**A.THUẬT TOÁN CƠ BẢN**

**1. Giá trị Max/ Min của mảng.**

1. *Khởi gán giá trị ban đầu cho mảng a:*

**memset(a,0,sizeof(a));**

1. *Tìm giá trị lớn nhất của mảng a:*

**\*max\_element(a+start,a+end+1)**

1. *Tìm vị trí phần tử lớn nhất trong a:*

**max\_element(a\_start,a+end+1)-a**

1. *Tìm giá trị nhỏ nhất của mảng a:*

**\*min\_element(a+start,a+end+1)**

1. *Tìm vị trí phần tử nhỏ nhất của a:*

**min\_element(a+start,a+end+1)-a**

1. **Sắp xếp (Sorting)**:

//Sắp xếp toàn bộ mảng a có n phần tử

**sort(a, a + n); sort(a+1,a+1+n)**

//Sắp xếp mảng a từ chỉ số x tới chỉ số y

**sort(a + x, a + y + 1);**

//Sắp xếp mảng giảm dần

**sort(a, a + n, greater<data\_type>());**

1. **Tìm kiếm (Searching)**:
   * Tìm kiếm tuần tự (Sequential Search)

**int** sequentialSearch(**int** arr[], **int** n, **int** x) {

    // Duyệt qua từng phần tử của mảng

**for** (**int** i = 0; i < n; i++) {

        // Kiểm tra nếu phần tử có giá trị bằng x

**if** (arr[i] == x) {

**return** i;  // Trả về chỉ số của phần tử được tìm thấy

        }

    }

    // Nếu không tìm thấy, trả về -1

**return** -1;

}

* + Tìm kiếm nhị phân (Binary Search) : dãy đã được sắp xếp theo thứ tự.

// Dạng 1:

**binary\_search(l, r, val);**

// Dạng 2:

