HASHMAP

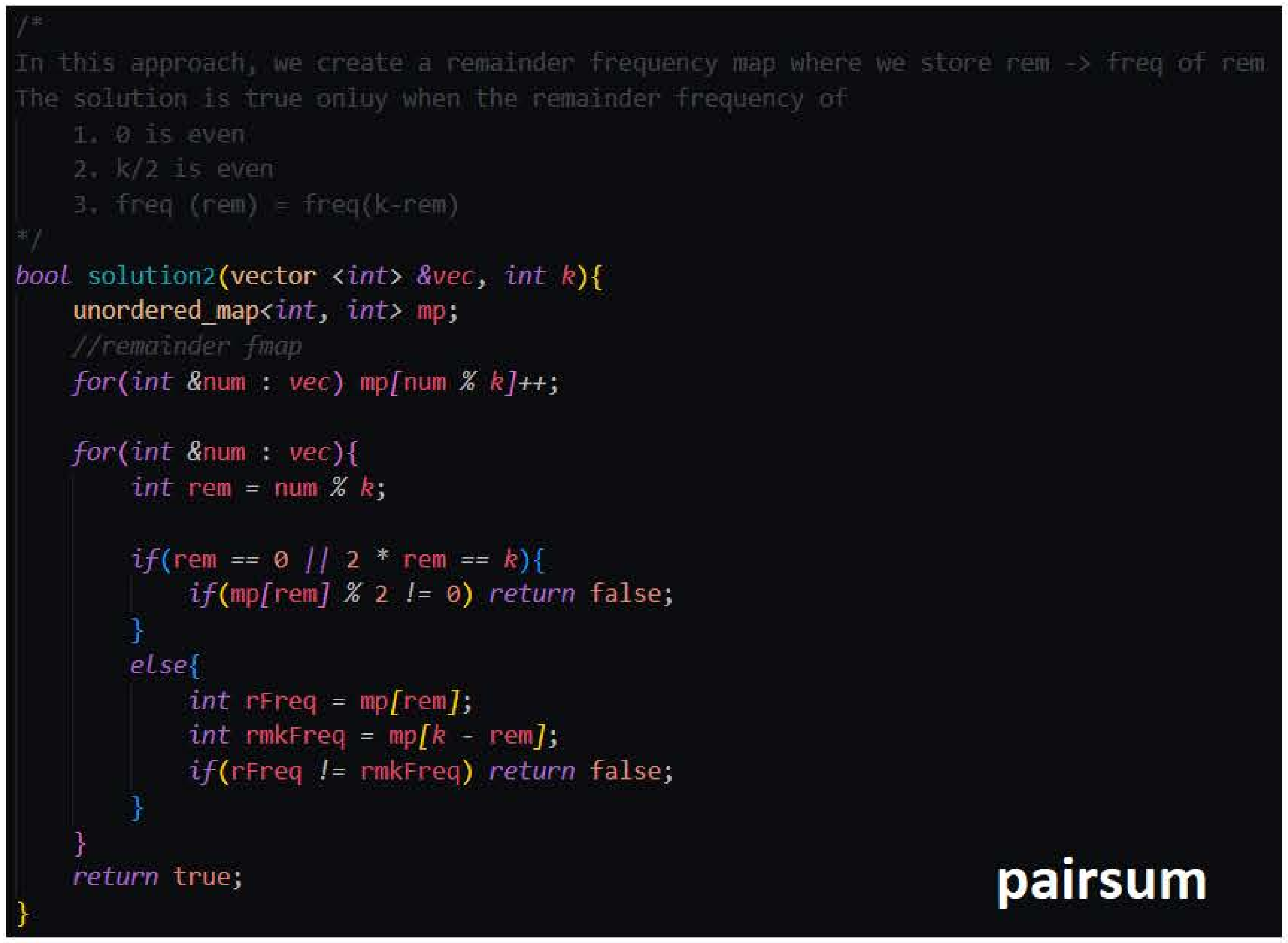
1. Decide which strategy to use

    1. freq map

        whenever we encounter question related to divisibility, it can most probably be solved using fmap (remainder frequency map)

    2. Acquire and release - Acquire till condition **is valid** -> print ans -> release till the condition **just gets invalid**

    3. prefix sum

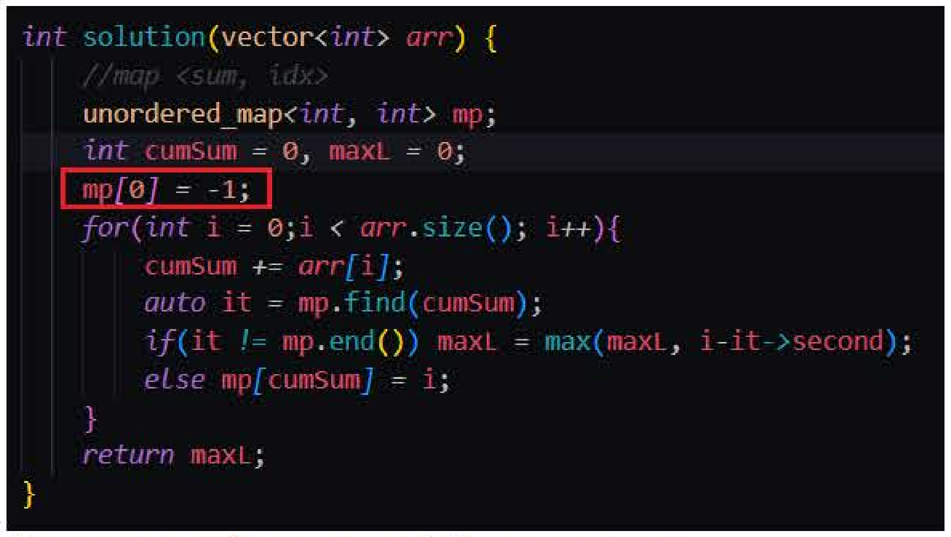


```cpp
/*
In this approach, we create a remainder frequency map where we store rem -> freq of rem
The solution is true onluy when the remainder frequency of
    1. 0 is even
    2. k/2 is even
    3. freq (rem) = freq(k-rem)
*/
bool solution2(vector <int> &vec, int k){
    unordered_map<int, int> mp;
    //remainder fmap
    for(int &num : vec) mp[num % k]++;

    for(int &num : vec){
        int rem = num % k;

        if(rem == 0 || 2 * rem == k){
            if(mp[rem] % 2 != 0) return false;
        }
        else{
            int rFreq = mp[rem];
            int rmkFreq = mp[k - rem];
            if(rFreq != rmkFreq) return false;
        }
    }
    return true;
}
```

pairsum

array can be divided into pairs such that the sum of every pair is divisible by k.

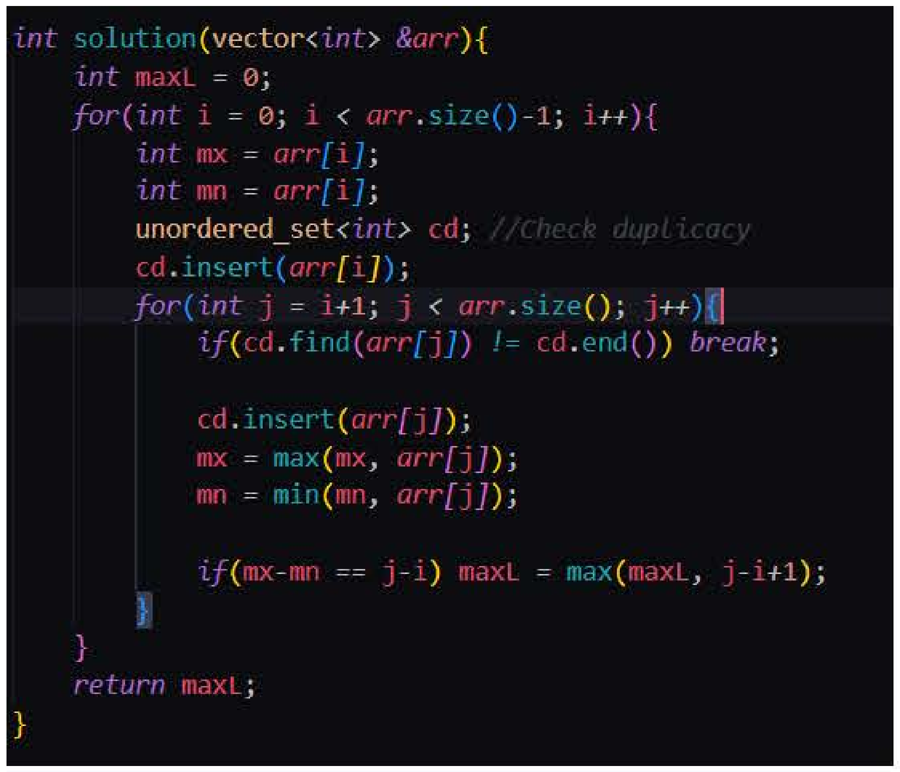
if sum of (num1 % k) + (num2 % k) == k
then num1+num2 is divisible by k



```cpp
int solution(vector<int> arr) {
    //map <sum, idx>
    unordered_map<int, int> mp;
    int cumSum = 0, maxL = 0;
    mp[0] = -1;
    for(int i = 0;i < arr.size(); i++){
        cumSum += arr[i];
        auto it = mp.find(cumSum);
        if(it != mp.end()) maxL = max(maxL, i-it->second);
        else mp[cumSum] = i;
    }
    return maxL;
}
```

<u>Largest subarray with zero sum</u>

We calculate cumulative sum of the array
while doing so, if value of cumulative sum repeats => sum after prev occurance
till curridx is 0
we update max len if the length of substring is greater then prev length

