

# LAB PROGRAM-3

DATE:06/06/2024

1. You are given a string *s*, and an array of pairs of indices in the string *pairs* where *pairs[i] = [a, b]* indicates 2 indices(0-indexed) of the string. You can swap the characters at any pair of indices in the given pairs any number of times. Return the lexicographically smallest string that *s* can be changed to after using the swaps.

CODING:

```
1 class UnionFind:
2     def __init__(self, n):
3         self.parent = [i for i in range(n)]
4         self.rank = [0] * n
5
6     def find(self, x):
7         if self.parent[x] != x:
8             self.parent[x] = self.find(self.parent[x])
9         return self.parent[x]
10
11     def union(self, x, y):
12         root_x = self.find(x)
13         root_y = self.find(y)
14         if root_x == root_y:
15             return False
16         if self.rank[root_x] < self.rank[root_y]:
17             self.parent[root_x] = root_y
18         elif self.rank[root_x] > self.rank[root_y]:
19             self.parent[root_y] = root_x
20         else:
21             self.parent[root_y] = root_x
```

```

16         if self.rank[root_x] < self.rank[root_y]:
17             self.parent[root_x] = root_y
18         elif self.rank[root_x] > self.rank[root_y]:
19             self.parent[root_y] = root_x
20         else:
21             self.parent[root_y] = root_x
22             self.rank[root_x] += 1
23         return True
24
25     def smallestStringWithSwaps(s, pairs):
26         n = len(s)
27         uf = UnionFind(n)
28         for a, b in pairs:
29             uf.union(a, b)
30         groups = {}
31         for i in range(n):
32             root = uf.find(i)
33             if root in groups:
34                 groups[root].append(i)
35             else:
36                 groups[root] = [i]
37
38         result = list(s)
39         for group in groups.values():
40             chars = sorted([s[i] for i in group])
41             for i, c in zip(group, chars):
42                 result[i] = c
43         return "".join(result)
44
45     # Example usage
46     s = "dcab"
47     pairs = [[0, 3], [1, 2], [0, 2]]
48     print(smallestStringWithSwaps(s, pairs))

```

OUTPUT:

```

Run LAB-3 QUESTION
"C:\Users\jacin\PycharmProjects\DAA-Design analysis of algorithm\.venv\Scripts\python.exe" "C:\Users\jacin\AppData\Roaming\JetBrains\PyCharmCE202
abcd
Process finished with exit code 0

```

2. Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break

s1 or vice-versa. A string x can break string y (both of size n) if  $x[i] \geq y[i]$  (in alphabetical order) for all i between 0 and n-1.

CODING:

```
3 usages
1  def checkIfCanBreak(s1, s2):
2      if len(s1) != len(s2):
3          return False
4      s1_sorted = sorted(s1)
5      s2_sorted = sorted(s2)
6      can_break_s2 = True
7      can_break_s1 = True
8      for i in range(len(s1)):
9          if s1_sorted[i] < s2_sorted[i]:
10             can_break_s2 = False
11             if s2_sorted[i] < s1_sorted[i]:
12                 can_break_s1 = False
13             return can_break_s1 or can_break_s2
14  s1 = "abc"
15  s2 = "xya"
16  print(checkIfCanBreak(s1, s2)) # Output: True

13      return can_break_s1 or can_break_s2
14  s1 = "abc"
15  s2 = "xya"
16  print(checkIfCanBreak(s1, s2)) # Output: True
17  s1 = "abe"
18  s2 = "acd"
19  print(checkIfCanBreak(s1, s2)) # Output: False
20  s2 = "interview"
21  print(checkIfCanBreak(s1, s2)) # Output: True
```

OUTPUT:

```
C:\Users\vinot\PycharmProjects\pythonP
True
False
True

Process finished with exit code 0
```

3. You are given a string s.  $s[i]$  is either a lowercase English letter or '?'. For a string t having length m containing only lowercase English letters, we define the function  $\text{cost}(i)$  for an index i as the number of characters equal to  $t[i]$  that appeared before it,

i.e. in the range  $[0, i - 1]$ . The value of  $t$  is the sum of  $\text{cost}(i)$  for all indices  $i$ . For example, for the string  $t = \text{"aab"}$ :

$\text{cost}(0) = 0$

$\text{cost}(1) = 1$

$\text{cost}(2) = 0$

Hence, the value of  $\text{"aab"}$  is  $0 + 1 + 0 = 1$ . Your task is to replace all occurrences of '?' in  $s$  with any lowercase English letter so at the value of  $s$  is minimized.