**binary\_search(l, r, val, comp);**

* *Tìm giá trị đầu tiên lớn hơn k của mảng a:*

**\*upper\_bound(a,k,a+1+n)**

* *Tìm vị trí giá trị đầu tiên lớn hơn k của mảng a:*

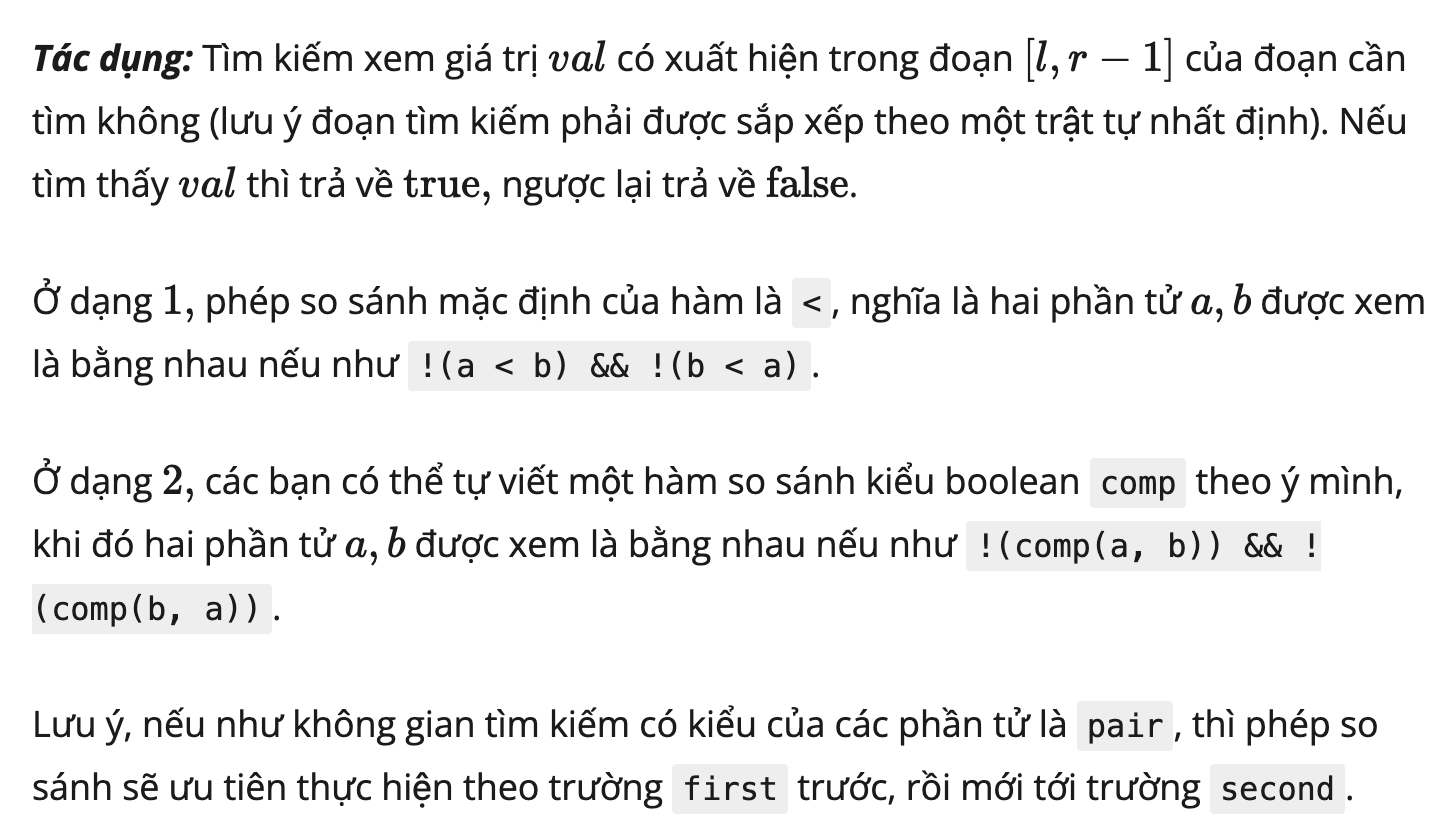
**upper\_bound(a,a+1+n)-a**

* *Tìm giá trị đầu tiên lớn hơn hoặc bằng k của mảng a:*

**\*lower\_bound(a,a+1+n)**

* *Tìm vị trí giá trị đầu tiên lớn hơn hoặc bằng k của mảng a:*

**lower\_bound(a,a+1+n)-a**



1. **Thuật toán đệ quy (Recursion)**:

* **Một hàm được gọi là đệ quy nếu bên trong thân hàm có một lời gọi đến chính nó**.
* Hàm đệ quy luôn có **điều kiện dừng** được gọi là “điểm neo”. Khi đạt tới điểm neo, hàm sẽ không gọi chính nó nữa.
* Khi được gọi, hàm đệ quy thường được truyền cho một tham số, thường là kích thước của bài toán lớn ban đầu. Sau mỗi lời gọi đệ quy, tham số sẽ nhỏ dần, nhằm phản ánh bài toán đã nhỏ hơn và đơn giản hơn. Khi tham số đạt tới một **giá trị cực tiểu** (tại điểm neo), hàm sẽ chấm dứt.

**Bài toán tính giai thừa**

Cho**n** là một số tự nhiên (n>=0). Hãy tính giai thừa của n (**n!**) biết rằng 0! = 1 và n! = (n-1)! \* n.

**Phân tích:**

Theo giả thiết, ta có :**n! = (n-1)! \* n**. Như vậy :

Để tính**n!** ta cần phải tính **(n-1)!**

Để tính **(n-1)!**ta phải tính **(n-2)!**

…

**long** GiaiThua(**int** n)

{

**if** (n == 0)

{

**return** 1; // điều kiện dừng

}

**return** n \* GiaiThua(n - 1); // gọi đệ quy

}

**Dãy Fibonaci** là dãy vô hạn các số tự nhiên. Số Fibonaci thứ **n**, ký hiệu **F(n)**, được định nghĩa như sau :

* **F(n) = 0**, nếu **n = 0**
* **F(n) = 1,**nếu**n = 1**
* **F(n) = F(n-1) + F(n-2)**, nếu **n > 1**

**Yêu cầu:** tính số fibonaci thứ n với n cho trước.

**int** fibonacci(**int** number)

{

**if** (number == 0)

**return** 0; // điều kiện dừng

**if** (number == 1)

**return** 1; // điều kiện dừng

**return** fibonacci(number - 1) + fibonacci(number - 2);

}

**B. CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC**

1. Map

a. Khai báo:

map<kiểu dữ liệu, kiểu dữ liệu> tên\_map;

int main ()

{

map<string,int> mp = {

{ "alpha", 20 },

{ "beta", 20 },

{ "alpha", 10 },

{ "gamma", 30 },

{ "alpha", 10 },

};

for (auto x: mp) {

cout << x.first << ": " << x.second << endl;

}

cout<< mp.size() <<endl;

return 0;

}

Kq:

alpha: 20

beta: 20

gamma: 30

3

**b. Tìm phần tử trong map C++ bằng hàm find**

**Hàm count** là một hàm thành viên trong class std:map, có tác dụng đếm số lần xuất hiện của phần tử trong map C++ thông qua khóa của nó.

