VECTOR CLASS

```
#include <vector>
```

NỘI DUNG

KHÁI QUÁT VỀ VECTOR CLASS

Vector được coi như mảng động có thể tự động thay đổi kích thước khi thêm hay xóa phần tử khỏi vector.

Lưu trữ được đa dạng các kiểu dữ liệu: int, string, pair, class, struct, vector, set,...

Các phần tử trong vector được lưu trữ trong các ô nhớ liên tiếp và truy cập được thông qua chỉ số.

KHÁI QUÁT VỀ VECTOR CLASS

VECTOR

Phù hợp để thay thế mảng, dễ dàng thêm phần tử vào vector.

Không phù hợp khi bài toán yêu cầu nhiều thác tác tìm kiếm hay xóa phần tử được thực hiện nhiều lần.

1. KHAI BÁO VECTOR

Syntax: `vector <data_type> vector_name;`

Khai báo vector rỗng: `vector <int> v;`

Khai báo vector với các phần tử cho trước:

```
vector <int> v = { 1, 2, 3, 4, 5 };
```

Khai báo vector với n phần tử: `vector <int> v (n);`

Khai báo vector với n phần tử có cùng giá trị val: `vector <int> v (n, val);`

Khai báo vector từ mảng a[ ] có n phần tử, hoặc từ một vector a khác:

```
vector <int> v (a, a + n);
vector <int> v (a.begin(), a.end());
```

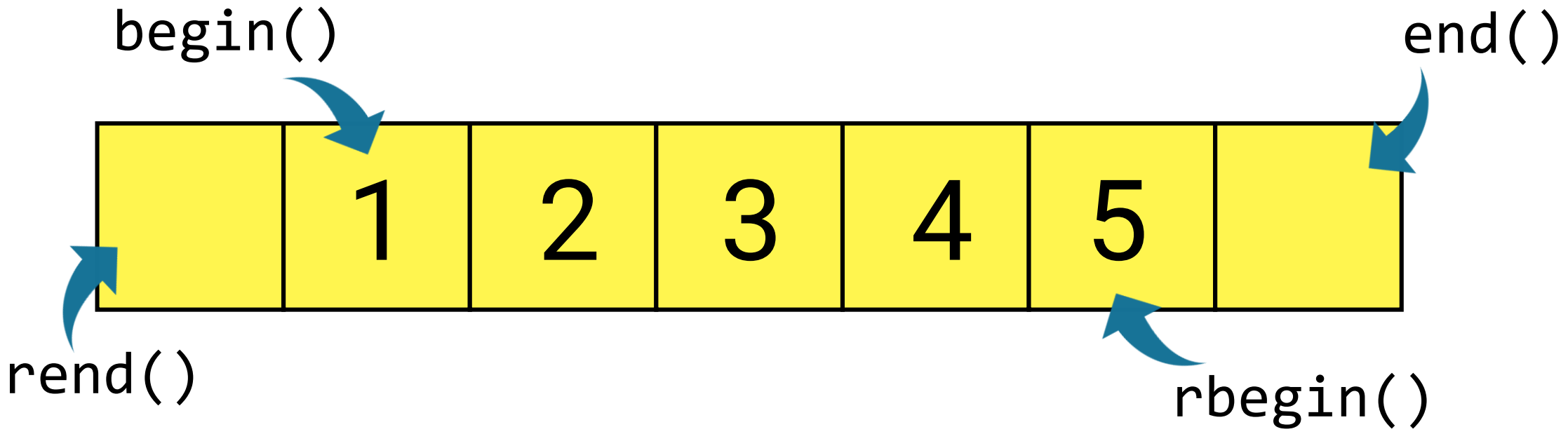
2. VECTOR ITERATOR

Iterator được coi như là một con trỏ thông minh, các bạn có thể hiểu đơn giản iterator là một con trỏ trỏ tới phần tử trong vector, tức là thông qua iterator bạn có thể truy cập, thay đổi phần tử mà nó đang quản lý.

`begin()`	Iterator đến phần tử đầu tiên
`end()`	Iterator đến phần tử sau phần tử cuối
`rbegin()`	Iterator đến phần tử cuối trong vector
`rend()`	Iterator đến phần tử trước phần tử đầu tiên

2. VECTOR ITERATOR

```
vector <int> v = { 1, 2, 3, 4, 5 };
```



2. VECTOR ITERATOR

- Để truy cập vào phần tử mà iterator đang quản lý các bạn phải dùng toán tử giải tham chiếu : *
- Các bạn có thể cộng trừ iterator, hoặc sử dụng toán tử ++ và -- với iterator, đối với vector thì việc khai báo iterator tương đối dài nên các bạn có thể dùng auto cho tiện.
- Nếu muốn truy cập đến iterator trỏ vào phần tử có chỉ số x trong mảng ta có thể sử dụng iterator : begin() + x

VÍ DỤ:

