[LẬP TRÌNH C/C++]

CÁC KĨ THUẬT MẢNG 1 CHIỀU

Một số lưu ý:
// <=> comment ghi chú, giải thích
; <=> kết thúc câu lệnh, luôn luôn có nếu không sẽ bị lỗi.

- Nếu các bạn đang xem bài viết này của mình, thì mình tin rằng các bạn đã được tiếp xúc qua các khái niệm cơ bản và nền tảng của mảng 1 chiều. Các bạn cần phải nắm rõ các khái niệm đầu tiên của mảng: Mảng là gì ? Học mảng để làm gì ? Tại sao phải dùng kiến thức mảng ? Chỉ số của mảng 1 chiều ? Phần tử của mảng 1 chiều ?.....

- Bài viết này mình xin được giới thiệu đến các bạn các kĩ thuật cơ bản mà nền tảng của mảng 1 chiều:

+ Kĩ thuật Thêm 1 phần tử vào vị trí bất kì trong mảng 1 chiều.

+ Kĩ thuật Xóa 1 phần tử ở vị trí bất kì trong mảng 1 chiều.

+ Kĩ thuật Sắp xếp mảng 1 chiều.

+ Kĩ thuật Tìm kiếm trong mảng 1 chiều.(Tìm kiếm tuyến tính)

- Khi tôi làm 1 bài toán nào đó về lập trình thì tôi làm theo các bước như sau:

+ Đọc kĩ đề và ngồi đó nghiền ngẫm xem cái đề bài người ta yêu cầu mình làm cái gì – phải hiểu đề, để có thể xem xét hết các tình huống có thể xảy ra cho bài toán – đi đúng hướng yêu cầu mà bài toán đưa ra chứ không hoang mang mà hiểu sai đề.

+ Lấy giấy nháp ra và cho 3 4… cái **ví dụ mẫu** về bài toán đó – ngồi và tìm ra cái quy tắc chung của lời giải bài toán dựa vào các ví dụ đó.

- Ví dụ mẫu nghĩa là sao – tôi sẽ trình bày ngay sao đây để các bạn hình dung về " **các ví dụ mẫu** "

I. KĨ THUẬT THÊM 1 PHẦN TỬ X VÀO VỊ TRÍ BẤT KÌ TRONG MẢNG 1 CHIỀU

- Trong tất cả các bài toán đưa ra thì công việc đầu tiên mà chúng ta cần làm đó chính là suy nghĩ ra cách giải(thuật toán) cho bài toán đó. Thì kĩ thuật thêm 1 phần tử vào vị trí bất kì trong mảng cũng được xem như là 1 bài toán. Chúng ta cần phải đi tìm ra các bước giải cho bài toán này.

Đề bài toán như sau:

Nhập vào mảng 1 chiều các số nguyên. Hãy thêm 1 phần tử vào vị trí bất kì trong mảng.

Tôi sẽ cho các ví dụ mẫu để tìm ra quy tắc của thuật toán Thêm 1 phần tử vào vị trí bất kì trong mảng

VD_1:

Mảng ban đầu có 5 phần tử: 5 6 7 8 9

Chỉ số(vị trí):	0	1	2	3	4
Phần tử mảng:	5	6	7	8	9

Yêu cầu : Thêm phần tử x vào vị trí 2
* Mảng sau khi thêm phần tử x vào vị trí 2 *

Chỉ số(vị trí):	0	1	2	3	4	5
Phần tử mảng:	5	6	x	7	8	9

* SỰ KHÁC NHAU CỦA MẢNG BAN ĐẦU VÀ MẢNG SAU KHI THÊM *

==== **MẢNG BAN ĐẦU** ====

Chỉ số(vị trí):	0	1	2	3	4
Phần tử mảng:	5	6	7	8	9

==== **MẢNG SAU KHI THÊM** ====

Chỉ số(vị trí):	0	1	2	3	4	5
Phần tử mảng:	5	6	x	7	8	9

Nhận xét:

