AmazonOA2-solution

Saturday, November 2, 2019 10:03 PM

1. **Two-sum**:
    a. Find Pair With Given Sum
        i. Return pair with largest number
        ii.
```
def giveSum(nums, target):
    target -= 30
    res = {}
    out = []
    for index, value in enumerate(nums):
        print(index,value)
        if target-value in res:
            print('target-value in res',target-value)
            print('any(out)',any(out),out)
            if res[target-value]>any(out) or index>any(out):
                out.clear()
                out.extend([res[target-value],index])
        else:
            print('out',out)
            print('else',res)
            res[value] = index
    return out
```

        iii.
```
def findSum(nums, target):
    target -= 30
    map = {}
    maximum = -1
    ans = [-1,-1]
    for i in range(len(nums)):
        if nums[i] not in map:
            map[target - nums[i]] = i
        else:
            if nums[i] > maximum or target - nums[i] > maximum:
                ans[0] = map[nums[i]]
                ans[1] = i
                maximum = max(nums[i],target - nums[i])
    if ans != [-1,-1]:
        return ans
    else:
        return []
```

    b. Movies on flight:
        i. Return closest output
        ii. 飞机飞行时长小于30分钟
        iii.
```
def moviesOnFlight(movieDurations, d):
    d = d-30
    newM = movieDurations
    newM = sorted(newM, reverse = True)
    maximum = 0
    ans = []
    for i in range(len(newM)):
        for j in range(len(newM)-1, i, -1):
            sum = newM[i]+ newM[j]
            if sum <= d:
                if sum > maximum:
                    maximum = newM[i] + newM[j]
                    ans = [movieDurations.index(newM[i]),len(movieDurations) - movieDurations[::-1].index(newM[j]) - 1]
            else:
                break

    return sorted(ans)
```

2. **Subarrays with k distinct elements**
    a. Input are integers

```
class Solution(object):
    def subarraysWithKDistinct(self, arr, k):
        lastIdx={}
        res=leftmost_last_idx=l=0
        for r, ch in enumerate(arr):
            lastIdx[ch]=r
            while lastIdx[arr[leftmost_last_idx]]!=leftmost_last_idx:#update
leftmost_last_idx
```

i.

```
                leftmost_last_idx+=1
            if len(lastIdx)==k+1:
                l=leftmost_last_idx+1
                del lastIdx[arr[leftmost_last_idx]]
                leftmost_last_idx+=1
                while lastIdx[arr[leftmost_last_idx]]!=leftmost_last_idx:
                    leftmost_last_idx+=1
            if len(lastIdx)==k:#calculate the numbers.
                res+=leftmost_last_idx-l+1
        return res
```

b. Input are chars

i.

```
def kdistinct(s, k):
    if k > len(set(s)):
        return 0
    res = 0

    for i in range(len(s)):
        dist = set()
        for j in range(i, len(s)):
            if s[j] not in dist:
                dist.add(s[j])

            if len(dist) == k:
                res += 1
            elif len(dist) > k:
                break

    return res
```

3. Path with Maximum Score:
   a. Testcase
      i. [[1]]
   b. DFS: https://leetcode.com/playground/CaAxhcPJ

```
def pathMaxScore(m):
    startVal = m[0][0]
    startCoor = (0,0)
    stack = [(startVal, startCoor, [startVal])]
    dirs = [(1,0), (0,1)] # only right and down are allowed
    maxScore = 0
    seen = set()
    count = 0
    while stack:
        node, coor, path = stack.pop()
        x,y = coor

        # check if leaf node (aka the mth, nth cell) is reached
        if y == len(m)-1 and x == len(m[0])-1 and str(path) not in seen:
            maxScore = max(maxScore, min(path))
            seen.add(str(path))
            count += 1

        for dir in dirs:
            newX, newY = x+dir[0], y+dir[1]
            # within bounds:
            if newX >= 0 and newX <= len(m)-1 and newY >= 0 and newY <= len(m[0])-1:
                stack.append((m[newX][newY], (newX, newY), path+[m[newX][newY]]))

    print('num of paths = ', count)
    return maxScore
```

c.

```
def max_min_path(matrix):
    if not matrix or not matrix[0]:
        return 0

    n, m = len(matrix), len(matrix[0])

    dp = [[0] * m for _ in range(n)]

    for i in range(n):
        for j in range(m):
            if i == 0 and j == 0:
                continue
            elif i == 1 and j == 0 or i == 0 and j == 1:
                dp[i][j] = matrix[i][j]
            elif i == 0:
                dp[i][j] = min(matrix[i][j], matrix[i][j - 1])
            elif j == 0:
                dp[i][j] = min(matrix[i][j], matrix[i - 1][j])
            else:
                dp[i][j] = min(matrix[i][j], max(dp[i - 1][j], dp[i][j - 1]))

    if n == 1:
        return dp[0][-2]
    elif m == 1:
        return dp[-2][0]
    else:
```

```
        return max(dp[-2][-1], dp[-1][-2])
```

