LRU CACHE

We have to implement a class LRU Cache where constructor takes input of capacity.

Values are stored in key value pairs If while inserting, we reach max capacity, we removed the least recently used value and add new one.

Fetching a value using its key counts as a use so it is brought to front of cache.

We use a DLL (head and tail) along with a map to facilitate insertion and getting in O(1).

At each insert step, we check if key already has a pair in the map. If it doesn't then we add it before the tail and store its address in the map.

At each get step, we look for key in map, if we find it, we go to its address and put it at the front.

Pseudocode :

```
struct Node {
    public :
     int key ;
     int value ;
     Node * next ;
     Node * back ;

     Node(int key , int value) {
         key = key ;
         value = value ;
         next = NULL ;
         back = NULL ;
     }
} ;

class LRUCache {
    public :
     Node * head ;
     Node * tail ;
     int capacity
     map <int , Node* > mpp ;
     LRUCache(int c) {
         capacity = c ;
         head -> next = tail ;
         tail -> back = head ;
     }

     get (int key) {
         if ( mpp.find(key) == mpp.end()) {
             return -1 ;
         } else {
```

```
            Node * n = mpp[key];
            removeNode(n);
            insertAtHead(head, n);
            return n->value;
        }
    }

    push(int key, int value) {
        if(mpp.find(key) == mpp.end()) {
            if(mpp.size() == capacity) {
                mpp.del(tail->back->key);
                deleteNode(tail->back);
            }
            Node * nN = new Node(key, value);
            insertAtHead(head, nN);
            mpp[key] = nN;
        } else {
            mpp[key]->value = value;
            Node * nN = mpp[key];
            deleteNode(nN);
            insertAtHead(head, nN);
        }
    }
```