Assignment-6

03 June 2023 14:27

Question 1

A permutation perm of n + 1 integers of all the integers in the range [0, n] can be represented as a string s of length n where:

- s[i] == 'I' if perm[i] < perm[i + 1], and
- s[i] == 'D' if perm[i] > perm[i + 1].

Given a string s, reconstruct the permutation perm and return it. If there are multiple valid permutations perm, return **any of them**.

Example 1:

Input: s = "IDID"

Output: [0,4,1,3,2]

```
class Solution {
public:
    vector<int> diStringMatch(string s) {
        int low=0;
        int high=s.size();
        vector<int>ans;

        for(int i=0;i<s.size();i++){
            if(s[i]=='I')
                ans.push_back(low++);

        else
            ans.push_back(high--);
        }
        ans.push_back(low);
        return ans;
    }
};
```

Question 2 You are given an m x n integer matrix matrix with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true *if* target *is in* matrix *or* false *otherwise*.

You must write a solution in O(log(m * n)) time complexity.

Example 1:

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

Output: true

```
class Solution {
public:
     bool binSearch(vector<vector<int>> res,int n,int m,int row,int key){
        int start=0;
        int end=m-1;

        while(start<=end){
            int mid=start+(end-start)/2;

            if(res[row][mid]==key)
                return true;
            else if(res[row][mid]<key)
                start=mid+1;
            else
                end=mid-1;
            mid=start+(end-start)/2;
        }
        return false;
    }
    bool search(vector<vector<int>>res,int n,int m,int target){
        int s=0;
    int e=n-1;
    while(s<=e){
      int mid=s+(e-s)/2;

      if(target>=res[mid][0] && target<=res[mid][m-1]){
        bool ans=binSearch(res,n,m,mid,target);
        return ans;
      }
      if(target<res[mid][0])
      e=mid-1;
      if(target>res[mid][m-1])
      s=mid+1;
      mid=s+(e-s)/2;

    }
    return false;

  }

  bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int n=matrix.size();
    int m=matrix[0].size();

    return (search(matrix,n,m,target));
  }
};
```

Question 3

Given an array of integers arr, return *true if and only if it is a valid mountain array*.

Recall that arr is a mountain array if and only if:

- arr.length >= 3
- There exists some i with 0 < i < arr.length - 1 such that:
  - arr[0] < arr[1] < ... < arr[i - 1] < arr[i]
  - arr[i] > arr[i + 1] > ... > arr[arr.length - 1]

Example 1:

Input: arr = [2,1]

Output:

false

```
class Solution {
public:
  bool validMountainArray(vector<int>& arr) {
    if(arr.size()<3)
      return false;
    int n=arr.size();
    int i=0;
    bool inc=false;
    bool dec=false;
    while(i+1<n && arr[i] < arr[i+1]){
      i++;
      inc=true;
    }
    while(i+1<n && arr[i]> arr[i+1]){
      i++;
      dec=true;
    }
    if(inc==true && dec==true && i==n-1)
      return true;
    return false;
  }
};
```

Question 4

Given a binary array nums, return *the maximum length of a contiguous subarray with an equal number of* 0 *and* 1.

Example 1:

Input: nums = [0,1]

Output: 2

Explanation:

[0, 1] is the longest contiguous subarray with an equal number of 0 and 1.

```
class Solution {
public:
    int findMaxLength(vector<int>& nums) {
        unordered_map<int,int>mp;
        int sum=0;
        int longest_subarray=0;

        for(int i=0;i<nums.size();i++){

            if(nums[i]==1){
                sum+=1;
            }
            if(nums[i]==0){
                sum-=1;
            }
            if(sum==0){
                longest_subarray=max(longest_subarray,i+1);
            }
            else if(mp.find(sum)!=mp.end()){
                longest_subarray=max(longest_subarray,i-mp[sum]);
            }
            else{
                mp[sum]=i;
            }

        }
        return longest_subarray;
    }
};
```

Question 5

The **product sum** of two equal-length arrays a and b is equal to the sum of a[i] * b[i] for all 0 <= i < a.length (**0-indexed**).