- Phần tử nằm ngay vị trí cần thêm(vị trí 2) và các phần tử nằm sau vị trí cần thêm(vị trí 2) sẽ bị dịch chuyển về phía sau. Cụ thể là các phần tử 7 8 9 dịch về sau 1 đơn vị chỉ số ban đầu. Rồi sau đó phần tử cần thêm x đó sẽ nằm ở vị trí 2

+ Ban đầu 7 nằm ở vị trí 2 – sau khi thêm: 7 nằm ở vị trí 3

+ Ban đầu 8 nằm ở vị trí 3 – sau khi thêm: 8 nằm ở vị trí 4

+ Ban đầu 9 nằm ở vị trí 4 – sau khi thêm: 9 nằm ở vị trí 5

VD_2:

Mảng ban đầu có 5 phần tử: 5 6 7 8 9

Chỉ số(vị trí):	0	1	2	3	4
Phần tử mảng:	5	6	7	8	9

Yêu cầu : Thêm phần tử x vào vị trí 3
* Mảng sau khi thêm phần tử x vào vị trí 3 *

Chỉ số(vị trí):	0	1	2	3	4	5
Phần tử mảng:	5	6	7	x	8	9

* SỰ KHÁC NHAU CỦA MẢNG BAN ĐẦU VÀ MẢNG SAU KHI THÊM *

==== **MẢNG BAN ĐẦU** ====

Chỉ số(vị trí):	0	1	2	3	4
Phần tử mảng:	5	6	7	8	9

==== **MẢNG SAU KHI THÊM** ====

Chỉ số(vị trí):	0	1	2	3	4	5
Phần tử mảng:	5	6	7	x	8	9

Nhận xét:

- Phần tử nằm ngay vị trí cần thêm(vị trí 3) và các phần tử nằm sau vị trí cần thêm(vị trí 3) sẽ bị dịch chuyển về phía sau. Cụ thể là các phần tử 8 9 dịch về sau 1 đơn vị chỉ số ban đầu. Rồi sau đó phần tử cần thêm x đó sẽ nằm ở vị trí 3

+ Ban đầu 8 nằm ở vị trí 3 – sau khi thêm: 8 nằm ở vị trí 4

+ Ban đầu 9 nằm ở vị trí 4 – sau khi thêm: 9 nằm ở vị trí 5

========== TÌM THUẬT TOÁN SAU KHI XEM 2 VÍ DỤ MẪU ==========

==== **MẢNG BAN ĐẦU** ====

Chỉ số(vị trí):	0	1	2	3	4
Phần tử mảng:	5	6	7	8	9

Yêu cầu : Thêm phần tử x vào vị trí 2
==== **MẢNG SAU KHI THÊM** ====

Chỉ số(vị trí):	0	1	2	3	4	5
Phần tử mảng:	5	6	x	7	8	9

Số lượng phần tử mảng: n = 5

arr[5] = arr[4]

arr[4] = arr[3]

arr[3] = arr[2]

arr[2] = x

==========

arr[n] = arr[n-1]

arr[n-1] = arr[n-2]

arr[n-2] = arr[n-3]

arr[2] = x

========= THUẬT TOÁN THÊM 1 PHẦN TỬ X VÀO VỊ TRÍ BẤT KÌ TRONG MẢNG 1 CHIỀU =========

n-1: giá trị khởi tạo

vt: điều kiện lặp lặp đến vt(vị trí cần xóa) <=> Chỉ có phần tử nằm ngay vị trí cần thêm và các phần tử nằm phía sau vị trí cần thêm - mới bị dịch chuyển về phía sau 1 đơn vị chỉ số.

```
for(int i = n - 1; i >= vt; i--)
{
        a[i + 1] = a[i];
}
a[vt] = x;  // thêm phần tử x vào vị trí vt
n++; // sau khi thêm thì số lượng phần tử mảng sẽ tăng lên 1 đơn vị
```

Nhận xét:

- Chỉ có phần tử nằm ngay vị trí cần thêm và các phần tử nằm phía sau vị trí cần thêm - mới bị dịch chuyển về phía sau 1 đơn vị chỉ số.

- Vị trí cần thêm có thể nằm trong đoạn [0, n]

- Sau khi thêm thì số lượng phần tử mảng n tăng lên 1 đơn vị

- Để có thể hiểu rõ hơn về thuật toán thì các bạn hãy cho nhiều ví dụ mẫu ra và áp dụng vào thuật toán của tôi thì các bạn sẽ hiểu rõ hơn về cách trình bày thuật toán bên trên.

