Big-O Blue

LECTURE 15

MAP – TREEMAP - DICT

Phạm Nguyễn Sơn Tùng

Email: 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:

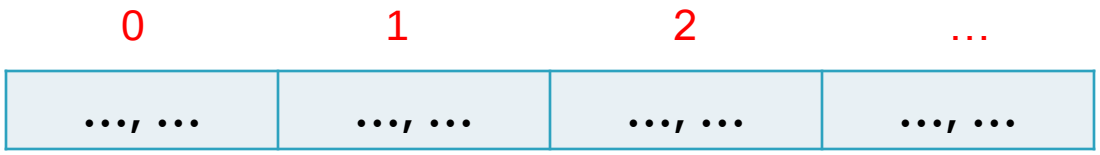
```
import java.util.TreeMap;
```

Khai báo: khai báo dạng mặc định.

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

Ví dụ:

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



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());
```

Sai:

```
map<int, string>
m1(m.begin(), m.begin() + 3);
```

Ví dụ:

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

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

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");
```

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**

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;
```

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. (≥)
- 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. (≤)
- ceilingEntry/ceilingKey: Trả về entry/key nhỏ nhất và lớn hơn hoặc bằng khóa tìm kiếm. (≥)
- 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;
```

abc

Các lưu ý khi sử dụng Map (3)

Duyệt: Duyệt bằng cách sử dụng iterator (duyệt 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<<", ";
}
```

10 mpk, 14 abc, 20 def, 25 po

Các lưu ý khi sử dụng TreeMap (3)

Duyệt: Duyệt bằng cách sử dụng iterator (duyệt 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 (duyệt 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 << ", ";
}
```

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 mutilmap 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.

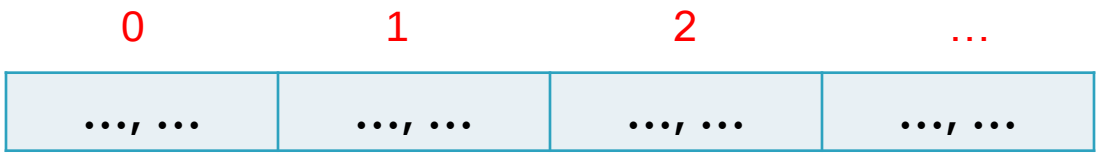
Khai báo và sử dụng

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

del d[key]: xóa mỗi lần 1 phần tử trong dict.

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.

10, “mpk”	14, “abc”	20, “def”

```
del m[14]
```

Kết quả

10, “mpk”	20, “def”

Các hàm thành viên của Dict

Kiểm tra key có tồn tại trong dict hay không:

<key> in <dict>

10, “mpk”	14, “abc”	20, “def”

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

exists

Các hàm thành viên của Dict

Kiểm tra key có tồn tại trong dict hay không:

<key> in <dict>

10, “mpk”	14, “abc”	20, “def”

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

exists

Các hàm thành viên của Dict

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

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)

10, “mpk”	14, “abc”	20, “def”

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


```
abc
None
```

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.

Lưu ý: nếu không truyền tham số cho default và key không tồn tại thì sẽ raise KeyError, vì vậy cần kiểm tra key tồn tại hay không trước.

10, “mpk”	14, “abc”	20, “def”

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

abc

Các hàm thành viên của Dict

update(dict2): Thêm các phần tử của dict2 vào dict, nếu như có key trùng thì value của key sẽ được lấy theo value trong dict2

m1	**10, “mpk”**	**14, “abc”**	**20, “def”**
m2	**10, “bigo”**	**22, “coding”**	

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

10, “bigo”	**20, “def”**	**22, “coding”**	**14, “abc”**

Các hàm thành viên của Dict

- Trả về số lượng phần tử hiện tại có trong dict: sử dụng len(<dict>) tương tự như với list.
- clear: Xóa hết tất cả các giá trị trong Dict.
- copy: Tạo một shallow copy của Dict.
- items: Trả về một list (key, value) của Dict.
- keys: Trả về một list các key của Dict.
- values: Trả về một list các value của Dict.

Hỏi đáp