Chủ đề 1: VÉT CẠN

Đã bao giờ bạn có ý định mò mẫm mật khẩu, mã pin thiết bị di động của ai đó và thử từng số một chưa? Hay bạn đã từng cố gắng lục tung mọi ngóc ngách trong nhà để tìm kiếm thứ gì đó? Thế thì bạn đã biết "vét cạn" rồi đấy!

I. Mục tiêu bài học

1	**Nắm được khái niệm về "vét cạn"**	**10%**
2	**Vận dụng được "vét cạn" trong mọi bài toán**	**55%**
3	**Biết cải thiện thuật toán để tối ưu hóa code**	**35%**

II. Lý thuyết

1. Khái niệm:

Thuật toán vét cạn hay duyệt trâu (Brute Force) được hiểu đúng như tên gọi của nó, chúng ta sẽ dùng những phương pháp đơn giản nhất để ***duyệt qua toàn bộ các trường hợp*** của bài toán. Khí đó, ta ***chắc chắn*** tìm được đáp án chính xác.

Ví dụ: Bạn muốn truy cập vào một chiếc laptop hay điện thoại mà không biết mật khẩu, bạn có thể thử qua từng dãy số một đến khi nhập đúng. Như "1", "2" ... "123", "124" ... N-1, N (N là mật khẩu chính xác)

2. Phân tích:

Qua ví dụ trên, có thể thấy rằng: trong mọi vấn đề, bạn luôn tìm được ***ít nhất một lời giải hợp lý***. Tuy nhiên, chính vì phải duyệt qua tất cả các đáp án nên cách này ***mất rất nhiều thời gian*** để thực hiện. Nếu mật khẩu của chiếc điện thoại càng dài thì càng mất nhiều thời gian để tìm ra được dãy số chính xác (giả sử không bị vô hiệu hóa nếu nhập sai).

Tương tự khi viết chương trình, nếu bạn sử dụng giải thuật vét cạn thì luôn đảm bảo tìm được kết quả đúng. Nhưng nếu dữ liệu đầu vào quá lớn sẽ dẫn đến lỗi TLE (Time Limit Exceeded - chạy quá thời gian cho phép).

3. Bài toán cụ thể:

Bài toán 1:

"Vừa gà vừa chó
Bó lại cho tròn
Ba mươi sáu con
Một trăm chân chẵn
Số gà số chó
Tính được bao nhiêu?"

- **Solution:**

Gọi x và y lần lượt là số chó và số gà. Lúc này ta duyệt 2 vòng lập: với mỗi x đang xét, ta tìm y sao cho x và y thỏa mãn điều kiện đề bài. Nếu không tìm được thì tiếp tục đến lần lặp tiếp theo.

- **Code mẫu:**

```
#include <bits/stdc++.h>
using namespace std;
int ga, cho;

int main()
{
    for (int x=1; x<=36; x++)
    {
        for (int y=1; y<=36; y++)
        {
            if (x+y == 36 && 4*x + 2*y == 100)
            {
                cho = x;
                ga = y;
            }
        }
    }
    cout << "So cho la: " << cho << '\n';
    cout << "So ga la: " << ga;
    return 0;
}
```

```
Output:

So cho la: 14
So ga la: 22
```

Bài toán 2: Cho dãy số A có N phần tử (A[i] < 101) nhập từ bàn phím. Tìm dãy con B có M phần tử từ dãy A (M ≤ N ≤ 1000) sao cho tổng các phần tử của dãy B lớn nhất có thể. ***(Dãy con là những phần tử liên tiếp nhau trong mảng)***.
- Dữ liệu vào: dòng đầu tiên lần lượt là 2 số N, M. Dòng thứ hai là N số nguyên A[i].
- Kết quả: dãy B có tổng lớn nhất tìm được.
- Ví dụ:

INPUT	OUTPUT
6 4 1 -3 2 -5 4 3	2 -5 4 3

- ***Solution:***

Trước tiên hãy phân tích đề bài và xét cần bao nhiêu vòng lặp để thực hiện, nhiệm vụ của mỗi vòng lặp là gì?

Ở bài toàn này, theo phương pháp vét cạn cơ bản, chắc chắn bạn sẽ cần 2 vòng lặp lồng nhau. Cụ thể như sau:

- **Code mẫu:**

```
#include <bits/stdc++.h>
using namespace std;
int a[1005], n, m;
int ketqua = 0, position = 0;

int main()
{
    cin>>n>>m;
    for (int i=1; i<=n; i++) cin>>a[i];

    for (int i=1; i<=m; i++) ketqua += a[i];
    for(int i=1; i<=n-m+1; i++)
    {
        int sum=0;
        for(int j=1; j<=m; j++) sum += a[j+i-1];
        if(sum >= ketqua)
        {
            ketqua = sum;
            position = i;
        }
    }

    for(int i=position; i<=position+m-1; i++) cout<<a[i]<<" ";
    return 0;
}
```

- Trước tiên, cho biến lưu kết quả (ketqua) chứa tổng của M phần tử đầu tiên trong mảng A. Vì M phần tử đầu tiên trong mảng A chính là **dãy con B khả thi đầu tiên**.

