Python Question Bank

ISS TAs

CS6.201: Introduction to Software Systems

Instructions

Implement the solutions for the following problems using standard Python 3. Do not use external libraries (like `pandas` or `numpy`). Focus on algorithmic efficiency and edge case handling.

Question 1: Spiral Matrix Traversal

Given an $m \times n$ matrix represented as a list of lists, return a list of all elements of the matrix in spiral order. Start from the top-left corner and move clockwise.

Input Format: A single line containing a Python list of lists representing the matrix.

Output Format: A single line containing the flattened list of elements in spiral order.

Sample Case 1:
Input: `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
Output: `[1, 2, 3, 6, 9, 8, 7, 4, 5]`
Sample Case 2:
Input: `[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]`
Output: `[1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7]`
Sample Case 3:
Input: `[[1]]`
Output: `[1]`

Question 2: Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Input Format: A single line containing a list of integers representing the height map.

Output Format: An integer representing the total units of trapped water.

Sample Case 1:
Input: `[0,1,0,2,1,0,1,3,2,1,2,1]`
Output: `6`
Sample Case 2:
Input: `[4,2,0,3,2,5]`
Output: `9`

Sample Case 3:
Input: `[1, 2, 3, 4, 5]`
Output: `0`

Question 3: Flatten Nested Dictionary

Write a program to flatten a nested dictionary. Keys should be combined using a dot notation. The input may contain dictionaries nested to arbitrary depth.

Input Format: A JSON-string or dictionary string representation.

Output Format: A dictionary string representation of the flattened key-value pairs.

Sample Case 1:
Input: `{"a": 1, "b": {"c": 2, "d": 3}}`
Output: `{"a": 1, "b.c": 2, "b.d": 3}`
Sample Case 2:
Input: `{"x": {"y": {"z": "deep"}}, "a": 5}`
Output: `{"x.y.z": "deep", "a": 5}`
Sample Case 3:
Input: `{"key": "value"}`
Output: `{"key": "value"}`

Question 4: Merge Overlapping Intervals

Given a collection of intervals represented as a list of tuples, merge all overlapping intervals.

Input Format: A list of tuples, where each tuple `(start, end)` represents an interval.

Output Format: A list of tuples representing the merged intervals, sorted by start time.

Sample Case 1:
Input: `[(1, 3), (2, 6), (8, 10), (15, 18)]`
Output: `[(1, 6), (8, 10), (15, 18)]`
Sample Case 2:
Input: `[(1, 4), (4, 5)]`
Output: `[(1, 5)]`
Sample Case 3:
Input: `[(1, 4), (0, 2), (3, 5)]`
Output: `[(0, 5)]`

Question 5: Group Anagrams

Given an array of strings, group the anagrams together. You can return the answer in any order, but the inner lists should be sorted internally for consistency in grading.

Input Format: A list of strings.

Output Format: A list of lists, where each inner list contains words that are anagrams of each other.

Sample Case 1:
Input: `["eat", "tea", "tan", "ate", "nat", "bat"]`
Output: `[["ate", "eat", "tea"], ["bat"], ["nat", "tan"]]`
Sample Case 2:
Input: `[""]`
Output: `[[""]]`
Sample Case 3:
Input: `["a"]`
Output: `[["a"]]`

Question 6: Product of Array Except Self

Given an integer array `nums`, return an array `answer` such that `answer[i]` is equal to the product of all the elements of `nums` except `nums[i]`. **Constraint:** You must write an algorithm that runs in $O(n)$ time and without using the division operation.

Input Format: A list of integers.

Output Format: A list of integers.

Sample Case 1:
Input: `[1, 2, 3, 4]`
Output: `[24, 12, 8, 6]`
Sample Case 2:
Input: `[-1, 1, 0, -3, 3]`
Output: `[0, 0, 9, 0, 0]`
Sample Case 3:
Input: `[5, 5]`
Output: `[5, 5]`

Question 7: Valid Sudoku Validator

Determine if a 9×9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

1. Each row must contain the digits 1-9 without repetition.
2. Each column must contain the digits 1-9 without repetition.
3. Each of the nine 3×3 sub-boxes of the grid must contain the digits 1-9 without repetition.

