CẤU TRÚC DỮ LIỆU TRONG THƯ VIỆN STL

STANDARD TEMPLATE LIBRARY

Giáo viên: Hoàng Đăng Quang - 0903781593

CẤU TRÚC DỮ LIỆU TRONG THƯ VIỆN STL

1. Giới thiệu thư viện STL
2. Cấu trúc dữ liệu pair
3. Cấu trúc dữ liệu vector
4. Cấu trúc dữ liệu set
5. Cấu trúc dữ liệu map

Giáo viên: Hoàng Đăng Quang - 0903781593

GIỚI THIỆU THƯ VIỆN STL

Standard Template Library - thư viện Template chuẩn của C++. STL chính là một thư viện chứa những template (khuôn mẫu) của cấu trúc dữ liệu cũng như thuật toán được xây dựng một cách tổng quát nhất, nhằm hỗ trợ cho người dùng trong quá trình lập trình.

Thư viện STL giúp người dùng thực hiện toàn bộ các công việc như vào ra dữ liệu, quản lý mảng động, xây dựng sẵn những cấu trúc dữ liệu cơ bản (ngăn xếp, hàng đợi, tập hợp,...) và bao gồm cả các giải thuật cơ bản như sắp xếp, tìm min - max, tính tổng, thậm chí là tìm ước chung lớn nhất...

Thư viện STL có 4 thành phần chính.

Containers Library: Thư viện chứa các cấu trúc dữ liệu mẫu như vector, stack, queue, deque, set, map,...

Algorithm Library: Chứa các thuật toán viết sẵn để thao tác với dữ liệu.

Iterator Library: Là các biến lặp, sử dụng để truy cập, duyệt các phần tử dữ liệu của các containers.

Numeric Library: Chứa các hàm toán học.

CẤU TRÚC DỮ LIỆU PAIR

Khái niệm

pair là một cấu trúc dữ liệu đặc biệt trong C++.

Nó lưu trữ hai giá trị có thể khác kiểu dữ liệu.

Được định nghĩa trong thư viện **<utility.h>** tuy nhiên có thể sử dụng thư viện tổng quát **<bits/stdc++.h>**

Khai báo

pair <tên kiểu 1, tên kiểu 2> tên biến pair;

Ví dụ

```
// Khai báo biến lưu thông tin học sinh có 2 trường dữ liệu là số thứ tự và họ tên
pair <int, string> hocsinh;

// Khai báo một mảng pair để lưu thông tin học sinh của một lớp có tối đa 50 học sinh
pair <int, string> lop[50];
```

CẤU TRÚC DỮ LIỆU PAIR

Lưu giá trị vào biến kiểu pair

Cách 1: gán thủ công

```
hocsinh.first = 1;
hocsinh.second = "Quang Minh";
```

Cách 2: sử dụng hàm make_pair

```
// Duyệt từ đầu đến cuối danh sách lớp, lưu thông tin học sinh vào mảng lớp bao gồm số tt và họ tên
for (int i = 1; i <= n; ++i) {
    cin >> hoten;
    lop[i] = make_pair(i, hoten);
}
```

CẤU TRÚC DỮ LIỆU PAIR

Truy cập giá trị phần tử trong pair

Tên biến pair.first

Tên biến pair.second

Ví dụ

```
cout << hocsinh.first << ") " << hocsinh.second;
```

```
for (int i = 1; i <= n; ++i)
    cout << lop[i].first << ") " << lop[i].second << "\n";
```

Lưu ý khi sử dụng biến kiểu pair

Không thể lưu nhiều hơn hai phần tử, dùng tuple hoặc struct nếu cần lưu nhiều hơn.

Có thể lồng pair bên trong array, vector, map, ...

CẤU TRÚC DỮ LIỆU PAIR

Bài tập ví dụ

Cho danh sách điểm cuối khóa cơ bản của n học sinh, mỗi học sinh có họ tên và điểm kiểm tra cuối khóa. Hãy tiến hành sắp xếp danh sách học sinh giảm dần theo điểm kiểm tra, nếu cùng điểm thì sắp xếp tăng dần theo họ tên.

