map vs unordered_map | When to choose one over another ?

Varun March 26, 2017 std::map, unordered_map 1 Comment

In this article we will compare std::map and std::unordered_map and will also discuss when to choose one over another.

Both std::map & std::unordered_map store elements in key value pair & provide member functions to efficiently insert, search & delete key value pairs.

But they are different in following areas,

- Internal Implementation
- Memory Usage
- Time Complexity
- Using user defined objects as keys

Internal Implementation :

std::map Internally store elements in a balanced BST. Therefore, elements will be stored in sorted order of keys.

std::unordered_map store elements using hash table. Therefore, elements will not be stored in any sorted order. They will be stored in arbitrary order . To know more about hashing check following article, What is Hashing and Hash Table?

Check following example,

C++

```cpp
#include <map>
#include <unordered_map>
#include <string>
#include <iterator>
#include <algorithm>

int main()
{

    // Map of string & int i.e. words as key & there
    // occurrence count as values
    std::map<std::string, int> wordMap = {
                        { "is", 6 },
                        { "the", 5 },
                        { "hat", 9 },
                        { "at", 6 }
                    };

    std::cout<<"std::map Contents : Elements are in sorted order of Keys"<<std::e

    // Print map contents
    std::for_each(wordMap.begin(), wordMap.end(), [](std::pair<std::string, int>
        std::cout<<elem.first<< " :: "<< elem.second<<std::endl;
    });



    // Unordered_map of string & int i.e. words as key & there
    // occurrence count as values
    std::unordered_map<std::string, int> wordUOMap = {
                        { "is", 6 },
                        { "the", 5 },
                        { "hat", 9 },
                        { "at", 6 }
                    };

    std::cout<<"std::unordered_map Contents : Elements are in Random order of Key

    // Print Unordered_map
    std::for_each(wordUOMap.begin(), wordUOMap.end(), [](std::pair<std::string, i
                                std::cout<<elem.first<< " :: "<< elem.second<<std
                            });
    return 0;
}
```

Output:

```
std::map Contents : Elements are in sorted order of Keys
at :: 6
hat :: 9
is :: 6
the :: 5
std::unordered_map Contents : Elements are in Random order of Keys
hat :: 9
the :: 5
at :: 6
is :: 6
```

Memory Usage :

Memory usage is more in unordered_map as compared to map because unordered_map need space for storing hash table too.

Time Complexity for Searching element :

Time complexity for searching elements in **std::map** is O(log n). Even in worst case it will be O(log n) because elements are stored internally as Balanced Binary Search tree (BST).

1.) What's std::vector and why to use it?
2.) Different ways to initialize a vector
3.) How does std::vector works internally
4.) User Defined Objects & std::vector
5.) How to use vector efficiently in C++?
6.) std::vector and Iterator Invalidation
7.) Remove repeated elements from a vector
8.) Fill a vector with random numbers
9.) Hidden cost of std::vector
10.) Adding elements in Vector

Popular Categories

Behavioral Design Patterns (6) Boost Date Time Library (6) Boost Library (17) C++ (107) C++ 11 (59) c++11 Threads (16) C++ Interview Questions (30) collections (17) Dataframe (10) Data Science (10) Debugging (6) Debugging Tutorial (6) Design Patterns (9) dictionary (17) Directories (13) FileHandling (27) Functions (10) gdb (6) gdb commands (6) gdb Tutorial (6) HashSet (8) java (29) linkedlist (5) Linux (8) Linux System Programming (8) List (27) Method Overriding (5) Multithreading (12) Numpy (24) Pandas (44) Python (182) Smart Pointers (6) std::list (12) std::map (16) std::set (10) std::string (12) std::thread (5) std::vector (18) STL (56) STL Algorithm (8) STL Interview Questions (16) strings (14) Uncategorized (11) unordered_map (8) unordered_set (6)

Search

Search … Search

Subscribe with us

Subscribe For latest Tutorials

* indicates required

Email Address *

Whereas, in **std::unordered_map** best case time complexity for searching is O(1). Where as, if hash code function is not good then, worst case complexity can be O(n) (In case all keys are in same bucket).

Subscribe

Using user define objects as Keys :

For std::map to use user defined object as keys, we need to override either < operator or pass external comparator i.e. a functor or function pointer that can be used by map for comparing keys.

Where as, For **std::unordered_map** we need to provide definition of function std::hash<K> for our key type K. Also we need to override == operator.

Check following article for detailed explanation :

Using User defined class objects as keys in std::map

Unordered container and Custom Hasher

When to choose map instead of unordered_map

When you need Low Memory:

Unordered_map consumes extra memory for internal hashing, so if you are keeping millions and billions of data inside the map and want to consume less memory then choose std::map instead of std::unordered_map.

When you are interested in Ordering too

As std::map internally use balanced BST, so all the elements inside it will be in sorted order based on the key. So, if you want all keys to be ordered then go for std::map.

When you need guaranted Performance

For searching an element, std::unordered_map gives the complexity O(1) in best case but O(n) in worst case (if hash implementation is not perfect).

So, if your hash implementation is not good and you have millions and billions of data then go for std::map because it will give you guaranteed O(log N).

When to choose unordered_map instead of map

When you have good hasher and no memory limitation

Unordered_map consumes extra memory for internal hashing. But to due to this searching complexity is O(1), if hasher function is good.

Python Recommendations:

- 5 Free Python Courses in 2020 | Best of the lot

- Reviews of Best Python Courses in 2020

C++ & C++11 Recommendations:

- Best C++ Courses for beginners to learn from scratch
- 5 Top Courses to learn C++11 / C+14
- Best books to learn C++11

If you didn't find what you were looking, then do suggest us in the comments below. We will be more than happy to add that.

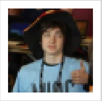
Subscribe with us to join 1500+ Python & C++ developers, to get more Tips & Tutorials like this.

Related Posts:

- Python : How to Sort a Dictionary by key or Value ?
- What is a dictionary in python and why do we need it?
- How to copy all Values from a Map to a Vector in C++
- Python : Filter a dictionary by conditions on keys or values
- C++ : How to find duplicates in a vector ?
- Python : 3 ways to check if there are duplicates in a List
- C++11 : std::array Tutorial and examples
- Python : Check if all elements in a List are same or matches a condition
- Python: Find duplicates in a list with frequency count & index positions
- Pandas : Check if a value exists in a DataFrame using in & not in operator | isin()
- C++ : How to check if a Set contains an element | set::find vs set::count?
- C++ : How to insert element in vector at specific position | vector::insert() examples
- C++ : How to find an element in vector and get its index ?
- C++11 : How to get a Thread ID ?
- Python : How to Remove Duplicates from a List
- C++ : How to compare two vectors | std::equal() & Comparators
- Python : How to convert a list to dictionary ?
- Python : Count elements in a list that satisfy certain conditions
- Python : How to Remove multiple keys from Dictionary while Iterating ?
- Python : How to iterate over the characters in string ?
- C++ : How to get element by index in vector | at() vs operator []
- C++ : How to reverse a List or sub-list in place?
- Java: How to get all keys by a value in HashMap ? | Search by Value in Map
- Python : Get number of elements in a list, lists of lists or nested list

- C++ : How to copy / clone a STL List or Sub List

1 Comment Already



Leonard Inkret - August 7th, 2019 at 12:35 pm

Thank you, this article was very useful

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.

« C++11 – Variadic Template Function | Tutorial & Examples

Different ways to iterate over a set in C++ »

Python tutorials

Strings
Lists
Tuple
Dictionary
Functions
File Handling
Lambda Functions

C++ / C++11 Tutorials

C++ Tutorials
C++11 Tutorial
C++ Interview Questions
Multithreading
C++17
STL Tutorials

Data Analysis in Python

Dataframe in Pandas
Numpy Arrays

Terms of Use

Terms of use

Terms and Policies

Disclaimer
Cookie policy (EU)
Cookie policy (UK)
Privacy statement (EU)
Privacy statement (UK)
Privacy statement (US)
Do Not Sell My Personal Information