210. Course Schedule II

```
from collections import defaultdict
class Solution:
    def findOrder(self, numCourses: int, prerequisites: List[List[int]]) -> List[int]:
        indegree = defaultdict(set)
        outdegree = defaultdict(set)
        for s, f in prerequisites:
          indegree[s].add(f)
          outdegree[f].add(s)
        begin = set(range(numCourses)) - set(indegree.keys())
        result = list(begin)
        total = len(begin)
        while begin:
          next_ite = set()
          for i in begin:
            if i in outdegree:
              for j in outdegree[i]:
                indegree[j].remove(i)
                if len(indegree[j]) == 0:
                  next_ite.add(j)
                  total += 1
                  result.append(j)
          begin = next_ite
        return result if total == numCourses else []
```

301. Remove Invalid Parentheses

```
class Solution:
    def removeInvalidParentheses(self, s: str) -> List[str]:
        result = []
        self.getRemovedParenthesesNum(s, '(', ')', 0, 0, result)
        return result


    def getRemovedParenthesesNum(self, s, p, q, last_i, last_j, result):
        max_diff = 0
        length = len(s)
        for i in range(last_i, length):
            if s[i] == p:
                max_diff += 1
            elif s[i] == q:
                max_diff -= 1
                if max_diff < 0:
                    for j in range(last_j, i + 1):
```

```
                if s[j] == q and (j == last_j or s[j - 1] != q):
                    self.getRemovedParenthesesNum(s[:j] + s[j + 1:], p, q, i, j, result)
            return
        reversed_s = s[::-1]
        if p == '(':
            self.getRemovedParenthesesNum(reversed_s, ')', '(', 0, 0, result)
        elif p == ')':
            print(reversed_s)
            result.append(reversed_s)
```

20. Valid Parentheses

```
class Solution(object):
    def isValid(self, s):
        """
        :type s: str
        :rtype: bool
        """
        table = {'(': ')', '{': '}', '[': ']'}
        length = len(s)
        stack = []
        for i in range(length):
            if s[i] in '({[':
                stack.append(s[i])
            else:
                if not stack or table[stack.pop()] != s[i]:
                    return False
        return len(stack) == 0
```

32. Longest Valid Parentheses

We will memorize all invalid parentheses' indexes and remove all valid parentheses from stack. Then we only need to get the maximum gap between indexes in our stack. TC is O(n), SC is O(n)

```
class Solution:
    def longestValidParentheses(self, s: str) -> int:
        stack = []
        left = 0
        for i, c in enumerate(s):
            if c == '(':
                stack.append(i)
                left += 1
            else:
                if left > 0:
                    stack.pop()
                    left -= 1
                else:
                    stack.append(i)
        if not stack:
            return len(s)
        prev = -1
        cur = 0
        stack.append(len(s))
        for i in stack:
            cur = max(cur, i - prev - 1)
            prev = i
        return cur
```

678. Valid Parenthesis String
We will use cmin and cmax to memorize maximum of close parenthesis and minimum least maximum we need. Cmax couldn't be less than zero and cmin should be zero in the end. TC is O(n), SC is O(1)

```
class Solution:
    def checkValidString(self, s: str) -> bool:
        cmin, cmax = 0, 0
        for i in s:
            if i == '(':
                cmax += 1
                cmin += 1
            elif i == ')':
                cmax -= 1
                cmin = max(0, cmin - 1)
            else:
                cmax += 1
                cmin = max(0, cmin - 1)
            if cmax < 0:
                return False
        return cmin == 0
```

921. Minimum Add to Make Parentheses Valid
Two pass:

```
class Solution:
    def minAddToMakeValid(self, S: str) -> int:
        def minParentheses(s, p):
            stack = 0
            max_stack = 0
            for i in s:
                if i == p:
                    stack += 1
                else:
                    stack -= 1
                    if stack < 0:
                        max_stack += 1
                        stack = 0
            return max_stack
        return minParentheses(S, '(') + minParentheses(S[::-1], ')')
```

One pass:

```
class Solution:
    def minAddToMakeValid(self, S: str) -> int:
        l, mis_match = 0, 0
        for i in S:
            if i == '(':
                l += 1
            elif l < 1:
                mis_match += 1
            else:
                l -= 1
        return mis_match + l
```