Dữ liệu vào:

Dòng đầu chứa số nguyên n (0 < n < 10000)

N dòng tiếp theo, mỗi dòng ghi 2 thông tin là họ tên và điểm kiểm tra của 1 học sinh ngăn cách giữa phần họ tên và điểm là kí tự dấu '-'

Dữ liệu ra: ghi danh sách đã được sắp xếp theo yêu cầu, mỗi họ tên và điểm số trên một dòng.

CẤU TRÚC DỮ LIỆU VECTOR

Khái niệm

Vector trong C++ là mảng động (dynamic array).

Có thể thay đổi kích thước khi chương trình đang chạy.

Được định nghĩa trong thư viện **<vector.h>** tuy nhiên có thể sử dụng thư viện tổng quát **<bits/stdc++.h>**

Khai báo vector rỗng

vector <**tên kiểu**> tên biến vector;

Ví dụ

```
// Khai báo một vector (mảng động) chứa các số thực
vector <double> m;
// Khai báo một vector có kiểu pair lưu danh sách nhân viên bao gồm mã và họ tên
vector <pair <int, string>> nhanvien;
```

CẤU TRÚC DỮ LIỆU VECTOR

Khai báo một vector mới đồng thời khởi tạo giá trị cho nó

```
5  vector <int> vec1 = {1, 2, 3, 4, 5};
```

Khai báo vector có giá trị rồi gán giá trị của nó cho một vector khác

```
5  vector <int> vec1 = {1, 2, 3, 4, 5};
6  vector <int> vec2 = vec1;
```

Khai báo một vector có sẵn số phần tử cho trước.

```
8  int spt = 1000;
9  vector <int> vec3(spt);
```

Khai báo một vector dạng ma trận

```
vector <vector <long long>> mt;
// Khai báo vector dang ma trân biết trước dòng cột
int row = 5, col = 7;
vector <vector <int>> mtrc(row, vector<int> (col));
// Khi khai báo vector dang ma trân biết trước dòng cột ta có thể truy cập trực tiếp mtrc[i][j]
```

CÁC THAO TÁC TRÊN VECTOR

Lấy số lượng phần tử đang có trong vector

Tên biến vector.**size**();

Thêm phần tử vào cuối vector

Tên biến vector.**push_back**(giá trị);

```
// Thêm giá trị 3.14 vào cuối vector m
m.push_back(3.14);
// Thêm thông tin nhân viên (kiểu pair) vào cuối vector nhân viên
nhanvien.push_back({1, "Minh Khoa"});
```

Gán phần tử vào vị trí đã có trong vector

Tên biến vector[chỉ số] = giá trị;

```
// Gán giá trị 9999 vào vị trí phần tử thứ 5 trong vector (phần tử phải tồn tại)
vec1[5] = 9999;
```

CÁC THAO TÁC TRÊN VECTOR

Truy xuất đến một phần tử trong vector

Tên biến vector[chỉ số];

```
// Xuất các phần tử từ vị trí 0 đến vị trí 9 trong vector
for (int i = 0; i < 10; i++)
    cout << vec1[i] << " ";
```

Xóa phần tử cuối cùng trong vector

Tên biến vector.**pop_back**();

Kiểm tra vector có rỗng hay không, nếu rỗng thì trả về True, nếu có phần tử thì trả về False

Tên biến vector.**empty**();

Xóa toàn bộ vector

Tên biến vector.**clear**();

ITERATOR

Khái niệm

iterator là bộ lặp: công cụ giúp duyệt qua từng phần tử của vector, list, set...

Nó hoạt động giống như con trỏ: trỏ đến từng phần tử trong dãy dữ liệu.

Ví dụ: Hãy tưởng tượng bạn có một hàng các chiếc hộp (vector), và bạn dùng ngón tay (iterator) để chạm vào từng chiếc hộp, xem bên trong có gì.

Iterator chỉ đến phần tử đầu tiên trong vector

Tên biến vector.**begin**();

Iterator chỉ đến phần tử cuối cùng trong vector

Tên biến vector.**end**();

Sắp xếp vector

sort(tên biến vector.**begin**(), tên biến vector.**end**());

CÁC THAO TÁC TRÊN VECTOR

Chèn phần tử mới vào trước phần tử tại vị trí được chỉ tới bởi iterator.

