

# 👉 Giới thiệu kỹ thuật Two Pointers (Hai con trỏ)

**Two Pointers** là kỹ thuật sử dụng hai biến (thường là `left` và `right`) để duyệt qua dữ liệu, cho phép giải quyết hiệu quả các bài toán tìm kiếm cặp phần tử, đoạn con hoặc nhóm phần tử thỏa mãn điều kiện nhất định.

Khi áp dụng đúng, Two Pointers có thể tối ưu độ phức tạp từ  $O(n^2)$  xuống  $O(n)$  hoặc  $O(n \log n)$ , đặc biệt trong các bài có mảng **đã sắp xếp** hoặc cần **xử lý đoạn liên tiếp**.

## 🔥 Các biến thể phổ biến

Biến thể	Mô tả
Two Pointers cùng chiều	<code>left</code> và <code>right</code> cùng tiến theo 1 hướng (Sliding Window)
Two Pointers đối đầu	<code>left</code> từ đầu, <code>right</code> từ cuối, tiến vào giữa
One-fixed One-moving	Một con trỏ cố định, con kia di chuyển để tìm điều kiện thỏa mãn

## 🔧 Công thức khung chuẩn

### a) Two Pointers cùng chiều


```
left = 0
for right in range(n):
    # Cập nhật trạng thái
    while (không thỏa điều kiện):
        left += 1
    # Xử lý đáp án tại đây
```

### b) Two Pointers đối đầu

```
left = 0
right = n - 1
while left < right:
    if (điều kiện đúng):
        # Xử lý đáp án
        left += 1
    else:
        right -= 1
```

## Danh sách luyện tập thêm cùng gợi ý

Bài tập	Link	Gợi ý áp dụng
Subarray Sum Equals K	<a href="#">LeetCode</a>	Sliding Window
Longest Subarray with Sum $\leq K$	<a href="#">CSES</a>	Sliding Window
3Sum	<a href="#">LeetCode</a>	Sắp xếp + Two Pointers
Count Number of Nice Subarrays	<a href="#">LeetCode</a>	Two Pointers có giãn cửa sổ



 Mẹo: Hãy đọc kỹ đề để nhận diện xem bài yêu cầu "cặp" hay "đoạn con", và dữ liệu có được sắp xếp hay không!

## Các lỗi sai học sinh thường gặp

- Không thu hẹp cửa sổ đúng lúc, dẫn tới bỏ sót đáp án.
- Nhầm điều kiện dừng trong while khi sliding window.
- Sai chỉ số: khi đề bài yêu cầu trả về chỉ số bắt đầu từ 1 thay vì 0.
- Quên xét trường hợp góc: mảng rỗng, không có đáp án,...

## Câu hỏi ôn tập nhanh

- Khi nào nên áp dụng Two Pointers đối đầu?
- Sliding Window là biến thể của Two Pointers như thế nào?
- Làm sao để nhận biết bài cần tối ưu từ  $O(n^2)$  xuống  $O(n)$ ?

 Học kỹ thuật Two Pointers không chỉ để giải bài nhanh hơn, mà còn để phát triển tư duy tối ưu hóa bài toán ngay từ lúc đọc đề! 

## Bài mẫu 1: Two Sum II – Input Array is Sorted

Link bài gốc: [LeetCode - Two Sum II](#)

### 1. Đề bài (dịch tiếng Việt)

Bạn được cho một mảng các số nguyên đã sắp xếp theo thứ tự tăng dần và một số nguyên `target`.

Yêu cầu: **Tìm hai số** trong mảng có **tổng đúng bằng target** và **trả về chỉ số** của chúng (lưu ý: chỉ số tính từ 1).

Giả sử rằng mỗi input có **chính xác một cặp** đáp án.

## Input

- `numbers` – Mảng số nguyên, đã được **sắp xếp tăng dần**.
- `target` – Số nguyên mục tiêu.

## Output

- Một mảng gồm hai số nguyên `[index1, index2]` (với `index1 < index2`), là chỉ số của hai phần tử thỏa mãn.

## Ràng buộc

- `2 <= numbers.length <= 3 * 10^4`
- `-1000 <= numbers[i] <= 1000`
- `-10^9 <= target <= 10^9`
- Luôn có đúng một cặp số thỏa mãn yêu cầu.

## Ví dụ

### Ví dụ 1:

Input: `numbers = [2,7,11,15]`, `target = 9`

Output: `[1,2]`

Giải thích:  $2 + 7 = 9$ .

### Ví dụ 2:

Input: `numbers = [2,3,4]`, `target = 6`

Output: `[1,3]`

Giải thích:  $2 + 4 = 6$ .

