Notebook

August 2, 2024

```
[ ]: from typing import List
```

```
[ ]: #### prime
     #### https://www.geeksforgeeks.org/problems/sum-of-prime4751/1
     #### given a target number, return prime numbers that sum to the target number

     class Solution:
         def countPrimes(self, n: int) -> int:
             if n < 3: return 0
             primes = [True] * n
             primes[0] = primes[1] = False
             for i in range(2, int(n**0.5)+1):
                 if primes[i]:
                     for j in range(i*i, n, i):
                         primes[j] = False
                     # primes[i*i:n:i] = [False] * len(primes[i*i:n:i])

             return sum(primes)
```

```
[ ]: ### https://leetcode.com/problems/
      ↪count-k-subsequences-of-a-string-with-maximum-beauty/description/
     ### Hard, Google , Math

     """
     suppose we have s = "aabbcc", k = 3
     then we can have 2 * 2 * 2 = 8 subsequences

     Now s = "aabbcc", k = 2
     combination of 3 to fill two slots 3C2 = 3 times (2*2) = 3 * 4 = 12
     refer below explanation - https://leetcode.com/problems/
      ↪count-k-subsequences-of-a-string-with-maximum-beauty/solutions/3993253/
      ↪pictures-greedy-math-easy-to-understand-guaranteed/

     """

     from math import comb
     from typing import Counter
```

```
class Solution:
    def countKSubsequencesWithMaxBeauty(self, s: str, k: int) -> int:
        mod = 10 **9 + 7
        counter = Counter(s)
        if len(counter) < k : return 0
        freq = Counter(counter.values())
        pairs = list(sorted(freq.items(), reverse=True))
        ans = 1
        for fc, occ in pairs:
            if occ <= k:
                ans = (ans * pow(fc, occ, mod)) % mod
                k -= occ
            else:
                ans = (ans * comb(occ, k) * pow(fc, k, mod)) % mod
                break

        return ans % mod
```

[ ]:
```
### https://leetcode.com/problems/minimum-number-of-k-consecutive-bit-flips/
### Hard

class Solution:
    def minKBitFlips(self, nums, k):
        n = len(nums)
        flipped = 0
        ans = 0
        isFlipped = [0] * n

        for i in range(n):
            if i >= k:
                flipped ^= isFlipped[i - k]

            if flipped == nums[i]:
                if i + k > n:
                    return -1
                isFlipped[i] = 1
                flipped ^= 1
                ans += 1

        return ans
```

[ ]:
```
# https://leetcode.com/problems/count-subarrays-with-fixed-bounds/description/
# Hard, Uber

"""
```

```
Input: nums = [1,3,5,2,7,5], minK = 1, maxK = 5
Output: 2
Explanation: The fixed-bound subarrays are [1,3,5] and [1,3,5,2].
"""

class Solution:
    def countSubarrays(self, A: List[int], minK: int, maxK: int) -> int:
        res = 0
        jmin = jmax = jbad = -1
        for i,a in enumerate(A):
            if not minK <= a <= maxK: jbad = i
            if a == minK: jmin = i
            if a == maxK: jmax = i
            res += max(0, min(jmin, jmax) - jbad)
        return res
```

[ ]:
```
# https://leetcode.com/problems/number-of-atoms/description/
# Important
# Start from end and keep track of the count of each element
"""
Input: formula = "K4(ON(SO3)2)2"
Output: "K4N2O14S4"
Explanation: The count of elements are {'K': 4, 'N': 2, 'O': 14, 'S': 4}.
"""

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)
                coeff *= cnt
                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 > 1 and v or "") for k, v in sorted(dic.
 ↪items()))
```

