Assignment-50 **A Job Ready Bootcamp in c++,DSA and IOT**

unordered_map

1. Write a c++ program to demonstrate functionality of unordered_map.

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    unordered_map <string, int> um;
    um["Akhtar"] = 100;
    um["Mukesh"] = 600;
    um["Gautam"] = 800;
    um["Tarun"] = 500;

    unordered_map <string, int> :: iterator it;

    for(it = um.begin(); it != um.end(); it++)
        cout << "Key: " << it->first << ", Value: " << it->second << endl;

    return 0;
}
===============================================================================
Output:
Key: Tarun, Value: 500
Key: Gautam, Value: 800
Key: Akhtar, Value: 100
Key: Mukesh, Value: 600
```

2. Given a string, Find the 1st repeated word in a string using unordered_map.
Example:
Input : "Ravi had been saying that he had been there"
Output : had
Input : "Ravi had been saying that"
Output : No Repetition

```
#include<bits/stdc++.h>
using namespace std;
void solve(string s)
{
    unordered_map <string, int> mp; // to store occurrences of word
    string t="",ans="";
     // traversing from back makes sure that we get the word which repeats
first as ans
    for(int i = s.length()-1; i >= 0; i--)
    {
        // if char present , then add that in temp word string t
        if(s[i]!=' ')
        {
            t += s[i];

        }
        // if space is there then this word t needs to stored in map
        else
        {
            mp[t]++;
            // if that string t has occurred previously then it is a possible
ans
            if(mp[t] > 1)
            ans = t;
            // set t as empty for again new word
            t = "";
        }
    }

    // first word like "he" needs to be mapped
            mp[t]++;
            if(mp[t] > 1)
            ans = t;

    if(ans != "")
    {
        // reverse ans string as it has characters in reverse order
        reverse(ans.begin(), ans.end());
        cout << ans << endl;
    }
    else
    cout << "No Repetition\n";
}
int main()
{
    string u="Ravi had been saying that he had been there";
    string v="Ravi had been saying that";
    string w="he had had he";
    solve(u);
    solve(v);
    solve(w);

    return 0;
}
=============================================================================
Output:
had
No Repetition
he
```

3. Write a c++ program to find freq of every word using unordered_map.

```
#include <bits/stdc++.h>
using namespace std;

void printFrequency(string str)
{
    // Define an unordered_map
    unordered_map<char, int> M;

    // Traverse string str check if
    // current character is present
    // or not
    for (int i = 0; str[i]; i++)
    {
        // If the current characters
        // is not found then insert
        // current characters with
        // frequency 1
        if (M.find(str[i]) == M.end())
            M.insert(make_pair(str[i], 1));

        // Else update the frequency
        else
            M[str[i]]++;
    }

    // Traverse the map to print the
    // frequency
    for (auto& it : M)
        cout << it.first << ' ' << it.second << '\n';
}

// Driver Code
int main()
{
    string str = "Shekh_Akhtar";

    // Function call
    printFrequency(str);
    return 0;
}
==============================================================================
Output:
r 1
a 1
h 3
S 1
e 1
k 2
t 1
A 1
_ 1
```

4. Write a c++ program to demonstrate implementation of find function in unordered_map.

```
#include <iostream>
#include <unordered_map>
using namespace std;

// Driver code
int main()
{
// Declaring umap to be of
// <string, int> type key
// will be of STRING type
// and mapped VALUE will
// be of int type
unordered_map<string, int> um;

// inserting values by using [] operator
    um["Akhtar"] = 100;
    um["Mukesh"] = 600;
    um["Gautam"] = 800;
    um["Tarun"] = 500;

// Traversing an unordered map
for (auto x : um)
    cout << x.first << " " <<
            x.second << endl;
string key;
cout << "Enter key to find element: ";
cin >> key;
if(um.find(key) != um.end())
    cout << "Key found" << endl;
else
    cout << "Key not found" << endl;
    return 0;
}
==============================================================================
Output:
Tarun 500
Gautam 800
Akhtar 100
Mukesh 600
Enter key to find element: vijendra
```