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

int main(){
    vector<int> a = {1, 2, 3, 4, 5, 6, 7, 8};
    vector<int>::iterator it1 = a.begin();
    cout << *it1 << endl; // 1

    vector<int>::iterator it2 = a.end();
    cout << *it2 << endl; // ?????

    vector<int>::iterator it3 = a.end();
    --it3; // dịch iterator này sang trái 1 phần tử
    cout << *it3 << endl; // 8

    cout << *(it1 + 4) << endl; // 5
}
```

3. TRUY CẬP PHẦN TỬ TRONG VECTOR

Truy cập thông qua chỉ số tương tự như mảng 1 chiều

```
vector <int> a =  {1, 2, 3, 4};
cout << a[2]; // 3
```

front ( ) Trả về phần tử đầu tiên trong vector

```
vector <int> a =  {1, 2, 3, 4};
cout << a.front(); // 1
```

back ( ) Trả về phần tử cuối cùng trong vector

```
vector <int> a =  {1, 2, 3, 4};
cout << a.back(); // 4
```

4. DUYỆT VECTOR

```
vector <int> v = { 1, 2, 3, 4, 5 };
```

Duyệt thông qua chỉ số:

```
for(int i = 0; i < a.size(); ++i){
    cout << a[i] << ' ';
}
```

Duyệt bằng for each:

```
for(int x : a){
    cout << x << ' ';
}
```

Duyệt bằng Iterator:

```
for(vector<int>::iterator it = a.begin(); it != a.end(); ++it){
    cout << *it << ' ';
}
```

Hoặc thay `vector<int>::iterator` bằng `auto`

5. CÁC HÀM PHỔ BIẾN TRONG VECTOR

1. Hàm push_back

- **Chức năng:** Thêm 1 phần tử vào cuối vector, vector sẽ tự động mở rộng kích thước.
- **Hàm emplace_back** cũng có chức năng tương tự như hàm push_back.

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

int main(){
  vector <int> v;
  v.push_back(1); // 1
  v.push_back(5); // 1 5
  v.push_back(3); // 1 5 3
  v.push_back(2); // 1 5 3 2
  v.push_back(4); // 1 5 3 2 4
  for(int i = 0; i < v.size(); i++){
     cout << v[i] << ' ';
  }
}
```

OUTPUT: 1 5 3 2 4

5. CÁC HÀM PHỔ BIẾN TRONG VECTOR

2. Hàm size, clear, empty

- **Hàm size:** Trả về số lượng phần tử trong vector
- **Hàm clear:** Xóa toàn bộ các phần tử trong vector
- **Hàm empty:** Trả về true nếu vector rỗng, ngược lại trả về false

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

int main(){
    vector<int> v;
    for(int i = 1; i <= 5; i++){
        v.push_back(i);
    }
    cout << v.size() << endl;
    v.clear(); // xoa toan bo
    if(v.empty()) cout << "EMPTY\n";
    else cout << "NOT EMPTY\n";
}
```

OUTPUT: 5
NOT EMPTY

5. CÁC HÀM PHỔ BIẾN TRONG VECTOR

3. Hàm insert

- **Chức năng:** Thêm 1 phần tử vào vị trí bất kì trong vector thông qua iterator.
- **Cú pháp:** insert(Vị trí iterator, val)

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

int main(){
    vector<int> v;
    for(int i = 1; i <= 5; i++){
        v.push_back(i);
    }
    for(int i = 0; i < v.size(); i++){
        cout << v[i] << ' ';
    }
    v.insert(v.begin() + 2, 100); // thêm
// 100 vào chỉ số 2
    for(int x : v){
        cout << x << ' ';
    }
}
```

OUTPUT:

```
1 2 3 4 5
1 2 100 3 4 5
```

5. CÁC HÀM PHỔ BIẾN TRONG VECTOR

4. Hàm erase

- **Chức năng:** Xóa 1 phần tử trong vector thông qua iterator.
- **Cú pháp:** erase (Vị trí iterator)

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

int main(){
    vector<int> v;
    for(int i = 1; i <= 5; i++){
        v.push_back(i);
    }
    for(int i = 0; i < v.size(); i++){
        cout << v[i] << ' ';
    }
    v.erase(v.begin() + 2); // xóa phần
// tử ở chỉ số 2
    for(int x : v){
        cout << x << ' ';
    }
}
```

OUTPUT:

1 2 3 4 5
1 2 4 5

5. CÁC HÀM PHỔ BIẾN TRONG VECTOR

5. Hàm pop_back

- **Chức năng:** Xóa phần tử cuối cùng trong vector

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

int main(){
    vector<int> v;
    for(int i = 1; i <= 5; i++){
        v.push_back(i);
    }
    for(int i = 0; i < v.size(); i++){
        cout << v[i] << ' ';
    }
    v.pop_back();
    for(int x : v){
        cout << x << ' ';
    }
}
```

OUTPUT:

```
1 2 3 4 5
1 2 3 4
```

5. CÁC HÀM PHỔ BIẾN TRONG VECTOR

6. Hàm assign

- **Chức năng:** Gán các phần tử trong vector với cùng một giá trị
- **Cú pháp:** assign (số phần tử, val)