[ ]:
```
# https://leetcode.com/problems/trapping-rain-water/description/
# create two arrays to store left max and right max
# then iterate over the array and calculate the trapped water with min of left␣
 ↪and right - height[i]

class Solution:
    def trap(self, height: List[int]) -> int:
        ans, n = 0, len(height)
        l, r = [0]*n, [0]*n
        l[0] = height[0]
        for i in range(1, n):
            l[i] = max(l[i-1], height[i])
        r[n-1] = height[n-1]
        for i in range(n-2, -1, -1):
            r[i] = max(r[i+1], height[i])
        for i in range(n):
            ans += min(l[i], r[i]) - height[i]
        return ans

# https://leetcode.com/problems/trapping-rain-water-ii/description/

# sink boundary cells first and we can trap water only in the middle cells␣
 ↪which are surrounded by cell with higher height

class Solution:
    def trapRainWater(self, heightMap: List[List[int]]) -> int:
        if not heightMap or not heightMap[0]:
            return 0


                # Initial
                # Board cells cannot trap the water
        m, n = len(heightMap), len(heightMap[0])
        if m < 3 or n < 3:
            return 0


                # Add Board cells first
        heap = []
        for i in range(m):
            for j in range(n):
                if i == 0 or i == m - 1 or j == 0 or j == n - 1:
                    heapq.heappush(heap, (heightMap[i][j], i, j))
                    heightMap[i][j] = -1


                # Start from level 0
        level, res = 0, 0

        while heap:
            height, x, y = heapq.heappop(heap)
            level = max(height, level)

            for i, j in [(x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)]:
                if 0 <= i < m and 0 <= j < n and heightMap[i][j] != -1:
                    heapq.heappush(heap, (heightMap[i][j], i, j))

                                        # If cell's height smaller than the␣
 ↪level, then it can trap the rain water
                    if heightMap[i][j] < level:
                        res += level - heightMap[i][j]

                                        # Set the height to -1 if the cell is␣
 ↪visited
                    heightMap[i][j] = -1

        return res


# https://leetcode.com/problems/container-with-most-water/description/
# Input: height = [1,8,6,2,5,4,8,3,7]   Output: 49

# Two pointer approach
class Solution:
    def maxArea(self, height: List[int]) -> int:
        l, r = 0, len(height)-1
        ans = 0
        while l < r:
            ans = max(ans, min(height[l], height[r]) * (r-l))
            if height[l] < height[r]: l += 1
            else : r -= 1
        return ans

# https://leetcode.com/problems/pour-water/description/
```

[ ]:

```
# https://leetcode.com/problems/
↪maximum-area-of-a-piece-of-cake-after-horizontal-and-vertical-cuts/
↪description/
# Input: h = 5, w = 4, horizontalCuts = [1,2,4], verticalCuts = [1,3]
# Output: 4

# simpliy sort the horizontal and vertical cuts and find the max difference␣
↪between consecutive cuts in both horizontal and vertical cuts
# ans is product of max horizontal and vertical cut

class Solution:
    def maxArea(self, h: int, w: int, hc: List[int], vc: List[int]) -> int:
        hc = [0] + sorted(hc) + [h]
        vc = [0] + sorted(vc) + [w]

        maxWidth = max([hc[i+1]-hc[i] for i in range(len(hc)-1)])
        maxHeight = max([vc[i+1]-vc[i] for i in range(len(vc)-1)])

        return (maxWidth * maxHeight) % ((10**9)+7)
```

[ ]:
```
# https://github.com/doocs/leetcode/blob/main/solution/0200-0299/0271.
↪Encode%20and%20Decode%20Strings/README_EN.md
# encode list of strings to a single string and decode it back to list of␣
↪strings

# during encoding we encode the length of string in fixed 4 digit format and␣
↪then the string
class Codec:
    def encode(self, strs: List[str]) -> str:
        """Encodes a list of strings to a single string."""
        ans = []
        for s in strs:
            ans.append('{:4}'.format(len(s)) + s)
        return ''.join(ans)

    def decode(self, s: str) -> List[str]:
        """Decodes a single string to a list of strings."""
        ans = []
        i, n = 0, len(s)
        while i < n:
            size = int(s[i : i + 4])
            i += 4
            ans.append(s[i : i + size])
            i += size
        return ans
```

