**Cấu trúc dữ liệu – Data structure**

**Tổng quan**

1.Data structure

- vector, queue, stack, list, hash, tree.

2.Advance structure

- map, unordered\_map, set, multilset, pair, priority\_queue, deque, heap,…

3. algorithm

- sort

-search

**Data structure**

**A.Vector**

**1, Định nghĩa**

* Nó là 1 mảng động với khả năng tự thay đổi kích thước khi thêm hoặc chèn vào phần tử vào trong mảng.
* Dùng khi ko muốn quản lý mảng động qua con trỏ , hay muốn liên tục resize lại kích thước của mảng
* Chỉ có thể thêm hoặc xóa vào cuối mảng

**2, Thao tác**

2.a Khai báo vector

Vd:

vector<int> v1 // vector sẽ trống

vector<double> v2(5, 3.4)

// vector có 5 phần tử mang giá trị 3.4

char x[] = "hanoimuathu";

vector<char> v3(x,x+5);

// vector chứa từ phần tử x đến x+5

2.b chèn, thêm, sửa vector

*2.b.1 chèn vào 1 vị trí*

BiếnVector.insert(vị trí muốn chèn, giá trị muốn chèn);

Vd : v2.insert(v2.begin(),8.2) // chèn vào vị trí đầu tiên giá trị 8.2 -> phần tử trong mảng hiện tại tăng lên 6;

*2.b.2 thêm vào cuối mảng vector*

BiếnVector.push\_back(giá trị muốn thêm);

Vd : v2.push\_back(10) // thêm giá trị 10 vào cuối mảng vector

*2.b.3 xóa phần tử cuối mảng*

BiếnVector.pop\_back()

*2.b.4 cài đăt lại kích thước của mảng*

BiếnVector.resize(số kích thước mảng muốn resize)

Vd v3.resize(2) // mảng chỉ còn 2 phần tử

v3.resize(20,a) // set lại kích thước của mảng là 20 và còn lại toàn a

2.c lặp các phần tử trong mảng vector

*2.c.a Lặp bình thường*

for(int i=0;i<v2.size();i++){

cout<<v2[i]<<” ”;

}

*2.c.b Vòng lặp foreach*

for(auto v:V3){

cout<<v<<” ”;

}

*2.c.c Vòng lặp suôi với begin() và end()*

for(auto it=v2.begin();it!=v2.end();it++){

cout<<\*it<<” “;

}

*2.c.d Vòng lặp ngược với rbegin() và rend()*

for(auto it=v2.rbegin();it!=v2.rend();it++){

cout<<\*it<<” “;

}

**B.LIST**

- Các thành phần không liên kết với nhau như vector chính vì vậy chúng ta không thể lấy thành phần cụ thể ở trong list

- List có thể thêm vào đầu và cuối, không giống như vector(vector chỉ thêm được vào cuối) -> insert và delete phần tử nhanh hơn vector

- Nhược điểm duyệt lâu và chậm hơn vì lúc nào cũng phải đi từ đầu list đến cuối list

1.push\_back();

2.push\_front();

3.pop\_back();

4.pop\_front();

5.reverse();

6.sort(l.begin(),l.end());

**C.Queue**

* Hoạt động theo nguyên lí first in first out (thằng nào vào trước thì thằng đấy ra trước)
* Đưa vào hàng đợi gọi là enqueue và lấy ra hàng đợi gọi là dequeue.

Q.empty() -> boolean

Q.size()

Q.swap(Q1) ->nó thay đổi 2 queue với nhau, với điều kiện phải cùng kiểu dữ liệu

Q.front() -> lấy ra phần tử thứ nhất

Q.back() -> lấy ra phần tử cuối cùng

Q.push() -> đưa vào queue 1 phần tử

Q.pop() -> xóa 1 phần tử cuối cùng

**D.Pair**

- pair được dùng để kết hợp với nhau mặc dù chúng khác kiểu giá trị với nhau

Khai báo

pair (data\_type1, data\_type2) Pair\_name;

pair (data\_type1, data\_type2) Pair\_name(value1,value2),..;

Thao tác

1. Đặt giá trị cho 1 pair
2. g2 = make\_pair(1, 'a');
3. g2 = {1, 'a'};

2. đổi 2 giá trị

pair1.swap(pair2) ;

3.sort pair

Pair<int,int> A[n+5]

Sort(A+1,A+n+1); // sắp xếp tăng dần theo first

**E. Stack**

**(Ngăn xếp)**

* Hoạt động theo nguyên lí  Last In First Out (LIFO)

1. Khai báo

stack<**int**> stack;

1. Thao tác
2. Empty()
3. Size()
4. Top() -> lấy ra phần tử ở đầu
5. Push()-> đưa phần tử vào
6. Pop() -> xóa phần tử ở đầu

***Tổng kết :***

***-*Ta có thể thấy Array thì khi lấy,thêm, sửa và tìm kiếm đều có độ phức tạp là O(n).**

**-Còn với Stack,queue,linkedlist khi lấy, thêm, bớt phần tử có độ phức tạp là O(1) còn tìm kiếm với độ phức tạp là O(n)**

* **Hashtable = sẽ có độ phức tạp khi tìm kiếm là O(1).**

**F. Map \* ưu tiền sử dụng map**

- Mỗi key đều có value riêng, trong map không thể có 2 key trùng nhau.

-Tìm kiếm có độ phức tạp là log(n) vì nó tìm kiếm theo cây nhị phân cân bằng.

- Lấy, thêm phần tử trong map có độ phức tạp là O(1).

1. Khai báo

Map<key, value> biến map{{key1,value1},…}

1. Thao tác

* K.find(Key cần tìm) == K.end() -> tìm từ đầu đến cuối nếu đầu bằng cuối tức là ko có phần tử nào được tìm thấy vào trả về giá trị true.

- empty() -> kiểm tra map có rỗng không

- Insert(pair<key,value>) ->

- erase(vị trí muốn xóa)

- bộ lặp it-> first : key, it->second: value