Today’s topics

1. Concept of Prefix
2. Problem: Highest altitude
3. Frequency arrays
4. Problem: Most frequent character
5. Handling ranges
6. Problem: Maximum platform needed
7. A technique to approach problems
8. Problem: Subarray sum divisible by K
9. Sliding window (If time permits)
10. Problem: Finding K beauty of a number (if time permits)

Concept of prefix

General definition

A prefix is a letter or group of letters, for example 'un-' or 'multi-', which is added to the beginning of a word in order to form a different word.

Example, **un**manageable, **un**happy

Prefix for arrays

Any **continuous** segment of array starting from index 0 is a prefix

Example:

Array = [1, 2, 3, 4, 5]

Prefixes:

[1]

[1, 2]

[1, 2, 3]

[1, 2, 3, 4]

[1, 2, 3, 4, 5]

Prefix sum

It is a sum array that we create from main array, where prefix_sum[i] = sum of all the elements of the array from 0 to i

How do we calculate prefix sum?

▼ Brute force

Steps:

1. Loop from 0 to n. Let's say looping variable is i
2. For each i, to get the prefix sum at i, we need to sum all elements from 0 to i. Run another loop from 0 to i and add all elements

3. Store the sum in prefix_sum[i]

```
int prefix_sum[n] = {0}
for(int i=0; i<n; i++) {
    sum = 0;
    for(int j=0; j<=i; j++) {
        sum += arr[j];
    }
    prefix_sum[i] = sum;
}
```

▼ Optimal

Let's think it a bit mathematically. What is prefix_sum[i] and prefix_sum[i-1]?

prefix_sum[i] = arr[0] + arr[1] + arr[2] + ... + arr[i-1] + arr[i]

prefix_sum[i-1] = arr[0] + arr[1] + arr[2] + ... + arr[i-1]

We can see the common parts, arr[0] + arr[1] + ... + arr[i-1]

So, prefix_sum[i] = prefix_sum[i-1] + arr[i].

🚧 Use the above for i = 1 to n-1 but not 0. As prefix_sum[i-1] is not valid. Populate prefix_sum[0] = arr[0] separately.

```
int prefix_sum[n] = {0};
prefix_sum[0] = arr[0];
for(int i=0; i<n; i++) {
    prefix_sum[i] = prefix_sum[i-1] + arr[i]
}
```

Problem 1: Highest altitude

Link: https://leetcode.com/problems/find-the-highest-altitude/description/

▼ **Solution**

Let’s say you are at xth point. What is your altitude right now?

It is gain[0] + gain[1] + gain[2] + .. + gain[x-1]. Let’s call it altitude[x]

What is maximum altitude point? It is Max(altitude[i]).

So we can create altitude[] using prefix sum and find the maximum altitude point.

Frequency arrays

Sometimes, we might want to maintain frequencies of something. Let's take this problem:

Given a list of integers, find the integer with maximum frequency.

For example, if we are given an array with [1, 2, 1, 1, 3, 3], then the answer will be 1 as frequency of 1 is 3 vs that of 2 and 3 which are 1 and 2 respectively.

In these type of problems, what we can do is create a **frequency array**

A **frequency array,** is a special array that stores the frequencies of all the elements.

So in our example the frequency array can be something like this:

`freq =  0, 3, 1, 2]`

Where,

freq[0] = how many times 0 occurs in the array = 0

freq[1] = how many times 1 occurs in the array = 3

freq[2] = how many times 2 occurs in the array = 1

freq[3] = how many times 3 occurs in the array = 2

We can clearly see, the array elements of the main array are **indices in the frequency array.**

So what should be the size of the frequency array?

Now, how to calculate this frequency array

▼ Solution

Maintain a frequency array freq[]. The size of this array should be the maximum value in nums list + 1 (Why?)

Iterate through all the elements and increase their frequency using: `freq[nums[i]] += 1`

Now loop through the frequency array and find which index has maximum frequency. The index is the answer.

▼ Code

```
int max_value = *(max_element(arr, arr+n));
int freq[max_value + 1] = {0};

for(int i=0; i<n; i++) {
   freq[arr[i]] += 1
}
```

💡 Remember, the size of the array is dependent on the array elements so in case the array elements are very large (~10^8) then don't use this technique directly otherwise it will give Memory Limit Exceeded errors.

💡 Frequency array is a special form of hash table. So we can also use it with hashing for other data types or large integer values. Hashing is a technique of mapping one thing to another using a hash function. A hash function takes an argument of one type, and converts it into another value either of same type or different type. Generally, the output of hash functions is integers.

