Bài: Stack - Cấu trúc dữ liệu ngăn xếp

Xem bài học trên website để ủng hộ Kteam: Stack - Cấu trúc dữ liệu ngăn xếp

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Trong bài này, Kteam sẽ giới thiệu đến bạn một cấu trúc dữ liệu được sử dụng rất nhiều trong các bài toán – đó chính là **ngăn xếp - Stack**. Cùng tìm hiểu xem **Stack** là gì và cách sử dụng như thế nào nhé!

Nội dung

Để có thể hiểu được bài học này một cách tốt nhất, các bạn nên có kiến thức cơ bản về các phần:

- Biến, kiểu dữ liệu, toán tử trong C++
- Câu điều kiện, vòng lặp, hàm trong C++
- Mảng trong C++
- Các kiến thức cần thiết để theo dõi khóa học
- Nhập, xuất dữ liệu qua file trong C++
- Và đừng quên Cài đặt môi trường CodeBlocks để thực hành theo hướng dẫn

Trong bài học này chúng ta sẽ tìm hiểu về:

- Khái niệm stack
- Cách cài đặt stack thủ công
- Cách sử dụng stack có trong C++

Bài toán đặt ra

Ta có một bài toán như sau:

Cho một dãy số gồm n số nguyên dương a_i $(n \leq 10^6,\ a_i \leq 10^9)$. Với mỗi vị trí i, hãy in ra vị trí j gần nhất về phía bên trái thoả mãn $a_i < a_j$. Nếu như không có phần tử nào thoả mãn in ra -1.

Ví dụ:

Input	Output
7 2 1 3 2 8 5 7	-1 1 -1 3 -1 5 5

Phương hướng suy nghĩ

Thông thường, khi giải quyết một bài toán, ta sẽ tiếp cận theo các bước sau:

- Đề bài bảo gì ta sẽ đi làm cái đó, cố gắng suy nghĩ ra một thuật toán đơn giản nhất đáp ứng được yêu cầu đề bài mà không cần quan tâm đến thời gian hay bộ nhớ (Cách này được gọi là "chạy trâu").
- Từ thuật toán ban đầu, tìm kiếm các cấu trúc dữ liệu để tối ưu thời gian hoặc đưa ra các tính chất, nhận xét để rút ra được một số đặc điểm của bài toán.
- Từ những nhận xét trên, tìm ra lời giải tối ưu của bài toán

Lời giải ban đầu

Theo như để bài yêu cầu thì ta sẽ cần tìm phần tử gần nhất bên trái mà lớn hơn phần tử hiện tại. Do đó, theo cách suy nghĩ thông thường, với mỗi vị trí, ta sẽ duyệt từ phải qua trái từ vị trí hiện tại, phần tử đầu tiên mà lớn hơn phần tử hiện tại chính là vị trí cần tìm.

Ta sẽ thực hiện thuật toán như sau: Với mỗi vị trí i, ta sẽ duyệt ngược tất cả các vị trí j theo thứ tự từ i – 1 đến 0 (Mảng trong C++ bắt đầu từ 0). Nếu gặp được một vị trí j mà $a_j > a_i$ thì ta sẽ in ra kết quả và huỷ bỏ vòng lặp. Nếu như duyệt đến 0 mà vẫn không thể tìm ra j thoả mãn thì in ra -1.

Một chút chú ý cho các bạn trước khi đọc code của mình. Trong suốt khoá học này, mình sẽ sử dụng đọc và ghi dữ liệu lần lượt ra hai file text "CTDL.inp" và "CTDL.out". Do đó, nếu các bạn copy code mình và muốn chạy trên máy bản thân thì cần tạo ra hai file này ở dạng text ở cùng thư mực chứa code của các bạn.