Chúng ta sử dụng hàm count trong C++ với cú pháp sau đây:

mp.count(key);

#include <iostream>

#include <map>

using namespace std;

int main() {

map<char,int> mp;

mp['a']=100;

mp['b']=200;

mp['c']=300;

//Đếm số lần xuất hiện của phần tử tồn tại trong map

cout << mp.count('b') <<endl;

//Đếm số lần xuất hiện của phần tử không tồn tại trong map

cout << mp.count('f') <<endl;

return 0;

}

Kq:

1

0

**Hàm equal\_range** là một hàm thành viên trong class std:map, có tác dụng tìm phạm vi của tất cả các phần tử có khóa giống với khóa chỉ định trong map.

Chúng ta sử dụng hàm equal\_range trong C++ với cú pháp sau đây:

mp.find(key);

#include <iostream>

#include <map>

using namespace std;

//Tạo hàm xuất map

void dump(map<char,int>& mp)

{

for (auto x: mp) {

cout << x.first << ":" << x.second << " ";

}

cout << endl;

}

int main() {

map<char,int> mp;

mp.insert(make\_pair('a', 1));

mp.insert(make\_pair('b', 2));

mp.insert(make\_pair('c', 3));

mp.insert(make\_pair('d', 1));

mp.insert(make\_pair('e', 2));

dump(mp);

//Tìm phần tử có khóa bằng 'c' trong map

auto ret = mp.equal\_range('c');

//Xóa phần tử vừa tìm thấy

mp.erase (ret.first,ret.second);

dump(mp);

return 0;

}

Kq:

a:1 b:2 c:3 d:1 e:2

a:1 b:2 d:1 e:2

**Hàm lower\_bound** là một hàm thành viên trong class std::map, có tác dụng tìm vị trí **phần tử đầu tiên** trong map có khóa lớn hơn hoặc bằng với khóa chỉ định.

Chúng ta sử dụng hàm lower\_bound trong C++ với cú pháp sau đây:

mp.lower\_bound(key);

Hàm lower\_bound() sẽ trả về trình lặp trỏ đến vị trí **phần tử đầu tiên** có khóa lớn hơn hoặc bằng với khóa chỉ định. Và nếu không tìm thấy, hàm sẽ trả về trình lặp trỏ đến vị trí cuối cùng trong map.

#include <iostream>

#include <map>

using namespace std;

//Tạo hàm xuất map

void dump(map<char,int>& mp)

{

for (auto x: mp) {

cout << x.first << ":" << x.second << " ";

}

cout << endl;

}

int main() {

map<char,int> mp;

mp.insert(make\_pair('a', 1));

mp.insert(make\_pair('b', 2));

mp.insert(make\_pair('c', 3));

mp.insert(make\_pair('d', 1));

mp.insert(make\_pair('e', 2));

mp.insert(make\_pair('f', 3));

dump(mp);

/\*Tìm vị trí phần tử đầu tiên có khóa lớn hơn hoặc bằng 'b' trong map\*/

auto itr1 = mp.lower\_bound('b'); // itr1 trỏ đến b:2

//Tìm vị trí phần tử đầu tiên có khóa lớn hơn 'e' trong map

auto itr2 = mp.upper\_bound('e'); // itr2 trỏ đến f:3

//Xóa các phần tử trong phạm vi [itr1, itr2)

mp.erase (itr1, itr2);

dump(mp);

return 0;

}

Kết quả, các phần tử trong phạm vi từ b:2 đến trước f:3 đã bị xóa đi.

a:1 b:2 c:3 d:1 e:2 f:3

a:1 f:3

**Hàm upper\_bound** là một hàm thành viên trong class std::map, có tác dụng tìm vị trí **phần tử đầu tiên** trong map có khóa lớn hơn khóa chỉ định.