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

int main(){
    vector<int> v;
    v.assign(5, 100);
    for(int x : v) cout << x << ' ';
}
```

OUTPUT:
100 100 100 100 100

5. CÁC HÀM PHỔ BIẾN TRONG VECTOR

7. Hàm resize

- **Chức năng:** Thay đổi kích thước của vector
- **Cú pháp:** resize(n)

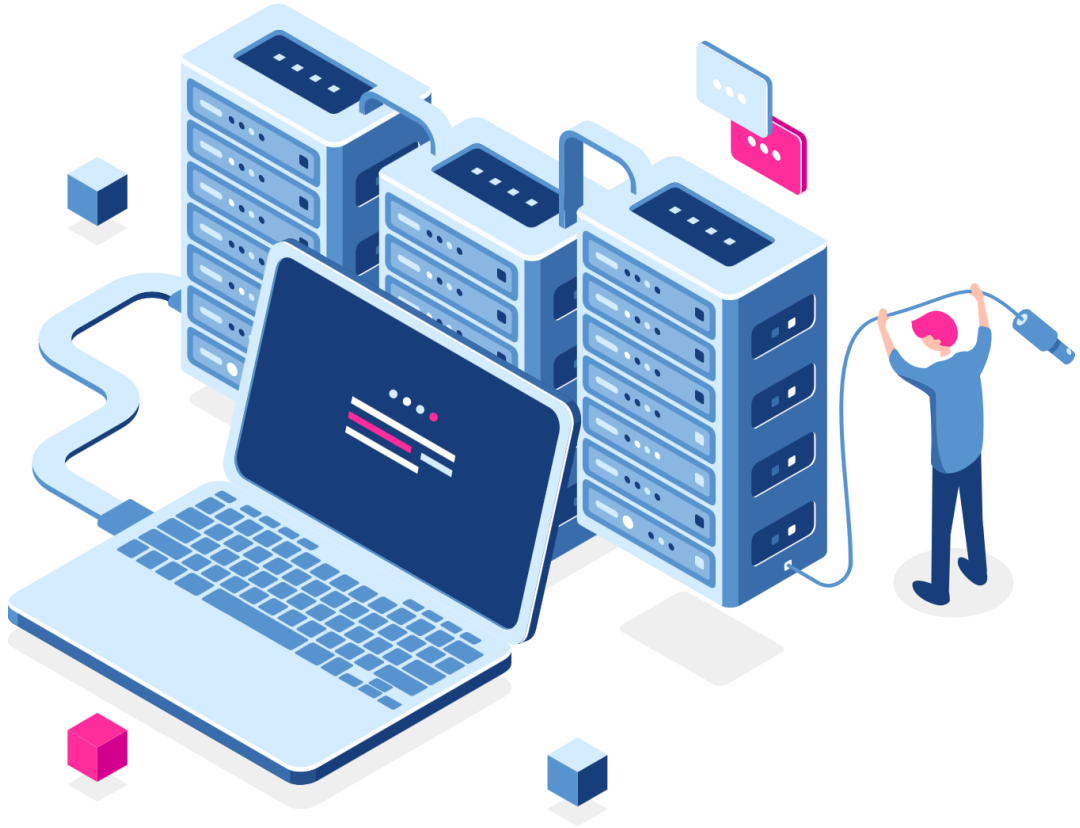
```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> v(5, 28);
    for(int x : v) cout << x << ' ';
    cout << endl;
    v.resize(3);
    for(int x : v) cout << x << ' ';
}
```

OUTPUT:

```
28 28 28 28 28
28 28 28
```

SET, MULTISET, UNORDERED_SET



1. SET:

Khái niệm

Set là một container cực kì mạnh mẽ trong thư viện STL của ngôn ngữ lập trình C++, sử dụng thành thạo Set là một kỹ năng cơ bản mà bạn cần đạt được. Set sẽ giúp code của các bạn trở nên tối ưu và ngắn gọn hơn rất nhiều.

Tính chất

Set là một container mà mỗi phần tử trong đó là duy nhất, tức là sẽ không có 2 phần tử có giá trị giống nhau tồn tại trong set.

Các phần tử trong set được sắp xếp theo thứ tự tăng dần về giá trị số và tăng dần về thứ tự từ điển nếu là xâu kí tự.

1. SET:

CÚ PHÁP

```
set <data_type> set_name;
```

SỬ DỤNG SET

- Các bài toán liên quan tới việc xóa, thêm, tìm kiếm một phần tử nào đó được thực hiện đi thực hiện lại nhiều lần.
- Các bài toán liên quan tới các giá trị khác nhau của mảng

MỘT SỐ HÀM TRONG SET

Hàm insert: thêm một phần tử trong set.

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

int main(){
    set<int> se;
    int a[7] = {1, 1, 2, 1, 3, 4, 5};
    for(int i = 0; i < 7; i++){
        se.insert(a[i]);
    }
}
```

```
se = {1, 2, 3, 4, 5}
```

MỘT SỐ HÀM TRONG SET

Hàm size: trả về số lượng phần tử trong set.