Problem 2: Most frequent character

Given a string containing lower case english alphabets, what is the most occurring character?

Example 1:

```
Input: s = "hello"
Output: l
Explanation: Frequencies - 'h': 1, 'e': 1, 'l': 2, 'o': 1
```

▼ Solution

Handling ranges: Prefix sum + frequency array

Often we are given problems around ranges. Let’s take this problem: https://leetcode.com/problems/maximum-population-year/description/

Essentially what do we want here? What is the year when maximum number of people are living.

The brute force solution can be:

```
1. Iterate through all the possible years
2. For an year, iterate through the list of ranges and see if the person is alive fo
3. Update the ans to count if greater
```

How can we optimise this? Think of this problem as range intersection, basically given a list of ranges, what is the maximum number of ranges that are intersecting at a point.

One way to solve it is using sorting the ranges and then iterating. However, we will do a better approach.

Let’s say we maintain a frequency array for the years, where freq[i] = number of people alive in year i.

To populate this array we can do something like:

```
freq[2051] = {0}                                    Copy

for [birth, death] in persons {
    for (year -> birth to death) freq[year] += 1
}
```

Now we can easily use the freq array to find the answer which is the index at which freq is maximum.

But can we optimise this further? Yes!

Think it like the altitude problem: For each year, we can have a gain[year] value, which is what is the gain / loss in population for this year.

If we have gain[year], calculating freq is simple right? Just do prefix sum! (Same logic as maximum altitude, where we found total altitude at every point using prefix sum)

Now how to calculate gain efficiently? Well, iterate through all the ranges and gain[birth] += 1, gain[death] -= 1. Because, when someone is born, we are gaining by 1, but when someone dies, we lose by 1.

▼ Final solution

```
Steps:                                                        Copy
1. Calculate gain array
        1.1 Create a gain array with all 0 initially
        1.2 Loop over all birth/death pairs. For each birth, death pair
2. Calculate prefix sum to find the freq of number of people alive ever
3. Calculate the index where frequency is maximum
```

Problem 3: Maximum platforms needed

Problem Link: https://www.geeksforgeeks.org/minimum-number-platforms-required-railwaybus-station/

▾ Solution

Same as previous problem. We will use the handling ranges using prefix sum and frequency array technique.

Create a frequency array for all possible times. Fill it up by calculating gain[] and then doing prefix sum to get actual frequency

The maximum value of frequency is the minimum number of platforms needed. Why? Because for any time, at max these number of trains can be at the station and no 2 trains can be in same platform at same time.

A technique to approach problems

Whenever you have a problem which involves finding **2 indices** (one or all pairs) from the array following some property, you can solve it by treating each element as the `right` element of the pair and trying to find an appropriate `left` element such that (left, right) satisfies the property. As we are moving from left to right in the loop, when we reach any element, the data structure only contains elements on the left hand side. So using the data structure, we should be able to quickly compute the left elements satisfying the property with the current element.

Let's understand this with the example of 2 sum problem. We are asked to find count of all possible pairs of elements whose sum is equal to K. How to solve for it?

It satisfies the above conditions we have mentioned, i.e. we need to find **count of all pairs** satisfying a property which is **A[i] + A[j] = K**

So, we can iterate and fix the right element, and for each right element try finding count of all possible left elements s/t the pairs satisfy the property.

In brute force terms, it will be something like

```
for(right = 0; right < n; right++) {
   // right is fixed. For this right, find count of all left's such that (left, righ
   for(left = 0; left < right; left++) {
           if (isPropertySatisfied(left, right)) {
                   ans += 1
           }
   }
}
```
Copy

Now we optimize this and completely get rid of the inner loop by using some data structure. Essentially, `isPropertySatisfied` is defined as `A[left] + A[right] == K` . Instead, we can maintain a frequency array, that stores the count of all the array elements we have seen till now in our loop. As we are moving from left to right, only the elements we have seen till now (i.e. the elements on the left hand side of the current element) are there in the frequency table.