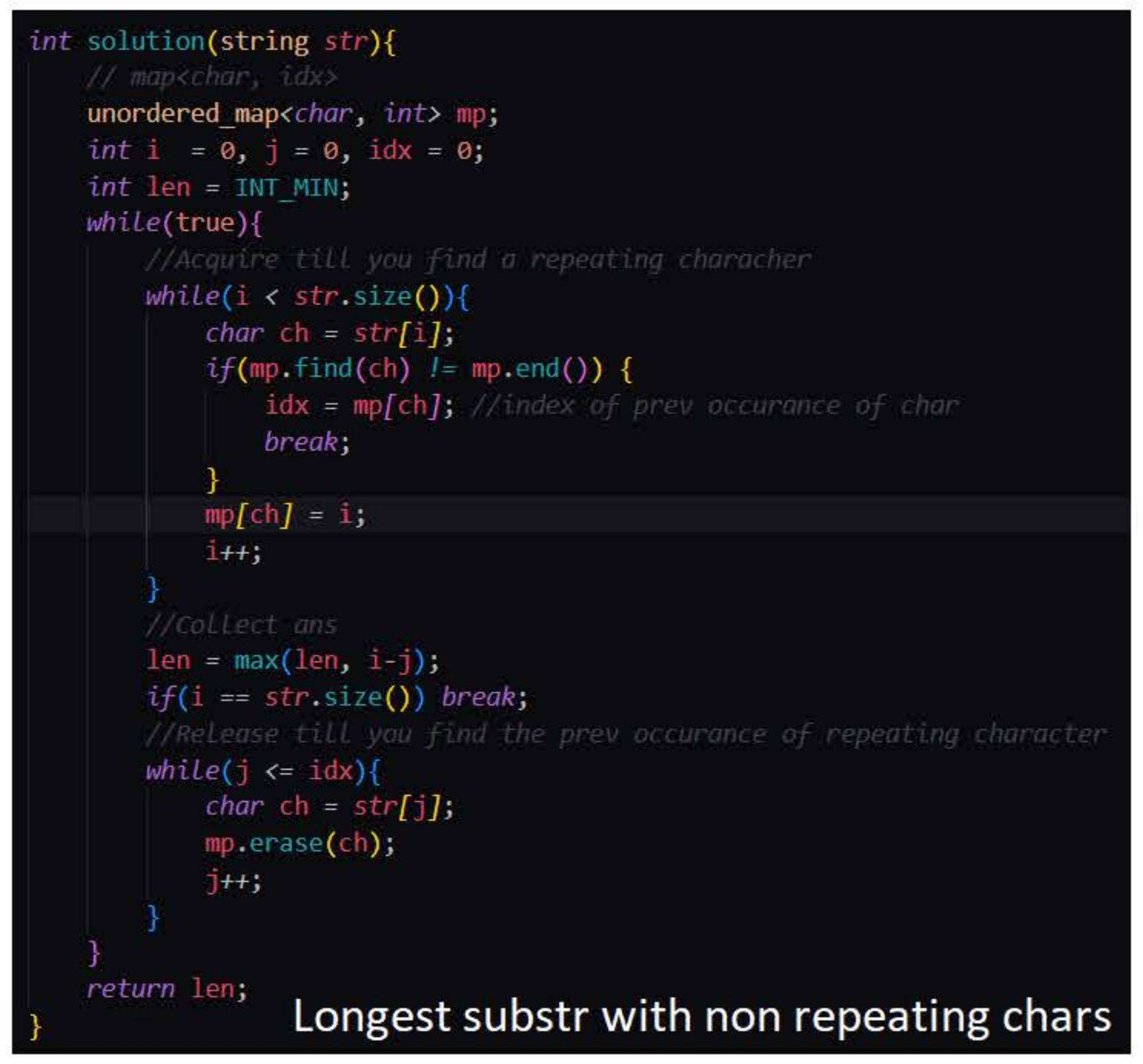


```cpp
int solution(vector<int> &arr){
    int maxL = 0;
    for(int i = 0; i < arr.size()-1; i++){
        int mx = arr[i];
        int mn = arr[i];
        unordered_set<int> cd; //Check duplicacy
        cd.insert(arr[i]);
        for(int j = i+1; j < arr.size(); j++){
            if(cd.find(arr[j]) != cd.end()) break;

            cd.insert(arr[j]);
            mx = max(mx, arr[j]);
            mn = min(mn, arr[j]);

            if(mx-mn == j-i) maxL = max(maxL, j-i+1);
        }
    }
    return maxL;
}
```

<u>07 LongestSubArrWithContiniousElements</u>

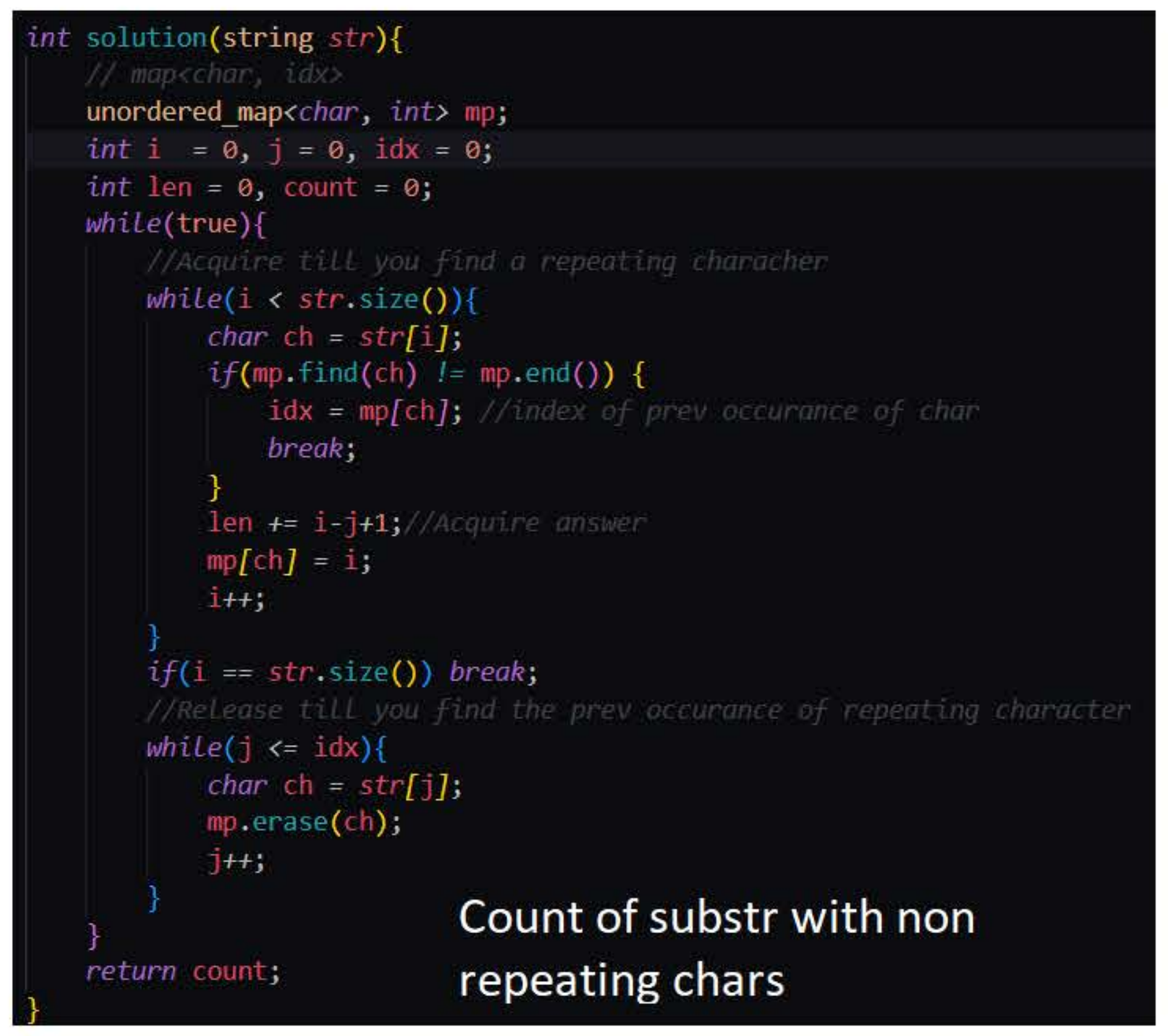
In an array, if a subarray [say from idx i to j]
contains continious elements, then it follows a
property
max-min = j - i

Questions related to SUBSTR WITH UNIQUE



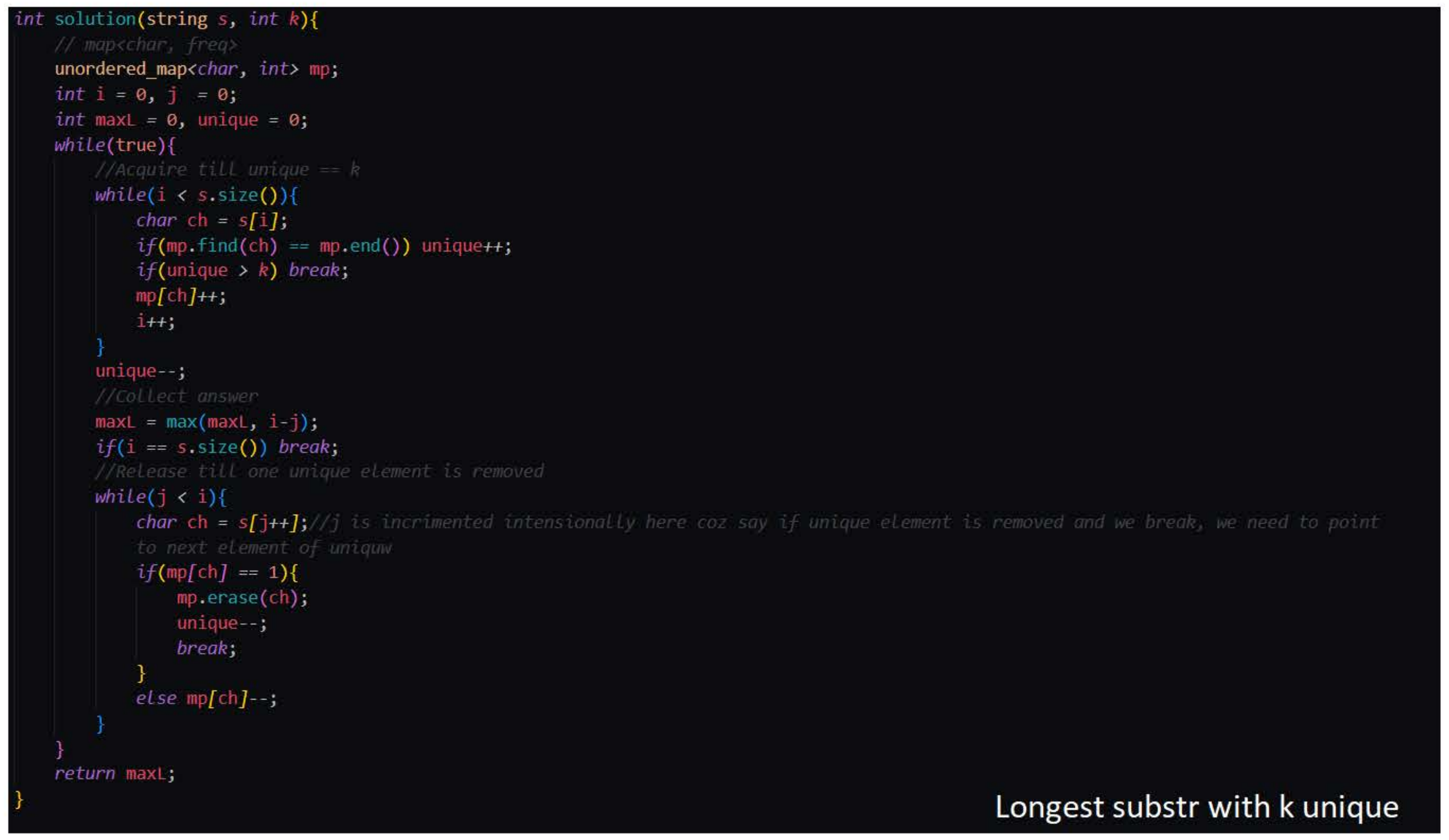
```cpp
int solution(string str){
    // map<char, idx>
    unordered_map<char, int> mp;
    int i  = 0, j = 0, idx = 0;
    int len = INT_MIN;
    while(true){
        //Acquire till you find a repeating characher
        while(i < str.size()){
            char ch = str[i];
            if(mp.find(ch) != mp.end()) {
                idx = mp[ch]; //index of prev occurance of char
                break;
            }
            mp[ch] = i;
            i++;
        }
        //Collect ans
        len = max(len, i-j);
        if(i == str.size()) break;
        //Release till you find the prev occurance of repeating character
        while(j <= idx){
            char ch = str[j];
            mp.erase(ch);
            j++;
        }
    }
    return len;
}
```