CODING:

```
1 def minimumValue(s):
2     n = len(s)
3     res = [0] * 26
4     for i in range(n):
5         if s[i] != '?':
6             res[ord(s[i]) - ord('a')] += 1
7     ans = 0
8     for i in range(n):
9         if s[i] != '?':
10            ans += res[ord(s[i]) - ord('a')] - (i != 0 and s[i] == s[i - 1])
11     return ans
12 s = "aa?b"
13 print(minimumValue(s))
```

OUTPUT:

```
C:\Users\vinot\PycharmProjects\pythonProjec
4

Process finished with exit code 0
```

4. You are given a string  $s$ . Consider performing the following operation until  $s$  becomes empty: For every alphabet character from 'a' to 'z', remove the first occurrence of that character in  $s$  (if it exists). For example, let initially  $s = \text{"aabcbbbca"}$ . We do the following operations: Remove the underlined characters  $s = \text{"aabcbbbca"}$ . The resulting string is  $s = \text{"abbca"}$ . Remove the underlined characters  $s = \text{"abbca"}$ . The resulting string is  $s = \text{"ba"}$ . Remove the underlined characters  $s = \text{"ba"}$ . The resulting string is  $s = \text{" "}$ . Return the value of the string  $s$  right before applying the last operation. In the example above, answer is  $\text{"ba"}$ .

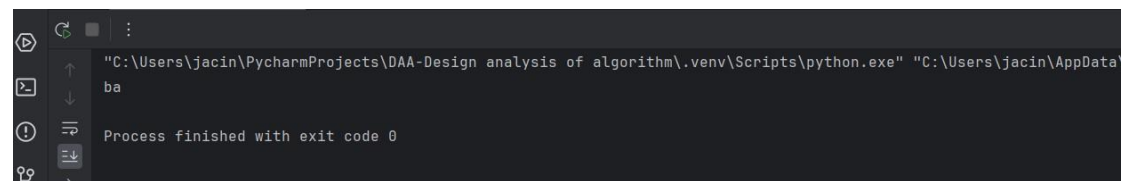
CODING:

```

1 def last_string_before_empty(s):
2     while True:
3         original_s = s
4         for char in set(s):
5             s = s.replace(char, '', 1)
6             if s == "":
7                 return original_s
8 s = "aabcbbca"
9 print(last_string_before_empty(s))

```

OUTPUT:



```

"C:\Users\jacin\PycharmProjects\DAA-Design analysis of algorithm\.venv\Scripts\python.exe" "C:\Users\jacin\AppData\Local\Programs\Python\Python310\python.exe ba"
Process finished with exit code 0

```

5. Given an integer array `nums`, find the subarray with the largest sum, and return its sum.

Example 1:

Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`

Output: 6

Explanation: The subarray `[4,-1,2,1]` has the largest sum 6.

CODING:

```

1 def maxSubArray(nums):
2     max_sum = nums[0]
3     current_sum = nums[0]
4
5     for num in nums[1:]:
6         current_sum = max(num, current_sum + num)
7         max_sum = max(max_sum, current_sum)
8
9     return max_sum
10
11
12 # Example usage
13 nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
14 print(maxSubArray(nums))

```

OUTPUT:

```
Run
subarray of largest sum
"C:\Users\jacin\PycharmProjects\DAA-Design analysis of algorithm\.venv\Scripts\python.exe" "C:\Users\jacin\AppData\Roaming\JetBra
6
Process finished with exit code 0
```

6. You are given an integer array `nums` with no duplicates. A maximum binary tree can be built recursively from `nums` using the following algorithm: Create a root node whose value is the maximum value in `nums`. Recursively build the left subtree on the subarray prefix to the left of the maximum value. Recursively build the right subtree on the subarray suffix to the right of the maximum value. Return the maximum binary tree built from `nums`.