[5]:
```
print('{:1}'.format("hello"))
```

hello

```
[ ]: # Calculator problems
     # https://leetcode.com/problems/basic-calculator/description/
     # Input: s = "(1+(4+5+2)-3)+(6+8)"    Output: 23

     # tricky question, keep track of sign and number and use stack to keep track of␣
      ↪previous results
     class Solution:
         def calculate(self, s: str) -> int:
             stk = []
             ans, num, sign = 0, 0, 1

             for c in s:
                 if c == ' ':
                     continue
                 if c >= '0' and c <= '9':
                     num *= 10
                     num += int(c)
                 if c == '+':
                     ans += sign * num
                     num = 0
                     sign = 1
                 if c == '-':
                     ans += sign * num
                     num = 0
                     sign = -1
                 if c == '(':
                     stk.append(ans)
                     stk.append(sign)
                     ans = 0
                     sign = 1
                 if c == ')':
                     ans += num * sign
                     prev_sign, prev_ans = stk.pop(), stk.pop()
                     ans *= prev_sign
                     ans += prev_ans
                     num = 0
                     sign = 1

             if num != 0:
                 ans += sign * num

             return ans

     # https://leetcode.com/problems/basic-calculator-ii/description/
     # Input: s = " 3+5 / 2 "  Output: 5
```

```python
class Solution:
    def calculate(self, s: str) -> int:
        num, ans, sz = 0, 0, len(s)
        stk = []
        sign = '+'

        for i in range(sz):
            c = s[i]
            if c.isdigit():
                num = (num * 10) + int(c)

            if (not c.isdigit() and not c.isspace()) or i == sz-1:
                if sign == '+':
                    stk.append(num)
                elif sign == '-':
                    stk.append(-num)
                elif sign == '*':
                    top = stk.pop()
                    stk.append(top * num)
                else:
                    top = stk.pop()
                    stk.append(int(top / num))

                num, sign = 0, c

        return sum(stk)

# https://leetcode.com/problems/evaluate-reverse-polish-notation/

class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        stack = []

        for token in tokens:
            if token in {"+", "-", "*", "/"}:
                b = stack.pop()
                a = stack.pop()
                if token == "+":
                    stack.append(a + b)
                elif token == "-":
                    stack.append(a - b)
                elif token == "*":
                    stack.append(a * b)
                elif token == "/":
                    stack.append(int(a / b))  # Using int() to truncate towards␣
  ↪zero
```

```
            else:
                stack.append(int(token))

        return stack[-1] if stack else 0

# https://leetcode.com/problems/different-ways-to-add-parentheses/description/
# Input: expression = "2-1-1" Output: [0,2]
# Explanation: ((2-1)-1) = 0 (2-(1-1)) = 2
# Input: expression = "2*3-4*5" Output: [-34,-14,-10,-10,10]
# Explanation: (2*(3-(4*5))) = -34  ((2*3)-(4*5)) = -14 ((2*(3-4))*5) = -10
# (2*((3-4)*5)) = -10    (((2*3)-4)*5) = 10

class Solution:
    def diffWaysToCompute(self, s: str) -> List[int]:
        n = len(s)
        if (n == 0):
            return []
        ans = []

        for idx, c in enumerate(s):
            if c == '+' or c == '-' or c == '*':
                left = self.diffWaysToCompute(s[:idx])
                right = self.diffWaysToCompute(s[idx+1:])

                for l in left:
                    for r in right:
                        if c == '*':
                            ans.append(int(l) * int(r))
                        if c == '+':
                            ans.append(int(l) + int(r))
                        if c == '-':
                            ans.append(int(l) - int(r))

        if (len(ans) == 0):
            ans.append(int(s))

        return ans
```

[ ]:
```
# https://github.com/doocs/leetcode/blob/main/solution/0300-0399/0339.
↪Nested%20List%20Weight%20Sum/README_EN.md
# Nested List Weight Sum

# Input: nestedList = [[1,1],2,[1,1]]    Output: 10
# Explanation: Four 1's at depth 2, one 2 at depth 1. 1*2 + 1*2 + 2*1 + 1*2 +␣
↪1*2 = 10.
# Input: nestedList = [1,[4,[6]]]      Output: 27
# Explanation: One 1 at depth 1, one 4 at depth 2, and one 6 at depth 3. 1*1 +␣
↪4*2 + 6*3 = 27.

# we have isInteger() and getList() methods to get the integer value and list␣
↪of nested integers
# we can use dfs to calculate the sum of nested integers
# if we encounter a nested integer, we calculate the sum of integer * depth
# if we encounter a list, we recursively call the dfs function with depth + 1
class Solution:
    def depthSum(self, nestedList: List[NestedInteger]) -> int:
        def dfs(nestedList, depth):
            depth_sum = 0
            for item in nestedList:
                if item.isInteger():
                    depth_sum += item.getInteger() * depth
                else:
                    depth_sum += dfs(item.getList(), depth + 1)
            return depth_sum

        return dfs(nestedList, 1)
```

[ ]: