

# LECTURE 15

## MAP – TREEMAP - DICT



Phạm Nguyễn Sơn Tùng

Email: [sontungtn@gmail.com](mailto:sontungtn@gmail.com)

# Định nghĩa Map

**Map** là một associative container chứa **danh sách** các phần tử mà mỗi phần tử là một cặp khóa-giá trị (key-value), 2 giá trị này có thể có kiểu dữ liệu khác nhau. Mỗi phần tử là 1 cấu trúc pair.

**C++:** Map

**Java:** TreeMap

**Python:** Dict (một cấu trúc tương tự, dùng Hash Table để quản lý).

# Khai báo và sử dụng (1)



Thư viện:

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

Khai báo: khai báo dạng mặc định.

```
map<data_type1, data_type2>
variable_name;
```

Ví dụ:

```
map<int, string> m;
```



Thư viện:

```
import java.util.TreeMap;
```

Khai báo: khai báo dạng mặc định.

```
TreeMap variable_name = new
TreeMap<object_type1, object_type2>();
```

Ví dụ:

```
TreeMap m = new
TreeMap<Integer, String>();
```



# Khai báo và sử dụng (2)



**Khai báo:** khởi tạo xong sau đó gán giá trị.

```
map<data_type1, data_type2>
```

```
variable_name;
```

```
variable_name[key] = value;
```

Ví dụ:

```
map<int, string> m;
```

```
m[10] = "abc";
```

```
m[20] = "def";
```

```
m[10] = "mpk";
```



**Khai báo:** khởi tạo xong sau đó gán giá trị.

```
variable_name.put(object1, object2);
```

Ví dụ:

```
TreeMap<Integer, String> m  
= new TreeMap<Integer, String>  
();
```

```
m.put(10, "abc");
```

```
m.put(20, "def");
```

```
m.put(10, "mpk");
```

10, "mpk"

20, "def"

# Sao chép map này sang map kia



Ví dụ:

```
map<int, string> m;  
map<int, string> m1(m);
```

Hoặc:

```
map<int, string>  
m1(m.begin(), m.end());
```



Ví dụ:

```
TreeMap<Integer, Integer> m  
= new TreeMap<Integer, Integer>();  
  
TreeMap<Integer, Integer> m1  
= new TreeMap<Integer, Integer>(m);
```

m	10, "mpk"	20, "def"
---	-----------	-----------

m1	10, "mpk"	20, "def"
----	-----------	-----------

# Thêm một phần tử

10, "mpk"

20, "def"



insert:

```
pair<int, string> p(14, "abc");
//Hoặc
p.insert(pair<int, string>(14, "abc"));
m.insert(p);
```



put:

```
m.put(14, "abc");
```

**Kết quả**

10, "mpk"

14, "abc"

20, "def"

# Xóa phần tử

10, "mpk"

14, "abc"

20, "def"



**erase**: Xóa giá trị trong **map** dựa vào khóa.

**erase(key)**: xóa mỗi lần 1 phần tử.

```
m.erase(14);
```

Hoặc:

**erase(iterator, iterator)**: dựa vào iterator xóa một loạt phần tử.



**erase**: Xóa giá trị trong **TreeMap** dựa vào khóa. Trả về object value của khóa tương ứng, nếu khóa không tồn tại thì trả về **null**

```
System.out.println(m.remove(14));
```

In ra: **abc**

**Kết quả**

10, "mpk"

20, "def"



# Tìm phần tử

10, "mpk"

14, "abc"

20, "def"



**find:** Tìm và trả về iterator của đối tượng cần tìm.

```
map<int, string>::iterator it;
it = m.find(20);
if (it == m.end())
    cout << "not found";
else
    cout << it->second;
```



**get:** Tìm và trả về value của đối tượng cần tìm, nếu không tìm thấy trả về **null**.

```
String value = m.get(20);
if (value == null)
    cout << "not found";
else
    cout << value;
```

**Kết quả**

**def**



# Các hàm thành viên của Map C++

- **size**: Trả về số lượng phần tử hiện tại có trong map.
- **empty**: Kiểm tra map có rỗng hay không.
- **clear**: Xóa hết tất cả các phần tử trong map.
- **swap**: Hoán đổi 2 map với nhau.
- **lower\_bound**: Trả về phần tử **đầu tiên không bé hơn** giá trị khóa tìm kiếm. ( $\geq$ )
- **upper\_bound**: Trả về phần tử **đầu tiên lớn hơn** giá trị khóa tìm kiếm. ( $>$ )

# Các hàm thành viên của TreeMap Java

- **size**: Trả về số lượng phần tử hiện tại trong TreeMap.
- **isEmpty**: Kiểm tra TreeMap có rỗng hay không.
- **clear**: Xóa hết tất cả các phần tử trong TreeMap.
- **floorEntry/floorKey**: Trả về entry/key lớn nhất và không lớn hơn khóa tìm kiếm. ( $\leq$ )
- **ceilingEntry/ceilingKey**: Trả về entry/key nhỏ nhất và lớn hơn hoặc bằng khóa tìm kiếm. ( $\geq$ )
- **lowerEntry/lowerKey**: Trả về entry/key lớn nhất và nhỏ hơn khóa tìm kiếm. ( $<$ )
- **higherEntry/higherKey**: Trả về entry/key nhỏ nhất và lớn hơn khóa tìm kiếm. ( $>$ )

# Các lưu ý khi sử dụng (1)

10, "mpk"	14, "abc"	20, "def"	25, "po"
-----------	-----------	-----------	----------



**KHÔNG** thể truy cập vào thành phần **map**.

```
m[2] = make_pair(14, "kmp");  
pair<int, string> p = m[2];  
cout << p.first;
```

**KHÔNG** có toán tử[] để sử dụng.

```
String value = m[20];
```

# Các lưu ý khi sử dụng Map (2)

10, "mpk"

14, "abc"

20, "def"

25, "po"



**CÓ THỂ** truy cập vào **key** để lấy giá trị của **value**. Nếu đầu vào là một key không có trong map thì nó sẽ tự tạo ra phần tử mới.

```
string s = m[14];  
cout << s;
```



**CÓ THỂ** dùng hàm **get** đã hướng dẫn ở phía trên để lấy dữ liệu.

```
String s = m.get(14);  
cout << s;
```



**Kết quả**

**abc**

# Các lưu ý khi sử dụng Map (3)



**Duyệt:** Duyệt bằng cách sử dụng **iterator** (duyet xuôi).

10, "mpk"	14, "abc"	20, "def"	25, "po"
-----------	-----------	-----------	----------

```
map<int, string>::iterator it;  
for (it=m.begin(); it!=m.end(); it++)  
{  
    cout<<it->first<<" "<<it->second<<" , ";  
}
```

**Kết quả**

**10 mpk, 14 abc, 20 def, 25 po**

# Các lưu ý khi sử dụng TreeMap (3)



**Duyệt:** Duyệt bằng cách sử dụng **iterator** (duyet xuôi).

10, "mpk"	14, "abc"	20, "def"	25, "po"
-----------	-----------	-----------	----------

```
for (Map.Entry<Integer, String> kvp : m.entrySet()) {  
    System.out.print(kvp.getKey() + " " + kvp.getValue() + ", " );  
}
```

**Kết quả**

**10 mpk, 14 abc, 20 def, 25 po**





# Các lưu ý khi sử dụng Map (4)

**Duyệt:** Duyệt bằng cách sử dụng **iterator** (duyet ngược).

10, "mpk"	14, "abc"	20, "def"	25, "po"
-----------	-----------	-----------	----------

```
map<int, string>::reverse_iterator it;  
for (it=m.rbegin(); it!=m.rend(); it++)  
{  
    cout << it->first << " " << it->second << ", ";  
}
```

**Kết quả**

**25 po, 20 def, 14 abc, 10 mpk**

*\*Java: Chưa có, nếu cần thì tạo ra cấu trúc mới và duyệt ngược → tốn chi phí.*



# Cấu trúc tương tự Map C++

## MULTIMAP

**Định nghĩa:** Tương tự như map, multimap có cách khai báo và sử dụng các hàm tương tự như map nhưng multimap các phần tử có thể có giá trị giống nhau.

Multimap cũng sử dụng thư viện `#include <map>`. Cách khai báo và sử dụng multimap tương tự như cách sử dụng map.

*\*Java: Chưa có multimap.*

# Cấu trúc tương tự Map

## UNORDERED\_MAP & HASHMAP & DICT

**Định nghĩa:** Tương tự như map/treemap, nhưng phần tử được đưa vào sẽ nằm ngẫu nhiên mà không sắp xếp sẵn. Dùng Hash Table để quản lý.

\*\*\*Lưu ý:

unordered\_map trong C++ thuộc thư viện `<unordered_map>`

HashMap trong Java thuộc thư viện `java.util.HashMap`

# Cấu trúc tương tự trong Python

## dict

**Định nghĩa:** Tương tự như unordered\_map trong C++ và HashMap trong Java, những phần tử được đưa vào sẽ nằm **ngẫu nhiên** mà không sắp xếp sẵn. Dùng Hash Table để quản lý.

