Striver's SDE Sheet

Day 4: Arrays Part-IV

Two Sum

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have *exactly* one solution, and you may not use the *same* element twice.

You can return the answer in any order.

Ans:

Observation and approach : (equation + process question)

In first go brute force approach will be apply two loop to find answer which take O(N^2) time

Code:

```
for (int i = 0; i < nums.size(); i++) {
            for (int j = i + 1; j < nums.size(); j++) {
                if (nums[i] + nums[j] == target) {
                    return {i, j};
                }
            }
        }
        return {};
```

optimize code:

to reduce time we have to increase space so e can optimize this question with help of hash map and use equation like x+y=target then x=target-y so find target-y in map if found then return answer

code:

```
  unordered_map<int, int> visited;
        int len = nums.size();
        for (int i = 0; i < len; ++i) {
            int n = nums[i];
            int complement = target - n;
            if (visited.count(complement)) {
                return {visited[complement], i};
            }
```

```
            visited[n] = i;  // assume that each input would have exactly one
solution
        }
        return {};
```

variation:

1) More than one pair exits (CD)
2) Instead of sum we take multiply , divided etc …

Code:

```
#include <bits/stdc++.h>

vector<vector<int>> pairSum(vector<int> &arr, int s){
    int cnt=0;
vector<vector<int>> ans;
unordered_map<int,int>mp;
for(int i=0;i<arr.size();i++){
if(mp[s-arr[i]]){
int val=mp[s-arr[i]];
vector<int> output;

while(val--){

output.push_back(min(arr[i],s-arr[i]));

output.push_back(max(arr[i],s-arr[i]));

ans.push_back(output);

}



}
mp[arr[i]]++;}
sort(ans.begin(),ans.end());
return ans;
}
```

4Sum

Given an array nums of n integers, return *an array of all the unique quadruplets* [nums[a], nums[b], nums[c], nums[d]] such that:

- 0 <= a, b, c, d < n
- a, b, c, and d are distinct.
- nums[a] + nums[b] + nums[c] + nums[d] == target

You may return the answer in any order.

Ans:

Observation and approach: process based

In first go brute force approach came in my mind is apply four loops to get our answer which take O(n^4) time 🤯

Second approach comes in my mind by seeing constraint is we can do it with help of recursion by pick non pick approach with some dp

Code recursion:

```
vector<vector<int>> nSum(vector<int> &nums, int target, int start, int n) {
  vector<vector<int>> result;
  if (start == nums.size() || nums[start] * n > target ||
      target > nums.back() * n) {
    return result;
  }

  if (n == 2) {
    // 3. We simply calculate twoSum when we need to.
    return twoSum(nums, target, start);
  }

  for (int i = start; i < nums.size(); i++) {
    if (i == start || nums[i - 1] != nums[i]) {
      for (auto &j : nSum(nums, target - nums[i], i + 1, n - 1)) {
        result.push_back({nums[i]});
        result.back().insert(end(result.back()), begin(j), end(j));
      }
    }
  }
```

```
    return result;
  }

  vector<vector<int>> fourSum(vector<int> &nums, int target) {
    sort(begin(nums), end(nums));
    return nSum(nums, target, 0, 4);
  }
};
```

Variation:

1) we can do for kth sum (recursion code contain kth sum)
2) what if repeated element is also considering

TODO:

Who we wise use space to reduce time means I know we use hash map to reduce time but who can I use it effectively

Space code:

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        set<vector<int>> st;
        int n = nums.size();
        for(int i=0;i<n;i++){
            for(int j=i+1;j<n;j++){
                set<long long> hashset;
                for(int k=j+1;k<n;k++){
                    long long sum = nums[i]+nums[j]; sum+=nums[k];
                    long long fourth = target-sum;
                    if(hashset.find(fourth)!=hashset.end()){
                   vector<int> temp={nums[i],nums[j],nums[k],(int)fourth};
                        sort(temp.begin(),temp.end());
                        st.insert(temp); }
                    hashset.insert(nums[k]);
                }
            }
        }
```

```
        vector<vector<int>> quadruplets(st.begin(),st.end());
        return quadruplets;

    }
};
```

Longest Consecutive Sequence

Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence*.

You must write an algorithm that runs in O(n) time.

Ans:

Observation and approach : process based

In first go brute force approach will be find all sequence and check whether the difference between them in one or not then return the answer take so much time nearly expo

Second approach strike in my mind is sort the array and travel the array if difference between a[i] and a[i+1] is one then ctn++ else if ctn =0

Code:

```
int longestConsecutive(vector<int>& arr) {
        int n = arr.size();
        if(n == 0) return 0;
        sort(arr.begin(), arr.end());
        int mxLen = 0, currLen = 1;
        for(int i = 1; i < n; i++) // traverse from the array
        {
            if(arr[i] == arr[i - 1] + 1)            {
                currLen++; // increase the curr Length by 1
            }
            else if(arr[i] != arr[i - 1]) // but if it is not equal
            {
                mxLen = max(mxLen, currLen); // update our mxLen
                currLen = 1; // and reset the currLen with 1
            }
        }

        mxLen = max(mxLen, currLen);
        return mxLen;
```

in this question I saw some solution which using=g unnecessary space , but why?

Largest subarray with 0 sum

Given an array having both positive and negative integers. The task is to compute the length of the largest subarray with sum 0.

Ans:

Observation and approach : process based

In first go brute force approach will be find all sub array and check whether the sum is zero or not , takes to much time

Second approach strike in my mind is similar to kadane's algo in which we travel a array and maintain sum and length if sum ==0 then check for max length so it take O(N) time

But but problem is who we can find length ? 😳

After watching solution I find that uses same approach and to keep track of length we map and .find map method to find last element index so we get start-last and get length

Code:

```
int maxLen(vector<int>&a, int n)
    {
        // Your code here
       unordered_map<int, int> mp;
       mp[0] = -1;
       int mx = 0, sum = 0;
       for(int i=0; i<n;i++)
       {
           sum += a[i];
           if(mp.find(sum) != mp.end())
           mx = max(mx, i-mp[sum]);
           else
           mp[sum] = i;
       }
       return mx;
```

this approach may fails is contains duplicate number because map .find confuse in two number means get two ranges

Count Subarrays with Given XOR

Given an array of integers 'ARR' and an integer 'X', youare supposed to find the number of subarrays of 'ARR' which have bitwise XOR of the elements equal to 'X'.

Ans:

Observation and approach : resemble to above questions

In first go brute force approach came in my mind is to apply two loops and calculate xor and matches than ctn ++;

Code:

```
int n=arr.size();
// sort(arr.begin(),arr.end());
int cur;
int k=0;
for(int i=0;i<n;i++)
{
    cur=arr[i];
    if(cur==x)
    k++;
    for(int j=i+1;j<n;j++)
    {
        cur=cur^arr[j];
        if(cur==x)
         {
             k++;
         }
    }
}
return k;
```

optimize code:

we can use approach of above question (similar to kadane's)

code:

```
unordered_map<int,int> m;
   m[0]=1;
   int count=0;
   int xorr=0;
   for(int i=0;i<arr.size();i++){
```

```
xorr^=arr[i];
if(m.find(xorr^x)==m.end())m[xorr]++;
else{
count+=m[xorr^x];
m[xorr]++;
}
    }

return count;
```

Longest Substring Without Repeating Characters

Given a string s, find the length of the longest substring without repeating characters.

Ans:

Observation and approach : process based

In first go brute force is to apply two loop and check for duplicated by map if there exit and check for max

Code:

For(int i=0;i<n;i++){

For(int j=i;j<n;j++){

If(map.find(a[j])==map.end()){

Ctn++;

}

Else{

Map.clear;

Max_=max(ctn,max_);

}

}

}

Second approach strike in my mind of sliding window concept in which we maintain a window (variable) to check if contain all unique and maintain size of window to find max

If a duplicate one is found then remove that window and start inserting new elements

Code:

```cpp
int lengthOfLongestSubstring(string s) {
        int n = s.size();
        int l=0, r = n-1;

        unordered_set<char> visited;

        int maxStr = 0;

        for(int r=0;r<n;r++)
        {
            if(visited.find(s[r])==visited.end())
            {
                visited.insert(s[r]);
                maxStr = max(maxStr,r-l+1);

            }
            else
            {
                while(l!=r && s[l]!=s[r])
                    visited.erase(s[l++]);

                //Removing and adding the same element
                visited.erase(s[l++]);
                visited.insert(s[r]);

                maxStr = max(maxStr,r-l+1);
            }
        }

        return maxStr;
    }
```

Observation in Above Question:

1) TODO : who to use space efficiently to reduce time
2) As I observer on previous day that algorithm are not used directly but there approaches is used , today most of the question use kadane's algo (not total but up to some extended)