139. Word Break

We will use DP to check whether we could achieve this position. When dp[i] and s[j:i] in wordDict, dp[j] = True. TC is O(n^2), SC is O(O(n))

```
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        wordDict = set(wordDict)
        length = len(s)
        dp = [False] * (length + 1)
        dp[0] = True
        for i in range(1, length + 1):
          for j in range(0, i):
            if dp[j] and s[j:i] in wordDict:
              dp[i] = True
              break
        return dp[length]
```

140. Word Break II

We will get all possibilities of current substring and use a memo to record all possible results. We will check every word in wordsDict and decompose our string if current string starts with word.

```
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> List[str]:
        memo = {}
        wordDictSet = set(wordDict)
        def dfs(s):
          if s in memo:
            return memo[s]
          if not s or len(s) == 0:
            return ['']
          ret = []
          for word in wordDictSet:
            if word == s[:len(word)]:
              res = dfs(s[len(word):])
              for w in res:
                if w:
                  ret.append(word + ' ' + w)
                else:
                  ret.append(word)
          memo[s] = ret
          return ret
        return dfs(s)
```

133. Clone Graph

We will use dfs to clone the graph, we will use memo to memorize value and their associated nodes. If the node we have created, we will return the node, if not, we will create new node. TC is O(n), SC is O(n)

```
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        memo = {}
        def dfs(node):
          if not node:
            return None
          if node.val in memo:
            return memo[node.val]
          new_node = Node(node.val, [])
          memo[node.val] = new_node
          for n in node.neighbors:
            new_node.neighbors.append(dfs(n))
          return new_node
        return dfs(node)
```

138. Copy List with Random Pointer

We will go through our original linked list and memorize our value-node. Then we will go through our linked list again and link new nodes according to their values. TC is O(n), SC is O(n)

```
class Solution:
    def copyRandomList(self, head: 'Node') -> 'Node':
        if not head:
          return head

        memo = {}
        head_memo = head

        while head:
          memo[head.val] = Node(head.val, None, None)
          head = head.next

        head = head_memo
        new_head = memo[head.val]
        new_head_memo = new_head
        while head:
          if head.next:
            new_head.next = memo[head.next.val]
          if head.random:
            new_head.random = memo[head.random.val]
          head = head.next
          new_head = new_head.next
        return new_head_memo
```

200. Number of Islands

We will use bfs to count islands. We will iterate through all nodes in grid. When there is connected node we will reset them to ‘0’ until there are all ‘zero’ around. TC is O(n), SC is O(1)

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
      if not grid or not grid[0]:
        return 0
      rows = len(grid)
      cols = len(grid[0])
      count = 0
      for i in range(rows):
        for j in range(cols):
          if grid[i][j] == '1':
            count += 1
            grid[i][j] = '0'
            cords = [[i, j]]
            while cords:
              new_cords = []
              for _i, _j in cords:
                for d_i, d_j in [[0, 1], [0, -1], [1, 0], [-1, 0]]:
                  new_i, new_j = _i + d_i, _j + d_j
                  if 0 <= new_i < rows and  0 <= new_j < cols and grid[new_i][new_j] == '1':
                    grid[new_i][new_j] = '0'
                    new_cords.append([new_i, new_j])
              cords = new_cords

      return count
```