tổng hợp C++

Nhanh - gọn - rối

I: Cơ bản (phải biết):

- Cấu trúc của 1 bài làm:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(nullptr); cout.tie(nullptr);

  <!-- your code here -->

  return 0;
}
```

- Kiểu dữ liệu:

Kiểu dữ liệu	Bộ nhớ	Phạm vi giá trị
char	1 byte	$-127 \dots 127$
short char	1 byte	$-127 \dots 127$
unsigned char	1 byte	$0 \dots 255$
sign char	1 byte	$-127 \dots 127$
short int	2 byte	$-32768 \dots 32767$
unsigned short int	2 byte	$0 \dots 65535$
signed short int	2 byte	$-32768 \dots 32767$
int	4 byte	$-2147483648 \dots 2147483648$
unsigned int	4 byte	$0 \dots 4294967295$
signed int	4 byte	$-2147483648 \dots 2147483648$
long long	8 byte	$-9223372036854775808 \dots 9223372036854775808$
unsigned long long	8 byte	$0 \dots 18446744073709551615$
signed long long	8 byte	$-9223372036854775808 \dots 9223372036854775808$
float (âm)	4 byte	$-(3{,}40282 \times 10^{38}) \dots -(1{,}17549 \times 10^{-38})$
float (dương)	4 byte	$1{,}17549 \times 10^{-38} \dots 3{,}40282 \times 10^{38}$
double (âm)	8 byte	$-(1{,}79769 \times 10^{308}) \dots -(2{,}22507 \times 10^{-308})$
double (dương)	8 byte	$2{,}22507 \times 10^{-308} \dots 1{,}79769 \times 10^{308}$

- Kiểu dữ liệu (có cấu trúc):

  A: MẢNG

  1. Mảng 1 chiều (10 mũ 6)

```
 x = *max_element(a + start, a + 1 + end);
 // giá trí lớn nhất của mảng a từ start đến end
 auto v = max_element(a + start, a + 1 + end);
 // vị trí lớn nhất của mảng a từ start đến end
```

```
//ngược lại
x = *min_element(a + start, a + 1 + end);
v = min_element(a + start, a + 1 + end);
```

```
bool cmp(int x, int y){
      return (x > y);
}

sort(a + start, a + 1 + end, cmp);
// sắp a giảm dần từ start đến end
```

```
binary_search(a + start, a + 1 + end, val);
//Tìm kiếm nhị phân val trong mảng đã được sort

*lower_bound(a + start, a + 1 + end, val);
//Tìm kiếm giá trị đầu tiên lớn hơn hoặc bằng val trong mảng đã sort

lower_bound(a + start, a + 1 + end, val) - a;
//Vị trí đầu tiên lớn hơn hoặc bằng val trong mảng đã sort

*upper_bound(a + start, a + 1 + end, val);
//tương tự nhưng mà lớn hơn hẳn

upper_bound(a + start, a + 1 + end, val) - a;
//Tương tự nhưng mà lớn hơn hẳn
```

2. Vector

```
vector<kiểu dữ liệu> a;

a.push_back(val);
a.pop_back(val);
a.emplace_back(val);
```

```
a.at(0); //Truy cập phần tử tại 0, có ktra lỗi)
a[0]; //giống a.at, nhưng ko ktra lỗi)
a.front() //phần tử đầu
a.back() //phần tử cuối

vector<kiểu dữ liệu> vec;

vec.erase(vec.begin() + 1); //xóa phần tử thứ 2
vec.clear(); //clear toàn bộ vector
vec.empty(); //ktra vector vec có rỗng ko
sort(vec.begin(), vec.end()); //sort nhưng ở vector
reverse(vec.begin(), vec.end()); //đảo ngược các ký tự
```

3. Map

```
map<kiểu dữ liệu của key, kiểu dữ liệu của giá trị> tên_map

map<int, int, greater<int>> mp; //sắp xếp giảm dần
map<int, map<int, int>> mp; //map lồng nhau (nested map)
unordered_map<int, int> mp; //giống map nhưng ko sắp xếp
mp.find(key) != mp.end() //ktra key có tồn tại trong map ko

for(auto it = mp.begin(); it != mp.end(); ++it){
}
//Duyệt từ đầu đến cuối map
for(auto x : mp){
}
// ko khác gì cái ở trên
```

4. Xâu (string, hoặc là 1 chuỗi char)