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

int main(){
    set<int> se;
    int a[7] = {1, 1, 2, 1, 3, 4, 5};
    for(int i = 0; i < 7; i++){
        se.insert(a[i]);
    }
    cout << se.size() << endl;
}
```

OUTPUT: 5

MỘT SỐ HÀM TRONG SET

Hàm empty: kiểm tra set rỗng, nếu rỗng trả về true, ngược lại trả về false.

Hàm clear: xóa mọi phần tử trong set.

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

int main(){
    set<int> se;
    int a[7] = {1, 1, 2, 1, 3, 4, 5};
    for(int i = 0; i < 7; i++)
        se.insert(a[i]);
    cout << se.size() << endl;
    if(se.empty())
        cout << "Empty !\n";
    else
        cout << "Not empty !\n";
    se.clear(); // set rỗng
    if(se.empty())
        cout << "Empty !\n";
    else
        cout << "Not empty !\n";
}
```

OUTPUT: 5
Not empty !
Empty !

MỘT SỐ HÀM TRONG SET

Duyệt set

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

int main(){
    set<int> se;
    int a[7] = {1, 1, 2, 1, 3, 4, 5};
    for(int i = 0; i < 7; i++)
        se.insert(a[i]);
    //Cách 1: Dùng For each
    for(int x : se)
        cout << x << ' ';
    cout << endl;
    //Cách 2: Dùng Iterator
    for(set<int>::iterator it = se.begin(); it != se.end(); ++it)
        cout << *it << ' ';
    cout << endl;
    //Cách 3: Dùng Auto
    for(auto it = se.begin(); it != se.end(); ++it)
        cout << *it << ' ';
}
```

OUTPUT:
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

MỘT SỐ HÀM TRONG SET

Hàm find: kiểm tra sự tồn tại của một phần tử nào đó trong set, đây là một hàm được sử dụng rất nhiều của set vì độ phức tạp của nó là O(logN)

Hàm này trả về iterator tới phần tử nếu nó tìm thấy, ngược lại nó trả về iterator end() của set khi phần tử đó không tồn tại trong set

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

int main(){
    set<int> se;
    int a[7] = {5, 5, 1, 2, 3, 4, 5};
    for(int i = 0; i < 7; i++){
        se.insert(a[i]);
    }
    auto it = se.find(3);
    if(it == se.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << "FOUND\n";
    }
}
```

OUTPUT: FOUND

MỘT SỐ HÀM TRONG SET

Hàm count: Hàm này dùng để đếm số lần xuất hiện của 1 phần tử trong set, đối với set thì 1 phần tử sẽ xuất hiện 1 hoặc 0 lần, có thể sử dụng hàm này để thay cho hàm find. Độ phức tạp O(logN).

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

int main(){
    set<int> se;
    int a[7] = {5, 5, 1, 2, 3, 4, 5};
    for(int i = 0; i < 7; i++)
        se.insert(a[i]);
    cout << se.count(1) << endl;
    cout << se.count(6) << endl;
    if(se.count(3) != 0)
        cout << "FOUND\n";
    else
        cout << "NOT FOUND\n";
}
```

OUTPUT:
```
1
0
FOUND
```

MỘT SỐ HÀM TRONG SET

Hàm erase: Xóa 1 phần tử khỏi set với độ phức tạp là O(logN), trước khi sử dụng hàm erase hãy đảm bảo phần tử bạn cần xóa tồn tại trong set nếu không sẽ xảy ra lỗi runtime error.

Xóa bằng giá trị

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

int main(){
    set<int> se;
    int a[7] = {5, 5, 1, 2, 3, 4, 5};
    for(int i = 0; i < 7; i++)
        se.insert(a[i]);
    for(int x : se)
        cout << x << ' ';
    cout << endl;
    se.erase(3);
    for(int x : se)
        cout << x << ' ';
}
```

OUTPUT: 1 2 3 4 5
1 2 4 5

MỘT SỐ HÀM TRONG SET

Hàm erase: Xóa 1 phần tử khỏi set với độ phức tạp là O(logN), trước khi sử dụng hàm erase hãy đảm bảo phần tử bạn cần xóa tồn tại trong set nếu không sẽ xảy ra lỗi runtime error.