Tên biến vector.**insert**(vị trí iterator, giá trị cần chèn);

```
// Chèn giá trị 5.4 vào trước vị trí thứ 5 trong vector m
m.insert(m.begin()+5, 5.4);
```

Xóa các phần tử được chỉ định trong vector

Tên biến vector.**erase**(vị trí iterator cần xóa);

```
// Xóa phần tử tại vị trí 5 (phần tử thứ 6) trong vector
m.erase(m.begin()+5);
// Xóa các phần tử từ vị trí 5 đến vị trí 15 trong vector
m.erase(m.begin()+5, m.begin()+15);
// Xóa các phần tử từ vị trí 5 đến cuối trong vector
m.erase(m.begin()+5, m.end());
```

Hoán đổi giá trị 2 vector cùng kiểu (có thể khác kích thước)

Tên biến vector 1. **swap**(tên biến vector 2);

DUYỆT VECTOR

Duyệt vector bằng index

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

Duyệt vector bằng iterator và kiểu dữ liệu auto

```
for (auto nv = nhanvien.begin(); nv != nhanvien.end(); ++nv)
    cout << nv->first << ")" << nv->second << "\n";
```

Sử dụng vector với index âm (vector phải được khởi tạo trước số phần tử)

```
for (int i = -10; i <= 10; i++)
    vec3[i] = i;
for (int i = -10; i <= 10; i++)
    cout << vec3[i] << " ";
```

CẤU TRÚC DỮ LIỆU VECTOR

Ưu điểm

- Không cần phải khai báo kích thước vì vector là mảng động.
- Không cần tạo biến quản lý số lượng phần tử riêng.
- Thực hiện các thao tác, thêm, xóa nhanh bằng hàm có sẵn
- Cho phép sử dụng chỉ số âm (chẳng hạn như A[-6], A[-5],…)

Nhược điểm

- Không sử dụng được tính năng debug
- Nếu không quản lý tốt bộ nhớ dễ xảy ra lỗi runtime
- Tốc độ xử lý chậm hơn mảng tĩnh 1 ít.
- Phải nhớ nhiều cú pháp để sử dụng.

CẤU TRÚC DỮ LIỆU VECTOR

Bài tập ví dụ

Trong kì thi tuyển sinh vào lớp 10 trường Phổ Thông Năng Khiếu, để bảo mật điểm số của học sinh người ta chỉ lưu số báo danh và tổng điểm 4 môn văn, toán, anh và môn chuyên. Sau khi hoàn tất phần chấm bài, nhà trường có một danh sách gồm N học sinh tham gia thi, mỗi học sinh có 2 thông tin là số báo danh S và tổng điểm T. Để thực hiện việc xét tuyển, nhà trường cần thực hiện các yêu cầu sau:

- Xóa tất cả các học sinh có tổng điểm nhỏ hơn 20.

- Do sai sót trong khâu nhập điểm, nhà trường cần tìm và điều chỉnh tổng điểm của học sinh có số báo danh cụ thể.

- Chèn thêm thông tin của học sinh thi bổ sung vào danh sách tại vị trí đầu tiên.

- Xuất ra 10 học sinh có tổng điểm cao nhất theo thứ tự từ trên xuống để vinh danh.

Dữ liệu vào:

Mỗi dòng ghi 2 số S (1000 <= S <= 9999) là số báo danh và T(0 <= T <= 50) là tổng điểm của 1 học sinh cách nhau khoảng trắng.

Hai dòng cuối:

Dòng đầu ghi số báo danh và tổng điểm của học sinh bị sai sót.

Dòng kế tiếp ghi số báo danh và tổng điểm của học sinh thi bổ sung

Dữ liệu ra:

Ghi danh sách sau khi điều chỉnh và bổ sung thí sinh, mỗi học sinh trên 1 dòng gồm 2 giá trị là số báo danh và tổng điểm cách nhau khoảng trắng.

Cách một dòng trống ghi danh sách top 10 tổng điểm theo thứ tự giảm dần, mỗi học sinh trên 1 dòng gồm 2 giá trị là số báo danh và tổng điểm cách nhau khoảng trắng.

