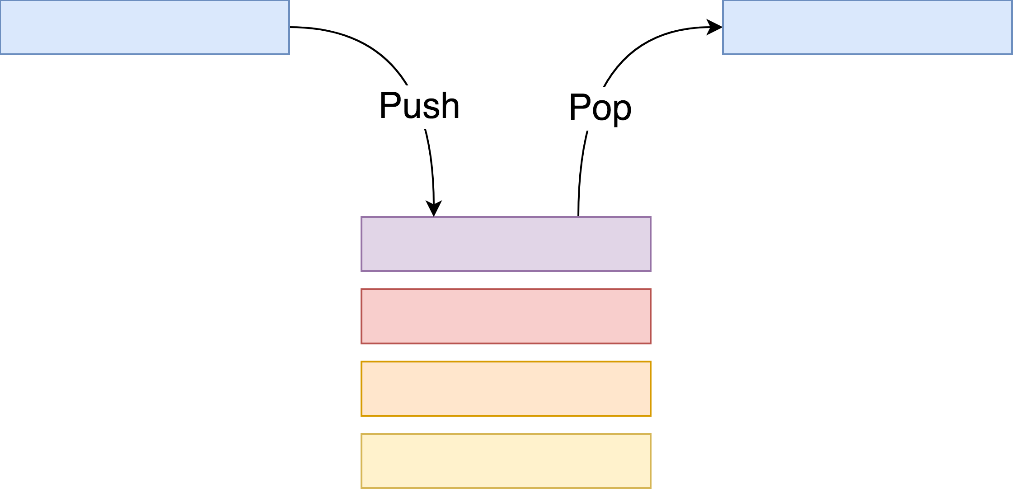
**TUẦN 5: NGĂN XẾP - STACK**

*Câu hỏi kì này*

*Tìm hiểu cách hoạt động của stack, các thao tác trên Stack.*

*Độ phức tạp của từng thao tác trên Stack.*

*Ý tưởng giải quyết Bài toán tìm max min trên đoạn tịnh tiến.*

1. **TÌM HIỂU VỀ STACK.**
2. **Khái niệm:**

* Stack là một cấu trúc dữ liệu container (thùng chứa) có thể chứa đa dạng các phần tử với cùng 1 kiểu dữ liệu.
* Đặc điểm của stack là LIFO (Last in first out) hoặc FILO (First in last out). Hãy tưởng tượng Stack giống như 1 thùng chứa đồ. Đồ nào được đưa vào cuối cùng ở trên top thì sẽ là đồ vật được lấy ra trước tiên.

1. **Thao tác với stack:**
2. Khai báo:

* Thư viện #include<stack>
* Khai báo stack: stack<*data*> *name*;

Vd: stack<int> st;

1. Hàm trong stack:

s.size() : trả về kích thước hiện tại của stack (số phần tử trong stack).

s.empty() : true (1) stack nếu rỗng, và (0) ngược lại.

s.push(x) : đẩy phần tử x vào đỉnh stack.

s.pop() : loại bỏ phẩn tử ở đỉnh của stack.

s.top() : truy cập tới phần tử ở đỉnh stack.

Các hàm bên trên đều có độ phức tạp là O(1).

Có thể nghiên cứu bài BCSTACK trên SPOJ để rõ hơn cách hoạt động của stack.

1. **CÁC BÀI MẪU.**
2. **Làm quen với stack:**

Link bài: [BCSTACK - SPOJ](https://www.spoj.com/PTIT/problems/BCSTACK/)

Code mẫu:

int main()

{

    stack<ll> ss;

    string a;

    do{

        cin >> a;

        if(a == "end") break;

        if(a == "init"){

            while(ss.size() > 0){

                ss.pop();

            }

        }

        if(a == "push"){

            int x;

            cin >> x;

            ss.push(x);

        }

        if(a == "pop"){

            if(ss.size() > 0){

                ss.pop();

            }

        }

        if(a == "top"){

            if(ss.size() != 0) cout << ss.top() << endl;

            else cout << "-1" << endl;

        }

        if(a == "size"){

            cout << ss.size() << endl;

        }

        if(a == "empty"){

            cout << ss.empty() << endl;

        }

    }

    while(a != "end");

}

1. **Tìm min, max trên đoạn tịnh tiến:**

Cho một dãy gồm n số nguyên a_{i} (n\leq 10^{5},a_{i}\leq 10^{9})  và một số nguyên dương k (k\leq n). Với mỗi phần tử i thuộc đoạn [1, n-k+1], hãy in ra giá trị nhỏ nhất của dãy a trong đoạn [i, i+k-1]

Example:

|  |  |
| --- | --- |
| Input | Output |
| 7 4  2 1 4 3 6 5 2 | 1 1 3 2 |

Giải thích ví dụ:

* Đoạn [1, 4] gồm các phần tử [2, 1, 4, 3] và giá trị nhỏ nhất là 1
* Đoạn [2, 5] gồm các phần tử [1, 4, 3, 6] và giá trị nhỏ nhất là 1
* Đoạn [3, 6] gồm các phần tử [4, 3, 6, 5] và giá trị nhỏ nhất là 3
* Đoạn [4, 7] gồm các phần tử [3, 6, 5, 2] và giá trị nhỏ nhất là 2

*Hướng dẫn:*

Nếu làm theo tìm kiếm tuần tự, sẽ rất lâu với độ phức tạp O(k\*(n-k)). Nên ta áp dụng stack vào bài để tối ưu như sau.