```
Key not found
```

5. Given a positive integer N, the task is to print the nearest power of 2 of the frequencies of each digit present in N. If there exists two nearest powers of 2 for any frequency, print the larger one using unordered_map.
   Example:
   Input: N = 344422
   Output:
   2 -> 2
   3 -> 1
   4 -> 4
   Explanation:
   Frequency of the digit 3 is 1. Nearest power of 2 is 1.
   Frequency of the digit 4 is 3. Nearest power of 2 is 4.
   Frequency of the digit 2 is 2. Nearest power of 2 is 2.

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the nearest power of
// 2 for all frequencies in the Map freq
void nearestPowerOfTwoUtil(unordered_map <char, int> & freq)
{
    // Traverse the Map
    for (auto& it : freq)
    {
        cout << it.first << " -> ";

        // Calculate log of the
        // current array element
        int lg = log2(it.second);
        int a = pow(2, lg);
        int b = pow(2, lg + 1);

        // Find the nearest power of 2
        // for the current frequency
        if ((it.second - a) < (b - it.second))
            cout << a << endl;
        else
            cout << b << endl;
    }
}

// Function to find nearest power of 2
// for frequency of each digit of num
void nearestPowerOfTwo(string& S)
{
    // Length of string
    int N = S.size();

    // Stores the frequency of each
    // character in the string
    unordered_map<char, int> freq;

    // Traverse the string S
    for (int i = 0; i < N; i++)
        freq[S[i]]++;

    // Function call to generate
    // nearest power of 2 for each
    // frequency
    nearestPowerOfTwoUtil(freq);
}

// Driver Code
int main()
{
    string N = "344422";
    nearestPowerOfTwo(N);

    return 0;
}
==============================================================================
Output:
2 -> 2
3 -> 1
4 -> 4
```

6. Given two integers L, R, and an integer K, the task is to print all the pairs of Prime Numbers from the given range whose difference is K using unordered_map.
   Example:
   Input: L = 1, R = 19, K = 6
   Output: (5, 11) (7, 13) (11, 17) (13, 19)
   Explanation: The pairs of prime numbers with difference 6 are (5, 11), (7, 13), (11, 17), and (13, 19).

```
#include <bits/stdc++.h>
using namespace std;

// Function to generate prime numbers
// in the given range [L, R]
void findPrimeNos(int L, int R, unordered_map<int, int>& M)
{
    // Store all value in the range
    for (int i = L; i <= R; i++)
        M[i]++;

    // Erase 1 as its non-prime
    if (M.find(1) != M.end())
        M.erase(1);

    // Perform Sieve of Eratosthenes
    for (int i = 2; i <= sqrt(R); i++)
    {
        int multiple = 2;

        while ((i * multiple) <= R)
        {
            // Find current multiple
            if (M.find(i * multiple)
                != M.end())
                {
                    // Erase as it is a
                    // non-prime
                    M.erase(i * multiple);
                }
            // Increment multiple
            multiple++;
        }
    }
}

// Function to print all the prime pairs
// in the given range that differs by K
void getPrimePairs(int L, int R, int K)
```

```
{
    unordered_map<int, int> M;

    // Generate all prime number
    findPrimeNos(L, R, M);

    // Traverse the Map M
    for (auto& it : M)
    {
        // If it.first & (it.first + K)
        // is prime then print this pair
        if (M.find(it.first + K) != M.end())
            cout << "(" << it.first << ", " << it.first + K << ") ";
    }
}

// Driver Code
int main()
{
    // Given range
    int L = 1, R = 19;

    // Given K
    int K = 6;

    // Function Call
    getPrimePairs(L, R, K);

    return 0;
}
==============================================================================
Output:
(5, 11) (7, 13) (11, 17) (13, 19)
```

7. Create an unordered_map and Initialize it from another map using the copy constructor

```
#include <iostream>
#include <unordered_map>
using namespace std;

// Driver code
int main()
{
    // Initialize an unordered_map
    // using default constructor
    unordered_map <string, string> old_map;

    // Adding key-value pairs using
    // subscript([]) and assignment(=)
    // operators
    old_map["Shekh"] = "Akhtar";
    old_map["Gautam"] = "Sharma";
    old_map["Mukesh"] = "Rathore";


    // Create a new_map where contents of
    // the previous map will be copied using
    // copy constructor
    unordered_map <string, string> New_map(old_map);

    // Traverse through the unordered_map
    for(auto x: New_map)
        cout << x.first << " -> " << x.second <<endl;

    return 0;
}
==============================================================================
Output:
Mukesh -> Rathore
Shekh -> Akhtar
Gautam -> Sharma
```