II. KĨ THUẬT XÓA 1 PHẦN TỬ TẠI VỊ TRÍ BẤT KÌ TRONG MẢNG 1 CHIỀU

Đề bài toán như sau:

Nhập vào mảng 1 chiều các số nguyên. Hãy xóa 1 phần tử tại vị trí bất kì trong mảng.

Tôi sẽ cho các ví dụ mẫu để tìm ra quy tắc của thuật toán Xóa 1 phần tử tại vị trí bất kì trong mảng

VD_1:

Mảng ban đầu có 5 phần tử: 5 6 7 8 9

Chỉ số(vị trí):	0	1	2	3	4
Phần tử mảng:	5	6	7	8	9

Yêu cầu : Xóa phần tử tại vị trí 2 <=> Xóa số 7
* Mảng sau khi xóa phần tử tại vị trí 2 *

Chỉ số(vị trí):	0	1	2	3
Phần tử mảng:	5	6	8	9

* SỰ KHÁC NHAU CỦA MẢNG BAN ĐẦU VÀ MẢNG SAU KHI XÓA *

==== **MẢNG BAN ĐẦU** ====

Chỉ số(vị trí):	0	1	2	3	4
Phần tử mảng:	5	6	7	8	9

==== **MẢNG SAU KHI XÓA** ====
Chỉ số(vị trí): 0 1 2 3
Phần tử mảng: 5 6 8 9

Nhận xét:

- Phần tử nằm ngay vị trí cần xóa(vị trí 2) sẽ bị mất đi(bản chất là gán đè giá trị 8 lên), các phần tử nằm sau vị trí cần xóa(vị trí 2) sẽ bị dịch chuyển về phía trước. Cụ thể là các phần tử 8 9 dịch về trước 1 đơn vị chỉ số ban đầu.

+ Ban đầu 7 nằm ở vị trí 2 – sau khi xóa: 7 bị giá trị 8 đè lên

+ Ban đầu 8 nằm ở vị trí 3 – sau khi xóa: 8 nằm ở vị trí 2

+ Ban đầu 9 nằm ở vị trí 4 – sau khi xóa: 9 nằm ở vị trí 3

VD_2:

Mảng ban đầu có 5 phần tử: 5 6 7 8 9

Chỉ số(vị trí): 0 1 2 3 4
Phần tử mảng: 5 6 7 8 9

Yêu cầu : Xóa phần tử tại vị trí 1 <=> Xóa số 6
* Mảng sau khi xóa phần tử tại vị trí 1 *

Chỉ số(vị trí): 0 1 2 3
Phần tử mảng: 5 7 8 9

* SỰ KHÁC NHAU CỦA MẢNG BAN ĐẦU VÀ MẢNG SAU KHI XÓA *

==== **MẢNG BAN ĐẦU** ====
Chỉ số(vị trí): 0 1 2 3 4
Phần tử mảng: 5 6 7 8 9

==== **MẢNG SAU KHI XÓA** ====
Chỉ số(vị trí): 0 1 2 3
Phần tử mảng: 5 7 8 9

Nhận xét:

- Phần tử nằm ngay vị trí cần xóa(vị trí 1) sẽ bị mất đi(bản chất là gán đè giá trị 7 lên), các phần tử nằm sau vị trí cần xóa(vị trí 1) sẽ bị dịch chuyển về phía trước. Cụ thể là các phần tử 7 8 9 dịch về trước 1 đơn vị chỉ số ban đầu.