Cách này sẽ được code như sau:

C++:

```
#include<bits/stdc++.h>
using namespace std;

const int MaxN = 1e6 + 1;

int n, a[MaxN];

int main(){
    freopen("CTDL.inp","r",stdin);
    freopen("CTDL.out","w",stdout);
    cin >> n;
    for(int i = 0 ; i < n ; ++i) cin >> a[i];
    for(int i = 0 ; i < n ; ++i){
        int pos = -1; // Khởi tạo giá trị kết quả ban đầu
        for(int j = i - 1 ; j >= 0 ; --j)
        if(a[j] > a[i]){
            // Nếu tìm thấy vị trí j thoả mãn thì ghi lại và dừng vòng lặp
            pos = j;
            break;
        }
        // Kiểm tra xem có tồn tại vị trí j không, nếu có thì giá trị pos sẽ nằm trong đoạn [0, n – 1]
        // In ra phải cộng 1 do chỉ số mảng trong C++ bắt đầu từ 0
        if(pos >= 0) cout << pos + 1 << " ";
        else cout << -1 << " ";
    }

    return 0;
}
```

Sau khi chạy, ta thấy cách này là một cách hoàn toàn đúng. Tuy nhiên ta thấy chương trình sử dụng hai vòng lặp lồng nhau. Trong trường hợp xấu nhất, vòng lặp j sẽ phải lặp lại n-1 lần. Do đó, độ phức tạp của cách này lên tới $O(n^2)$. Nếu trong các kì thi giới hạn thời gian là 1s cho mỗi chương trình và $n = 10^6$ thì cách này là không đủ tốt.

Nhận xét

Hãy xét một dãy như sau: [5, 1, 2, 3, 4]

Giả xử ta đang ở vị trí i = 4 *(a_i=4)*, theo như code ban đầu của chúng ta thì ta sẽ phải xét cả các vị trí j = 1, 2. Ta thấy các giá trị tại j = 1, 2 còn nhỏ hơn tại j = 3 nên nếu j = 3 không thoả mãn thì xét qua các vị trí j = 1, 2 là hoàn toàn vô nghĩa. Do đó, ta nghĩ đến việc làm sao để chỉ xét các vị trí có khả năng thỏa mãn chứ không xét toàn bộ.

Cách giải cải tiến

Từ nhận xét trên, ta sẽ có một tập b gọi là "ứng cử viên" thể hiện cho vị trí trong dãy a. Tập này sẽ có tính chất sau:

> $b_i<b_j$ và $a_{bi}>a_{bj}$ ($\forall i<j$).

Ví dụ: Dãy b là [2, 4, 6] thì $a_2 > a_4 > a_6$.

Khi đó, ta có một thuật toán như sau:

- Với mỗi vị trí i, ta sẽ loại bỏ tất cả các ứng cử viên j ở cuối tập mà $a_j < a_i$.
- Xét tập ứng viên, ứng viên cuối cùng trong tập chính là vị trí cần tìm. Nếu như tập rỗng có nghĩa là không tồn tại vị trí thoả mãn.
- Đẩy vị trí i vào cuối tập.

Hãy hình dung luồng chạy của chương trình trong ví dụ trên:

- Đầu tiên, xét i = 0. Hiện tại tập ứng viên đang rỗng. Do đó sẽ không có số nào lớn hơn a_0 mà đứng trước nó. In ra -1 và đẩy i = 0 vào tập ứng cử viên
- Xét i = 1. Hiện tại, tập ứng viên đang có 1 đề cử là vị trí j = 0. Ta thấy đề cử thỏa mãn do $a_0 > a_1$. Do đó in ra đề cử và đẩy i = 1 vào tập ứng viên. Lúc này tập vẫn đáp ứng tính chất nêu trên.
- Xét i = 2. Hiện tại, tập ứng viên là [0, 1], đề cử cho vị trí i = 2 đang là vị trí j = 1. Sẽ có bạn đặt ra câu hỏi tại sao lại xét ứng cử viên cuối cùng. Lí do là do dãy b tăng dần nên vị trí ở cuối sẽ là vị trí gần i nhất về phía bên trái. Tuy nhiên, vị trí j này không đáp ứng yêu cầu. Đây là lúc mà ta cần chú ý. Do tất cả các vị trí i về sau đều tìm về vị trí gần nhất bên trái nên một khi tồn tại i sao cho $a_i > a_j$ thì vị trí j sẽ không bao giờ được lựa chọn. Thật vậy, nếu như có một vị trí k nhận vị trí j làm kết quả mà không phải là vị trí i thì có nghĩa là ($a_j > a_k$ và $a_k > a_i$) hay $a_j > a_i$ (Trái với giả thiết). Do đó loại bỏ vị trí j = 1 ra khỏi tập. Tương tự với j = 0. Lúc này tập ứng viên rỗng. In ra -1 và đẩy i = 2 vào tập.
- Cách thực hiện với các giá trị i = 3, 4, 5, 6 là tương tự