The optimized code becomes something like the below:

```
freq = [] // All 0's initially
for(right = 0; right < n; right++) {
   // right is fixed. For this right, find count of all left's such that (left, righ
   need = K - A[right]
   ans += freq[need]
   freq[A[right]]++
}
```
Copy

💡 A subarray can also be treated as 2 indices, in case of subarrays we use the prefix computation concept as well. To compute for a subarray (i, j), we can use the prefix computation (0, j) and from that remove the computation for (0, i-1)

Problem 4: Subarray sums divisible by K

Problem Link: https://leetcode.com/problems/subarray-sums-divisible-by-k/description/

▼ Code

Copy

```
class Solution {
public:
    int subarraysDivByK(vector<int>& nums, int k) {
        int n = nums.size(); // Size of the array
        vector<int> psum(n, 0); // Prefix sum array
        psum[0] = nums[0];
        for(int i=1; i<n; i++) {
            psum[i] = psum[i-1] + nums[i];
        }

        // psum[i] % k
        for(int i=0; i<n; i++) {
            psum[i] = (psum[i] + k) % k;
        }

        // In the array psum, we want to find all pairs (i, j)
        // s/t psum[i] == psum[j]
        // all the elements in psum are in the range [0, k-1]
        vector<int> freq(k, 0);

        int ans = 0;
        for(int j=0; j<n; j++) {
            // Here we have fixed j in the pair (i, j)
            // We want to find all the i's that are satisfying the prop
            // So, we want all the i's such that psum[i] == psum[j] and
            // Let's say we are index 5 and psum[5] = 3
            // To reach 5, we have already covered 0, 1, 2, 3, 4 in the
            // So we have already executed freq[psum[j]] ++ for all j =
            // So the freq table contains the correct frequency of all
            ans += freq[psum[j]];
            freq[psum[j]] ++;
            if (psum[j] == 0) {
                // psum[j] alone satisifies the property and it represe
                ans ++;
            }
        }

        return ans;
    }
};
```

Sliding window

We use sliding window in problems where we have to solve it for **fixed size** subarrays.

We are given an window (subarray) size of K. For each window, we have to compute some value. Using sliding window algorithm, this computation is optimised.

💡 As mentioned, sliding window is only used for fixed sized subarray problems, where you need to compute something for all subarrays of size K. However, this does not mean that the problem has to be directly of this type. Rather, if you are able to reduce the problem to this or a part of the problem involves computation of all subarrays of size K, then you can use sliding window. An example can be Binary search + Sliding window. We will discuss more about this when we learn about Binary search.

The logic is, we should be able to design 2 operations: `in` and `out` .

`in` means being able to add something to the result. add doesn't mean addition operation, rather if we have a result calculated for a set of elements, how can we recalculate quickly if one element is added to the set.

`out` means recalculating when one element is removed from the set.

We calculate manually for the first window i.e loop through all the K elements [0, K-1] and calculate the answer. If needed, we maintain some data structure as well.

Now we start with the sliding window algorithm. As the window slides, we can see from the picture one element gets added and one element gets removed. We use our `in` and `out` operations respectively to recalculate the answer

For example,

when the window moves for the first time, element at index K is added and element at index 0 is removed. So we call `in(K)` and `out(0)`

Example

Given an array nums of size n and an integer K, calculate the maximum sum subarray of size K.

▼ Solution

We can use sliding window approach as we have to calculate the sum for every subarray of size K

`in` : We are maintaining the sum of a set of elements. As new elements are added, we can add them to the sum.

`out` : As elements are removed we can subtract their values

▼ Code

Copy

```
int sum = 0;
int max_sum = 0;
```

```
// Calculate for first window
for(int i=0; i<K; i++) sum += arr[i];
max_sum = sum;

// Now do sliding window
for(int i=K; i<n; i++) {
    // Do in operation
    sum += arr[i];

    // Do out operation
    sum -= arr[i-K];

    // Now we have the updated value for current window. Update ans
    max_sum = max(max_sum, sum);
}
```

Problem 5: Find the K-Beauty of a Number

Problem Link: https://leetcode.com/problems/find-the-k-beauty-of-a-number/description/

Problem 6: Number of substrings containing all 3 characters

Q - https://leetcode.com/problems/number-of-substrings-containing-all-three-characters/description/

▼ Code

Copy

```
int numberOfSubstrings(string s) {
        vector<int> last(3, -1);
        int ans = 0;
        for(int i=0; i<s.length(); i++) {
            last[(int)(s[i] - 'a')] = i;
            if(last[0] != -1 and last[1] != -1 and last[2] != -1) {
                int from = min(last[0], min(last[1], last[2]));
                ans += (from + 1);
            }
        }
        return ans;
    }
```

Problem 7: Count number of nice subarrays

Q - https://leetcode.com/problems/count-number-of-nice-subarrays/description/