Longest substr with non repeating chars



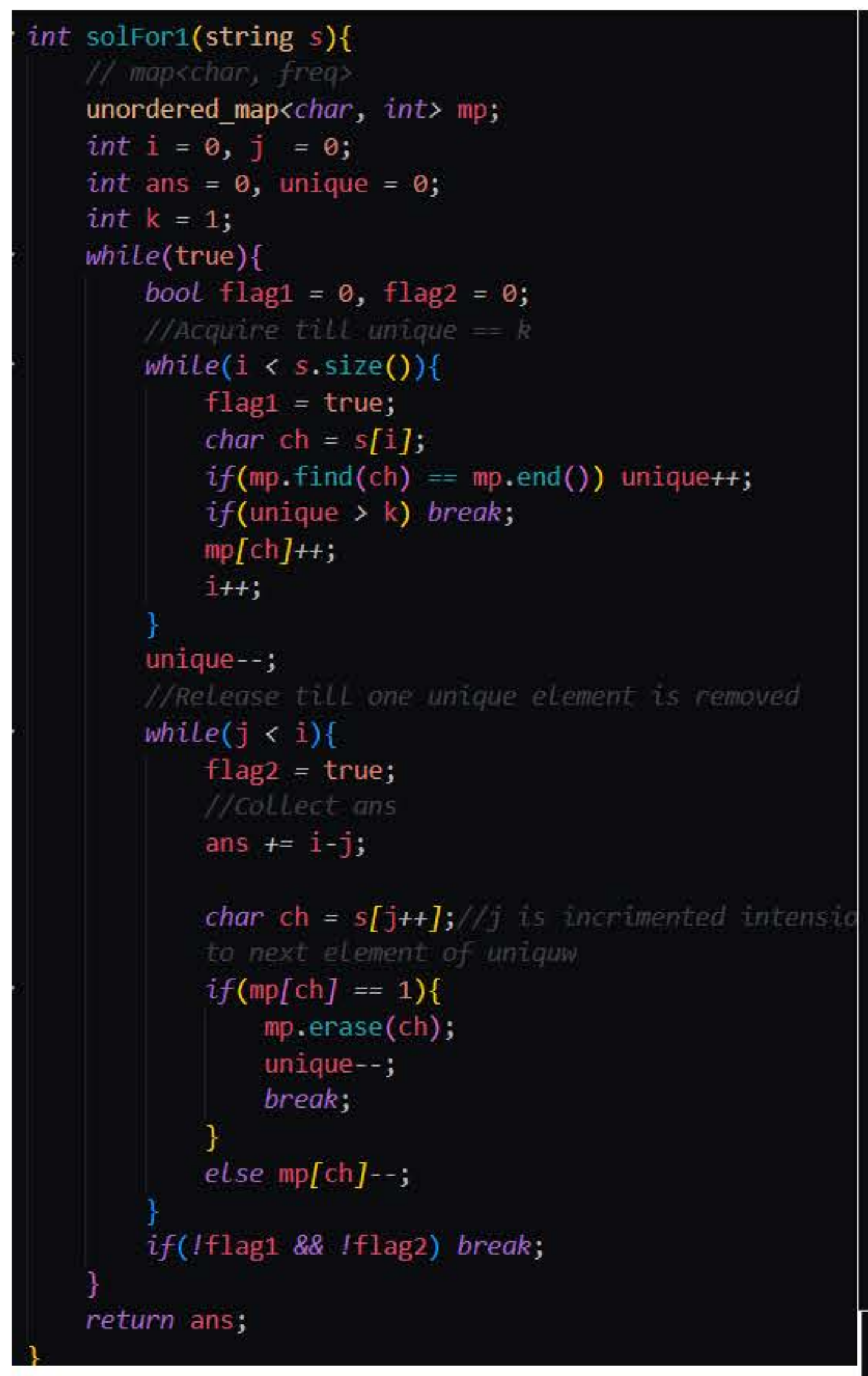
```cpp
int solution(string str){
    // map<char, idx>
    unordered_map<char, int> mp;
    int i  = 0, j = 0, idx = 0;
    int len = 0, count = 0;
    while(true){
        //Acquire till you find a repeating characher
        while(i < str.size()){
            char ch = str[i];
            if(mp.find(ch) != mp.end()) {
                idx = mp[ch]; //index of prev occurance of char
                break;
            }
            len += i-j+1;//Acquire answer
            mp[ch] = i;
            i++;
        }
        if(i == str.size()) break;
        //Release till you find the prev occurance of repeating character
        while(j <= idx){
            char ch = str[j];
            mp.erase(ch);
            j++;
        }
    }
    return count;
}
```

Count of substr with non repeating chars

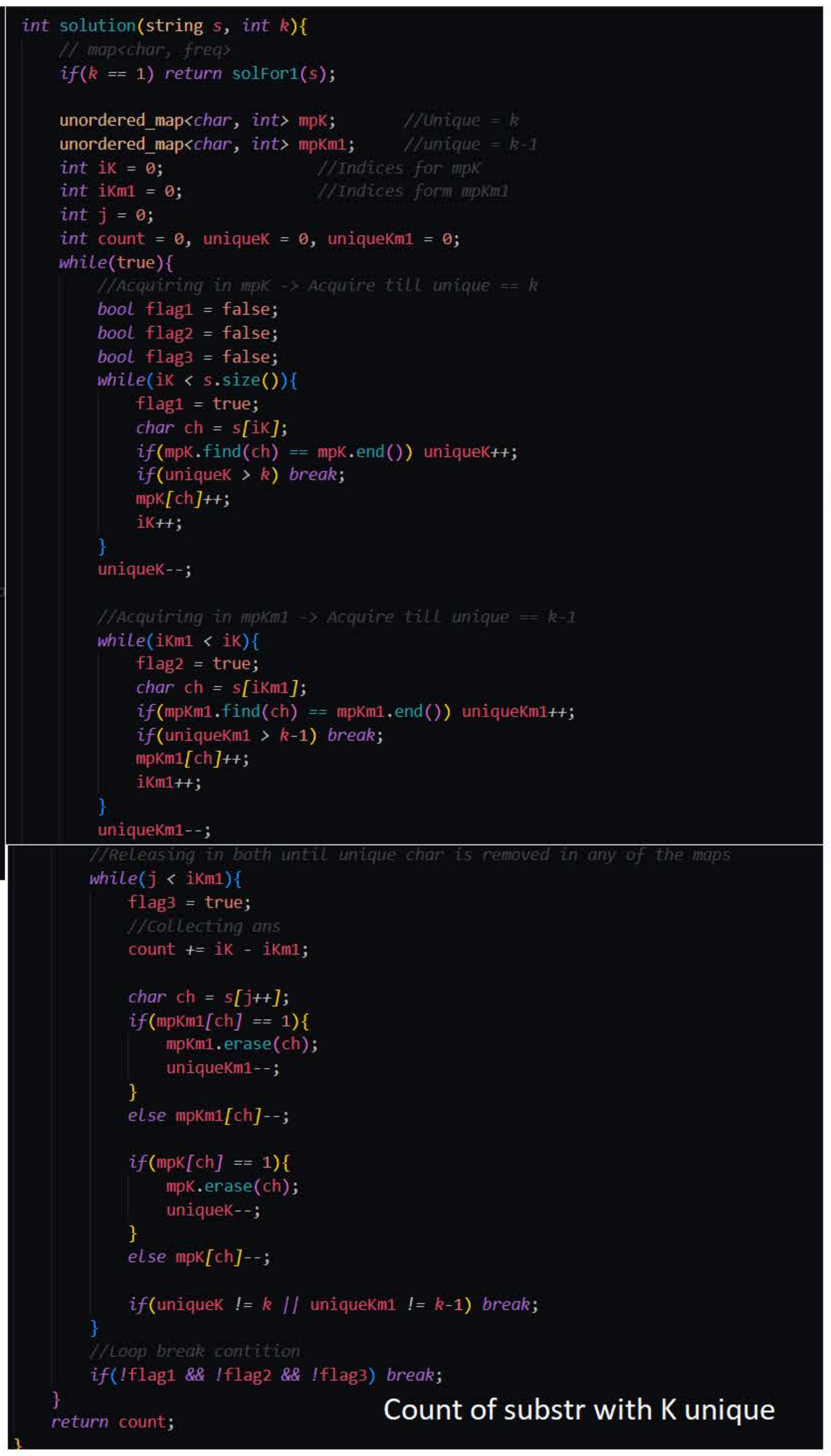


```cpp
int solution(string s, int k){
    // map<char, freq>
    unordered_map<char, int> mp;
    int i = 0, j  = 0;
    int maxL = 0, unique = 0;
    while(true){
        //Acquire till unique == k
        while(i < s.size()){
            char ch = s[i];
            if(mp.find(ch) == mp.end()) unique++;
            if(unique > k) break;
            mp[ch]++;
            i++;
        }
        unique--;
        //Collect answer
        maxL = max(maxL, i-j);
        if(i == s.size()) break;
        //Release till one unique element is removed
        while(j < i){
            char ch = s[j++];//j is incrimented intensionally here coz say if unique element is removed and we break, we need to point
            to next element of uniquw
            if(mp[ch] == 1){
                mp.erase(ch);
                unique--;
                break;
            }
            else mp[ch]--;
        }
    }
    return maxL;
}
```

Longest substr with k unique

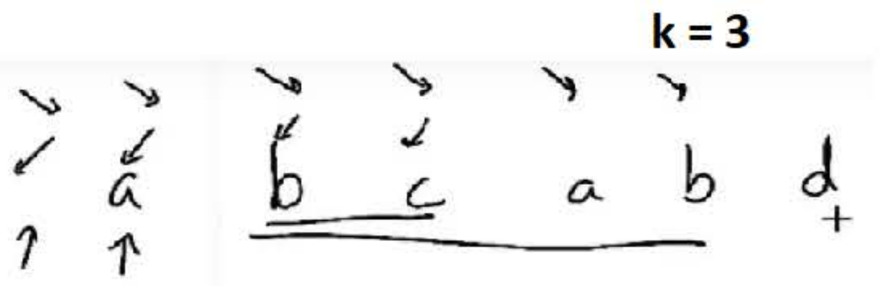


```cpp
int solFor1(string s){
    // map<char, freq>
    unordered_map<char, int> mp;
    int i = 0, j  = 0;
    int ans = 0, unique = 0;
    int k = 1;
    while(true){
        bool flag1 = 0, flag2 = 0;
        //Acquire till unique == k
        while(i < s.size()){
            flag1 = true;
            char ch = s[i];
            if(mp.find(ch) == mp.end()) unique++;
            if(unique > k) break;
            mp[ch]++;
            i++;
        }
        unique--;
        //Release till one unique element is removed
        while(j < i){
            flag2 = true;
            //Collect ans
            ans += i-j;

            char ch = s[j++];//j is incrimented intensio
            to next element of uniquw
            if(mp[ch] == 1){
                mp.erase(ch);
                unique--;
                break;
            }
            else mp[ch]--;
        }
        if(!flag1 && !flag2) break;
    }
    return ans;
}
```



```cpp
int solution(string s, int k){
    // map<char, freq>
    if(k == 1) return solFor1(s);

    unordered_map<char, int> mpK;       //Unique = k
    unordered_map<char, int> mpKm1;     //unique = k-1
    int iK = 0;                     //Indices for mpK
    int iKm1 = 0;                   //Indices form mpKm1
    int j = 0;
    int count = 0, uniqueK = 0, uniqueKm1 = 0;
    while(true){
        //Acquiring in mpK -> Acquire till unique == k
        bool flag1 = false;
        bool flag2 = false;
        bool flag3 = false;
        while(iK < s.size()){
            flag1 = true;
            char ch = s[iK];
            if(mpK.find(ch) == mpK.end()) uniqueK++;
            if(uniqueK > k) break;
            mpK[ch]++;
            iK++;
        }
        uniqueK--;

        //Acquiring in mpKm1 -> Acquire till unique == k-1
        while(iKm1 < iK){
            flag2 = true;
            char ch = s[iKm1];
            if(mpKm1.find(ch) == mpKm1.end()) uniqueKm1++;
            if(uniqueKm1 > k-1) break;
            mpKm1[ch]++;
            iKm1++;
        }
        uniqueKm1--;
        //Releasing in both until unique char is removed in any of the maps
        while(j < iKm1){
            flag3 = true;
            //Collecting ans
            count += iK - iKm1;

            char ch = s[j++];
            if(mpKm1[ch] == 1){
                mpKm1.erase(ch);
                uniqueKm1--;
            }
            else mpKm1[ch]--;

            if(mpK[ch] == 1){
                mpK.erase(ch);
                uniqueK--;
            }
            else mpK[ch]--;

            if(uniqueK != k || uniqueKm1 != k-1) break;
        }
        //Loop break contition
        if(!flag1 && !flag2 && !flag3) break;
    }
    return count;
}
```