Giáo viên: Hoàng Đăng Quang - 0903781593

CẤU TRÚC DỮ LIỆU SET

Khái niệm

set là một cấu trúc dữ liệu chứa các phần tử không trùng nhau.

Các phần tử trong set được sắp xếp tăng dần

Được định nghĩa trong thư viện **<set.h>** tuy nhiên có thể sử dụng thư viện tổng quát **<bits/stdc++.h>**

Khai báo

set <tên kiểu> tên set;

Ví dụ

```
set <int> myset; // Khai báo set có thứ tự tăng dần
set <int, greater<int>> decset; // Khai báo set có thứ tự giảm dần
```

CÁC HÀM CƠ BẢN CỦA SET

Hàm cho biết số phần tử có trong set

```
8       //In ra số lượng phần tử đang có trong set
9       cout << id.size();
```

Hàm thêm phần tử vào set

```
 9      id.insert(2); // {1}
10      id.insert(1); // {1, 2} phần tử 1 được thêm vào sau phần tử 2 nhưng được tự động sắp xếp
11      id.insert(1); // {1, 2} phần tử 1 đã có trong set sẽ không được thêm vào
12      id.insert(3); // {1, 2, 3}
13      id.insert(2); // {1, 2, 3} phần tử 2 đã có trong set sẽ không được thêm vào
```

Hàm kiểm tra set có rỗng không, nếu rỗng thì trả về true, ngược lại là false

```
15      if (id.empty()) cout << "Set rong";
16      else cout << "Set co phan tu";
```

CÁC HÀM CƠ BẢN CỦA SET

Hàm kiểm tra phần tử x có tồn tại trong set hay không

```
myset.count(x); // Trả về 1 nếu x có tồn tại trong set, 0 nếu không có
myset.find(x); // Trả về 1 nếu x có tồn tại trong set, 0 nếu không có
```

Hàm xóa một phần tử trong set

```
int x = 10;
myset.erase(x); // Xóa phần tử có giá trị 10 trong set
```

Hàm xóa tất cả các phần tử trong set

```
18      id.clear();
```

DUYỆT SET

Duyệt set bằng range based for loop kết hợp với auto

```
for (auto x: myset)
    cout << x << " ";
```

Duyệt set bằng iterator

```
for (auto it = myset.begin(); it != myset.end(); ++it)
    cout << *it << " ";
```

Duyệt ngược set bằng iterator

```
for (auto it = myset.rbegin(); it != myset.rend(); --it)
    cout << *it << " ";
```

CẤU TRÚC DỮ LIỆU SET

set đặc biệt phù hợp với những bài toán liên quan tới việc loại bỏ giá trị trùng nhau hoặc tìm kiếm nhanh.

Ngoài ra cấu trúc dữ liệu set còn có 2 dạng

multiset: Các phần tử cũng được tự động sắp xếp tăng dần như set nhưng cho phép trùng nhau.

unordered_set: Các phần tử chỉ tồn tại duy nhất như set nhưng không được tự động sắp xếp.

Tùy theo nhu cầu sử dụng mà ta chọn cấu trúc **set** phù hợp

CẤU TRÚC DỮ LIỆU SET

Bài tập ví dụ

Đếm phần tử khác biệt trong mảng, in ra các phần tử không trùng lặp.

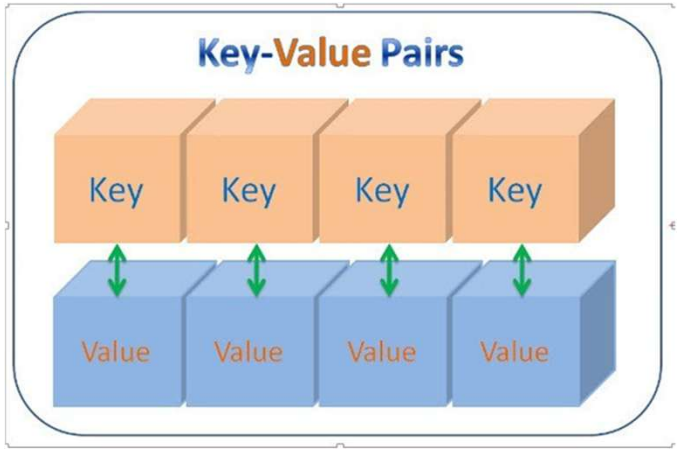
Tạo tập hợp hợp nhất và giao nhau.

