正则表达式匹配

In [ ]:

```python
# 10.正则表达式匹配
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
        meno = {}
        def dp(s,i,p,j):
            ls = len(s)
            lp = len(p)
            if lp == j: return ls == i
            if ls == i:
                if (lp-j) % 2 == 1: return False
                while j < lp-1:
                    if p[j+1] != '*': return False
                    j += 2
                return True
            res = False
            key = str(i)+','+str(j)
            if key in meno: return meno[key]
            if s[i] == p[j] or p[j] == '.':
                if j < lp-1 and p[j+1] == '*':
                    res = dp(s,i,p,j+2) or dp(s,i+1,p,j)
                else:
                    res = dp(s,i+1,p,j+1)
            else:
                if j < lp-1 and p[j+1] == '*':
                    res = dp(s,i,p,j+2)
                else:
                    return False
            meno[key] = res
            return res
        return dp(s,0,p,0)
```

LRU

In [ ]:

```python
class ListNode:
    def __init__(self,key=0,val=0):
        self.key = key
        self.val = val
        self.next = None
        self.pre = None
class LRUCache:

    def __init__(self, capacity: int):
        self.capacity = capacity
        self.hashMap = {}
        self.head = ListNode()
        self.tail = ListNode()
        self.head.next = self.tail
        self.tail.pre = self.head

    def removeNode(self,node):
        node.next.pre = node.pre
        node.pre.next = node.next

    def addNodetoTail(self,node):
        self.tail.pre.next = node
        node.next = self.tail
        node.pre = self.tail.pre
        self.tail.pre = node

    def moveTotail(self,node):
        self.removeNode(node)
        self.addNodetoTail(node)

    def get(self, key: int) -> int:
        if key not in self.hashMap: return -1
        node = self.hashMap[key]
        self.moveTotail(node)
        return node.val

    def put(self, key: int, value: int) -> None:
        if key in self.hashMap:
            node = self.hashMap[key]
            node.val = value
            self.moveTotail(node)
            return
        if len(self.hashMap) == self.capacity:
            del self.hashMap[self.head.next.key]
            self.removeNode(self.head.next)
        node = ListNode(key,value)
        self.addNodetoTail(node)
        self.hashMap[key] = node



# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```