Như vậy, chương trình của chúng ta sẽ đòi hỏi một cấu trúc dữ liệu có khả năng đáp ứng các yêu cầu sau:

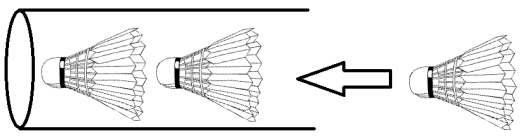
- Cho một phần tử vào cuối dãy
- Đẩy phần tử cuối dãy ra ngoài

Đó chính là lúc mà cấu trúc dữ liệu stack phát huy sức mạnh của mình. Bây giờ hãy cùng nhau tìm hiểu chi tiết về stack nhé.

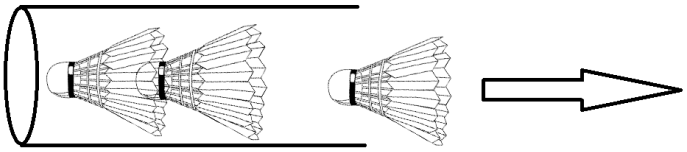
Khái niệm stack

Stack là một cấu trúc dữ liệu hoạt động theo nguyên tắc Last In First Out. Hiểu đơn giản là phần tử sẽ được thêm vào cuối stack và khi lấy ra ta cũng sẽ lấy phần tử cuối stack (phần tử được thêm vào gần nhất).

Một ví dụ trong thực tế của stack mà các bạn có thể dễ hình dung được đó chính là hộp cầu lông. Khi muốn cho cầu vào trong hộp ta sẽ cho cầu vào đáy hộp cầu và khi muốn lấy cầu ra thì ta sẽ lấy quả cầu gần nhất được cho vào.



Cho cầu vào đáy hộp



Lấy cầu ra khỏi hộp từ đáy

Một stack sẽ hỗ trợ các thao tác cơ bản sau:

- Thêm phần tử vào cuối stack
- Loại bỏ phần tử cuối ra khỏi stack
- Lấy giá trị cuối trong stack
- Lấy kích thước stack

Cách cài đặt stack thủ công

Cơ chế hoạt động của stack

Ta sẽ có mảng a giả làm stack và giá trị sz thể hiện kích thước của stack như sau:

Chỉ số	0	1	2	3	4	…
sz	*					
a						

Ban đầu, mảng a rỗng và sz = 0

Chèn phần tử cuối stack

Để chèn phần tử cuối stack thì ta chỉ cần gán phần tử ở vị trí sz (vị trí trống ở cuối) giá trị cần thêm vào rồi tăng giá trị sz thể hiện việc stack tăng kích thước.

Ví dụ như khi ta thêm 3 vào stack thì ta được

Chỉ số	0	1	2	3	4	…
sz		*				
a	3					

Sau đó nếu như ta thêm 5 vào stack thì ta được

Chỉ số	0	1	2	3	4	…
sz			*			
a	3	5				

Loại bỏ phần tử cuối stack

Nếu muốn bỏ phần tử cuối stack thì đơn giản là ta sẽ giảm giá trị sz đi 1. Khi này, nếu như chèn phần tử mới vào thì phần tử 5 tự nhiên sẽ bị loại bỏ.

Chỉ số	0	1	2	3	4	…
sz		*				
a	3	5				

Ở đây có chú ý quan trọng cho các bạn: Phải đảm bảo stack không rỗng (sz > 0). Nếu không, có thể sz sẽ có giá trị âm dẫn đến Runtime Error (mảng không có chỉ số âm).

