Unordered Maps in C++

unordered_map is an associated container data structure that stores elements formed by the combination of a key value and a mapped value. It's similar to a dictionary but with no apparent order of key and value pairs.

Properties

1. **Key-Value Map**: **unordered_map** stores a key-value pair. Where the keys are used to uniquely find the value. The unique "keys" are mapped to the "values", the whole "key-value" pair is treated as one element.

2. **Associative**: An individual element has no indexing instead they are associated with a Key which is used for their look up.

3. **Unordered**: This data structure organizes its elements using a hash table which does not arrange elements in a specific order i.e. There is no order to the element storage.

4. **Unique Keys**: Duplicate Keys are not allowed in **unordered_map**, if a duplicate key is pushed into, then the existing key is updated with a new value.

5. **Look Up**: Unlike its ordered cousin, this data structure is not internally sorted as it uses hash tables to store key-value mapping, organized into buckets depending upon their Hash Values to allow for fast access.

6. **Time-Complexity**: The performance of this data structure depends on Hash Function it implements internally, On an average the cost of search, insert, and delete is $O(1)$.

Note: In **unordered_map**, the average cost of basic operations is $O(1)$ but it can slip to worst case of $O(n)$ for some cases, in such cases it is advised to use **map** to avoid getting TLE. In most cases **unordered_map** are 4 times faster than **map**.

Still in most cases **map** is used.

Implementation

Creation

An **unordered_map** is defined using the following syntax:

```
unordered_map<key_type, value_type> map_name;
```

Allowed key-value type: all integer types (int, long long, unsigned long long), char, floating point types, strings, pointers, vectors, bitset etc.

Example:

```
unordered_map<int,int> ump;
```

Access

The map elements can be created and accessed using the `[]` operator.

Usage

```
// C++ program to demonstrate functionality of 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> umap;

    // Inserting values by using [] operator
    umap["Abhishek Kumar Yadav"] = 4;
    umap["Aditya Kumar Singh"] = 5;
    umap["Aditya Narayan Rai"] = 6;
    umap["Aditya Raj Gupta"] = 7;
    umap["Akshit Raj Singh"] = 9;

    // Traversing an unordered map
    for (auto x : umap)
      cout << x.first << " " <<  x.second << endl;
}
```

Output:

```
Aditya Raj Gupta 7
Aditya Narayan Rai 6
Akshit Raj Singh 9
Aditya Kumar Singh 5
Abhishek Kumar Yadav 4
```

Iterators

The Iterators of **unordered_map** are created using the following syntax:

```
      unordered_map<key_type, value_type>::iterator name;
```

Example:

```
        unordered_map<string, double>::iterator itr;
```

Using Iterators

- **(it).first** – key
- **(it).second** – value

In For Loop

```
for (auto x : umap)
    cout << x.first << " " << x.second;
```

Important Functions

Methods/Functions	**Description**
at()	This function in C++ **unordered_map** returns the reference to the value with the element as key k
Begin()	Returns an iterator pointing to the first element in the container in the **unordered_map** container
end()	Returns an iterator pointing to the position past the last element in the container in the **unordered_map** container
bucket()	Returns the bucket number where the element with the key k is located in the map
bucket_count()	bucket_count() is used to count the total number of buckets in the **unordered_map**. No parameter is required to pass into this function
bucket_size()	Returns the number of elements in each bucket of the **unordered_map**
count()	Count the number of elements present in an **unordered_map** with a given key
equal_range()	Return the bounds of a range that includes all the elements in the container with a key that compares equal to k
find()	Returns iterator to the element
empty()	Checks whether the container is empty in the **unordered_map** container
erase()	Erase elements in the container in the **unordered_map** container

An Example:

```
// C++ program to demonstrate initialization, indexing, and iteration
#include <iostream>
#include <unordered_map>
using namespace std;

// Driver code
int main()
{
    // Declaring umap to be of <string, double> type key will be of string type and mapped val-
ue will be of double type
    unordered_map<string, double> umap = {
        //inserting element directly in map
        {"One", 1},
        {"Two", 2},
        {"Three", 3}
    };
    // inserting values by using [] operator
    umap["PI"] = 3.14;
    umap["root2"] = 1.414;
    umap["root3"] = 1.732;
    umap["log10"] = 2.302;
    umap["loge"] = 1.0;

    // inserting value by insert function
    umap.insert(make_pair("e", 2.718));

    string key = "PI";

  // If key not found in map iterator to end is returned
    if (umap.find(key) == umap.end())
        cout << key << " not found"<<endl;

    // If key found then iterator to that
    // key is returned
    else
        cout << "Found " << key << endl;

    key = "lambda";
    if (umap.find(key) == umap.end())
        cout << key << " not found"<<endl;
    else
        cout << "Found " << key << endl;

    // iterating over all value of umap
    unordered_map<string, double>::iterator itr;
    cout << "\nAll Elements : \n";
    for (itr = umap.begin();itr != umap.end(); itr++)
    {
      // itr works as a pointer to pair<string, double> type itr->first stores the key part and
itr->second stores the value part
      cout << itr->first << "  " << itr->second << endl;
    }
}
```

Output:

```
sqlCopy code
CFound PI
lambda not found
All Elements :
e  2.718
loge  1
log10  2.302
Two  2
One  1
Three  3
PI  3.14
root2  1.414
root3  1.732
```

Must Reads

1. Difference Between Ordered Maps and unordered Maps
2. Fast Use of unordered map in CP
3. Pairs in unordered maps
4. Allowed Hash Functions

5. Codeforces Blog

Problems to Practice:

1. **Still Finding Good Questions to put here.**