The board contains strings of digits 1-9, or "." for empty cells.

Input Format: A list of 9 lists, each containing 9 strings.

Output Format: `True` if valid, `False` otherwise.

Sample Case 1:
Input: (Standard Valid Board)
Output: `True`
Sample Case 2:
Input: (Board with duplicates in top-left 3x3 box)
Output: `False`
Sample Case 3:
Input: (Board with duplicates in first row)
Output: `False`

Question 8: Longest Consecutive Sequence

Given an unsorted array of integers, find the length of the longest consecutive elements sequence. The algorithm should ideally run in $O(n)$ complexity.

Input Format: A list of integers.

Output Format: An integer representing the length of the longest sequence.

Sample Case 1:
Input: `[100, 4, 200, 1, 3, 2]`
Output: `4` *(Sequence is 1, 2, 3, 4)*
Sample Case 2:
Input: `[0, 3, 7, 2, 5, 8, 4, 6, 0, 1]`
Output: `9`
Sample Case 3:
Input: `[]`
Output: `0`

Question 9: Matrix Rotation

You are given an $n \times n$ 2D matrix representing an image. Rotate the image by 90 degrees (clockwise). You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

Input Format: A list of lists representing the $n \times n$ matrix.

Output Format: The modified matrix (list of lists).

Sample Case 1:
Input: `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
Output: `[[7, 4, 1], [8, 5, 2], [9, 6, 3]]`
Sample Case 2:
Input: `[[5, 1, 9, 11], [2, 4, 8, 10], [13, 3, 6, 7], [15, 14, 12, 16]]`
Output: `[[15, 13, 2, 5], [14, 3, 4, 1], [12, 6, 8, 9], [16, 7, 10, 11]]`
Sample Case 3:

Input: `[[1]]`
Output: `[[1]]`

Question 10: Sparse Vector Dot Product

Represent sparse vectors (vectors with mostly zero values) using dictionaries where keys are indices and values are the non-zero numbers. Given two such dictionary representations, compute their dot product.

Input Format: Two dictionaries on separate lines. Line 1: Vector A `{index: value, ...}` Line 2: Vector B `{index: value, ...}`

Output Format: An integer or float representing the dot product.

Sample Case 1:
Input:
```
{0: 1, 3: 2, 5: 3}
{0: 4, 3: 5, 1: 2}
```
Output: `14` *(1*4 + 2*5 = 14)*

Sample Case 2:
Input:
```
{100: 5}
{99: 5}
```
Output: `0`

Sample Case 3:
Input:
```
{0: 1, 1: -1}
{0: -1, 1: 1}
```
Output: `-2`

Question 11: Longest Palindromic Substring

Given a string `s`, return the longest palindromic substring in `s`. A palindrome is a string that reads the same backward as forward.

Input Format: A single string.

Output Format: A single string representing the longest palindrome.

Sample Case 1:
Input: `"babad"`
Output: `"bab"` *(Note: "aba" is also a valid answer)*

Sample Case 2:
Input: `"cbbd"`
Output: `"bb"`

Sample Case 3:
Input: `"a"`
Output: `"a"`

Question 12: Edit Distance (Recursive)

Given two strings `word1` and `word2`, return the minimum number of operations required to convert `word1` to `word2`. You have the following three operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

Constraint: Implement this using a recursive approach (memoization is allowed/encouraged).

Input Format: Two strings on separate lines.

Output Format: An integer representing the minimum operations.

Sample Case 1:
Input:
`"horse"`
`"ros"`
Output: `3` *(horse -> rorse -> rose -> ros)*

Sample Case 2:
Input:
`"intention"`
`"execution"`
Output: `5`

Sample Case 3:
Input:
`""`
`"abc"`
Output: `3`