Lấy giá trị cuối trong stack

Ta thấy giá trị cuối trong stack nằm ở vị trí sz – 1 nên đơn giản là trả về phần tử ở vị trí sz – 1

Ở đây có chú ý quan trọng cho các bạn: Phải đảm bảo stack không rỗng (sz > 0). Nếu không, có thể sz sẽ có giá trị âm dẫn đến Runtime Error (mảng không có chỉ số âm).

Lấy kích thước stack

Ta thấy kích thước chính là biến sz do đó chỉ cần trả về biến sz là được.

Code

C++:

```
struct CustomStack{
    int sz = 0; // Kích thước stack
    int a[int(1e6 + 1)]; // Mảng được giả làm stack với kích thước tối đa 1e6

    // Thêm phần tử vào stack
    void push(int element){
        a[sz] = element;
        sz++;
    }

    // Xoá phần tử khỏi stack
    void pop(){
        if(sz) sz--;
    }

    // Lấy giá trị cuối cùng trong stack
    int top(){
        if(sz) return a[sz - 1];
    }

    // Lấy kích thước stack
    int getSize(){
        return sz;
    }
};
```

Cách sử dụng stack có trong C++

Stack được cài đặt ở trên là tương đối ổn. Tuy nhiên, trong C++ đã được xây dựng sẵn stack với hiệu quả tốt nên việc xây dựng trên là không thật sự cần thiết trong đa phần các trường hợp.

Khai báo stack

Thông thường để thêm stack vào chương trình, chúng ta sẽ thêm thư viện như sau:

#include<stack>

Tuy nhiên, ở trong suốt khoá học này mình sẽ sử dụng header sau:

#include<bits/stdc++.h>

Header này sẽ giúp chúng ta thêm tất cả các thư viện về các cấu trúc dữ liệu mà chúng ta sẽ học trong khóa học này.

Ta sẽ khai báo stack như sau:

> **stack <{kiểu dữ liệu}> {tên stack};**

Ví dụ: stack<int> myStack;

Ngoài ra có thể khởi tạo stack với mảng giá trị cho trước. Tuy nhiên mình ít khi gặp phải trường hợp này trong thực tế. Các bạn có thể tìm hiểu thêm về các cách khởi tạo stack khác.

Các phương thức trong stack

Các phương thức cơ bản trong stack của C++:

- **push**: Thêm phần tử vào cuối stack
- **pop**: Loại bỏ phần tử cuối stack
- **top**: Trả về giá trị là phần tử cuối trong stack
- **size**: Trả về giá trị nguyên là số phần tử đang có trong stack
- **empty**: Trả về một giá trị bool, true nếu stack rỗng, false nếu stack không rỗng

Các phương thức trên sẽ đều mất độ phức tạp O(1).

Mình có một đoạn code demo về các phương thức cơ bản của stack như sau:

C++:

```
#include<bits/stdc++.h>
using namespace std;

stack<int> st;

int main(){
    // Thêm các phần tử vào stack
    st.push(1);
    st.push(3);
    st.push(5);
    // Hiện tại stack là [1, 3, 5]

    // In ra phần tử cuối cùng trong stack và kích thước stack
    cout << "Phan tu cuoi cung trong stack la:" << st.top() << endl;
    cout << "Kich thuoc hien tai cua stack la:" << st.size() << endl;

    // Loại bỏ 1 phần tử ra khỏi stack
    st.pop();
    cout << "Loai bo phan tu cuoi ra khoi stack" << endl;
    // Hiện tại stack là [1, 3]

    // Kiểm tra stack có rỗng không
    if(st.empty()) cout << "Stack rong" << endl;
    else cout << "Stack khong rong" << endl;

    // Sau khi loại bỏ 1 phần tử ra in ra phần tử cuối cùng trong stack và kích thước stack
    cout << "Phan tu cuoi cung trong stack la:" << st.top() << endl;
    cout << "Kich thuoc hien tai cua stack la:" << st.size() << endl;

    // Loại bỏ tất cả các phần tử ra khỏi stack
    while(st.size() > 0) st.pop();

    // Kiểm tra stack có rỗng không
    if(st.empty()) cout << "Stack rong" << endl;
    else cout << "Stack khong rong" << endl;
}
```