Xóa bằng iterator

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

int main(){
    set<int> se;
    int a[7] = {5, 5, 1, 2, 3, 4, 5};
    for(int i = 0; i < 7; i++)
        se.insert(a[i]);
    for(int x : se)
        cout << x << ' ';
    cout << endl;
    auto it = se.find(3);
    if(it != se.end())
        se.erase(it);
    for(int x : se)
        cout << x << ' ';
}
```

OUTPUT: 1 2 3 4 5
1 2 4 5

Dự đoán output của một số chương trình sau

Quiz 1

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

int main(){
    int a[7] = {5, 5, 1, 2, 3, 4, 5};
    set<int> se(a, a + 7);
    auto it = se.find(7);
    --it;
    cout << *it << endl;
}
```

Quiz 2

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

int main(){
    int a[7] = {5, 5, 1, 2, 3, 4, 5};
    set<int> se(a, a + 7);
    auto it = se.find(4);
    it -= 2;
    cout << *it << endl;
}
```

Dự đoán output của một số chương trình sau

Quiz 3

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

int main(){
    int a[7] = {5, 5, 1, 2, 3, 4, 1};
    set<int> se(a, a + 7);

    cout << *se.begin() << endl;

    auto it = se.end(); --it;
    cout << *it << endl;
}
```

Quiz 4

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

int main(){
    int a[7] = {5, 5, 1, 2, 3, 4, 1};
    set<int> se(a, a + 7);

    cout << *se.begin() << endl;
    cout << *se.rbegin() << endl;
}
```

Dự đoán output của một số chương trình sau

Quiz 5

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

int main(){
    int a[7] = {5, 5, 1, 2, 3, 4, 1};
    set<int> se(a, a + 7);
    auto it = se.find(3);
    --it;
    se.erase(it);
    for(int x : se){
        cout << x << ' ';
    }
}
```

Quiz 6

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


int main(){
    int a[7] = {5, 5, 1, 2, 3, 4, 1};
    set<int> se(a, a + 7);
    for(auto it = se.rbegin(); it != se.rend(); ++it){
        cout << *it << ' ';
    }
}
```

2. MULTISET:

Multiset tương tự như set nhưng có thể lưu nhiều phần tử có giá trị giống nhau, các phần tử này cũng được sắp xếp theo thứ tự tăng dần. Các hàm của multiset giống y hệt các hàm của set chỉ khác một chút ở hàm find, count, erase.

MỘT SỐ HÀM TRONG MULTISET

Hàm find: Vì multiset có thể chứa nhiều phần tử giống nhau nên hàm find sẽ trả về iterator đến vị trí đầu tiên của phần tử có giá trị cần tìm kiếm.

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

int main(){
    int a[7] = {5, 5, 1, 1, 2, 2, 2};
    multiset<int> se(a, a + 7);
    //se ={1, 1, 2, 2, 2, 5, 5}
    auto it = se.find(2);
    cout << *it << endl;
    --it;
    cout << *it << endl;
}
```

OUTPUT: 2
1

MỘT SỐ HÀM TRONG MULTISET

Hàm count

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

int main(){
    int a[7] = {5, 5, 1, 1, 2, 2, 2};
    multiset<int> se(a, a + 7);
    //se ={1, 1, 2, 2, 2, 5, 5}
    cout << se.count(1) << endl;
    cout << se.count(2) << endl;
    cout << se.count(4) << endl;
}
```

OUTPUT:
```
2
3
0
```

MỘT SỐ HÀM TRONG MULTISET

Hàm erase: Khi sử dụng hàm erase nếu bạn xóa bằng giá trị multiset sẽ xóa hết mọi phần tử có cùng giá trị bị xóa, vì thế nếu muốn 1 phần tử bạn phải xóa bằng iterator

Xóa bằng giá trị

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

int main(){
    int a[7] = {5, 5, 1, 1, 2, 2, 2};
    multiset<int> se(a, a + 7);
    //se ={1, 1, 2, 2, 2, 5, 5}
    for(int x : se)
        cout << x << ' ';
    cout << endl;
    se.erase(2);
    for(int x : se)
        cout << x << ' ';
}
```

OUTPUT: 1 1 2 2 2 5 5
1 1 5 5

MỘT SỐ HÀM TRONG MULTISET

Hàm erase: Khi sử dụng hàm erase nếu bạn xóa bằng giá trị multiset sẽ xóa hết mọi phần tử có cùng giá trị bị xóa, vì thế nếu muốn 1 phần tử bạn phải xóa bằng iterator

Xóa bằng iterator

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

int main(){
    int a[7] = {5, 5, 1, 1, 2, 2, 2};
    multiset<int> se(a, a + 7);
    //se ={1, 1, 2, 2, 2, 5, 5}
    for(int x : se)
        cout << x << ' ';
    cout << endl;
    auto it = se.find(2);
    se.erase(it);
    for(int x : se)
        cout << x << ' ';
}
```

OUTPUT: 1 1 2 2 2 5 5
1 1 2 2 5 5

3. UNORDERED_SET

Unordered_set tương tự như set, nó chỉ có thể lưu các phần tử đôi một khác nhau nhưng lại không có thứ tự. Không có thứ tự tức là các phần tử ở trong unordered_set sẽ xuất hiện một cách bất kì, không phải là theo thứ tự bạn thêm phần tử vào.

Ví dụ:

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

int main(){
    int a[7] = {5, 5, 1, 1, 2, 2, 2};
    unordered_set<int> se(a, a + 7);
    //se ={1, 1, 2, 2, 2, 5, 5}
    for(int x : se){
        cout << x << ' ';
    }
}
```

OUTPUT: 2 1 5

3. UNORDERED_SET

Sự khác biệt

Unordered_set khác với set và multiset ở tốc độ của các hàm phổ biến như count, find, và erase. Còn cách sử dụng thì không có gì khác biệt so với set.