```
string s; //khai báo string

s.size(); //chiều dài xâu
s.find(T); //tìm vị trí xuất hiện đầu tiên xâu T trong s, n
s.rfind(T); //tương tự s.find nhưng mà được tìm từ bên phải
s.substr(vt, n); //sao chép xâu s từ vị trí vt với n phần t
s.c_str(); //trả về 1 mảng kí tự biểu diễn s
s.erase(vt, n); //xóa xâu s từ vị trí vt với n phần tử
t.insert(vt, s); //Chèn xâu s vào t tại vị trí vt

atoll(s.c_str()); //Chuyển xâu thành số
itoa(x, k, 10); //Chuyển số thành xâu
```

```
string s = to_string(i); //ko khác gì itoa

getline(cin, s) //đọc một dòng từ thiết bị vào cho xâu a
```

5. Hàng đợi (queue)

```
//Định nghĩa: Phần tử vào trước sẽ ra trước
queue<int> q; //khái báo queue

q.push(val); //thêm phần tử vào cuối
q.pop(); //loại bỏ phần từ đầu
q.front(); //truy cập phần tử đầu tiên
q.back(); //truy cập phần tử cuối
q.empty(); //ktra hàng đợi có rỗng ko
q.size(); //trả về số lượng phần tử trong hàng đợi
```

6. Hàng đợi ưu tiên (priority queue)

```
//hàng đợi ưu tiên giảm dần:
priority_queue<int> pq; //vd: 30; 10; 20 -> 10; 20; 30
//hàng đợi ưu tiên tăng dần:
priority_queue<int, vector<int>, greater<int>> pq; //vd: 3

pq.top(); //giống hệt q.front();
```

7. Ngăn xếp (stack)

```
//Định nghĩa: Phần tử vào sau sẽ ra trước:
stack<int> st; //khai báo stack

st.push(val);
st.pop();
st.top();
st.empty();
s.size();
//giống hệt cái queue nhưng hoạt động theo định nghĩa của s
```

8. Set

```
//Định nghĩa: tự động sắp xếp theo thứ tự tăng dần + ko có
set<kiểu dữ liệu> se;
```

```
se.size(); //trả về số lượng phần tử trong set
se.insert(); //thêm phần tử vào trong set
se.empty(); //ktra set có rỗng ko
se.clear(); //xóa toàn bộ phần tử trong set

se.find(val); //tìm kiếm 1 phần tử trong set
//se.find(val) != se.end();
se.count(val); //số lần xuất hiện của phần tử trong set (t
se.erase(val); //xóa phần tử val trong set
```

9. struct

```
//Định nghĩa: Dùng để nhóm nhiều kiểu dữ liệu khác nhau và
struct Adso{
    int Age;
    string Name;
};
int main(){
    Adso a = {18, "adsovetzky"};
    cut << a.Age << " " << a.Name;
}
```

10. Vector 2 chiều

```
int n, m; //độ dài, độ rộng của vector 2 chiều
vector<vecto <int>> vec(n, vector<int>(m));

//nhập vector 2 chiều
for(int i = 0; i < n; i++){
  for(int j = 0; j < m; j++){
    cin >> a[i][j];
  }
}
//xuất vector 2 chiều
for(int i = 0; i < n; i++){
  for(int j = 0; j < m; j++){
    cout << a[i][j] << " ";
  } cout << "\n";
}
```

B: Các hàm thông dụng

1. Mảng thống kế

```
int n;
cin >> n;
vector<int> a(n), d(n, 0);

for(int i = 1; i <= n; i++){
    cin >> a[i];
    d[a[i]]++;
}
```

2. Mảng cộng dồn

```
int n;
cin >> n;
vector<int> a(n), d(n, 0);

for(int i = 1; i <= n; i++){
    cin >> a[i];
    d[i] = d[i - 1];
    if(điều kiện){
        d[i]++;
    }
}
```

3. Mảng tổng

```
int n;
cin >> n;
vector<int> a(n), d(n, 0);

for(int i = 1; i <= n; i++){
    cin >> a[i];
    d[i] = d[i - 1] + a[i];
}
```

4. Mảng tổng v2 (nếu kích thước mảng > 10 mũ 6) (dùng map)

