2020 Interview

Classical Problem Set

0. Maximum Sum of Subarray

```python
class Solution:
    def maxSubArray(self, nums):
        for i in range(1, len(nums)):
            nums[i] += max(nums[i-1], 0)
        return max(nums)
```

1. Merge Two Sorted Arrays

```python
class Solution:
    def merge(self, A, m, B, n):
        pa, pb = m-1, n-1
        tail = m + n - 1
        while pa >= 0 or pb >= 0:
            if pa == -1:
                A[tail] = B[pb]
                pb -= 1
            elif pb == -1:
                A[tail] = A[pa]
                pa -= 1
            elif A[pa] > B[pb]:
                A[tail] = A[pa]
                pa -= 1
            else:
                A[tail] = B[pb]
                pb -= 1

            tail -= 1
```

2. Best Timing of Buying and Selling

```python
class Solution:
    def maxProfit(self, prices):
        minprice = int(1e9)
        maxprofit = 0

        for price in prices:
            maxprofit = max(price - minprice, maxprofit)
            minprice = min(price, minprice)

        return maxprofit
```

3. Validate Palindrome String

```python
class Solution:
    def validPalindrome(self, s: str) -> bool:
        n = len(s)
        left, right = 0, n - 1

        while left < right:
            while left < right and not s[left].isalnum():
                left += 1
            while left < right and not s[right].isalnum():
                right -= 1
            if left < right:
                if s[left].lower() != s[right].lower():
                    return False
                left, right = left + 1, right - 1

        return True
```

4. Binary Tree Level Order Traversal

```python
class Solution:
    def levelOrderBFS(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        res = []
        queue = [root,]

        while queue:
            size = len(queue)
            lvl = []

            for _ in range(size):
                node = queue.pop(0)
                lvl.append(node.val)
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)

            res.append(lvl)
        return res

    def levelOrderDFS(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        res = []

        def dfs(index, r):
            if len(res) < index:
                res.append([])
            res[index-1].append(r.val)
            if r.left:
                dfs(index+1, r.left)
            if r.right:
                dfs(index+1, r.right)

        dfs(1, root)
        return res
```

5. Copy List with Random Pointer

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

        ptr = head
        while ptr:
            new_node = Node(ptr.val, None, None)
            new_node.next = ptr.next
            ptr.next = new_node

            ptr = new_node.next

        ptr = head
        while ptr:
            ptr.next.random = ptr.random.next if ptr.random else None
            ptr = ptr.next.next

        ptr_old_list = head
        ptr_new_list = head.next
        head_old = head.next

        while ptr_old_list:
            ptr_old_list.next = ptr_old_list.next.next
            ptr_new_list.next = ptr_new_list.next.next if ptr_new_list.next else None
            ptr_old_list = ptr_old_list.next
            ptr_new_list = ptr_new_list.next

        return head_old
```

6. LRU Cache

```
class LRUCache:
    def __init__(self, capacity: int):
        self.cache = dict()
        # Use fake head and fake tail.
        self.head = DLinkedNode()
        self.tail = DLinkedNode()
        self.head.next = self.tail
        self.tail.prev = self.head
        self.capacity = capacity
        self.size = 0

    def get(self, key: int) -> int:
        if key not in self.cache:
            return -1
        node = self.cache[key]
        self.moveToHead(node)
        return node.value

    def put(self, key: int, value: int) -> None:
        if key not in self.cache:
            # Add new node to the hash table.
            node = DLinkedNode(key, value)
            self.cache[key] = node
            self.addToHead(node)
            self.size += 1

            if self.size > self.capacity:
                removed = self.removeTail()
                self.cache.pop(removed.key)
                self.size -= 1
        else:
            node = self.cache[key]
            node.value = value
            self.moveToHead(node)
```

7. **Number of Islands**

```
class UnionFind:
    def __init__(self, grid):
        m, n = len(grid), len(grid[0])
        self.count = 0
        self.parent = [-1] * (m * n)
        self.rank = [0] * (m * n)
        for i in range(m):
            for j in range(n):
                if grid[i][j] == "1":
                    self.parent[i * n + j] = i * n + j
                    self.count += 1

    def find(self, i):
        if self.parent[i] != i:
            self.parent[i] = self.find(self.parent[i])
        return self.parent[i]

    def union(self, x, y):
        rootx = self.find(x)
        rooty = self.find(y)
        if rootx != rooty:
            if self.rank[rootx] < self.rank[rooty]:
                rootx, rooty = rooty, rootx
            self.parent[rooty] = rootx
            if self.rank[rootx] == self.rank[rooty]:
                self.rank[rootx] += 1
            self.count -= 1

    def getCount(self):
        return self.count
```

```python

class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        nr = len(grid)
        if nr == 0:
            return 0
        nc = len(grid[0])

        uf = UnionFind(grid)
        num_islands = 0
        for r in range(nr):
            for c in range(nc):
                if grid[r][c] == "1":
                    grid[r][c] = "0"
                    for x, y in [(r - 1, c), (r + 1, c), (r, c - 1), (r, c + 1)]:
                        if 0 <= x < nr and 0 <= y < nc and grid[x][y] == "1":
                            uf.union(r * nc + c, x * nc + y)

        return uf.getCount()
```

8. Minimum Window Containing String

```python
from collections import defaultdict

class Solution:
    def __init__(self):
        self.ori = defaultdict(int)
        self.cnt = defaultdict(int)

    def isFit(self):
        for k, v in self.ori.items():
            if k not in self.cnt.keys() or v >= self.cnt[k]:
                return False
        return True

    def minWindow(self, s: str, t: str) -> str:
        for i in t:
            self.ori[i] += 1
        l, r = 0, -1
        length, ansL, ansR = float('inf'), -1, -1
        sLen, tLen = len(s), len(t)

        while r < sLen:
            r += 1
            if r < sLen and s[r] in self.ori.keys():
                self.cnt[s[r]] += 1
            while self.isFit() and l <= r:
                if r - l + 1< length:
                    length = r - l + 1
                    ansL = l
                    ansR = l + length
                if s[l] in self.ori.keys():
                    self.cnt[s[l]] -= 1
                l += 1
        return "" if ansL == -1 else s[ansL:ansR]
```

9. Trapping Rain Water

```
class Solution:
    def trapWater(self, height: List[int]) -> int:
        left, right = 0, len(height) - 1
        lmax, rmax = 0, 0
        ans = 0

        while left < right:
            if height[left] < height[right]:
                if height[left] >= lmax:
                    lmax = height[left]
                else:
                    ans += lmax - height[left]
                left += 1   # update left pointer
            else:
                if height[right] >= rmax:
                    rmax = height[right]
                else:
                    ans += rmax - height[right]
                right -= 1  # update right pointer

        return ans
```