4th feb 2023

FIRST:-

Find pairs with given sum in doubly linked list : : Easy

Given a sorted doubly linked list of positive distinct elements, the task is to find pairs in a doubly-linked list whose sum is equal to given value **target**.

Example 1:

```
Input:
1 <-> 2 <-> 4 <-> 5 <-> 6 <-> 8 <-> 9
target = 7
Output: (1, 6), (2,5)
Explanation: We can see that there are two pairs
(1, 6) and (2,5) with sum 7.
```

Example 2:

```
Input:
1 <-> 5 <-> 6
target = 6
Output: (1,5)
Explanation: We can see that there is one pairs  (1, 5)
with sum 6.
```

Your Task:
You don't need to read input or print anything. Your task is to complete the function **findPairsWithGivenSum()** which takes head node of the doubly linked list and an integer target as input parameter and returns an array of pairs. If there is no such pair return empty array.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(1)

Constraints:

$1 <= N <= 10^5$

$1 <= target <= 10^5$

CODE SECTION :-

```
vector<pair<int, int>> findPairsWithGivenSum(Node *head, int target)
   {
       // code here
       vector<pair<int, int>> ans;

       Node *p = head;
       Node *q = head->next;

       while (q->next)
       {

           q = q->next;
       }

       while (p != q)
       {

           if (p->data + q->data == target)
           {
               ans.push_back(make_pair(p->data, q->data));
               p = p->next;
           }

           else
           {

               if (p->data + q->data < target)
               {

                   p = p->next;
               }
               else
               {
                   q = q->prev;
               }
           }
       }

       return ans;
   }
```

SECOND :-

Max Sum without Adjacents : : Easy

Given an array **Arr** of size **N** containing positive integers. Find the maximum sum of a subsequence such that no two numbers in the sequence should be adjacent in the array.

Example 1:

```
Input:
N = 6
Arr[] = {5, 5, 10, 100, 10, 5}
Output: 110
Explanation: If you take indices 0, 3
and 5, then Arr[0]+Arr[3]+Arr[5] =
5+100+5 = 110.
```

Example 2:

```
Input:
N = 4
Arr[] = {3, 2, 7, 10}
Output: 13
Explanation: 3 and 10 forms a non
continuous  subsequence with maximum
sum.
```

Your Task:

You don't need to read input or print anything. Your task is to complete the function **findMaxSum()** which takes the array of integers **arr** and **n** as parameters and returns an integer denoting the answer. It is guaranteed that your answer will always fit in the 32-bit integer.

Expected Time Complexity: O(N)
Expected Auxiliary Space: O(1)

Constraints:
$1 \leq N \leq 10^6$
$1 \leq Arr_i \leq 10^7$

CODE SECTION:-

```
 int findMaxSum(int *arr, int n)
    {
        // code here (USING DP)
        if (n < 1)
        {
            return 0;
        }
        if (n == 1)
            return arr[0];
        /*

           we're just checking if (n>=2),then changing the value of arr[1],so
that when
           they will have to compared,it will compare the previous element and
the sum of current and 2nd previous
           element..

        */
        if (n >= 2)
        {
            arr[1] = max(arr[0], arr[1]);
        }
        for (int i = 2; i < n; i++)
        {
            /*
               changing the value of arrays from second position,so that each
previous element wll containt the
               maximum  sum till their position
            */
            arr[i] = max(arr[i] + arr[i - 2], arr[i - 1]);
        }
        return arr[n - 1];
    }
```

-: DONE FOR THE DAY :-