Khi chạy đoạn code trên ta thu được kết quả như sau:

```
Phan tu cuoi cung trong stack la:5
Kich thuoc hien tai cua stack la:3
Loai bo phan tu cuoi ra khoi stack
Stack khong rong
Phan tu cuoi cung trong stack la:3
Kich thuoc hien tai cua stack la:2
Stack rong
```

> Lưu ý: Các phương thức như pop, top nếu được gọi khi stack rỗng sẽ dẫn đến Runtime Error. Do đó, cần phải đảm bảo stack không rỗng trước khi gọi các phương thức này. Để kiểm tra stack có rỗng hay không thì các bạn có thể sử dụng phương thức **empty()** mà mình đã nêu ở trên.

Ứng dụng stack vào giải quyết bài toán ban đầu

Vậy sau khi biết về stack thì bài toán ban đầu sẽ được code như sau:

C++:

```
#include<bits/stdc++.h>
using namespace std;

const int MaxN = 1 + 1e6;

int n, a[MaxN];
stack<int> st;

int main(){
    freopen("CTDL.inp","r",stdin);
    freopen("CTDL.out","w",stdout);
    cin >> n;
    for(int i = 0 ; i < n ; ++i) cin >> a[i];
    for(int i = 0 ; i < n ; ++i){
        // Các bạn chú ý phương thức kiểm tra empty() luôn phải được đặt
        trước khi gọi top() hoặc pop()
        while(!st.empty() && a[st.top()] < a[i]) st.pop();
        if(!st.empty()) cout << st.top() + 1 << " ";
        else cout << "-1" << " ";
        st.push(i);
    }

    return 0;
}
```

Hãy chú ý vào độ phức tạp của thuật toán này: Ta thấy dù có vòng lặp while trong vòng lặp for tuy nhiên hai vòng lặp này hoàn toàn không phụ thuộc vào nhau (Khác với ở lời giải ban đầu). Các phần tử trong mảng a chỉ ra và vào stack tổng cộng tối đa là 2 lần đo đó trong toàn bộ chương trình, vòng lặp while sẽ chỉ thực hiện tối đa 2n thao tác O(1). Do đó thuật toán sẽ có độ phức tạp O(n). So với thuật toán ban đầu có độ phức tạp $O(n^2)$ thì đây là cải tiến đáng kể.

Về việc sử dụng stack có sẵn và stack tự xây dựng

Như mình có nói, stack trong C++ sẽ tốt hơn trong đa phần các trường hợp. Vậy có khi nào chúng ta sẽ cần stack tự xây dựng không? Câu trả lời là vẫn có. Vậy thì đó là khi nào? Các bạn có thể thấy, bản chất của stack tự xây dựng vẫn là mảng. Do đó, ta có thể truy cập ngẫu nhiên vào bất cứ phần tử nào, khác với stack trong C++ chỉ có thể lấy phần tử cuối cùng ra. Do đó, nếu các bạn vì lý do gì đó cần truy cập nhiều hơn 1 phần tử ở cuối thì nên dùng stack tự xây dựng.

Ngoài ra, trong C++ có một cấu trúc dữ liệu là vector. Các bạn hoàn toàn có thể đọc về vector và sử dụng nó như stack. vector sẽ là sự dung hòa tốt các điểm lợi giữa stack tự xây dựng (truy cập ngẫu nhiên) và stack có sẵn (tiện lợi, an toàn).

Kết luận

Qua bài này chúng ta đã nắm được khái niệm stack cũng như là cách cài đặt và áp dụng trong thực tế.

Bài sau chúng ta sẽ bắt đầu tìm hiểu về cấu trúc dữ liệu **Queue và Deque**.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".