- For example, if a = [1,2,3,4] and b = [5,2,3,1], the **product sum** would be 15 + 22 + 33 + 41 = 22.

Given two arrays nums1 and nums2 of length n, return *the **minimum product sum** if you are allowed to **rearrange** the **order** of the elements in* nums1.

Example 1:

Input: nums1 = [5,3,4,2], nums2 = [4,2,2,5]

Output: 40

Explanation:

We can rearrange nums1 to become [3,5,4,2]. The product sum of [3,5,4,2] and [4,2,2,5] is 34 + 52 + 42 + 25 = 40.

```
class Solution {
public:
    int minProductSum(vector<int>& A, vector<int>& B) {
        sort(begin(A), end(A));
        sort(begin(B), end(B), greater<>());
        int ans = 0;
        for (int i = 0; i < A.size(); ++i) ans += A[i] * B[i];
        return ans;
    }
};
```

Question 6

An integer array original is transformed into a **doubled** array changed by appending **twice the value** of every element in original, and then randomly **shuffling** the resulting array.

Given an array changed, return original *if* changed *is a **doubled** array. If* changed *is not a **doubled** array, return an empty array. The elements in* original *may be returned in **any** order*.

Example 1:

Input: changed = [1,3,4,2,6,8]

Output: [1,3,4]

Explanation: One possible original array could be [1,3,4]:

- Twice the value of 1 is 1 * 2 = 2.
- Twice the value of 3 is 3 * 2 = 6.
- Twice the value of 4 is 4 * 2 = 8.

Other original arrays could be [4,3,1] or [3,1,4]

```
class Solution {
public:
  vector<int> findOriginalArray(vector<int>& changed) {
    int n=changed.size();
    vector<int>ans;
    if(n%2==1)
      return ans;

    unordered_map<int,int>mp;

    for(int i=0;i<n;i++){
      mp[changed[i]]++;
    }
    sort(changed.begin(),changed.end());
    for(auto x:changed){
      if(mp[x]==0)
        continue;

      if(mp[2*x]==0)
        return {};

      if(mp[2*x] && mp[x]){
        mp[2*x]--;
        ans.push_back(x);
        mp[x]--;
      }
    }
    return ans;

  }
};
```

Question 7

Given a positive integer n, generate an n x n matrix filled with elements from 1 to n2 in spiral order.

Example 1:

Input: n = 3

Output: [[1,2,3],[8,9,4],[7,6,5]]

```
class Solution {
public:
  vector<vector<int>> generateMatrix(int n) {
    vector<vector<int>> matrix (n,vector<int>(n,0));
    int top=0;
    int left=0;
    int right=n-1;
    int down=n-1;
    int val=1;

    while(left<=right && top<=down){
    for(int col=left;col<=right;col++){
      matrix[top][col]=val++;

    }
    top++;
    for(int row=top;row<=down;row++) {
      matrix[row][right]=val++;
```

```
        }
        right--;
        if(top<down && left<right){
        for(int col=right;col>=left;col--){
            matrix[down][col]=val++;
        }
        down--;

        for(int row=down;row>=top;row--){
            matrix[row][left]=val++;
        }
        left++;
        }

        }
         return matrix;
    }
};
```

Question 8

Given two sparse matrices mat1 of size m x k and mat2 of size k x n, return the result of mat1 x mat2. You may assume that multiplication is always possible.

Example 1:

Input: mat1 = [[1,0,0],[-1,0,3]], mat2 = [[7,0,0],[0,0,0],[0,0,1]]

Output:

[[7,0,0],[-7,0,3]]

```
class Solution {
public:
    vector<vector<int>> multiply(vector<vector<int>>& A, vector<vector<int>>& B) {
        int M = A.size(), K = A[0].size(), N = B[0].size();
        vector<vector<int>> ans(M, vector<int>(N));
        for (int i = 0; i < M; ++i) {
            for (int j = 0; j < N; ++j) {
                for (int k = 0; k < K; ++k) {
                    ans[i][j] += A[i][k] * B[k][j];
                }
            }
        }
        return ans;
    }
};
```