8. Create an unordered_map and Initialize it using assignment and subscript operator

```
#include <unordered_map>
using namespace std;

// Driver code
int main()
{
    // Initialize unordered_map
    // using default constructor
    unordered_map<string,string>New_map;

    // Adding key-value pairs using
    // subscript([]) and assignment(=)
    // operators
    New_map["Shekh"] = "Akhtar";
    New_map["Gautam"] = "Sharma";
    New_map["Mukesh"] = "Rathore";

    // Traverse through the unordered_map
    for(auto x: New_map)
        cout << x.first << "->" << x.second <<endl;

    return 0;
}
==============================================================================
Output:
Mukesh->Rathore
Shekh->Akhtar
Gautam->Sharma#include <iostream>

```

9. Given string str, the task is to find the minimum count of characters that need to be deleted from the string such that the frequency of each character of the string is unique using unordered_map.
   Example:
   Input: str = “ceabaacb”
   Output: 2
   Explanation:
   The frequencies of each distinct character are as follows:
   c —> 2
   e —> 1
   a —> 3
   b —> 2
   Possible ways to make frequency of each character unique by minimum number of moves are:
   ● Removing both occurrences of ‘c’ modifies str to “eabaab”
   ● Removing an occurrence of ‘c’ and ‘e’ modifies str to “abaacb”
   Therefore, the minimum removals required is 2.

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the minimum count of
// characters required to be deleted to make
// frequencies of all characters unique
int minCntCharDeletionsfrequency(string& str, int N)
{
    // Stores frequency of each
    // distinct character of str
    unordered_map<char, int> mp;

    // Store frequency of each distinct
    // character such that the largest
    // frequency is present at the top
    priority_queue<int> pq;

    // Stores minimum count of characters
    // required to be deleted to make
    // frequency of each character unique
    int cntChar = 0;

    // Traverse the string
    for (int i = 0; i < N; i++)
    {
        // Update frequency of str[i]
        mp[str[i]]++;
    }

    // Traverse the map
    for (auto it : mp)
    {
        // Insert current
        // frequency into pq
        pq.push(it.second);
    }

    // Traverse the priority_queue
    while (!pq.empty())
    {

        // Stores topmost
        // element of pq
        int frequent
            = pq.top();

        // Pop the topmost element
        pq.pop();

        // If pq is empty
        if (pq.empty())
        {
            // Return cntChar
            return cntChar;
        }

        // If frequent and topmost
        // element of pq are equal
        if (frequent == pq.top())
        {

            // If frequency of the topmost
            // element is greater than 1
            if (frequent > 1)
            {
                // Insert the decremented
                // value of frequent
                pq.push(frequent - 1);
            }
            // Update cntChar
            cntChar++;
        }
    }
    return cntChar;
}

// Driver Code
int main()
{

    string str = "ceabaacb";

    // Stores length of str
    int N = str.length();
    cout << minCntCharDeletionsfrequency(str, N);
    return 0;
}
==============================================================================
Output:
2
```

10. Given an array arr[] consisting of N integers, the task is to find the maximum element with the minimum frequency using unordered_map.
Example:
Input: arr[] = {2, 2, 5, 50, 1}
Output: 50
Explanation:
The element with minimum frequency is {1, 5, 50}. The maximum element among these element is 50.

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the maximum element
// with the minimum frequency
int maxElementWithMinFreq(int* arr, int N)
{
    // Stores the frequency of array
    // elements
    unordered_map<int, int> mp;

    // Find the frequency and store
    // in the map
    for (int i = 0; i < N; i++)
        mp[arr[i]]++;

    // Initialize minFreq to the maximum
    // value and minValue to the minimum
    int minFreq = INT_MAX;
    int maxValue = INT_MIN;

    // Traverse the map mp
    for (auto x : mp)
    {

        int num = x.first;
        int freq = x.second;

        // If freq < minFreq, then update
        // minFreq to freq and maxValue
        // to the current element
        if (freq < minFreq)
        {
            minFreq = freq;
            maxValue = num;
        }

        // If freq is equal to the minFreq
        // and current element > maxValue
        // then update maxValue to the
        // current element
        else if (freq == minFreq && maxValue < num)
            maxValue = num;
    }

    // Return the resultant maximum value
    return maxValue;
}

// Driver Code
int main()
{
    int arr[] = { 2, 2, 5, 50, 1 };
    int N = sizeof(arr) / sizeof(arr[0]);
    cout << maxElementWithMinFreq(arr, N);

    return 0;
}
=============================================================================
Output:
50
```