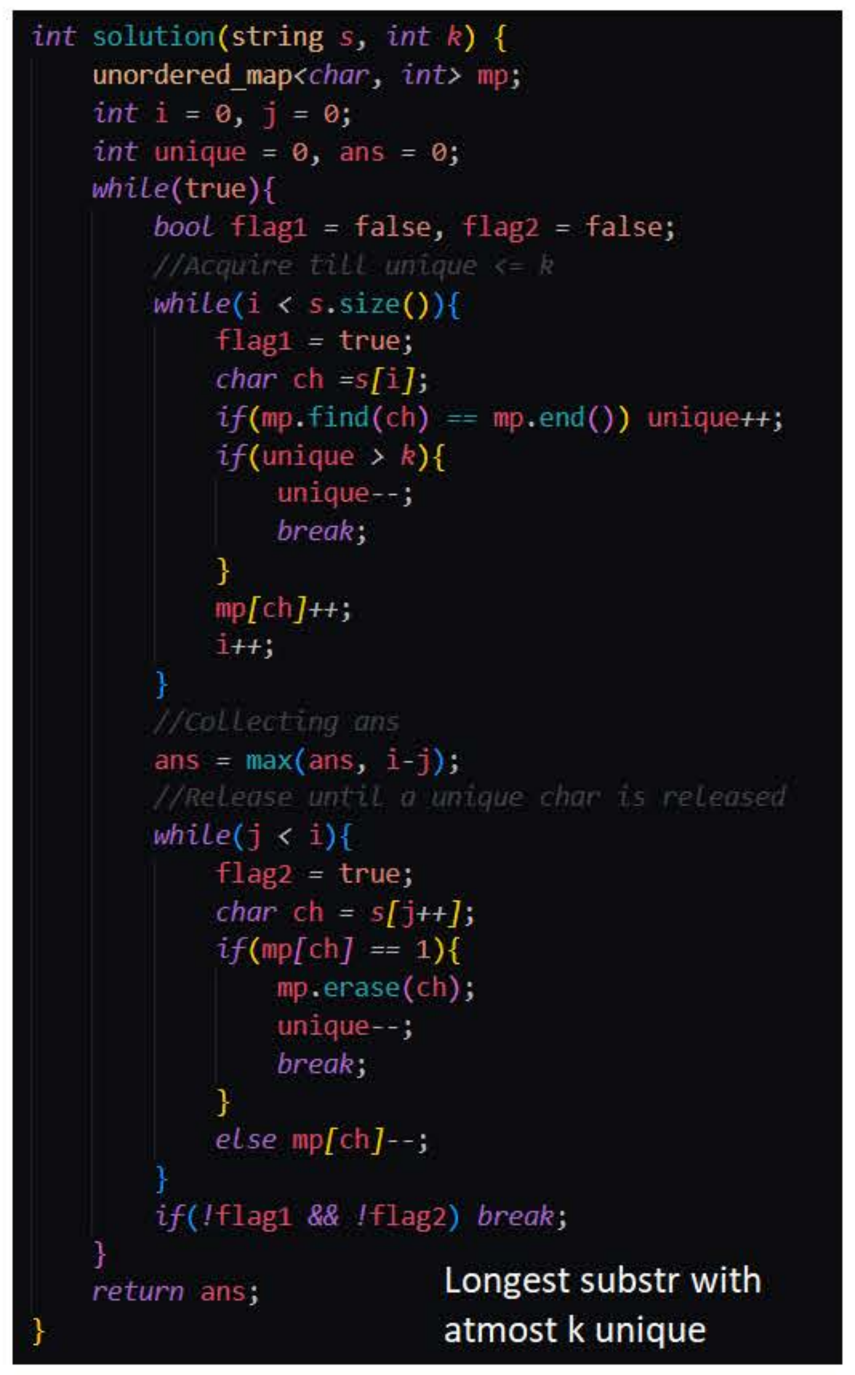
Count of substr with K unique



We create maps here to store substr with k unique and k-1 unique coz as shown above

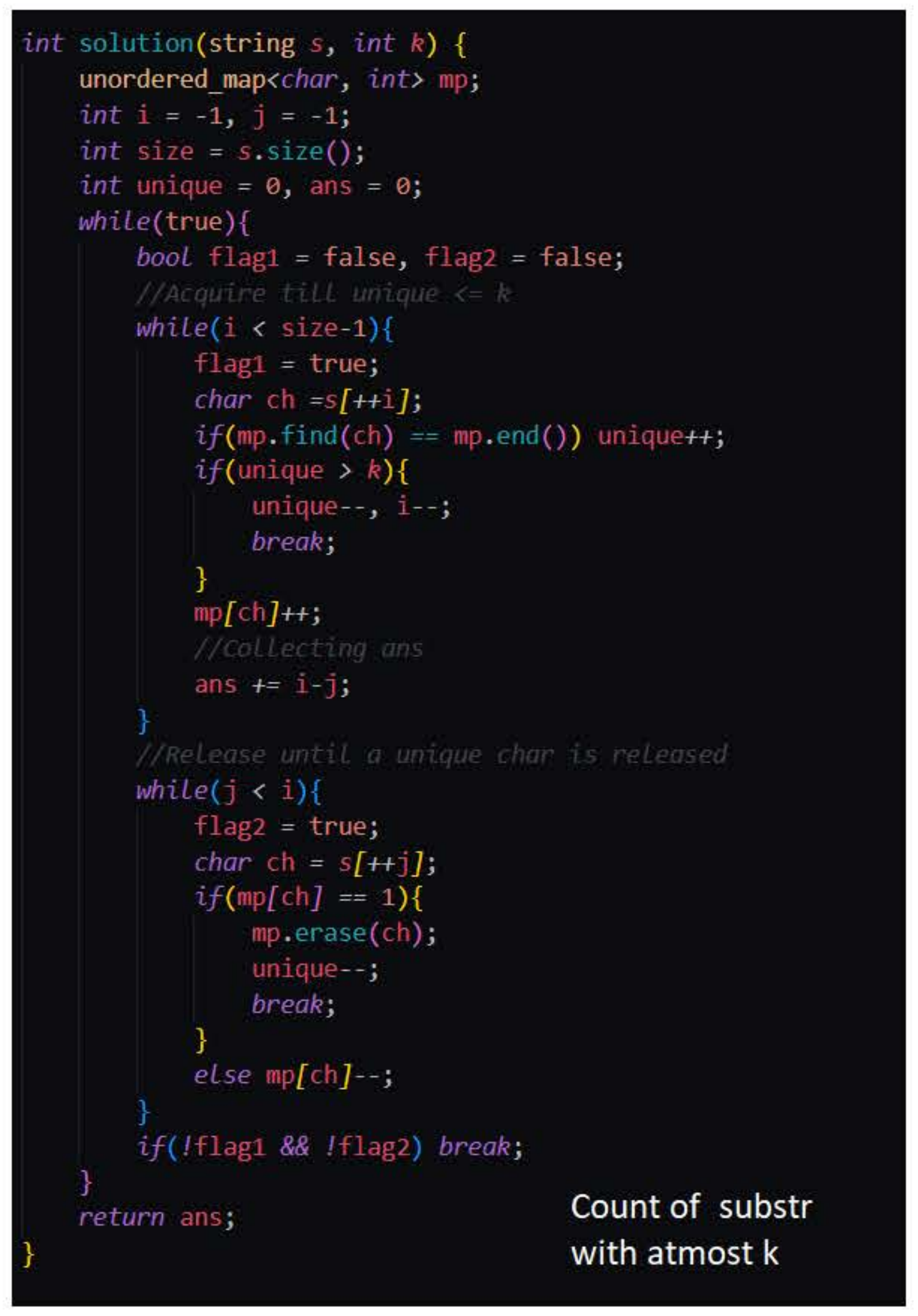
No of Substrs with k unique = difference of len of substrs of k unique and k-q unique

substrs with k unique in above case is
bca
bcab



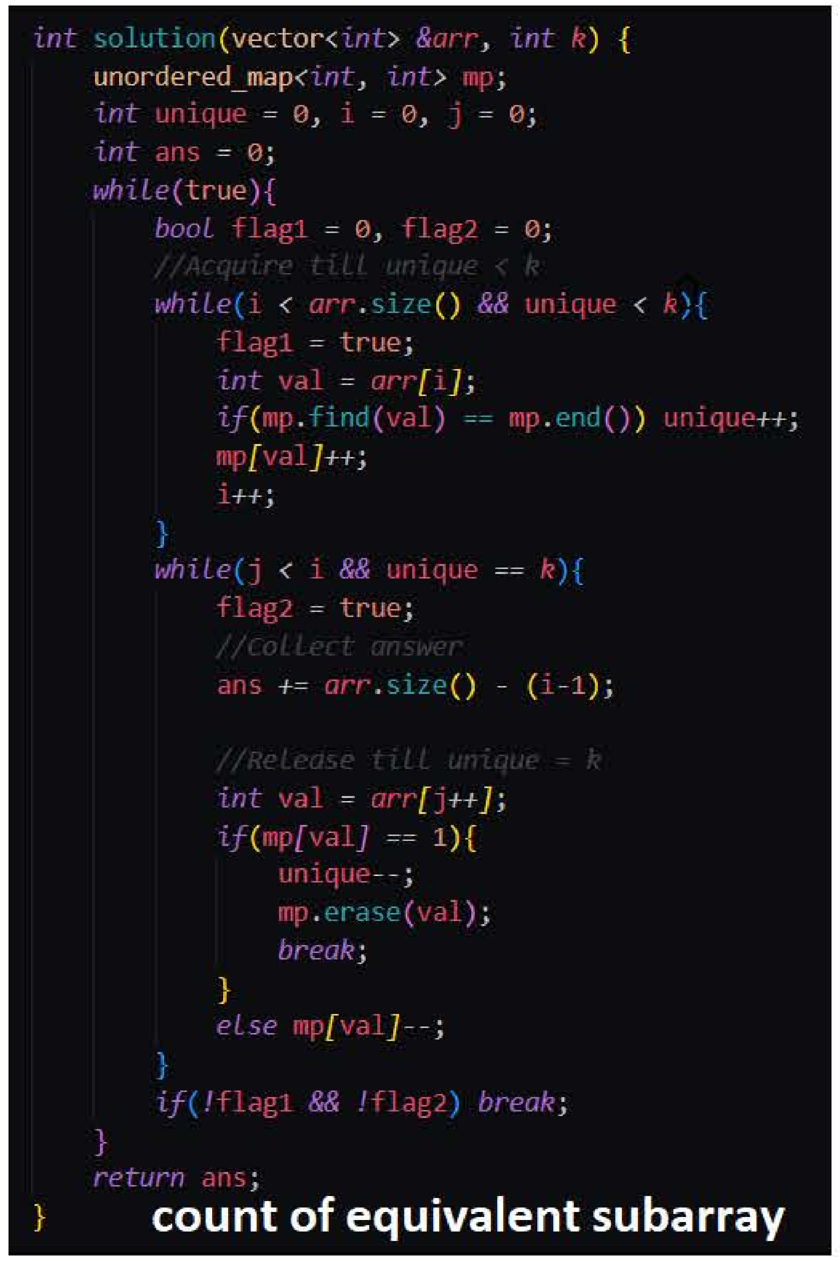
```cpp
int solution(string s, int k) {
    unordered_map<char, int> mp;
    int i = 0, j = 0;
    int unique = 0, ans = 0;
    while(true){
        bool flag1 = false, flag2 = false;
        //Acquire till unique <= k
        while(i < s.size()){
            flag1 = true;
            char ch =s[i];
            if(mp.find(ch) == mp.end()) unique++;
            if(unique > k){
                unique--;
                break;
            }
            mp[ch]++;
            i++;
        }
        //Collecting ans
        ans = max(ans, i-j);
        //Release until a unique char is released
        while(j < i){
            flag2 = true;
            char ch = s[j++];
            if(mp[ch] == 1){
                mp.erase(ch);
                unique--;
                break;
            }
            else mp[ch]--;
        }
        if(!flag1 && !flag2) break;
    }
    return ans;
}
```

