Notebook

August 6, 2024

[ ]:
```
# https://leetcode.com/problems/valid-parentheses/description/
# ([{}]) -> True ; ([{]}) -> False

class Solution:
    def isValid(self, s: str) -> bool:
        stack = []
        for c in s:
            if c == ")" and stack and stack[-1] == "(":
                stack.pop()
            elif c == "]" and stack and stack[-1] == "[":
                stack.pop()
            elif c == "}" and stack and stack[-1] == "{":
                stack.pop()
            else:
                stack.append(c)

        return len(stack) == 0

# https://leetcode.com/problems/generate-parentheses/description/
# n = 3 -> ["((()))","(()())","(())()","()(())","()()()"]

class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        ans = []
        def dfs(open, close, s):
            if len(s) == 2*n:
                ans.append(s)
                return

            if open < n:
                dfs(open+1, close, s+'(')
            if close < open:
                dfs(open, close+1, s+')')
        dfs(0, 0, '')
        return ans
```

[ ]:
```
# https://leetcode.com/problems/number-of-atoms/description/
import collections

class Solution:
    def countOfAtoms(self, formula: str) -> str:
        dic, coeff = collections.defaultdict(int), 1
        stack, elem = [], ""
        cnt, i = 0, 0

        for c in formula[::-1]:
            if c.isdigit():
                cnt += int(c) * (10 ** i)
                i += 1
            elif c == ")":
                stack.append(cnt or 1)  # Handle the case where cnt is 0
                coeff *= stack[-1]
                i = cnt = 0
            elif c == "(":
                coeff //= stack.pop()
                i = cnt = 0
            elif c.isupper():
                elem += c
                dic[elem[::-1]] += (cnt or 1) * coeff
                elem = ""
                i = cnt = 0
            elif c.islower():
                elem += c

        return "".join(k + (str(v) if v > 1 else "") for k, v in sorted(dic.
↪items()))
```

[ ]:
```
# https://leetcode.com/problems/word-break/description/

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

# https://leetcode.com/problems/word-break-ii/description/
```

```python
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> List[str]:
        memo = {}
        def dfs(s, wordDict):
            if s in memo: return memo[s]
            if not s : return []
            res = []
            for word in wordDict:
                if not s.startswith(word): continue
                if len(word) == len(s): res.append(word)
                else:
                    resultOfRest = dfs(s[len(word):], wordDict)
                    for item in resultOfRest:
                        item = word + ' ' + item
                        res.append(item)
            memo[s] = res
            return res
        return dfs(s, wordDict)
```