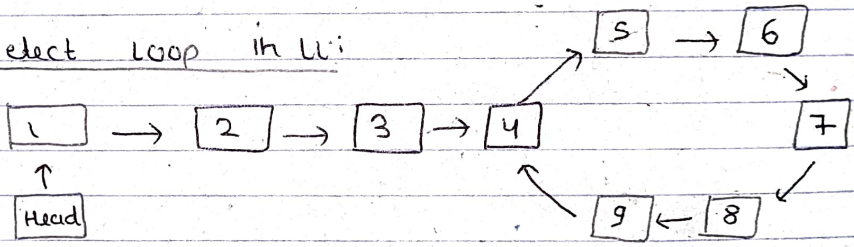
Day - 121

Linked List - 7

* **Circular Linked List:**



* **Detect Loop in LL:**



⇒ To find out the cycle, we can traverse the LL and store the data if any data will repeat that means cycle exist.

⇒ But here there is a possibility that the data is repeating.

⇒ But the thing that will not repeat is the address.

Code

```
Node *curr = head;
vector<Node*> visited;
while (curr != NULL){
        if (check(visited, curr))
                return 1;
        visited.push_back(curr);
        curr = curr->next;
}; return 0; }

bool check(vector<Node*> visited, Node *curr){
        for(i=0; i< visited.size(); i++){
                if (visited[i] == curr)
                        return 1;
        }  ~~visited~~
        return 0;
}
```

⇉ Above code will give Time Limit Exceeded.

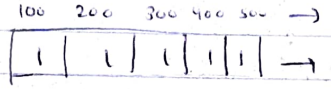
⇉ Because we are solving the question in $O(n^2)$ by using above approach.

⇉ To solve the question in $O(n)$, we will use unordered map.

⇉ Unordered map is just like a dictionary where we can store data in (key, value) pair.

⇉ And we can search any data by using key in $O(1)$ time.

⇒ So, it will store values like this —



Code

```
Node *curr = head;
unordered_map <Node *, bool> visited;
while(curr != NULL){
        if( visited[curr] == 1 )
                return 1;
        visited[curr] = 1;
        curr = curr → next;
}
return 0;
```

⇒ ~~Brute~~ This code taking O(n) space complexity.

⇒ Now we have to solve this question in O(1) space complexity.

⇒ Now we will use slow & fast approach.

⇒ So, if both the pointers meet then the loop exist otherwise not.

⇒ Everytime distance reduces.

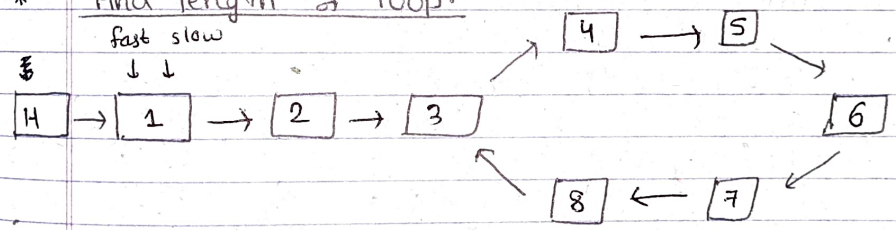
⇒ So, ~~after doing~~ while traversing if there is a loop then the pointers will meet.

Code

```
Node * slow = head;
Node * fast = head;

while (fast != NULL && fast → next != NULL){
        slow = slow → next;
        fast = fast → next → next;
        if( slow == fast)
            return 1;
}
return 0;
```

* Find length of loop:



- We will take two pointers fast & slow.
- First, we will check, if there is a loop in LL or not.
- If there is a loop then we will count the nodes.
- By using the current position of fast & slow, we will move slow & maintain a count variable.
- And increase the count until slow reached fast.

Code

```
Node * slow = head, * fast = head;
while (fast != NULL && fast -> next != NULL){
        slow = slow -> next;
        fast = fast -> next -> next;
        if(slow == fast)
            break;
}
if( fast == NULL || fast -> next == NULL)
        return 0;
int count = 1;
slow = slow -> next;
while( slow != fast) {
    count ++;
    slow = slow -> next;
}
return count;
```