Longest substr with atmost k unique



```cpp
int solution(string s, int k) {
    unordered_map<char, int> mp;
    int i = -1, j = -1;
    int size = s.size();
    int unique = 0, ans = 0;
    while(true){
        bool flag1 = false, flag2 = false;
        //Acquire till unique <= k
        while(i < size-1){
            flag1 = true;
            char ch =s[++i];
            if(mp.find(ch) == mp.end()) unique++;
            if(unique > k){
                unique--, i--;
                break;
            }
            mp[ch]++;
            //Collecting ans
            ans += i-j;
        }
        //Release until a unique char is released
        while(j < i){
            flag2 = true;
            char ch = s[++j];
            if(mp[ch] == 1){
                mp.erase(ch);
                unique--;
                break;
            }
            else mp[ch]--;
        }
        if(!flag1 && !flag2) break;
    }
    return ans;
}
```

Count of substr with atmost k



```cpp
int solution(vector<int> &arr, int k) {
    unordered_map<int, int> mp;
    int unique = 0, i = 0, j = 0;
    int ans = 0;
    while(true){
        bool flag1 = 0, flag2 = 0;
        //Acquire till unique < k
        while(i < arr.size() && unique < k){
            flag1 = true;
            int val = arr[i];
            if(mp.find(val) == mp.end()) unique++;
            mp[val]++;
            i++;
        }
        while(j < i && unique == k){
            flag2 = true;
            //Collect answer
            ans += arr.size() - (i-1);

            //Release till unique = k
            int val = arr[j++];
            if(mp[val] == 1){
                unique--;
                mp.erase(val);
                break;
            }
            else mp[val]--;
        }
        if(!flag1 && !flag2) break;
    }
    return ans;
}
```

count of equivalent subarray

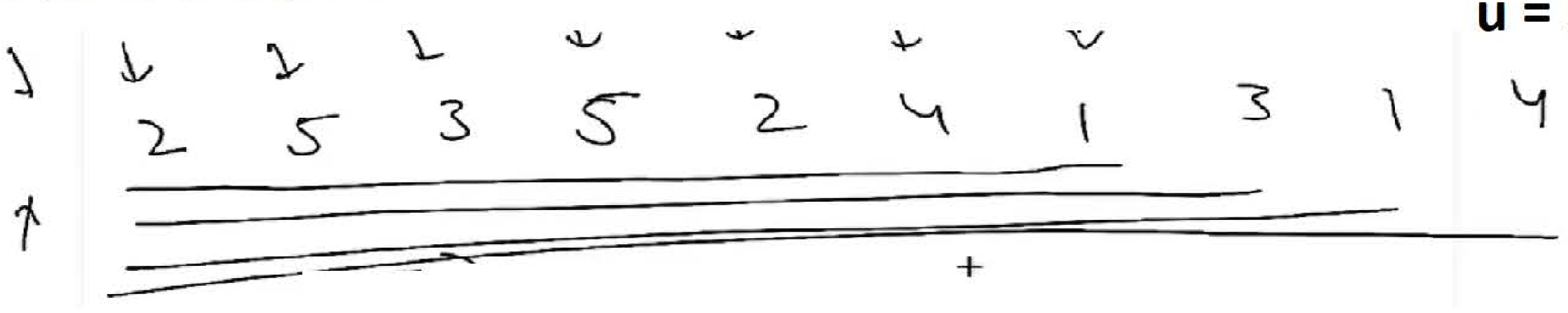
A subarray is equivalent if,

count of unique integers in the subarray = count of unique integers in the given array.

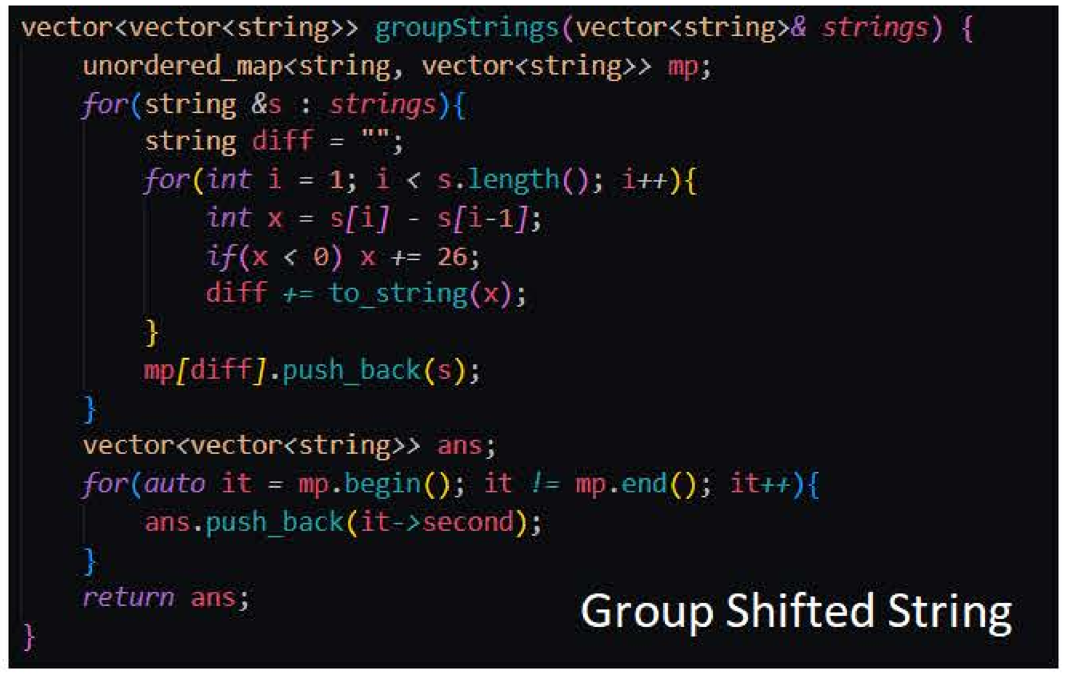
This questions looks same as prev question with k=no of unique elements in arr -> Yes it is but we can solve with a better approach

Here we first count no of unique chars in array (say u)
acquire till all unique = u

u = 5



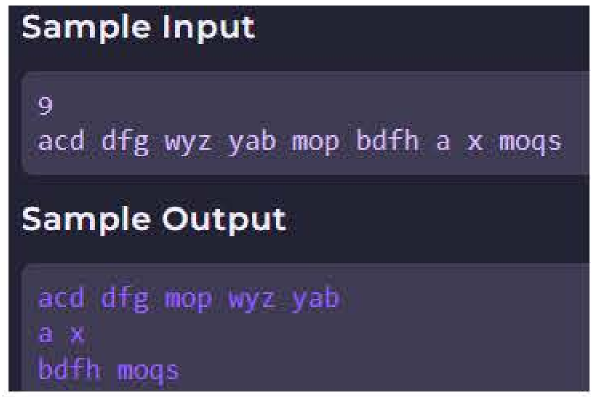
ans += arrSize - sizeofSubstr + 1



```cpp
vector<vector<string>> groupStrings(vector<string>& strings) {
    unordered_map<string, vector<string>> mp;
    for(string &s : strings){
        string diff = "";
        for(int i = 1; i < s.length(); i++){
            int x = s[i] - s[i-1];
            if(x < 0) x += 26;
            diff += to_string(x);
        }
        mp[diff].push_back(s);
    }
    vector<vector<string>> ans;
    for(auto it = mp.begin(); it != mp.end(); it++){
        ans.push_back(it->second);
    }
    return ans;
}
```

Group Shifted String

Two strings s1 and s2 are shifted if -
-> Length of both the strings is the same.
-> The difference between ASCII values of every character of s1 and s2 is constant.



Sample Input

```
9
acd dfg wyz yab mop bdfh a x moqs
```

Sample Output

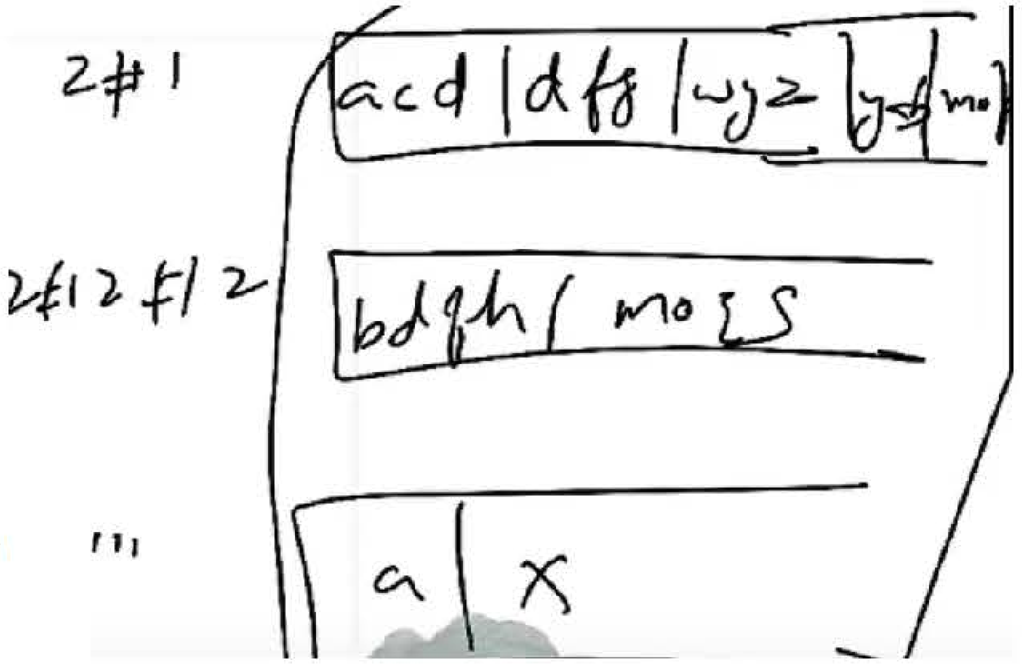
```
acd dfg mop wyz yab
a x
bdfh moqs
```

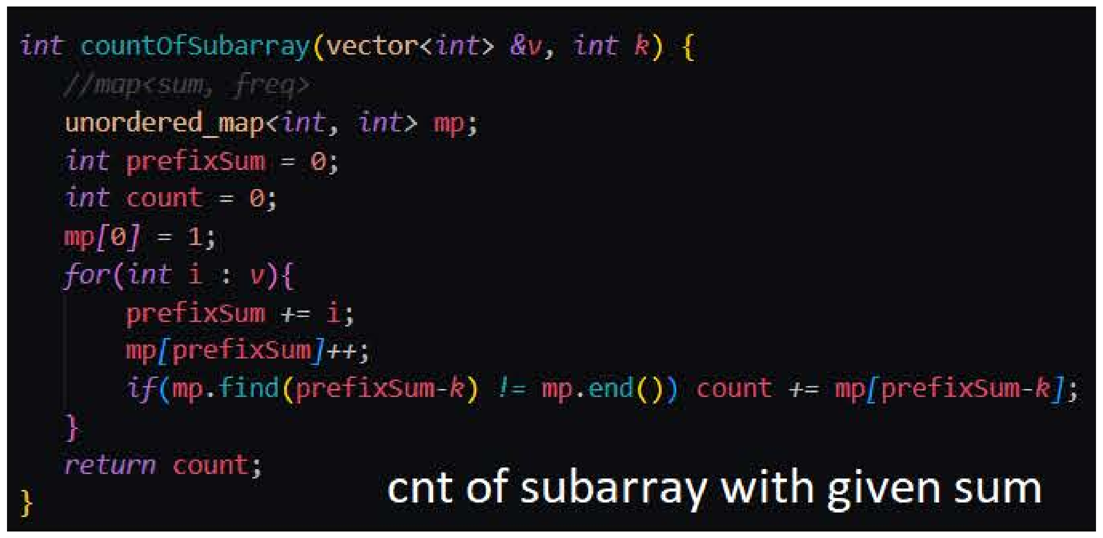
a c d
d f g

diff of(a, d) = diff(c, f) = diff(d, g) = 3

=> diff(a, c) = diff(d, f)
diff(c, d) = diff(f, g)