Chúng ta sử dụng hàm upper\_bound trong C++ với cú pháp sau đây:

**mp.upper\_bound(key);**

#include <iostream>

#include <map>

using namespace std;

int main() {

map<char,int> mp;

mp['a']=100;

mp['b']=200;

mp['c']=300;

mp['d']=300;

mp['e']=300;

mp['f']=300;

/\*Duyệt map mp\*/

for (auto x: mp) {

cout << x.first << ":" << x.second << " ";

}

cout << endl; //a:1 b:2 c:3 d:1 e:2

//Tìm phần tử có khóa bằng 'b' trong map

auto itr1 = mp.lower\_bound('b'); // itr1 trỏ đến b:200

//Tìm vị trí phần tử đầu tiên có khóa lớn hơn 'e' trong map

auto itr2 = mp.upper\_bound('e'); // itr2 trỏ đến f:300

//In các phần tử trong phạm vi (itr1, itr2)

for (auto it=itr1; it!=itr2; ++it)

cout << (\*it).first << ":" << (\*it).second << ' ';

return 0;

}

Kq:

a:100 b:200 c:300 d:300 e:300 f:300

b:200 c:300 d:300 e:300

**2.VECTOR**

**a.Nhập xuất vector.**

#include <bits/stdc++.h>

#define int int\_fast64\_t

using namespace std;

main()

{

int n;

cin >> n;

vector<int> a;

for (int i = 0; i < n; ++i)

{

int x;

cin >> x ;

a.push\_back(x);

}

for (int i = 0; i < a.size(); i++) cout<<a[i]<<" ";

//hoac

for (int i : a) cout<<i<<" ";

return 0;

}

**b. Xoá phần tử, vị trí.**

//Xoa phan tu tai vi tri m

int m;

cin>>m;

a.erase(a.begin() + m);

for (int i : a) cout<<i<<" ";

// Xoa phan tu co gia tri bang k trong vector

int k;

cin>>k;

a.erase(remove(a.begin(), a.end(), k), a.end());

for (int i : a) cout<<i<<" ";

**c. Chèn phần tử trong vector**

// Chen phan tu s vao vi tri t

int s,t;

cin>>s>>t;

a.insert(a.begin() +s, t);

for (int i : a) cout<<i<<" ";

**d. Tìm giá trị lớn nhất nhỏ nhất trong vector**

//Tim Min, Max

int mina=\*min\_element(a.begin(), a.end());

int maxa=\*min\_element(a.begin(), a.end());

**e. Tìm vị trí giá trị lớn nhất nhỏ nhất trong vector**

//Tim vi tri Min, Max

auto x=min\_element(a.begin(), a.end());

auto y=max\_element(a.begin(), a.end());

int vtmin= distance(a.begin(), x);

int vtmax= distance(a.begin(), x);

**f. Sắp xếp vector**

//Sap xep vector

sort(a.begin(), a.end());

**g.Tìm kiếm nhị phân**

int value;

cin>>value;

cout<< binary\_search(a.begin(), a.end(), value);

**h. Tìm giá trị đầu tiên lớn hơn hoặc bằng value**

int value;

cin>>value;

auto it = lower\_bound(a.begin(), a.end(), value);

//Giá trị:

cout<<\*it<<endl;

//Vị trí

cout<<distance(a.begin(), it);

**i. Tìm giá trị đầu tiên lớn hơn value**

int value;

cin>>value;

auto it = upper\_bound(a.begin(), a.end(), value);

cout<<\*it<<endl;

cout<<distance(a.begin(), it);

**3. DỮ LIỆU KIỂU PAIR CẶP SỐ**

***a.Khai báo phần tử kiểu pair***

pair<kiểu dữ liệu, kiểu dữ liệu> tên\_pair

***b.Mảng pair***

int n;

//Khai báo

pair<int,int> a[100];

// Nhập mảng pair

cin >> n;

for (int i = 1; i <= n; i++)

{

cin>>a[i].first>>a[i].second;

}

//Xuất mảng pair

for (int i = 1; i <= n; i++)

{

cout<<a[i].first<<" "<<a[i].second<<endl;

}

// Hàm sắp xếp thep First

sort(a+1,a+1+n);

**c. Vector Pair**

**// Khai bao**

`