- Vòng lặp bên ngoài để tìm **tất cả dãy con B khả thi**: Ta thấy M phần tử cuối của mảng A chính là dãy con B cuối cùng **có thể là đáp án**. Do đó, ta chỉ cần duyệt đến N-M+1.

- Vòng lặp bên trong để tính tổng các phần tử của **dãy B đang xét** (sum). Sau mỗi lần lặp, phải so sánh **tổng vừa tìm được** với **tổng đang lưu** (ketqua).

- Nếu tổng vừa tính lớn hơn hoặc bằng kết quả đang lưu, ta cập nhật kết quả mới và lưu vị trí của dãy B mới vào position (vị trí của một dãy con thường được hiểu là vị trí phần tử đầu tiên trong dãy).

- Cuối cùng, ta xuất ra M phần tử từ vị trí đã lưu. Đây là đáp án cần tìm.

→ Với 2 vòng lặp lồng nhau thì độ phức tạp của bài toán là O(N*M).

III. Cải thiện thuật toán - Vét cạn "nâng cao"

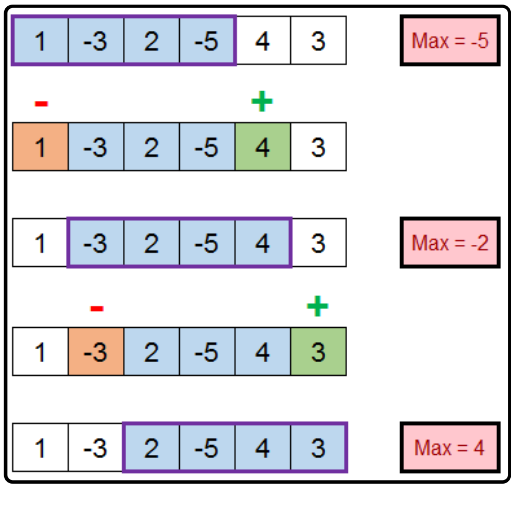
1. Vì sao phải tối ưu hóa vét cạn?

Ở một số bài toán phức tạp, đôi khi bạn cần đến ba, bốn hay nhiều hơn những vòng lặp lồng nhau. Vậy nên, việc sử dụng tư duy toán học và kĩ năng lập trình để giảm bớt số vòng lặp lồng nhau là vô cùng cần thiết. Nên nhớ rằng với mỗi vòng lặp lồng nhau bớt đi, độ phức tạp chương trình sẽ giảm đi (N lần - thông thường ta duyệt từ 1 đến N).

2. Ví dụ:

Với bài toán 2 ở phần trên, bạn có thể tối ưu hóa giải thuật còn một vòng lặp bằng kỹ thuật **cửa sổ trượt**:

+ Kỹ thuật cửa sổ trượt (cụ thể ở bài này) tức là khi ta **đang biết** tổng của đoạn [L...R] bất kì, để tính đoạn [L+1...R+1], ta chỉ cần lấy tổng **đang biết** trừ cho phần tử ở vị trí L, và cộng phần tử ở vị trí R+1 vào. Xem ảnh minh họa:



+ Cụ thể, với **A[6] = {1; -3; 2; -5; 4; 3}** như trên, ban đầu ta có tổng từ A[1] đến A[4] là ketqua = 1 + (-3) + 2 + (-5) = -5. Để tính đoạn từ A[2] đến A[5], ta làm như sau: (-5) - 1 + 4 = -2. Ta lặp câu lệnh đó đến khi duyệt hết mảng A.

+ Khi tính được tổng mới, ta **so sánh với tổng trước đó** để **lưu đáp án** và **vị trí của dãy đáp án**.

+ Xuất ra kết quả như bình thường.

- **Code mẫu:**

```
#include <bits/stdc++.h>
using namespace std;
int n,m,a[1001];
int ketqua=0, position=0;

int main()
{
    cin>>n>>m;
    for (int i=1; i<=n; i++) cin>>a[i];

    for (int i=1; i<=m; i++) ketqua += a[i];
    int sum = ketqua;
    for (int i=m+1; i<=n; i++)
    {
        sum = sum - a[i-m] + a[i];
        if (sum>=ketqua)
        {
            ketqua = sum;
            position = i-m+1;
        }
    }
    for (int i=position; i<=position+m-1; i++) cout<<a[i]<<" ";
    return 0;
}
```

→ Theo cách trên, độ phức tạp của chương trình đã giảm từ O(N*M) thành O(N).