CẤU TRÚC DỮ LIỆU MAP

map là một cấu trúc dữ liệu lưu trữ dữ liệu tương tự như một từ điển, mỗi phần tử trong **map** là một cặp **key - value**.

Ta có thể hình dung **map** là mở rộng của **set** nhưng thêm thành phần **value**. Lúc này mỗi phần tử trong **map** là 1 **pair**

Ví dụ: danh bạ {tên – số điện thoại}, từ điển {từ - nghĩa của từ} danh sách học sinh {mã học sinh – tên học sinh}.



Khai báo

map <tên kiểu key, tên kiểu value > tên map;

Ví dụ

```
map <int, string> dssinhvien;
map <int, string, greater<int>> decsv;
```

CÁC HÀM CƠ BẢN CỦA MAP

Hàm thêm phần tử vào map

```
dssinhvien.insert({1, "A"}); // Thêm sinh viên có key 1 với value A vào map
dssinhvien.insert({1, "B"}); // Không thêm được do trùng key
dssinhvien.insert(make_pair(2, "A")); // Sử dụng make_pair để thêm sinh viên có key 2 với value A vào map
dssinhvien[3] = "C"; // Thêm sinh viên có key 3 với value C vào map
dssinhvien[2] = "D"; // Sử value của sinh viên có key 2 thành D
```

Hàm cho biết số phần tử có trong map

```
38      cout << dssv.size();
```

Hàm kiểm tra map có rỗng không, nếu rỗng thì trả về true, ngược lại là false

```
40      if (dssv.empty()) cout << "map rong";
41      else cout << "map co phan tu";
```

CÁC HÀM CƠ BẢN CỦA MAP

Hàm xóa một phần tử trong map dựa trên giá trị key

```
43      dssv.erase(4); // Xóa sinh viên có mã là 4
```

Hàm xóa tất cả các phần tử trong map

```
45      dssv.clear();
```

DUYỆT MAP

Xuất giá trị value khi biết key

```
cout << dssinhvien[3];
```

Sử dụng range based for loop để duyệt map

```
for (auto sv: dssinhvien)
    cout << "Key: " << sv.first << " - value: " << sv.second << "\n";
```

Duyệt map kết hợp thay đổi giá trị value

```
// Duyệt danh sách sinh viên, thêm "CTIN - " vào đầu họ tên mỗi sinh viên
for (auto &sv: dssinhvien)
    sv.second =+ "CTIN - ";
```

Duyệt map bằng iterator

```
for (auto it = dssinhvien.begin(); it != dssinhvien.end(); ++it)
    cout << "Key: " << it->first << " - value: " <<it->second << "\n";
```

CẤU TRÚC DỮ LIỆU MAP

Các tính chất quan trọng của map:

- Các key trong map là riêng biệt, không có 2 key trùng nhau
- Map duy trì thứ tự các phần tử theo giá trị key tăng dần
- Map tìm kiếm giá trị key với độ phức tạp O(logN)
- Map không hỗ trợ truy cập thông qua chỉ số như mảng hay vector, string.
- Map cũng có dạnh multimap và unordered_map như set

Ứng dụng của map:

- Cho một danh sách các số điện thoại kèm theo tên của chủ thuê bao đó. Yêu cầu đầu vào là một số điện thoại (key), hãy đưa ra tên của chủ thuê bao (value)
- Cho danh sách thể hiện lịch sử đi muộn của các nhân viên một công ty nào đó. Hãy tìm xem nhân viên (key) nào có số lần đi muộn (value) nhiều nhất?
- Cho một danh sách các IP kèm theo các domain. Hãy trả ra ip (value) tương ứng domain (key) hoặc ngược lại trong thời gian nhanh nhất?

CẤU TRÚC DỮ LIỆU MAP

Bài tập ví dụ

Xác định số lượng phần tử trùng nhau

Gom các nhóm từ đảo kí tự với nhau.