# Khai báo và sử dụng

**Khai báo:** Khai báo dạng mặc định:

```
variable_name = dict()
```

**Ví dụ:**

```
m = dict()
```



# Khai báo và sử dụng

**Khai báo:** khởi tạo xong sau đó gán giá trị.

```
m = dict()
```

```
variable_name[key] = value
```

**Ví dụ:**

```
m = dict()
```

```
m[10] = 'abc'
```

```
m[20] = 'def'
```

```
m[10] = 'mpk'
```

10, "mpk"	20, "def"
-----------	-----------

# Các hàm thành viên của Dict

Thêm/cập nhật một phần tử vào Dict thông qua toán tử []  
với cú pháp:

```
variable_name[data_type1] = data_type2
```

10, "mpk"	20, "def"
-----------	-----------

```
m[14] = 'abc'
```

**Kết quả**

10, "mpk"	14, "abc"	20, "def"
-----------	-----------	-----------

# Các hàm thành viên của dict

`pop(key[, default])`: Xóa phần tử có key trong dict và trả về value của nó. Nếu key không tồn tại thì trả về default.

10, "mpk"	14, "abc"	20, "def"
-----------	-----------	-----------

```
print (m.pop(14) )
```



**Kết quả**

**abc**

**Lưu ý:** hàm này sẽ raise một **KeyError** nếu như key không tồn tại trong dict, do đó cần kiểm tra trước key có tồn tại trong dict hay không.



# Các hàm thành viên của dict

KeyError là gì?

10, "mpk"	14, "abc"	20, "def"
-----------	-----------	-----------

```
if d.pop(30, None) == None:  
    print('Key not found')  
else:  
    print('deleted')
```

**Kết quả**

**Key not found**

*Lưu ý: tham số thứ 2 của hàm pop chúng ta có thể thay thế bất kỳ chữ gì nếu muốn. Nhưng không được trùng giá trị trong dict.*

# Các hàm thành viên của Dict

`<key> in <dict>`: Kiểm tra key có tồn tại trong dict hay không.

10, "mpk"	14, "abc"	20, "def"
-----------	-----------	-----------

```
if 20 in m:  
    print("exists")  
else:  
    print("not exists")
```



**Kết quả**

**exists**

# Các hàm thành viên của dict

`get(key[, default])`: Lấy giá trị của dict theo key.

10, "mpk"	14, "abc"	20, "def"
-----------	-----------	-----------

```
print(m.get(14))  
print(m.get(1))
```



**Kết quả**

**abc**  
**None**

*Lưu ý: nếu key không có trong dict thì khi dùng toán tử `[]` sẽ raise `KeyError`, hàm `get()` sẽ return về default (mặc định là `None`)*

# Các hàm thành viên của dict

**update(dict)**: Thêm các phần tử của dict này vào dict khác, nếu như có key trùng thì value của key sẽ được lấy theo value trong dict thứ 2.

<b>m1</b>	10, "mpk"	14, "abc"	20, "def"
-----------	-----------	-----------	-----------

<b>m2</b>	10, "bigo"	22, "coding"
-----------	------------	--------------

```
m1.update(m2)
```

**Kết quả**

10, "bigo"	20, "def"	22, "coding"	14, "abc"
------------	-----------	--------------	-----------

# Các hàm thành viên của dict

**items**: Trả về danh sách các phần tử (item) trong dict.

10, "mpk"	14, "abc"	20, "def"
-----------	-----------	-----------

```
for item in m.items():  
    print(item)
```



**Kết quả**

```
(10, 'mpk')  
(20, 'def')  
(14, 'abc')
```

# Các hàm thành viên của dict

**keys**: Trả về danh sách các khóa tìm kiếm (key) trong dict.

10, "mpk"	14, "abc"	20, "def"
-----------	-----------	-----------

```
for key in m.keys():  
    print(key, end = ', ')
```

**Kết quả**

**mpk, def, abc,**

# Các hàm thành viên của dict

**values**: Trả về danh sách các giá trị (value) trong dict.

10, "mpk"	14, "abc"	20, "def"
-----------	-----------	-----------

```
for value in m.values():  
    print(value, end = ', ')
```

**Kết quả**

10, 20, 14,



# Các hàm sử dụng cho dict

- `len(<dict>)` Trả về số lượng phần tử hiện tại có trong dict.
- `clear`: Xóa hết tất cả các giá trị trong dict.
- `copy`: Tạo một shallow copy của dict.

# Hỏi đáp