IV. Bài tập vận dụng

Bài tập 1: **Dãy con tăng:** EASY

Cho một dãy số nguyên A có N phần tử (1 ≤ N, A[i] ≤ 10000). Viết chương trình tìm chiều dài của dãy con tăng liên tiếp dài nhất trong dãy A. (Dãy con tăng liên tiếp có dạng A[i-1] < A[i] < A[i+1] với mọi i).

INPUT	OUTPUT	GIẢI THÍCH
6 1 3 2 4 5 10	4	4 là độ dài của dãy con: 2 4 5 10

Bài tập 2: Dãy cấp số cộng: EASY

Cho một dãy A gồm N số nguyên (1 ≤ N, A[i] ≤ 10000). In ra dãy con cấp số cộng dài nhất tìm được từ dãy A. (Dãy con cấp số cộng có dạng: A[i] = A[i-1] + d với d là số nguyên bất kì. Ví dụ: {1; 2; 3; 4}, {2; 5; 8}, {1; 5; 9; 13} ...)

INPUT	OUTPUT
8 1 2 1 4 5 6 2 1	4 5 6

Bài tập 3: Vòng tròn số: MEDIUM

Cho N số nguyên nằm trên 1 vòng tròn được đánh số thứ tự theo chiều kim đồng hồ. Tìm dãy con có tổng các phần tử lớn lớn nhất (chỉ được đi 1 vòng, tức độ dài tối đa của dãy con không vượt quá N). (1 ≤ N ≤ 10^7; A[i] ≤ 1000).

- Dữ liệu vào: dòng đầu tiên là số N; dòng thứ 2 là các số trên vòng tròn.
- Kết quả:
  - Dòng đầu tiên ghi tổng của dãy con tìm được.
  - Dòng thứ hai ghi hai số d và k: d là vị trí của số hạng đầu tiên trong dãy con và k là độ dài của dãy tìm được.

INPUT	OUTPUT	GIẢI THÍCH
7 2 -4 1 -7 4 6 -1	11 5 4	4 + 6 + (-1) + 2 = 11

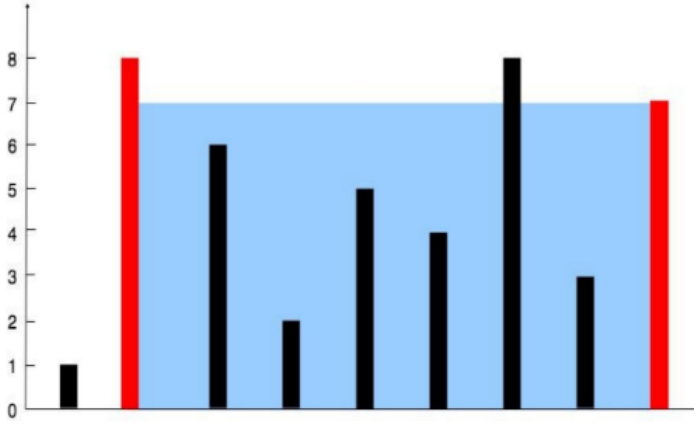
Bài tập 4: Diện tích lớn nhất: MEDIUM

Cho N đoạn thẳng nằm trên một mặt phẳng với A[i] là độ dài của đoạn thẳng thứ i. Tìm 4 điểm thuộc 2 đoạn thẳng bất kì để tạo thành hình chữ nhật có diện tích lớn nhất.

- Dữ liệu vào: số N và N số nguyên dương ($2 \le N \le 1000$; $1 \le A[i] \le 10^5$).
- Kết quả: diện tích hình chữ nhật lớn nhất tìm được.

INPUT	OUTPUT
9 1 8 6 2 5 4 8 3 7	49

- Giải thích: Phần tô xanh là hình chữ nhật lớn nhất tìm được, hình này có diện tích là 49.



V. Dặn dò:

- Ở bất kì bài toán nào, nhất là trong các kỳ thi, hãy vét cạn trước rồi mới tối ưu hóa thành các giải thuật khác sau.
- Hãy cố gắng tinh thông thuật toán vét cạn, vì nó sẽ **luôn cho kết quả đúng**. Do đó, bạn có thể sử dụng thuật toán này để làm gốc, đối chiếu output với các đoạn code đã cải tiến.
- Trong các cuộc thi, trừ khi bạn rất chắc chắn về giải thuật của mình thì hãy dùng, còn không hãy ưu tiên dùng vét cạn. Điều này sẽ giúp bạn đạt giải hoặc đậu trong hầu hết các cuộc thi ở quy mô thành phố trở xuống. Đó cũng là lý do mà Brute force được mệnh danh là vua của mọi thuật toán.
- Theo dõi contest luyện tập sắp tới của Căng TIN 2326 trên GDOJ.
- Mọi thắc mắc xin liên hệ:
  + Email: cangtin2326@gmail.com
  + Confession: https://forms.gle/dvByc9UQ2bRsPQHn8
  + Instagram: __cang_tin_2326__
  + Inbox fanpage: Căng TIN 2326