Hence we hash value of difference btw chars of each string to string

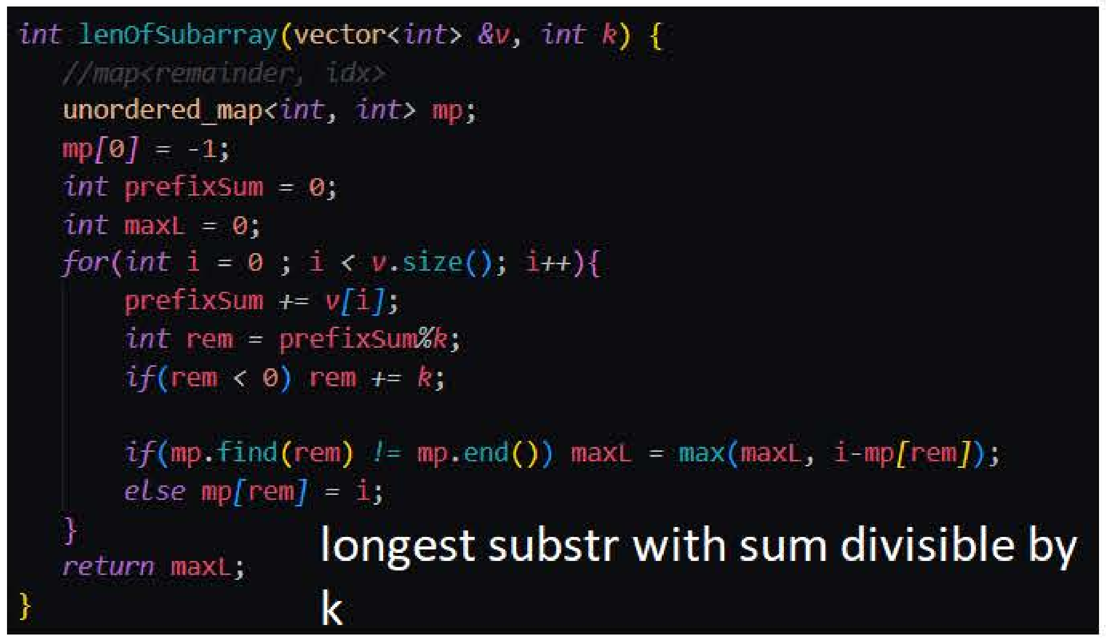




```
int countOfSubarray(vector<int> &v, int k) {
    //map<sum, freq>
    unordered_map<int, int> mp;
    int prefixSum = 0;
    int count = 0;
    mp[0] = 1;
    for(int i : v){
        prefixSum += i;
        mp[prefixSum]++;
        if(mp.find(prefixSum-k) != mp.end()) count += mp[prefixSum-k];
    }
    return count;
}
```

cnt of subarray with given sum

if (curr sum - k) occured before, if yes that means the sum of elements after that element is k



```
int lenOfSubarray(vector<int> &v, int k) {
    //map<remainder, idx>
    unordered_map<int, int> mp;
    mp[0] = -1;
    int prefixSum = 0;
    int maxL = 0;
    for(int i = 0 ; i < v.size(); i++){
        prefixSum += v[i];
        int rem = prefixSum%k;
        if(rem < 0) rem += k;

        if(mp.find(rem) != mp.end()) maxL = max(maxL, i-mp[rem]);
        else mp[rem] = i;
    }
    return maxL;
}
```

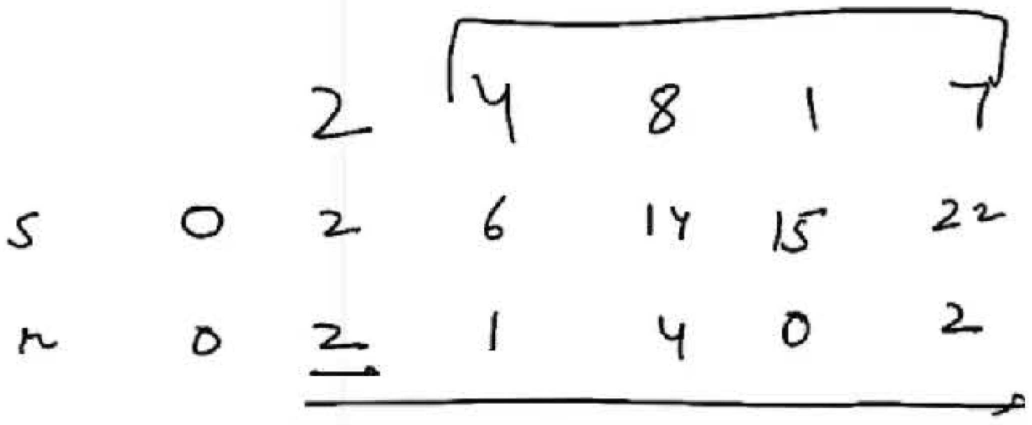
longest substr with sum divisible by k

say sum till idx i is S1 and S1%k = x

say sum till idx j is S2 and S2%k = x

=> S1 = k*n + x

S2 = k*m + x

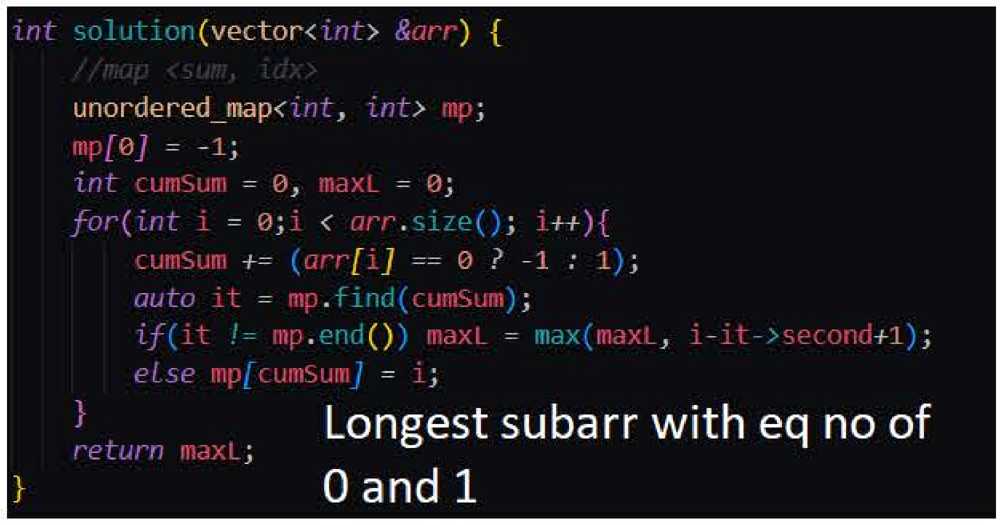


		2	4	8	1	7
S	0	2	6	14	15	22
r	0	2	1	4	0	2

sum of substr (from idx i->j) = S2-S1 = k(m-n) which is a multiple of k => Divisible by K

Therefore if prefixSum % k repeats, then sum pf substr from prev occurance to curr occurance is divisible by k

if rem is negative then we add k. Say rem = -5
then the number can be represented as
kn-5 = kn-5 +k-k
=k(n-1) +k-5
so effective rem is k-5



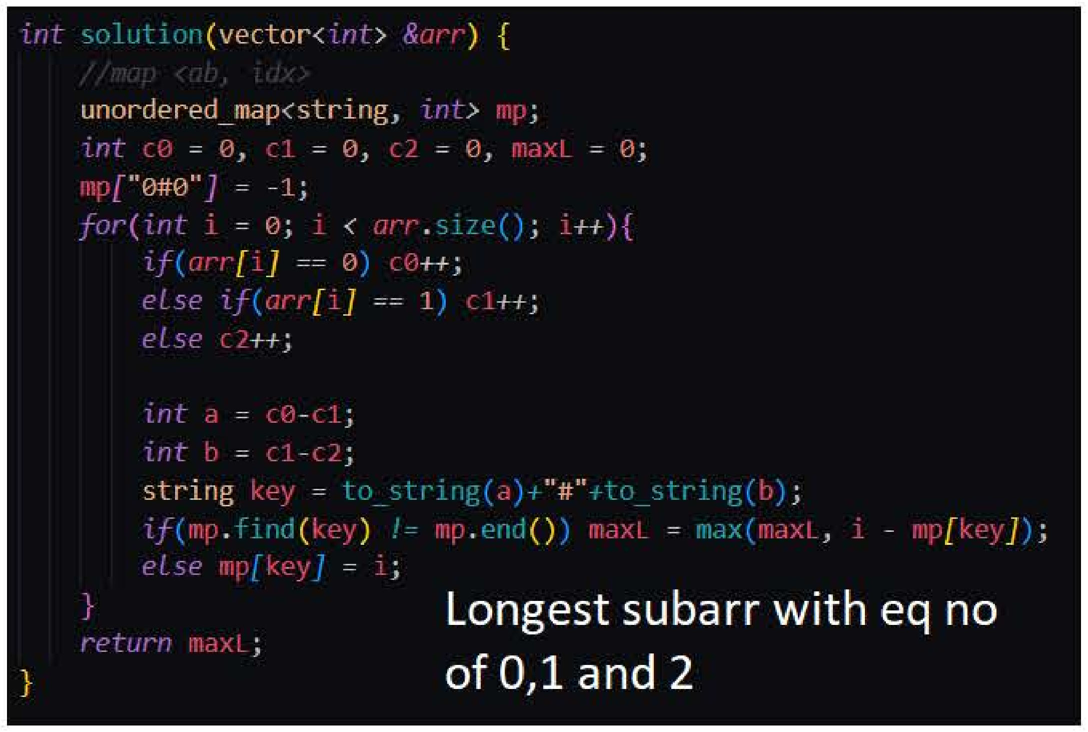
```
int solution(vector<int> &arr) {
    //map <sum, idx>
    unordered_map<int, int> mp;
    mp[0] = -1;
    int cumSum = 0, maxL = 0;
    for(int i = 0;i < arr.size(); i++){
        cumSum += (arr[i] == 0 ? -1 : 1);
        auto it = mp.find(cumSum);
        if(it != mp.end()) maxL = max(maxL, i-it->second+1);
        else mp[cumSum] = i;
    }
    return maxL;
}
```

Longest subarr with eq no of 0 and 1

We treat 0 as -1 and 1 as 1

Now if a subarray contains equal no of 0s and 1s => Sum = 0

Now the problem is reduced to LONGEST SUBARRAY WITH SUM = 0 which we have solved before



```
int solution(vector<int> &arr) {
    //map <ab, idx>
    unordered_map<string, int> mp;
    int c0 = 0, c1 = 0, c2 = 0, maxL = 0;
    mp["0#0"] = -1;
    for(int i = 0; i < arr.size(); i++){
        if(arr[i] == 0) c0++;
        else if(arr[i] == 1) c1++;
        else c2++;

        int a = c0-c1;
        int b = c1-c2;
        string key = to_string(a)+"#"+to_string(b);
        if(mp.find(key) != mp.end()) maxL = max(maxL, i - mp[key]);
        else mp[key] = i;
    }
    return maxL;
}
```