+ Ban đầu 6 nằm ở vị trí 1 – sau khi xóa: 6 bị giá trị 7 đè lên

+ Ban đầu 7 nằm ở vị trí 2 – sau khi xóa: 7 nằm ở vị trí 1

+ Ban đầu 8 nằm ở vị trí 3 – sau khi xóa: 8 nằm ở vị trí 2

+ Ban đầu 9 nằm ở vị trí 4 – sau khi xóa: 9 nằm ở vị trí 3

========== TÌM THUẬT TOÁN SAU KHI XEM 2 VÍ DỤ MẪU ==========

==== **MẢNG BAN ĐẦU** ====
Chỉ số(vị trí): 0 1 2 3 4
Phần tử mảng: 5 6 7 8 9

Yêu cầu : Xóa phần tử tại vị trí 2 <=> Xóa số 7
==== **MẢNG SAU KHI XÓA** ====
Chỉ số(vị trí): 0 1 2 3
Phần tử mảng: 5 6 8 9

Số lượng phần tử mảng: n = 5

arr[2] = arr[3]

arr[3] = arr[4]

========= THUẬT TOÁN XÓA 1 PHẦN TỬ TẠI VỊ TRÍ BẤT KÌ TRONG MẢNG 1 CHIỀU =========

vt + 1: giá trị khởi tạo <==> dịch chuyển các giá trị ở phía sau vt lên 1 đơn vị

```
for(int i = vt + 1; i < n; i++)
{
        a[i-1] = a[i];
}
n--;  // sau khi xóa số lượng phần tử n giảm xuống 1 đơn vị
```

Nhận xét:

- Phần tử nằm ngay vị trí cần xóa sẽ mất đi(bản chất là bị giá trị phía sau nó đè lên bằng toán tử gán trong thuật toán).

- Chỉ có các phần tử nằm phía sau vị trí cần xóa - mới bị dịch chuyển về phía trước 1 đơn vị chỉ số.

- Vị trí cần xóa phải nằm trong đoạn [0, n-1]

- Sau khi xóa thì số lượng phần tử mảng n giảm 1 đơn vị

- Để có thể hiểu rõ hơn về thuật toán thì các bạn hãy cho nhiều ví dụ mẫu ra và áp dụng vào thuật toán của tôi thì các bạn sẽ hiểu rõ hơn về cách trình bày thuật toán bên trên.

III. KĨ THUẬT SẮP XẾP TRONG MẢNG 1 CHIỀU

- Khi xây dựng 1 thuật toán sắp xếp cần chú ý tìm cách giảm thiểu những **phép so sánh và đổi chỗ** không cần thiết để tăng hiệu quả của thuật toán. Sau đây là một số phương pháp sắp xếp thông dụng:

+ **Chọn trực tiếp – Selection Sort**

+ Chèn trực tiếp – Insertion Sort

+ Đổi chỗ trực tiếp – Interchange Sort

+ Nổi bọt – Bubble Sort

+ Shaker Sort

+ Chèn nhị phân – Binary Insertion Sort

+ Shell Sort

+ Heap Sort

+ Quick Sort

+ Merge Sort

+ Radix Sort

- Dưới góc độ thuật toán thì những phương pháp được trình bày trong phần này có chi phí là **O(n^2)**, với n là kích thước của mảng dữ liệu. Đây được xem là nhóm giải thuật sắp xếp có tốc độ thực thi kém hiệu quả nhất chỉ nên áp dụng khi lượng dữ liệu đầu vào là nhỏ.

- Sau đây tôi xin giới thiệu đến các bạn 2 thuật toán sắp xếp dễ hiểu, dễ cài đặt mà thường được sử dụng nhất:

1. Phương pháp sắp xếp chọn trực tiếp – Selection Sort

Ý tưởng thuật toán

- Phương pháp này duyệt để tìm phần tử nhỏ nhất(đang nằm đâu đó trong mảng), lấy ra và đặt vào vị trí đầu tiên của mảng(có chỉ số là 0), tương tự như vậy, phần tử nhỏ thứ 2 trong mảng cũng sẽ được duyệt chọn ra và đặt vào vị trí thứ hai(có chỉ số là 1),….Cứ như thế cho đến khi toàn bộ mảng trở thành có thứ tự(trường hợp này là có thứ tự tăng dần).