CODING:

```
1  from typing import List
2  3 usages
3  class TreeNode:
4      def __init__(self, val=0, left=None, right=None):
5          self.val = val
6          self.left = left
7          self.right = right
8  1 usage
9  class Solution:
10     3 usages
11     def constructMaximumBinaryTree(self, nums: List[int]) -> TreeNode:
12         if not nums:
13             return None
14         max_val = max(nums)
15         max_idx = nums.index(max_val)
16         root = TreeNode(max_val)
17         root.left = self.constructMaximumBinaryTree(nums[:max_idx])
18         max_idx = nums.index(max_val)
19         root = TreeNode(max_val)
20         root.left = self.constructMaximumBinaryTree(nums[:max_idx])
21         root.right = self.constructMaximumBinaryTree(nums[max_idx + 1:])
22         return root
23     3 usages
24     def printTree(self, root: TreeNode) -> None:
25         if root:
26             print(root.val, end=' ')
27             self.printTree(root.left)
28             self.printTree(root.right)
29     nums = [3, 2, 1, 6, 0, 5]
30     solution = Solution()
31     root = solution.constructMaximumBinaryTree(nums)
32     solution.printTree(root)
```

OUTPUT:

```
C:\Users\vinot\PycharmProjects\pythonProject3\.venv
6 3 2 1 5 0
Process finished with exit code 0
```

7. Given a circular integer array `nums` of length `n`, return the maximum possible sum of a non-empty subarray of `nums`. A circular array means the end of the array connects to the beginning of the array. Formally, the next element of `nums[i]` is `nums[(i + 1) % n]` and the previous element of `nums[i]` is `nums[(i - 1 + n) % n]`. A subarray may only include each element of the fixed buffer `nums` at most once. Formally, for a subarray `nums[i]`, `nums[i + 1]`, ..., `nums[j]`, there does not exist  $i \leq k_1$ ,  $k_2 \leq j$  with  $k_1 \% n == k_2 \% n$ .

CODING:

```
3 usages
1 def max_subarray_sum_circular(nums):
2     # Helper function to find maximum subarray sum using Kadane's algorithm
3     def kadane_max_subarray(arr):
4         max_ending_here = max_so_far = arr[0]
5         for x in arr[1:]:
6             max_ending_here = max(x, max_ending_here + x)
7             max_so_far = max(max_so_far, max_ending_here)
8         return max_so_far
9
10    # Helper function to find minimum subarray sum using Kadane's algorithm
11    def kadane_min_subarray(arr):
12        min_ending_here = min_so_far = arr[0]
13        for x in arr[1:]:
14            min_ending_here = min(x, min_ending_here + x)
15            min_so_far = min(min_so_far, min_ending_here)
16        return min_so_far
17
18    total_sum = sum(nums) # Step 3: Total sum of the array
19    max_kadane = kadane_max_subarray(nums) # Step 1: Max subarray sum (non-wrapping)
20    min_kadane = kadane_min_subarray(nums) # Step 2: Min subarray sum (for wrapping)
21
22    # Step 4: Max subarray sum (wrapping)
23    max_wraparound = total_sum - min_kadane
24
25    # Step 5: Handle edge case where all elements are negative
26    if max_wraparound == 0:
27        return max_kadane
28    return max(max_kadane, max_wraparound)
29
30 nums = [1, -2, 3, -2]
31 print(max_subarray_sum_circular(nums)) # Output: 3
32 nums = [5, -3, 5]
33 print(max_subarray_sum_circular(nums)) # Output: 10
34 nums = [-3, -2, -3]
35 print(max_subarray_sum_circular(nums)) # Output: -2
```

OUTPUT:



```
C:\Users\vinot\PycharmProjects\pythonProject3
3
10
-2

Process finished with exit code 0
```

8. You are given an array `nums` consisting of integers. You are also given a 2D array `queries`, where `queries[i] = [posi, xi]`. For query `i`, we first set `nums[posi]` equal to `xi`, then we calculate the answer to query `i` which is the maximum sum of a subsequence of `nums` where no two adjacent elements are selected. Return the sum of the answers to all queries. Since the final answer may be very large, return it modulo  $10^9 + 7$ . A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

