

正则表达式匹配

```
In [ ]: 1 # 10. 正则表达式匹配
2 class Solution:
3     def isMatch(self, s: str, p: str) -> bool:
4         memo = {}
5         def dp(s, i, p, j):
6             ls = len(s)
7             lp = len(p)
8             if lp == j: return ls == i
9             if ls == i:
10                 if (lp-j) % 2 == 1: return False
11                 while j < lp-1:
12                     if p[j+1] != '*': return False
13                     j += 2
14                 return True
15             res = False
16             key = str(i)+'_'+str(j)
17             if key in memo: return memo[key]
18             if s[i] == p[j] or p[j] == '.':
19                 if j < lp-1 and p[j+1] == '*':
20                     res = dp(s, i, p, j+2) or dp(s, i+1, p, j)
21                 else:
22                     res = dp(s, i+1, p, j+1)
23             else:
24                 if j < lp-1 and p[j+1] == '*':
25                     res = dp(s, i, p, j+2)
26                 else:
27                     return False
28             memo[key] = res
29             return res
30         return dp(s, 0, p, 0)
```

LRU

In []:

```
1 class ListNode:
2     def __init__(self, key=0, val=0):
3         self.key = key
4         self.val = val
5         self.next = None
6         self.pre = None
7 class LRUCache:
8
9     def __init__(self, capacity: int):
10        self.capacity = capacity
11        self.hashMap = {}
12        self.head = ListNode()
13        self.tail = ListNode()
14        self.head.next = self.tail
15        self.tail.pre = self.head
16
17    def removeNode(self, node):
18        node.next.pre = node.pre
19        node.pre.next = node.next
20
21    def addNodetoTail(self, node):
22        self.tail.pre.next = node
23        node.next = self.tail
24        node.pre = self.tail.pre
25        self.tail.pre = node
26
27    def moveTotail(self, node):
28        self.removeNode(node)
29        self.addNodetoTail(node)
30
31    def get(self, key: int) -> int:
32        if key not in self.hashMap: return -1
33        node = self.hashMap[key]
34        self.moveTotail(node)
35        return node.val
36
37    def put(self, key: int, value: int) -> None:
38        if key in self.hashMap:
39            node = self.hashMap[key]
40            node.val = value
41            self.moveTotail(node)
42            return
43        if len(self.hashMap) == self.capacity:
44            del self.hashMap[self.head.next.key]
45            self.removeNode(self.head.next)
46        node = ListNode(key, value)
47        self.addNodetoTail(node)
48        self.hashMap[key] = node
49
50
51
52
53 # Your LRUCache object will be instantiated and called as such:
54 # obj = LRUCache(capacity)
55 # param_1 = obj.get(key)
56 # obj.put(key, value)
```