Ví dụ: Cho mảng arr: 5 3 2 1 4

5 3 2 1 4

i=0

1 3 2 5 4

i=1

1 2 3 5 4

i=2

1 2 3 5 4

i=3

1 2 3 4 5

========== THUẬT TOÁN SELECTION SORT ==========

```
void Hoan_Vi(int &x, int &y)
{
	int temp = x;
	x = y;
	y = temp;
}

void SelectionSort(int arr[], int n)
{
	int min; // chỉ số của phần tử có giá trị nhỏ nhất trong mảng hiện tại
	for(int i = 0; i < n - 1; i++)
	{
		min = i; // giả sử phần tử tại vị trí thứ i hiện tại có giá trị nhỏ nhất
		for(int j = i + 1; j < n; j++)
		{
			if(arr[min] > arr[j])
			{
				min = j; // ghi nhận vị trí của phần tử hiện đang nhỏ nhất
			}
		}
		Hoan_Vi(arr[min], arr[i]); // đổi chỗ vị trí i hiện tại và phần tử có giá trị nhỏ nhất tại chỉ số min
	}
}
```

2. Phương pháp sắp xếp đổi chỗ trực tiếp – Interchange Sort

Ý tưởng thuật toán

- Phương pháp này liên tục so sánh và đổi chỗ các cặp phần tử đứng kề nhau cho đến khi toàn bộ dãy được sắp. Nếu a[i] được theo sau bởi a[i+1] và a[i] > a[i+1] thì phải hoán đổi giá trị của 2 phần tử này. (đây là cách sắp xếp tăng dần các phần tử từ trái sang phải).
- Các phần tử có giá trị lớn hơn sẽ được đẩy về phía cuối của mảng, còn phần tử có giá trị nhỏ hơn sẽ được đẩy về phía đầu của mảng.

Ví dụ: Cho mảng arr: 5 3 2 1 4

Yêu cầu sắp xếp mảng arr tăng dần các phần tử

5 3 2 1 4

i=0 j=1

- Vì arr[i] > arr[j], nên phần tử arr[i] sẽ hoán đổi cho arr[j]

3 5 2 1 4

i=0 j=2

- Vì arr[i] > arr[j], nên phần tử arr[i] sẽ hoán đổi cho arr[j]

2 5 3 1 4
i=0 j=3

- Vì arr[i] > arr[j], nên phần tử arr[i] sẽ hoán đổi cho arr[j]

1 5 3 2 4
i=1 j=2

- Vì arr[i] > arr[j], nên phần tử arr[i] sẽ hoán đổi cho arr[j]

1 3 5 2 4
i=1 j=3

- Vì arr[i] > arr[j], nên phần tử arr[i] sẽ hoán đổi cho arr[j]

1 2 5 3 4
i=2 j=3

- Vì arr[i] > arr[j], nên phần tử arr[i] sẽ hoán đổi cho arr[j]

1 2 3 5 4
i=3 j=4

- Vì arr[i] > arr[j], nên phần tử arr[i] sẽ hoán đổi cho arr[j]

1 2 3 4 5

- Kết thúc quá trình sắp xếp mảng, lúc này mảng đã tăng dần theo thuật toán

========== THUẬT TOÁN INTERCHANGE SORT ==========

```
// hàm có chức năng hoán vị 2 vị trí của 2 phần tử cho nhau
void Hoan_Vi(int &x, int &y)
{
	int temp = x;
	x = y;
	y = temp;
}

void InterchangeSort(int arr[], int n)
{
	// vòng lặp duyệt từ 0 đến phần tử kế cuối(i < n - 1)
	for(int i = 0; i < n - 1; i++)
	{
		// vòng lặp bắt đầu duyệt từ vị trí sau vị trí i 1 đơn vị <=> j = i + 1 đến cuối mảng
		for(int j = i + 1; j < n; j++)
		{
			// điều kiện có tác dụng đẩy các phần tử lớn hơn về phía cuối mảng <=> bên phải
			if(arr[i] > arr[j])
			{
				Hoan_Vi(arr[i], arr[j]); // đổi chỗ phần tử lớn hơn về bên phải, nhỏ hơn về bên trái
			}
		}
	}
}
```

Nhận xét:

- Có rất nhiều thuật toán sắp xếp khác nhau, nhưng thường thì chúng ta chỉ thao tác trên 1 hoặc 2 thuật toán sắp xếp, chúng ta lựa chọn thuật toán nào mà khi ta cài đặt code dễ hiểu, dễ hình dung.