Longest subarr with eq no of 0,1 and 2

say count of 0, 1 and 2 till idx I is x0 x1 and x2

and a = x0-x1, b = x1-x2

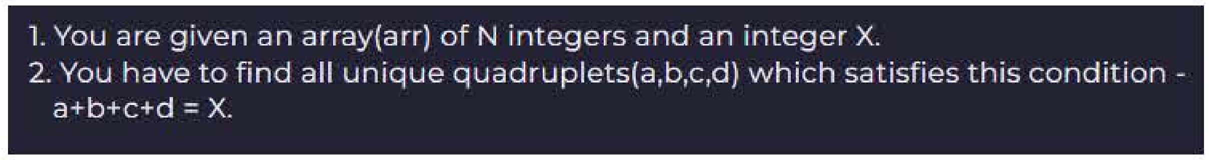
say count of 0, 1 and 2 till idx J is x0' x1'and x2'

and a' = x0'-x1', b' = x1'-x2'

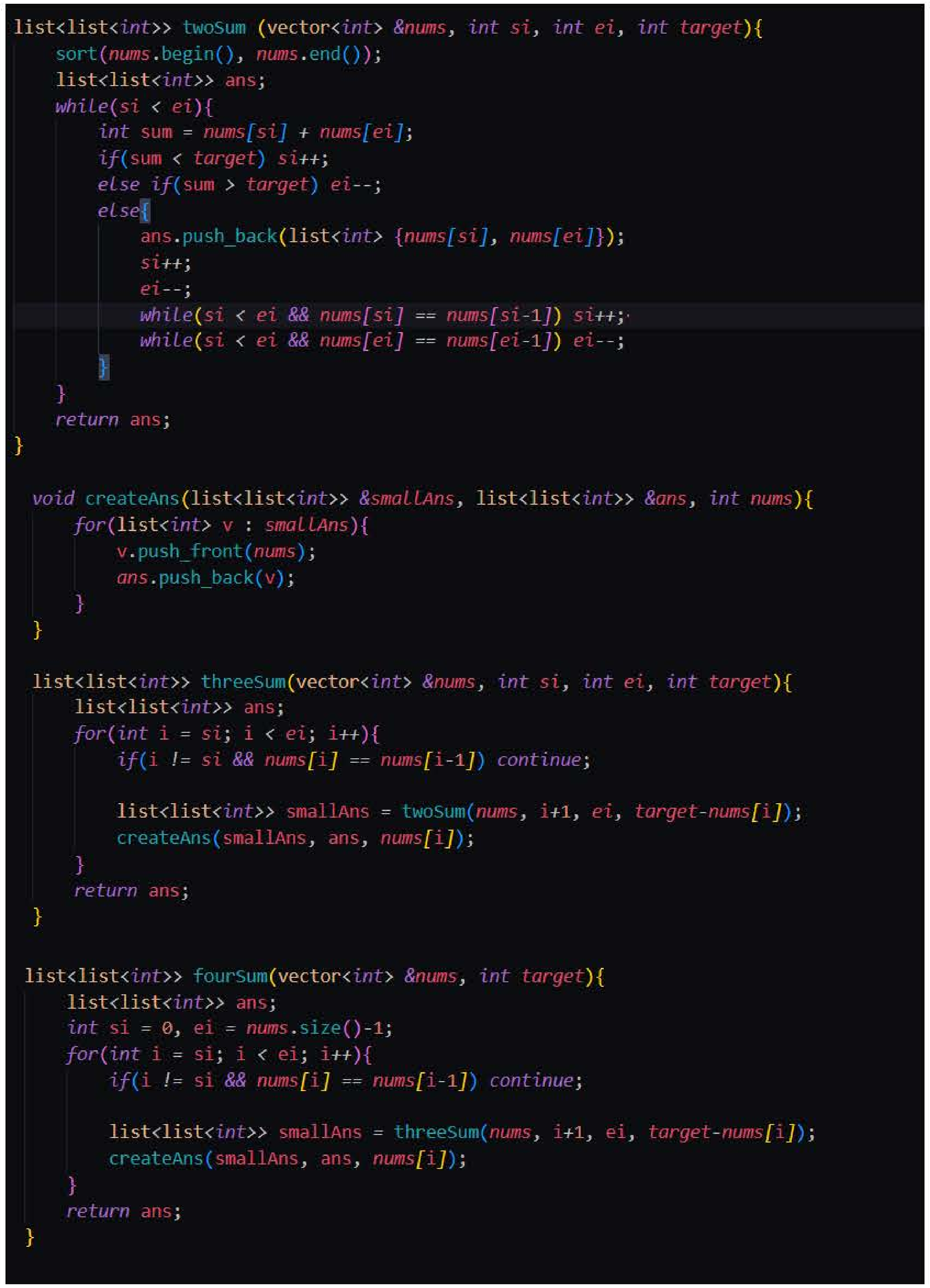
if a = a' and b = b' then that means subarray from idx I to J has equal no of 0 1 and 2's

This equallity implies that No of 0 has increased by same no of times as 1 and as well as 2 which is why the difference between count0-count1 and count1-count2 is same

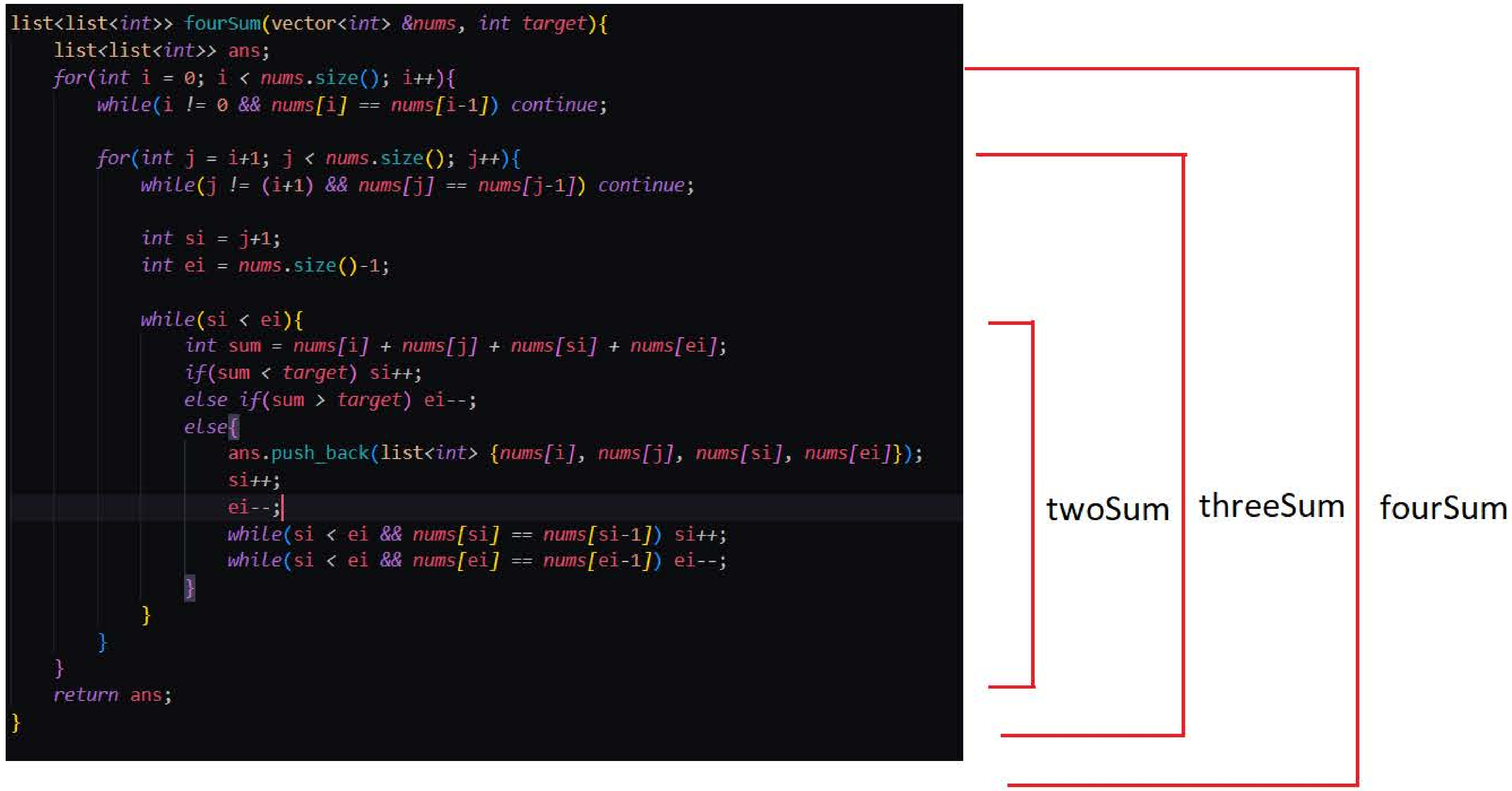
Quadraple Sum



1. You are given an array(arr) of N integers and an integer X.
2. You have to find all unique quadruplets(a,b,c,d) which satisfies this condition -
   a+b+c+d = X.

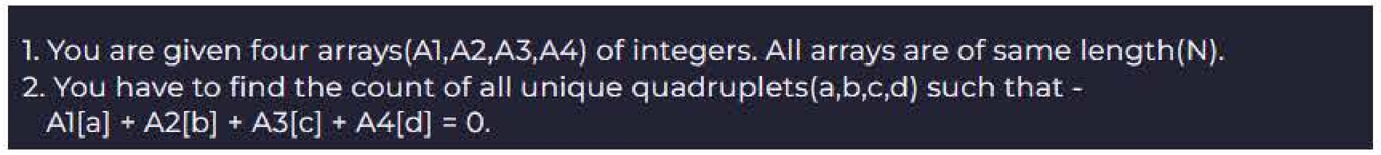


```
list<list<int>> twoSum (vector<int> &nums, int si, int ei, int target){
    sort(nums.begin(), nums.end());
    list<list<int>> ans;
    while(si < ei){
        int sum = nums[si] + nums[ei];
        if(sum < target) si++;
        else if(sum > target) ei--;
        else{
            ans.push_back(list<int> {nums[si], nums[ei]});
            si++;
            ei--;
            while(si < ei && nums[si] == nums[si-1]) si++;
            while(si < ei && nums[ei] == nums[ei-1]) ei--;
        }
    }
    return ans;
}

  void createAns(list<list<int>> &smallAns, list<list<int>> &ans, int nums){
      for(list<int> v : smallAns){
          v.push_front(nums);
          ans.push_back(v);
      }
  }

  list<list<int>> threeSum(vector<int> &nums, int si, int ei, int target){
      list<list<int>> ans;
      for(int i = si; i < ei; i++){
          if(i != si && nums[i] == nums[i-1]) continue;

          list<list<int>> smallAns = twoSum(nums, i+1, ei, target-nums[i]);
          createAns(smallAns, ans, nums[i]);
      }
      return ans;
  }

 list<list<int>> fourSum(vector<int> &nums, int target){
     list<list<int>> ans;
     int si = 0, ei = nums.size()-1;
     for(int i = si; i < ei; i++){
         if(i != si && nums[i] == nums[i-1]) continue;

         list<list<int>> smallAns = threeSum(nums, i+1, ei, target-nums[i]);
         createAns(smallAns, ans, nums[i]);
     }
     return ans;
 }
```