CODING:

```
1 usage
2 def max_sum_subsequence_no_adjacent(nums):
3     incl = 0
4     excl = 0
5     for num in nums:
6         new_excl = max(incl, excl)
7         incl = excl + num
8         excl = new_excl
9     return max(incl, excl)
10
11 usage
12 def sum_of_queries_results(nums, queries):
13     MOD = 10 ** 9 + 7
14     total_sum = 0
15     for pos, val in queries:
16         nums[pos] = val
17         max_sum = max_sum_subsequence_no_adjacent(nums)
18         total_sum = (total_sum + max_sum) % MOD
19
20     return total_sum
21
22 nums = [1, 2, 3]
23 queries = [[1, 5], [0, 4]]
24 print(sum_of_queries_results(nums, queries)) # Output sh
```

OUTPUT:



```
C:\Users\vinot\PycharmProjects\pythonProject
```

```
12
```

```
Process finished with exit code 0
```

9. Given an array of points where  $\text{points}[i] = [x_i, y_i]$  represents a point on the X-Y plane and an integer  $k$ , return the  $k$  closest points to the origin  $(0, 0)$ . The distance between two points on the X-Y plane is the Euclidean distance (i.e.,  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ ). You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).

CODING:

```
1  from typing import List
2  import heapq
3  import math
4  class Solution:
5      def kClosest(self, points: List[List[int]], k: int) -> List[List[int]]:
6          heap = []
7          output = []
8          for cord in points:
9              distance = math.sqrt((cord[0] - 0) ** 2 + (cord[1] - 0) ** 2)
10             distance_tuple = (-distance, cord)
11             if len(heap) == k:
12                 heapq.heappushpop(*args: heap, distance_tuple)
13             else:
14                 heapq.heappush(*args: heap, distance_tuple)
15             for item in heap:
16
17                 output.append(item[1])
18             return output
19
20 points = [[3, 3], [5, -1], [-2, 4]]
21 k = 2
22 solution = Solution()
23 result = solution.kClosest(points, k)
24 print(result) # Output: [[3, 3], [-2, 4]]
25 points = [[1, 3], [-2, 2]]
26 k = 1
27 solution = Solution()
28 result = solution.kClosest(points, k)
29 print(result) # Output: [[-2, 2]]
30
31 points = [[1, 1], [1, 1], [1, 1], [0, 0]]
32 k = 2
33 solution = Solution()
34 result = solution.kClosest(points, k)
35 print(result) # Output: [[0, 0], [1, 1]]
```

OUTPUT:

```
C:\Users\vinot\PycharmProjects\pythonP
[[-2, 4], [3, 3]]
[[-2, 2]]
[[1, 1], [0, 0]]

Process finished with exit code 0
```

10. Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return the median of the two sorted arrays. The overall run time complexity should be  $O(\log(m+n))$ .

**CODING:**

```
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    low, high = 0, m
    total_len = m + n
    while low <= high:
        partition_nums1 = (low + high) // 2
        partition_nums2 = (total_len + 1) // 2 - partition_nums1
        max_left_nums1 = float('-inf') if partition_nums1 == 0 else nums1[partition_nums1 - 1]
        min_right_nums1 = float('inf') if partition_nums1 == m else nums1[partition_nums1]
        max_left_nums2 = float('-inf') if partition_nums2 == 0 else nums2[partition_nums2 - 1]
        min_right_nums2 = float('inf') if partition_nums2 == n else nums2[partition_nums2]
        if max_left_nums1 <= min_right_nums2 and max_left_nums2 <= min_right_nums1:
            if total_len % 2 == 0:
                return (max(max_left_nums1, max_left_nums2) + min(min_right_nums1, min_right_nums2)) / 2
            else:
                return max(max_left_nums1, max_left_nums2)
        elif max_left_nums1 > min_right_nums2:
            high = partition_nums1 - 1
        else:
            low = partition_nums1 + 1
    nums1 = [1, 3]
    nums2 = [2]
    print(findMedianSortedArrays(nums1, nums2))
```

**OUTPUT:**

```
"C:\Users\jacin\PycharmProjects\DAA-Design analysis of algorithm\.venv\Scripts\python.exe" "C:\Users\jacin\AppData\Roaming\JetBrains\PyC
2
Process finished with exit code 0
```