- Khi gặp bài toán liên quan đến vấn đề sắp xếp thì tôi thường hay dùng thuật toán sắp xếp đổi chỗ trực tiếp – InterchangeSort, bởi vì đây là thuật toán dễ thao tác và dễ cài đặt.

IV. KĨ THUẬT TÌM KIẾM TRONG MẢNG 1 CHIỀU(TÌM KIẾM TUYẾN TÍNH – TÌM KIẾM TUẦN TỰ)

- Kĩ thuật tìm kiếm là 1 kĩ thuật quan trọng và ứng dụng trong nhiều bài toán thực tế và cụ thể như:

+ Tìm kiếm 1 sinh viên nào đó có mã sinh viên XXX trong lớp, trường học.

+ Tìm kiếm 1 quyển sách nào đó trong thư viện thông qua mã sách.

+ Tìm kiếm 1 món hàng hóa nào đó theo mã hàng hóa.

+ Tìm kiếm 1 công dân có mã CMND nào đó.

bla bla……

- Có 2 giải thuật thường được áp dụng để tìm kiếm dữ liệu là **Tìm kiếm tuyến tính và Tìm kiếm nhị phân**. Nhưng trong bài viết này thì tôi chỉ minh họa thuật toán **tìm kiếm tuyến tính** hay còn gọi là **tìm kiếm tuần tự** trong mảng 1 chiều - để giới thiệu đến các bạn, nhằm giúp các bạn nắm được bản chất và ý nghĩa của khái niệm Tìm kiếm.

Bài toán đặt ra

Nhập vào mảng 1 chiều các số nguyên. Sau đó nhập vào 1 giá trị x từ bàn phím, kiểm tra xem giá trị x đó có tồn tại trong mảng hay không ?

Thuật toán

+ Duyệt mảng từ vị trí 0 đến (số lượng phần tử mảng – 1). Duyệt mảng 1 cách **tuyến tính** hay còn gọi là duyệt **tuần tự**.

+ Trong quá trình duyệt từ đầu mảng đến cuối mảng đó, mỗi lần duyệt thì sẽ kiểm tra xem phần tử của mảng đó có bằng với x hay không, nếu bằng thì kết luận ngay lập tức là phần tử x đó có tồn tại trong mảng.

+ Còn nếu như ta duyệt từ đầu mảng đến cuối mảng mà vẫn chưa phát hiện ra phần tử nào trong mảng bằng với x thì ta kết luận là không tìm thấy phần tử x.

========== THUẬT TOÁN ==========

```
bool Check = false; // kĩ thuật đặt cờ hiệu <=> đầu tiên ta giả sử là phần tử x không tồn tại trong mảng

// vòng lặp duyệt từ đầu mảng đến cuối mảng(duyệt 1 cách tuần tự) để kiểm tra xem phần từ x có tồn tại trong mảng hay không
for(int i = 0; i < n; i++)
{
	// nếu như điều kiện này không xảy ra thì phần tử x không tồn tại trong mảng <=> giả sử ban đầu Check = false là đúng
	if(arr[i] == x)
	{
		Check = true; // giả sử ban đầu đã sai ==> cập nhật lại biến Check là true <=> phần tử x có tồn tại trong mảng
		break; // thoát ra khỏi vòng lặp và dừng lại, vì đã kiểm tra và thấy x có tồn tại nên không cần đi kiểm tra tiếp
	}
}

if(Check == true)
{
	cout << "\nPhan tu " << x << " co ton tai trong mang";
}
else
{
	cout << "\nPhan tu " << x << " khong co ton tai trong mang";
}
```

---------- ***END*** ----------

- Đây là tài liệu dùng để học và tham khảo.

- Trong quá trình biên soạn bộ tài liệu có:

+ Tham khảo 1 số nguồn tài liệu(google search, ebook, sách chuyên ngành…).

+ Kiến thức – kinh nghiệm cá nhân của tôi.

- Có thể còn 1 số lỗi sai sót – sẽ luôn cập nhật phiên bản mới.

- Mong nhận được sự đóng góp và phản hồi tích cực từ các bạn đọc. Chân thành cảm ơn.