- Ở set và multiset các hàm có độ phức tạp là O(logN).
- Ở unordered_set tốc độ của các hàm trong trường hợp tốt nhất là O(1) còn tệ nhất có thể lên đến O(N).

MAP, MULTIMAP, UNORDERED_MAP

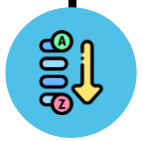
1. MAP:

Khái niệm

Map là container giúp lưu các phần tử theo cặp key, value (khóa - giá trị). Mỗi giá trị của key sẽ ánh xạ sang một value tương ứng. So với Set thì Map thậm chí còn mạnh mẽ và giải quyết được nhiều vấn đề hơn.

Tính chất

Các key trong map là những giá trị riêng biệt, không có 2 key nào có giá trị giống nhau, value thì có thể trùng nhau.



Các cặp phần tử trong map được sắp xếp theo thứ tự tăng dần của key.

Mỗi phần tử trong map thực chất là một pair, với first lưu key và second lưu value.

1. MAP:

CÚ PHÁP

```
map <key_data_type, value_data_type> map_name;
```

SỬ DỤNG MAP

/01 Các bài toán liên quan tới tần suất của các phần tử.

/02 Các bài toán cần tìm kiếm, thêm, xóa một cách nhanh chóng.

/03 Dùng map thay cho các bài toán sử dụng mảng đánh dấu khi dữ liệu không đẹp.

MỘT SỐ HÀM TRONG MAP

Thêm một phần tử vào trong map:

- Dùng **hàm insert**
- Dùng cú pháp **map[key] = value** nếu key chưa tồn tại trong map, hoặc sẽ thay đổi value nếu key đã tồn tại.

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

int main(){
    map<int, int> mp;
    mp.insert({1, 2}); //Thêm cặp (1,2)
    mp.insert({2, 4}); //Thêm cặp (2,4)
    mp.insert({1, 3}); //Không thêm được cặp (1,3)
    mp[3] = 10; // thêm cặp (3,10)
    mp[2] = 5; //Thay đổi cặp (2,4) thành (2,5)
}
```

```
mp = {(1,2),(2,5),(3,10)}
```

MỘT SỐ HÀM TRONG MAP

Hàm size: trả về số lượng phần tử trong map.

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

int main(){
    map<int, int> mp;
    mp.insert({1, 2}); //Thêm cặp (1, 2)
    mp.insert({2, 4}); //Thêm cặp (2, 4)
    mp.insert({1, 3}); //Thêm cặp (1, 3)
//nhưng không thêm được
    mp[3] = 10; // thêm cặp (3, 10)
    mp[2] = 5; //Thay đổi cặp (2, 4)
//thành (2, 5)
    cout << mp.size() << endl;
}
```

OUTPUT: 3

MỘT SỐ HÀM TRONG MAP

Hàm empty: kiểm tra map rỗng, nếu rỗng trả về true, ngược lại trả về false.

Hàm clear: xóa mọi phần tử trong map.

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

int main(){
    map<int, int> mp;
    mp.insert({1, 2}); //Thêm cặp (1, 2)
    mp.insert({2, 4}); //Thêm cặp (2, 4)
    mp.insert({1, 3}); //Thêm cặp (1, 3)
//nhưng không thêm được
    mp.insert({2, 5});
    mp.insert({3, 6});
    cout << mp.size() << endl;
    if(mp.empty()) cout << "Empty\n";
    else cout << "Not empty\n";
    mp.clear(); // xóa hết mọi phần tử
    if(mp.empty()) cout << "Empty\n";
    else cout << "Not empty\n";
}
```

OUTPUT:
```
3
Not empty !
Empty !
```

MỘT SỐ HÀM TRONG MAP

Duyệt map

mp = {(1,2),(2,4),(3,6)}

Duyệt bằng for each:

```
//for each
for(pair<int, int> it : mp){
   cout << it.first << ' ' << it.second << endl;
}
//for each dùng auto thay cho pair
for(auto it : mp){
   cout << it.first << ' ' << it.second << endl;
 }
//Từ key suy ra value
for(auto it : mp){
     int key = it.first;
     cout << key << ' ' << mp[key] << endl;
}
```

MỘT SỐ HÀM TRONG MAP

Duyệt map

mp = {(1,2),(2,4),(3,6)}

Duyệt bằng iterator:

```
//Dùng iterator
for(map<int, int>::iterator it = mp.begin(); it !=
mp.end(); ++it){
    cout << (*it).first << ' ' << (*it).second << endl;
}
//Thay bằng auto cho tiện
for(auto it = mp.begin(); it != mp.end(); ++it){
    cout << (*it).first << ' ' << (*it).second << endl;
}
```

MỘT SỐ HÀM TRONG MAP

Hàm find: Tìm kiếm sự xuất hiện của một key nào đó trong map. Độ phức tạp là O(logN).

Hàm này trả về iterator tới cặp phần tử nếu nó tìm thấy, ngược lại nó trả về iterator end() của map khi giá trị key tìm kiếm không tồn tại trong map.

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

int main(){
    map<int, int> mp;
    mp.insert({1, 2}); //Thêm cặp (1, 2)
    mp.insert({2, 4}); //Thêm cặp (2, 4)
    mp.insert({3, 6}); //Thêm cặp (3, 6)
    auto it = mp.find(1);
    if(it == mp.end()){
        cout << "NOT FOUND KEY\n";
    }
    else{
        cout << (*it).first << ' ' <<
(*it).second << endl;
    }
}
```

OUTPUT: 1 2

MỘT SỐ HÀM TRONG MAP

Hàm count: Hàm này dùng để đếm số lần xuất hiện của 1 key nào đó trong map. Đối với map hàm count trả về 0 hoặc 1, có thể sử dụng hàm này để thay cho hàm find. Độ phức tạp O(logN).

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

int main(){
    map<int, int> mp;
    mp.insert({1, 2}); //Thêm cặp (1, 2)
    mp.insert({2, 4}); //Thêm cặp (2, 4)
    mp.insert({3, 6}); //Thêm cặp (3, 6)
    cout << mp.count(1) << endl;
    cout << mp.count(5) << endl;
}
```

OUTPUT: 1
0

MỘT SỐ HÀM TRONG MAP

Hàm erase: Xóa một phần tử khỏi map với độ phức tạp là O(logN), trước khi sử dụng hàm erase hãy đảm bảo phần tử bạn cần xóa tồn tại trong map nếu không sẽ xảy ra lỗi runtime error.

Xóa thông qua giá trị của key

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

int main(){
    map<int, int> mp;
    mp.insert({1, 2}); //Thêm cặp (1, 2)
    mp.insert({2, 4}); //Thêm cặp (2, 4)
    mp.insert({3, 6}); //Thêm cặp (3, 6)
    mp.erase(1);
    for(auto it : mp){
        cout << it.first << ' ' <<
it.second << endl;
    }
}
```

OUTPUT: 2 4
3 6

MỘT SỐ HÀM TRONG MAP

Hàm erase: Xóa một phần tử khỏi map với độ phức tạp là O(logN), trước khi sử dụng hàm erase hãy đảm bảo phần tử bạn cần xóa tồn tại trong map nếu không sẽ xảy ra lỗi runtime error.

Xóa thông qua iterator

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

int main(){
    map<int, int> mp;
    mp.insert({1, 2}); //Thêm cặp (1, 2)
    mp.insert({2, 4}); //Thêm cặp (2, 4)
    mp.insert({3, 6}); //Thêm cặp (3, 6)
    auto it = mp.find(3);
    if(it != mp.end()){
        mp.erase(it);
    }
    for(auto it : mp){
        cout << it.first << ' ' <<
it.second << endl;
    }
}
```

OUTPUT: 1 2
2 4

Dự đoán output của một số chương trình sau

Quiz 1

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

int main(){
    map<int, int> mp;
    mp[1]++; mp[1]++; mp[1]++;
    mp[2] = 3;
    mp[2] = 4;
    for(auto it : mp){
        cout << it.first << ' ' <<
it.second << endl;
    }
}
```

Quiz 2

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

int main(){
    map<int, int> mp;
    cout << mp[1] << endl;
    mp.insert({1, 2});
    mp.insert({1, 3});
    mp.insert({1, 4});
    mp[1] = 5;
    cout << mp[1] << endl;
}
```

Dự đoán output của một số chương trình sau

Quiz 3

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

int main(){
    map<int, int> mp;
    mp.insert({1, 2});
    mp.insert({1, 3});
    mp.insert({1, 4});
    mp[2] = 3; mp[3] = 4; mp[3]++;
    auto it = mp.find(3);
    --it;
    cout << (*it).first << ' ' <<
(*it).second << endl;
}
```