4. Longest Palindromic String:

a.

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        l = len(s)
        if l == 0:
            return ''
        elif l == 1:
            return s

        dp = [[False for i in range(l)] for j in range(l)]
        max_len = 0
        max_ij = None

        for i in range(l):
            dp[i][i] = True

        for i in range(l-1):
            if s[i] == s[i+1]:
                dp[i][i+1] = True
                max_len = 2
                max_ij = (i, i+1)

        for k in range(2, l):
            for i in range(l):
                if i+k > l-1:
                    break
                else:
                    if s[i] == s[i+k] and dp[i+1][i+k-1]:
                        dp[i][i+k] = True
                        if len(s[i:i+k+1]) > max_len:
                            max_len = len(s[i:i+k+1])
                            max_ij = (i, i+k)

        if not max_ij:
            return s[0]

        x, y = max_ij
        return s[x:y+1]
```

5. Substrings of size K with K distinct chars

a.

```
def substringk(s, k):
    if not s or k == 0:
        return []

    letter, res = {}, set()
    start = 0
    for i in range(len(s)):
        if s[i] in letter and letter[s[i]] >= start:
            start = letter[s[i]]+1
        letter[s[i]] = i
        if i-start+1 == k:
            res.add(s[start:i+1])
            start += 1
    return list(res)
```

6. Most Common Word
    1. guaranteed there is at least one word that isn't banned
    2. the answer is unique and written in lowercase
    3. 注意，有的时候lowercase需要考虑，有的时候不需要
    4. corner case要注意输入为空的情况，要返回空list
    5. Solution
        i. Reg

1)

```
class Solution:
    def mostCommonWord(self, paragraph: str, banned: List[str]) -> str:
        ret = ""
        seen = {}       # {word: times seen}
        words = []
        # 1: populate dict with words, except banned
        words = re.findall(r"[\w]+", paragraph.lower())
        for word in words:
            if word not in banned:
                seen[word] = seen.get(word, 0) + 1

        # 2: find the most seen
        maxseen = 0
        for w in seen:
            if maxseen < seen[w]:
                maxseen = seen[w]
                ret = w
        return ret
```

ii.

```
class Solution:
    def mostCommonWord(self, p: str, b: List[str]) -> str:
        a = ''
        for i in p:
            if i in "!?',;.": i = ' '
            a += i
        a, M = a.lower().split(), 0
        for i in set(a):
            if i in b: continue
```

```
            m = a.count(i)
            if m > M: M, w = m, i
        return w
```

7. K Closest Points to Origin

1.

```
class Solution:
    def kClosest(self, points: List[List[int]], K: int) -> List[List[int]]:
        hq = []
        for i in range(len(points)):
            heapq.heappush(hq,(points[i][0]*points[i][0]+points[i][1]*points[i][1],i))

        res = []
        for i in range(K):
            temp = heapq.heappop(hq)
            res.append(points[temp[1]])

        return res
```

2. Faster

i.

```
import heapq

class Solution:
    def kClosest(self, points: List[List[int]], K: int) -> List[List[int]]:

        heap = []

        for (x, y) in points:
            dist = -(x*x + y*y)
            if len(heap) == K:
                heapq.heappushpop(heap, (dist, x, y))
            else:
                heapq.heappush(heap, (dist, x, y))

        return [(x,y) for (dist,x, y) in heap]
```

8. Merge Two sorted lists

1.

```
class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:

        head = sort_list = ListNode(0)

        while(l1 and l2):
            if (l1.val < l2.val):
                sort_list.next = l1
                l1 = l1.next
                sort_list = sort_list.next

            elif (l1.val >= l2.val):
                sort_list.next = l2
                l2 = l2.next
                sort_list = sort_list.next

        sort_list.next = l1 or l2
        return head.next
```