```
map<int, int> mp;
vector<int> a(n);
int n; cin >> n;
for(int i = 1; i <= n; i++){
```

```
    cin >> a[i];
    mp[a[i]]++;
}
```

5. Mảng với dữ liệu là kiểu cặp (dùng pair)

```
pair<int, int> p; //phần tử cặp
pair<int, int> a[val]; //mảng cặp

cin >> p.first >> p.second; //chỉ nhập 1 cặp

for(int i = 1; i <= n; i++){
    cin >> a[i].first >> a[i].second;
} // nhập nhiều cặp
//note: pair sẽ được sort theo .first nếu dùng hàm .sort()
```

6. Sàng nguyên tố (**gốc**)

```
#define ll long long //tuỳ chọn

const int MAXN = 1e6 + 1;
vector<bool> is_prime(MAXN, 1);

void sang(){
    is_prime[0] = is_prime[1] = 0;
    for(ll i = 2; i * i < MAXN; i++){
        if(is_prime[i]){
            for(ll j = i * i; j < MAXN; j+=i){
                is_prime[j] = 0;
            }
        }
    }
}

int main(){
    sang();
    vector<ll> primes, cntprimes(MAXN, 0);

    //Vector chứa toàn bộ số nguyên tố <= 10^6 (?)
    for(ll i = 2; i < MAXN; i++){
        if(is_prime[i]){
            primes.push_back(i);
        }
```

```
    }

    //Vector đếm số lượng số nguyên tố <= 10^6 (?)
    for(ll i = 2; i < MAXN; i++){
        cntprimes[i] = cntprimes[i - 1];
        if(is_prime[i]){
            cntprimes[i]++;
        }
    }
}
```

7. ktra số nguyên tố (nếu chỉ có 1 số):

```
  int n; cin >> n;

  if (n == 1) {
    cout << "NO";
    return 0;
  } else {
    for (int i = 2; i * i <= n; i++) {
      if (n % i == 0) {
        cout << "NO";
        return 0;
      }
    }
  }
  cout << "YES";
```

8. phân tích 1 số thành các thừa số nguyên tố

```
int n;
  cin >> n;
  vector<pair<int, int>> a;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      int cnt = 0;
      while (n % i == 0) {
        n /= i;
        cnt++;
        a.push_back({i, cnt});
      }
    }
  }
```

```
  if (n > 1) {
    a.push_back({n, 1});
  }
//vd 10 thành 2 x 5, 30 = 2 x 3 x 5, 45 = 3 ^ 2 x 5
```

C: Bài toán tổng ước, đếm ước của số N

1. Số lượng ước của N:

   nếu N được phân tích = $p1^{a1}p2^{a2}...pk^{ak}$
   thì d = $(a_1 + 1)(a_2 + 1)\dots(a_k + 1)$

2. Tổng các ước của N:

   S = $\frac{p_1^{a_1+1}-1}{p_1-1} \cdot \frac{p_2^{a_2+1}-1}{p_2-1} \dots \frac{p_k^{a_k+1}-1}{p_k-1}$

   hoặc

   S = $\frac{(p_1^{a_1+1}-1)(p_2^{a_2+1}-1)\cdots(p_k^{a_k+1}-1)}{(p_1-1)(p_2-1)\cdots(p_k-1)}$

   Ví dụ:

```
int tonguoc(int n){
 for(int i = 2; i * i <= n; i++){
   if(n % i == 0){
     p[++k];
     a[k] = 0;
     while(n % i == 0){
       a[k]++;
       n /= i;
     }
   }
   if(n > 1){
     p[++k] = n;
     a[k] = 1;
   }
 }
}
```

D: Các thuật toán số học:

1. Ước số - Bội số:
   1.1: Tính chất:

   - Nếu a là ước của b và b là ước của a → thì a = ±b
   - Nếu a là ước của b và b là ước của c → a là ước của c

- Nếu c là ước của a và c là ước của b → c là ước của p.a + q.b với p, q nguyên tùy ý

1.2: Tính chất của UCLN (gcd()):

- gcd(a.c, b.c) = gcd(a, b).c

1.3: Tính chất của BCNN (lcm()):

- lcm(a, b) = $\frac{ab}{gcd(a,b)}$
- Giả sử k là 1 bội chung của a, b -> k = lcm(a, b) khi và chỉ khi gcd($\frac{k}{a}$, $\frac{k}{b}$) = 1
- k.lcm(a, b) = lcm(a.k , b.k) với mọi k > 0
- Nếu d là ước của a, d là ước của b và d > 0 thì lcm($\frac{a}{d}$, $\frac{b}{d}$) = $\frac{lcm(a,b)}{d}$

E: Quy hoạch động:

1. Khái niệm:

```
Định nghĩa: Là chia nhỏ một bài toán phức tạp thành nhiều bài toán c

Điểm cốt lõi:

  • Một bài toán lớn có thể được giải bằng kết quả của những bài toá

  • Để tránh tính lại nhiều lần, ta lưu kết quả các bài toán con → g
```

2. Ví dụ:

2.1: Dãy Fibonacci:

```cpp
int fib(int n){
  vector<int> f(n + 1);
  f[1] = f[2] = 1;
  for(int i = 3; i <= n; i++){
    f[i] = f[i - 1] + f[i - 2];
  }
  return f[n];
}
```

2.2: Bậc thang:

```cpp
#include<bits/stdc++.h>
using namespace std;
int main(){
  int n; cin >> n;
```

```
  vector<int> dp(n + 1, 0);
  dp[0] = 1;
  dp[1] = 1;
  for(int i = 2; i <= n; i++){
    dp[i] = d[i - 1] + dp[i - 2];
  }
  cout << dp[n];
}
```

2.3: Tìm dãy con tăng dài nhất (LIS):

```
#include<bits/stdc++.h>
using namespace std;
int main(){
  int n; cin >> n;
  vector<int> a(n);
  for(int i = 0; i < n; i++) cin >> a[i];
  vector<int> tail;
  for(int x : a){
    auto it = lower_bound(tail.begin(), tail.end(), x);
    if(it = tail.end()) tail.push_back(x);
    else *it = x;
  }
  cout << tail.size() << "\n";
}
```

2.4: 4 bước học quy hoạch động:

- **B1**: *Xác định "trạng thái" (state)*
  - Trạng thái = một cách mô tả bài toán con
  - *vd*: Fibonacci: f[i] = số fibonacci thứ i
- **B2**: *Xác định "công thức chuyển" (Transition)*
  - Đây là mối quan hệ giữa các bài toán con
  - *vd*: Fibonacci: f[i] = f[i - 1] + f[i - 2];
- **B3**: *Điều kiện cơ sở (Base case)*
  - Là giá trị ban đầu
  - *vd*: Fibonacci: f[1] = 1, f[2] = 1
- **B4**: *Tính dần và lấy kết quả*
  - Tính từ nhỏ đến lớn(hoặc dùng đệ quy có nhớ)
  - kết quả cuối = f[n]

F: Đệ quy, quay lui, đồ thị:

1. Đệ quy:

```
//định nghĩa: là hàm được gọi lại chính nó bằng phương pháp quy

//vd đơn giản nhất
void DeQuy(int n){
 if(n > 0){
   cout << n << "  ";
   DeQuy(n - 1);
 }
}
int main(){
 DeQuy(4);
 return 0;
}

//vd2: Tính tổng số tự nhiên từ 1 tới N bằng đệ quy
int sum(int n){
 if(n == 0){
   return 0;
 } else {
   return n + sum(n - 1);
 }
}
int main(){
 cout << sum(3) << "\n";
 return 0;
}
```

2. Thuật toán Quay lui:

```
//Định nghĩa: dùng để giải bài toán liệt kê các cấu hình. Mỗi c

//mẫu, cơ sở của thuật toán:
   void backtrack(int pos)
 {
     // Trường hợp cơ sở
     if (<pos là vị trí cuối cùng>)
     {
         <output/lưu lại tập hợp đã dựng nếu thoả mãn>
         return;
     }

     //Phần đệ quy
     for (<tất cả giá trị i có thể ở vị trí pos>)
```

```
    {
        <thêm giá trị i vào tập đanh xét>
        backtrack(pos + 1);
        <xoá bỏ giá trị i khỏi tập đang xét>
    }
}
```

3: BFS, DFS, đồ thị:

- toi xin loi :broken_heart: