Revision - Arrays

Introduction

Arrays are data structures that store a collection of elements of the same type in contiguous memory locations.

Question - 1: Queries on an Array

Given an array A where all elements are initially 0. Perform queries q(i, j, x) - increment all the elements of the array from index i to j by x. Find the final array.

Example

Array A: [0, 0, 0, 0, 0]

Queries:

q(1, 3, 2) # Increment elements from index 1 to 3 by 2

q(0, 2, 1) # Increment elements from index 0 to 2 by 1

Final Array A: [1, 3, 3, 2, 0]

Solution:

To perform the queries efficiently, we can use the concept of prefix sums. Create an array prefix[] of size |A| + 1, initially filled with 0.

For each query q(i, j, x), increment prefix[i] by x and decrement prefix[j+1] by x. Then, iterate over the prefix array and calculate the cumulative sum to obtain the final array.

Pseudo Code:

```
n = length(A)
prefix = new Array(n + 1)
for i = 0 to n:
    prefix[i] = 0

for each query (i, j, x) in queries:
    prefix[i] += x
    prefix[j+1] -= x

cumulativeSum = 0
for i = 0 to n-1:
    cumulativeSum += prefix[i]
    A[i] += cumulativeSum

return A
```

- Time complexity: O(N)
- Space complexity: O(N)

Question - 2: Equilibrium index of an array

The equilibrium index of an array is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes. We have to find the equilibrium index of an array.

Example:

A = [-7, 1, 5, 2, -4, 3, 0]

The equilibrium index is 3 as the elements at lower index [-7, 1, 5] and at higher index [-4, 3, 0] have the same sum.

Solution:

The idea is to get the total sum of the array first. Then Iterate through the array and keep updating the left sum which is initialized as zero. In the loop, we can get the right sum by subtracting the elements one by one.

Pseudo Code:

```
1) Initialize leftsum  as 0
2) Get the total sum of the array as sum
3) Iterate through the array and for each index i, do following.
    a)  Update sum to get the right sum.
           sum = sum - arr[i]
       // sum is now right sum
    b) If leftsum is equal to sum, then return current index.
       // update leftsum for next iteration.
    c) leftsum = leftsum + arr[i]
4) return -1
// If we come out of loop without returning then
// there is no equilibrium index
```

- Time complexity: O(N)
- Space complexity: O(1)

Question - 3: Rainwater trapped problem

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.

Example:

A = [0, 1, 0, 2]

Here, the quantity of water that can be trapped is 1. It is trapped on top of the third building

Solution:

To solve the rainwater trapped problem, we can use the two-pointer technique. Initialize two pointers, left and right, at the beginning and end of the array respectively. Keep track of

the maximum left and right heights encountered so far. Move the pointers inward, updating the maximum heights and calculating the trapped water at each step.

Pseudo Code:

```
Initialize variables: left = 0, right = n-1 (where n is the length of the array)
Initialize variables: maxLeft = 0, maxRight = 0
Initialize variable: water = 0 (to store the total trapped water)

While left < right:
    If height[left] < height[right]:
        If height[left] > maxLeft:
            Set maxLeft = height[left]
        Else:
            Add maxLeft - height[left] to water
        Increment left by 1
    Else:
        If height[right] > maxRight:
            Set maxRight = height[right]
        Else:
            Add maxRight - height[right] to water
        Decrement right by 1

Return water (total trapped water)
```

- Time complexity: O(N)
- Space complexity: O(1)

Question - 4: Best Time to Buy and Sell Stocks

We are given the price of a stock for N days where the i-th element is the price on the i-th day. We are only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), what is the maximum possible profit we can get.

Example:

A = [1, 4, 5, 2, 4]

Here, the maximum profit is 4 as we can buy on day 0 and sell on day 2.

Solution:

To find the maximum profit from buying and selling stocks, we can use a greedy approach. Iterate over the array, keeping track of the minimum stock price encountered so far. Calculate the current profit by subtracting the minimum price from the current price. Update the maximum profit if the current profit is greater.

Pseudo Code:

```
Initialize variables minPrice = A[0] and maxProfit = 0.
Iterate over the array A from index 1 to N-1:
    If A[i] < minPrice, update minPrice.
    Else, calculate the current profit: profit = A[i] - minPrice.
    If profit > maxProfit, update maxProfit.
Return maxProfit.
```

- Time complexity: O(N)
- Space complexity: O(1)

2-D Array/Matrix

Question - 5

Given a matrix A of integers, find the sum of a given submatrix. There would be multiple sum queries.