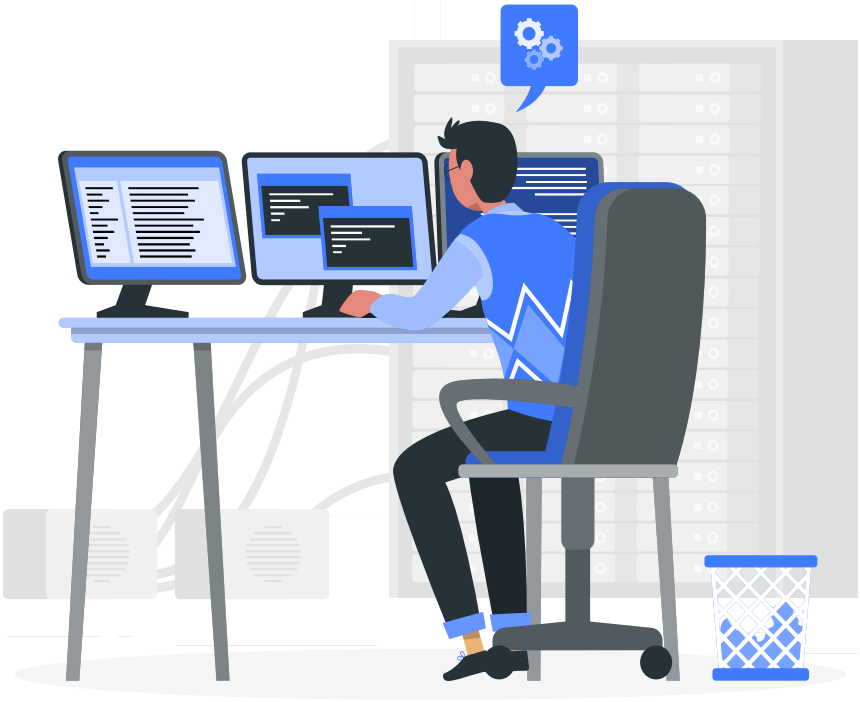
Quiz 4

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

int main(){
    map<int, int> mp;
    int a[] = {1, 2, 3, 4, 1, 2, 3, 4, 5, 5, 1};
    for(int x : a){
        mp[x]++;
    }
    for(auto it : mp){
        cout << it.first << ' ' << it.second <<
endl;
    }
}
```

2. MULTIMAP:

Multimap tương tự như map có điều khác biệt là trong multimap có thể tồn tại nhiều key có cùng giá trị. Các tính chất như key được sắp xếp tăng dần hay độ phức tạp, cách sử dụng hàm thì tương tự như map. Có 3 hàm có sự khác biệt so với map là : find, count, erase.



Chú ý: Trong multimap bạn không thể truy cập value thông qua key, hoặc gán theo cú pháp map[key] = value.

MỘT SỐ HÀM TRONG MULTIMAP

Hàm find: Vì trong multimap có nhiều key có cùng giá trị nên nếu ta tìm kiếm giá trị của 1 key nào đó nó sẽ trả về vị trí xuất hiện đầu tiên của key đó trong multimap.

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

int main(){
    multimap<int, int> mp;
    mp.insert({1, 2});
    mp.insert({1, 3});
    mp.insert({1, 4});
    mp.insert({2, 5});
    auto it = mp.find(1);
    cout << (*it).first << ' ' <<
(*it).second << endl;
}
```

OUTPUT: 1 2

MỘT SỐ HÀM TRONG MULTIMAP

Hàm count

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

int main(){
    multimap<int, int> mp;
    mp.insert({1, 2});
    mp.insert({1, 3});
    mp.insert({1, 4});
    mp.insert({2, 5});
    cout << mp.count(1) << endl;
    cout << mp.count(2) << endl;
}
```

OUTPUT:
```
3
1
```

MỘT SỐ HÀM TRONG MULTIMAP

Hàm erase: Xóa thông qua giá trị sẽ xóa tất cả các key tương ứng.

Xóa bằng giá trị

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

int main(){
    multimap<int, int> mp;
    mp.insert({1, 2});
    mp.insert({1, 3});
    mp.insert({1, 4});
    mp.insert({2, 5});
    mp.insert({2, 6});
    mp.erase(1);
    for(auto it : mp){
        cout << it.first << ' '
<< it.second << endl;
    }
}
```

OUTPUT:
```
2 5
2 6
```

MỘT SỐ HÀM TRONG MULTIMAP

Hàm erase: Xóa thông qua iterator sẽ xóa phần tử mà iterator đó chỉ vào.

Xóa bằng iterator

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

int main(){
    multimap<int, int> mp;
    mp.insert({1, 2});
    mp.insert({1, 3});
    mp.insert({1, 4});
    mp.insert({2, 5});
    mp.insert({2, 6});
    auto it = mp.find(1);
    mp.erase(it);
    for(auto it : mp){
        cout << it.first << ' '
<< it.second << endl;
    }
}
```

OUTPUT:

```
1 3
1 4
2 5
2 6
```

3. UNORDERED_MAP

Unordered_map cũng tương tự như map nhưng các key trong unordered_map không có thứ tự như map và multimap.

Ví dụ:

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

int main(){
    unordered_map<int, int> mp;
    mp.insert({1, 4});
    mp.insert({1, 2});
    mp.insert({1, 3});
    mp.insert({2, 5});
    mp.insert({2, 6});
    mp.insert({3, 5});
    for(auto it : mp){
        cout << it.first << ' '
<< it.second << endl;
    }
}
```

OUTPUT:
```
3 5
2 5
1 4
```

3. UNORDERED_MAP

Sự khác biệt

Unordered_map khác với map và multimap ở tốc độ của các hàm phổ biến như count, find, và erase. Còn cách sử dụng thì không có gì khác biệt so với map.

- Ở map và multimap các hàm có độ phức tạp là O(logN).
- Ở unordered_map tốc độ của các hàm trong trường hợp tốt nhất là O(1) còn tệ nhất có thể lên đến O(N).