hw#4_hui_duan

October 6, 2019

1 401. Binary Watch

```
In [3]: class Solution(object):
            def readBinaryWatch(self, num):
                times = []
                for h in range(12):
                    for m in range(60):
                        if(bin(h) + bin(m)).count('1')==num:
                            times.append('%d:%02d' % (h, m))
                return times
```

2 39. Combination Sum

```
In [5]: class Solution(object):
            def combinationSum(self, candidates, target):
                res = []
                candidates.sort()
                self.dfs(candidates, target, 0, [], res)
                return res


            def dfs(self, nums, target, index, path, res):
                for i in range(index,len(nums)):
                    remain = target - nums[i]
                    if target < 0:
                        break
                    elif target == 0:
                        res.append(path + [nums[i]])
                        break
                    self.dfs(nums, remain, i, path + [nums[i]], res)
```

3 46. Permutations

```
In [6]: class Solution(object):
            def permute(self, nums):
```

```
        res = []
        used = [0] * len(nums)
        if len(nums) == 0: return res

        def dfs(res,path,nums,used):
            if len(path) == len(nums) and path not in nums:
                res.append(path)
            for i in range(len(nums)):
                if not used[i]:
                    used[i] = 1
                    self.dfs(path + [nums[i]])
                    used[i] = 0

        self.dfs(res,[],nums,used)

        return res
```

4 47. Permutations II

```
In [11]: def backtrack(path):

            res = []
            used = [0] * len(nums)
            nums.sort()

            def dfs(res,path,nums,used):
                if len(path) == len(num) and path not in nums:
                    res.append(path)
                for i in range(len(nums)):
                    if not used[i]:
                        used[i] = 1
                        self.backtrack(path+[nums[i]])
                        used[i] = 0
                    elif i > 0 and used[i - 1] == False and nums[i - 1] == nums[i]:
                        continue
            self.dfs(res,[],nums,used)
            return res
```

5 52. N-Queens II

```
In [15]: class Solution:
            def totalNQueens(self, n: int) -> int:
                board=[-1 for i in range(n)]
                res = 0
                self.dfs(0,board,res)
                return res
```

```
            def dfs(self,col,board,res):
                if col==n:
                    res += 1
                for row in range(n):
                    board[col]=row
                    if self.check(col,row,board):
                        self.dfs(col+1,board,res)
                    board[col] = -1

            def check(self,k,j,board):
                for i in range(k):
                    if board[i]==j or abs(k-i)==abs(board[i]-j):
                        return False
                return True
```

6 232. Implement Queue using Stacks

```
In [16]: class Queue(object):
            def __init__(self):
                self.inStack = []
                self.outStack = []

            def push(self, x):
                self.inStack.append(x)

            def pop(self):
                self.peek()
                self.outStack.pop()

            def peek(self):
                if not self.outStack:
                    while self.inStack:
                        self.outStack.append(self.inStack.pop())
                return self.outStack[-1]

            def empty(self):
                return not self.inStack and not self.outStack
```