* Trong khoảng đang xét, ta cho biến i chạy lần lượt từng phần tử.
* Các phần tử lớn hơn a[i] trong khoảng thì ta loại khỏi stack, nếu có phần tử nhỏ hơn thì ta bỏ qua a[i]. Đến cuối, ta thêm a[i] thỏa mãn vào hàng đợi.
* Tiếp tục tịnh tiến sang khoảng kế tiếp, ta lại tìm được a[i] thỏa mãn.

Code mẫu:

#include <bits/stdc++.h>

using namespace std;

#define endl "\n"

#define ll long long

const ll big = 1e6;

int n, k, a[big];

int main(){

    cin >> n >> k;

    for(int i = 0 ; i < n ; i++) cin >> a[i];

    for(int i = 0 ; i < n - k + 1 ; i++)

    {

        int res = INT\_MAX;

        for(int j = i ; j < i + k ; j++) res = min(res, a[j]);

        cout << res << " ";

    }

}

#include <bits/stdc++.h>

using namespace std;

#define endl "\n"

#define ll long long

const ll big = 1e6;

int n, k, a[big];

int main()

{

    ios\_base::sync\_with\_stdio(false);

    cin.tie(0);

    cout.tie(0);

    int n, k;

    cin >> n >> k;

    int a[n + 5];

    for (int i = 0; i < n; i++) cin >> a[i];

    stack<int> st;

    for (int i = 0; i < n; i++)

    {

        while (st.size() != 0 && a[st.top()] >= a[i])

        {

            st.pop();

        }

        st.push(i);

        if (i - k + 1 >= 0)

        {

            stack<int> tmp;

            while (st.size() != 0)

            {

                tmp.push(st.top());

                st.pop();

            }

            while (tmp.top() < i - k + 1)

                tmp.pop();

            cout << a[tmp.top()] << "\n";

            while (tmp.size())

            {

                st.push(tmp.top());

                tmp.pop();

            }

        }

    }

}

1. **Bài toán bán dừa**

Link bài: [Bán dừa - SPOJ](https://vn.spoj.com/problems/KPLANK/)

Nếu các bạn biết câu chuyện thương tâm "ăn dưa leo trả vàng" của Pirate hẳn đã phải khóc hết nước mắt khi anh ấy, vì lòng thương chim, đã bán rẻ trái dưa leo siêu bự của mình.

Chart, histogram

Description automatically generatedDưa leo cũng đã bị chim to lấy đi rồi, Pirate giờ chuyển sang nghề bán dừa để bù lỗ. Bất đắc dĩ thôi, vì trên đảo toàn là dừa...

Chart, treemap chart

Description automatically generatedNhưng mà bán cái gì thì đầu tiên cũng phải có biển hiệu đã. Pirate quyết định lùng sục trên đảo các mảnh ván còn sót lại của những con tàu đắm để ghép lại thành tấm biển. Cuối cùng anh cũng tìm được N tấm ván hình chữ nhật, tấm thứ i có chiều rộng là 1 đơn vị và chiều dài là ai đơn vị. Pirate dựng đứng chúng trên mặt đất và dán lại với nhau để được một mảnh ván to hơn (xem hình minh họa).

Việc cuối cùng chỉ là đem mảnh ván này đi cưa thành tấm biển thôi. Nhưng hóa ra đây lại là công việc khó khăn nhất. Pirate rất thích hình vuông và muốn tấm biển của mình càng to càng tốt, nhưng khổ nỗi trên đảo lại không có nhiều dụng cụ đo đạc. Không êke, không thước đo độ, nên Pirate chỉ còn cách dựa vào cạnh của N tấm ván ban đầu để cưa cho thẳng thôi. Pirate chỉ có thể cưa theo những đoạn thẳng chứa một cạnh nào đó (dọc hoặc ngang) của các tấm ván.

Hãy giúp anh ấy cưa được tấm biển lớn nhất có thể. Input Dòng thứ nhất: ghi số nguyên N - số tấm ván. N dòng tiếp theo: mô tả độ cao của các tấm ván theo thứ tự trái sang phải sau khi đã dán lại. Output Một số nguyên duy nhất là độ dài cạnh của tấm biển lớn nhất có thể cưa được. Giới hạn Độ cao của các tấm ván là các số nguyên dương không vượt quá 109. 1 ≤ N ≤ 106. 60% số test có 1 ≤ N ≤ 2000. 80% số test có 1 ≤ N ≤ 105.

**Example**

**Input:**

7  
5  
2  
4  
3  
3  
1  
4  
  
**Output:**

3