Example:

Suppose the matrix is:

```
[[1, 3, 5, 2, -1],
 [4, 8, 5, 0, 6],
 [10, 20, -1, 3, 5],
 [1, 5, -5, 10, 6]]
```

Suppose you have to find the sum of the matrix with top left coordinate (1, 0) and bottom right coordinate (3, 2).

The elements in the submatrix are and their sum are:

4 + 8 + 5 + 10 + 20 - 1 + 1 + 5 - 5 = 47

Solution:

Brute force:

In the brute force approach, we would simply answer each query by iterating the submatrix given to us and finding the sum.

Pseudo Code:

```
function each_query(Mat, x, y, u, v):
    sum = 0
    for i from x to u:
        for j from y to v:
            sum = sum + Mat[i][j]
    return sum
```

- Time complexity for each query: O(N * M)
- Space complexity for each query: O(1)

Optimization:

We can use the prefix sum technique, we used in case of an array to solve each query in O(1), with precalculation in O(N * M).

Pseudo Code:

```
function calculate_prefix_matrix(mat):
    prefix_mat = mat
    N = rows in mat
    M = cols in mat

    for i from 1 to N:
        for j from 1 to M:
            prefix_mat[i][j] = prefix_mat[i - 1][j] + prefix_mat[i][j - 1] - pref

    return prefix_mat

function solve(mat, queries)
    prefix_mat = calculate_prefix_matrix(mat)
    for (x, y, u, v) in queries:
        // handle negative values
        answer = prefix_mat[u][v] - prefix_mat[u][y - 1] - prefix_mat[x - 1][v] +
        print(answer)
```

- Time complexity: O(1) per query, with precalculation of O(N * M)
- Space complexity: O(N * M)

Question - 6

Given N buildings, with heights of each building. There is a person on top of each building who can only move to right. Find max height a person can go from current location.

Example:

Suppose A = [6, 4, 3, 5, 2, 4]

Here answer = [0, 1, 2, 0, 3, 0]

Here:

- Person at index 0 can’t move to any other taller building.
- Person at index 1 would move to building at index 3.
- Person at index 2 would move to building at index 3.
- Person at index 3 can’t move to any other taller building.
- Person at index 4 would move to building at index 5.
- Person at index 5 can’t move to any other taller building.

Solution:

Brute force:

A simple naive solution would be for each person, we run a loop and check the tallest building to the right of it. The person would either stay at the current building if there is no taller building on right else would reach the tallest builind to the right.

Pseudo Code:

```
function(A):
    N = length of A
    ans = [] of length N
    for i from 0 to N - 1:
        tallest = -1
        for j from i to N - 1:
            tallest = max(tallest, A[j])
        ans[i] = tallest - A[i]
    return ans
```

- Time complexity: $O(N^2)$
- Space complexity: O(1)

Optimization:

For each index i, we can maintain a variable tallest[i], which would store the tallest building to the right of i (including i). We can do this using precalculation

Pseudo Code:

```
function(A):
    N = length of A
    tallest = [] of length N
    tallest[N - 1] = A[N - 1]
    for i from N - 2 to 0:
        tallest[i] = max(tallest[i + 1], A[i])

    ans = [] of length N
    for i from 0 to N - 1:
        ans[i] = tallest[i] - A[i]
    return ans
```

- Time complexity: O(N)
- Space complexity: O(N)

Question - 7

Given an integer array, find first missing positive integer. There are no duplicates.

Example - 1:

A = [10, 15, 3, 2, 8]

1 is missing in the array A.

Example - 2:

A = [10, 3, 1, 2, 5, -8, -3, 4]

6 is the first missing positive integer.

Solution:

Sol - 1: Brute force

In the brute force approach, we can simply search for each number in the array starting from 1 to N. If all numbers are present, then answer would be N + 1.

- Time complexity: $O(N^2)$
- Space complexity: O(1)

Sol - 2:

We will simply sort the array. Now, starting from index 0, we will find the first positive integer. Now, the numbers would be consecutive starting from 1. We will check them and report whenever we find a missing number.

- Time complexity: O(N * log(N))
- Space complexity: O(1)

Sol - 3:

Use extra array namely visited array to track elements from 1 to N.

- Time complexity: O(N)
- Space complexity: O(N)

Sol - 4:

We can improve the space complexity by modifying the array A itself. Since, there are no duplicates, whenever we encounter an element which is within the array [1, N], we simply swap it with the index.

Pseudo Code:

```
function(A):
    N = length of A
    for i from 0 to N - 1:
        if A[i] >= 1 and A[i] <= N:
            swap(A, A[A[i] - 1])
            i--

    for i from 0 to N - 1:
        if A[i] is not equal to i + 1:
            return i + 1

    return N + 1
```