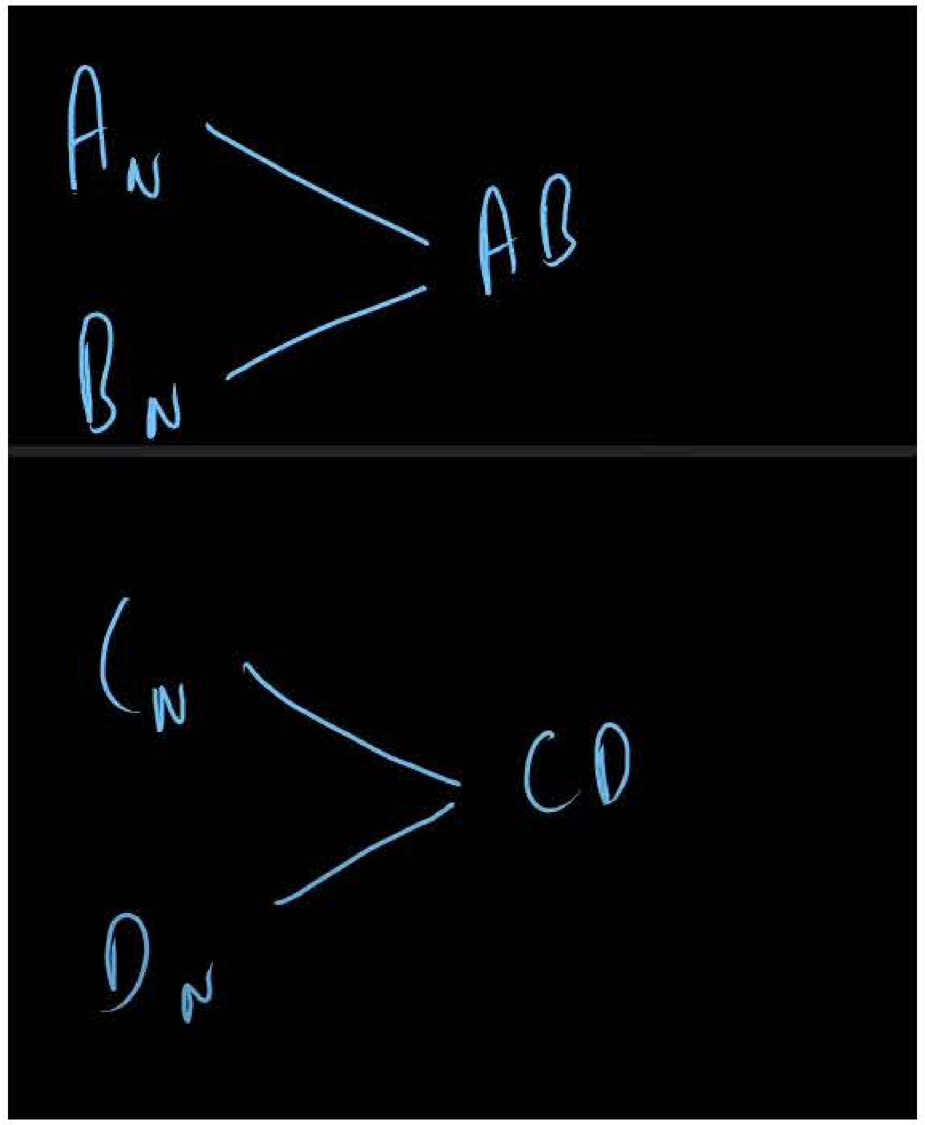


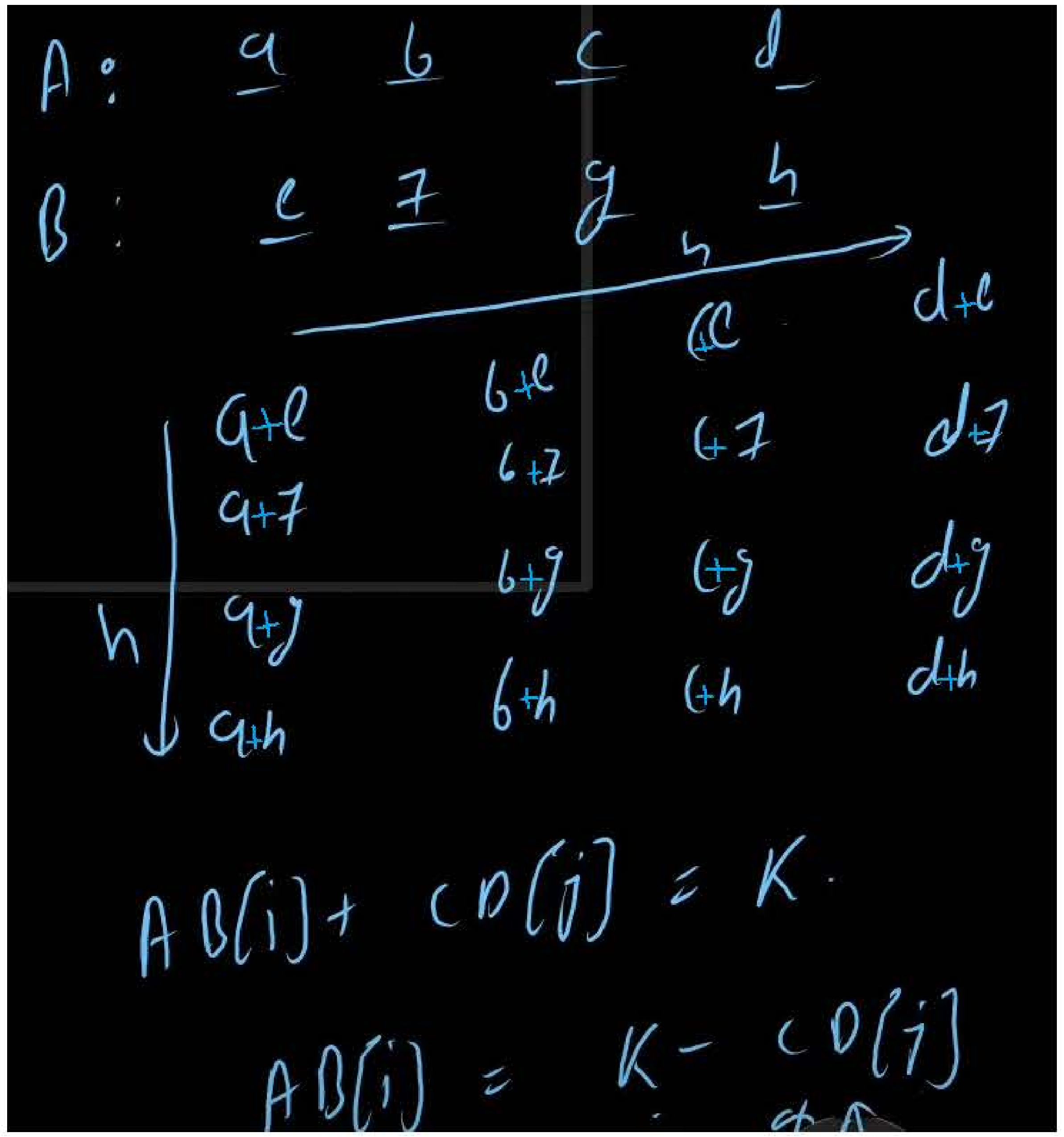
```
list<list<int>> fourSum(vector<int> &nums, int target){
    list<list<int>> ans;
    for(int i = 0; i < nums.size(); i++){
        while(i != 0 && nums[i] == nums[i-1]) continue;

        for(int j = i+1; j < nums.size(); j++){
            while(j != (i+1) && nums[j] == nums[j-1]) continue;

            int si = j+1;
            int ei = nums.size()-1;

            while(si < ei){
                int sum = nums[i] + nums[j] + nums[si] + nums[ei];
                if(sum < target) si++;
                else if(sum > target) ei--;
                else{
                    ans.push_back(list<int> {nums[i], nums[j], nums[si], nums[ei]});
                    si++;
                    ei--;
                    while(si < ei && nums[si] == nums[si-1]) si++;
                    while(si < ei && nums[ei] == nums[ei-1]) ei--;
                }
            }
        }
    }
    return ans;
}
```

Quadraplet Sum



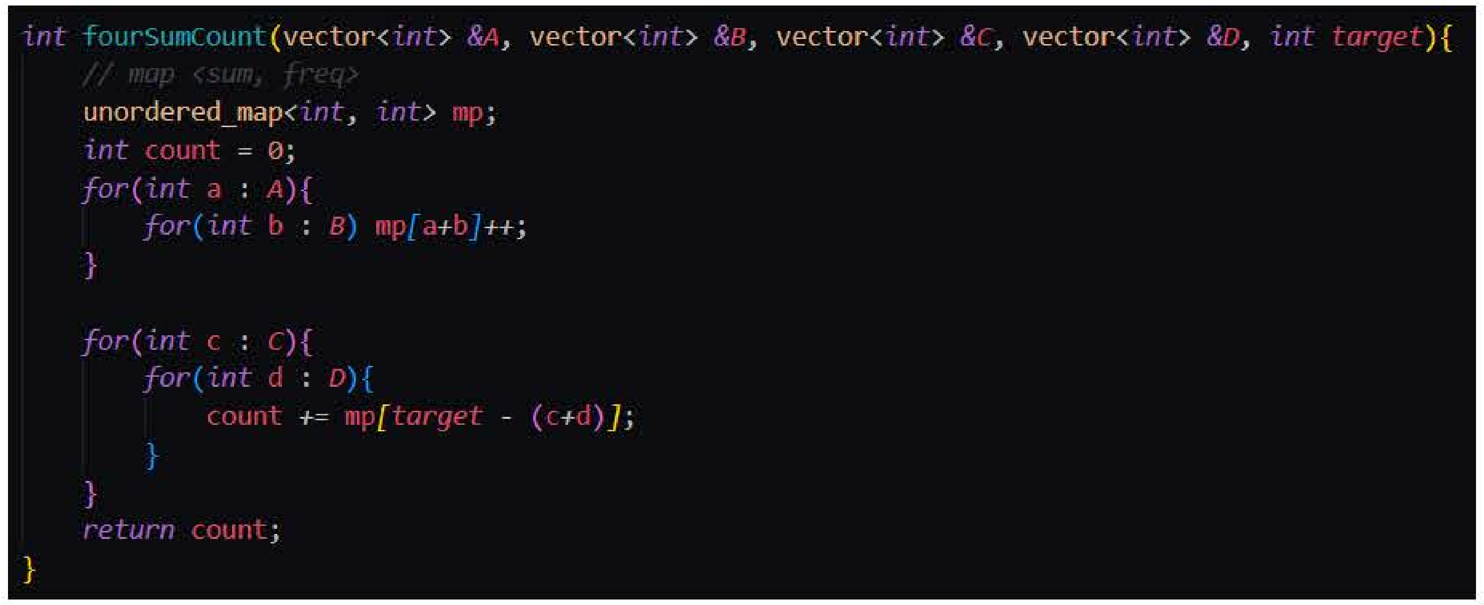
1. You are given four arrays(A1,A2,A3,A4) of integers. All arrays are of same length(N).
2. You have to find the count of all unique quadruplets(a,b,c,d) such that -
   A1[a] + A2[b] + A3[c] + A4[d] = 0.





We will hash sum of values of combintion of all elements of A and B (first two arrays)

While iterating through the combination of C and D (other two arrays) we search for target - (C[i] + D[j]), if found then that is a quadraplet



```
int fourSumCount(vector<int> &A, vector<int> &B, vector<int> &C, vector<int> &D, int target){
    // map <sum, freq>
    unordered_map<int, int> mp;
    int count = 0;
    for(int a : A){
        for(int b : B) mp[a+b]++;
    }

    for(int c : C){
        for(int d : D){
            count += mp[target - (c+d)];
        }
    }
    return count;
}
```