## 2. Gợi ý tư duy

### a) Cách Brute-force

- Duyệt tất cả các cặp `(i, j)` với `i < j`.
- Kiểm tra nếu `numbers[i] + numbers[j] == target`.
- Trả kết quả khi tìm được.

⌚ Độ phức tạp thời gian:  **$O(n^2)$**  — Không hiệu quả với mảng lớn.

## b) Cách tối ưu bằng Two Pointers

- Do mảng **đã sắp xếp**, ta áp dụng **hai con trỏ đối đầu**:
  - `left = 0`, `right = n-1`
- Tại mỗi bước:
  - Nếu tổng nhỏ hơn `target`: cần tổng lớn hơn → `left += 1`
  - Nếu tổng lớn hơn `target`: cần tổng nhỏ hơn → `right -= 1`
  - Nếu tổng đúng bằng `target`: trả chỉ số `[left+1, right+1]`

🕒 Độ phức tạp thời gian: **O(n)** — Rất tối ưu.

## 3. Code mẫu C++

```
#include <vector>
using namespace std;

vector<int> twoSum(vector<int>& numbers, int target) {
    int left = 0, right = numbers.size() - 1;
    while (left < right) {
        int current_sum = numbers[left] + numbers[right];
        if (current_sum == target) {
            return {left + 1, right + 1}; // chỉ số bắt đầu từ 1
        } else if (current_sum < target) {
            ++left;
        } else {
            --right;
        }
    }
    return {}; // không cần thiết vì luôn tồn tại đáp án
}
```

📌 Ghi nhớ: Khi dữ liệu **đã sắp xếp**, luôn kiểm tra xem có thể áp dụng **hai con trỏ đối đầu** để tối ưu hoá bài toán hay không!

## 📖 Bài mẫu 2: Subarray Sum

Link bài gốc: [CSES - Subarray Sum](#)

### 1. Đề bài (dịch tiếng Việt)

Bạn được cho một mảng số nguyên dương gồm `n` phần tử và một số nguyên `x`.

Yêu cầu: Đếm số lượng **đoạn con liên tiếp** trong mảng có tổng bằng `x`.

## Input

- Dòng đầu tiên: hai số nguyên  $n$  và  $x$  ( $1 \leq n \leq 2 \cdot 10^5$ ,  $1 \leq x \leq 10^9$ ).
- Dòng thứ hai:  $n$  số nguyên dương (mỗi số không vượt quá  $10^9$ ).

## Output

- Một số nguyên: số lượng đoạn con có tổng đúng bằng  $x$ .

## Ví dụ

### Input:

```
5 7
2 4 1 2 7
```

### Output:

```
2
```

**Giải thích:** Hai đoạn  $[2, 4, 1]$  và  $[7]$  có tổng bằng 7.

## 2. Gợi ý tư duy

### a) Cách Brute-force

- Duyệt tất cả các đoạn  $[i, j]$ , tính tổng các phần tử.
- Nếu tổng đúng bằng  $x$  thì tăng kết quả.

⌚ Độ phức tạp:  $O(n^2)$  — không khả thi với  $n$  lớn.

### b) Cách tối ưu bằng Two Pointers

- Cửa sổ trượt  $[left, right]$ :
  - Nếu tổng nhỏ hơn  $x$ , mở rộng  $right$ .
  - Nếu tổng lớn hơn  $x$ , co  $left$ .
  - Nếu tổng đúng bằng  $x$ , tăng kết quả.

⌚ Độ phức tạp:  $O(n)$


## 3. Code mẫu C++

```
#include <iostream>
#include <vector>
using namespace std;
```

```

int main() {
    int n;
    long long x;
    cin >> n >> x;
    vector<long long> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
    }
    int left = 0;
    long long sum = 0, result = 0;
    for (int right = 0; right < n; ++right) {
        sum += a[right];
        while (sum > x) {
            sum -= a[left++];
        }
        if (sum == x) result++;
    }
    cout << result << endl;
}

```

 Kỹ thuật **sliding window** thực chất là một biến thể của **two pointers cùng chiều**, rất hiệu quả khi các phần tử đều không âm!

## Bài mẫu 3: Longest Subarray with Sum $\leq K$

Link bài gốc: [CSES - Subarray Divisibility \(biến thể tương tự\)](#)

(Chú thích: Bài Subarray Divisibility của CSES tương tự bài tìm đoạn con thỏa mãn điều kiện tổng với ràng buộc.)

### 1. Đề bài (dịch tiếng Việt)

Cho một mảng gồm  $n$  số nguyên không âm và một số nguyên  $k$ .

Yêu cầu: Tìm độ dài lớn nhất của một **đoạn con liên tiếp** sao cho **tổng các phần tử trong đoạn không vượt quá  $k$** .

#### Input

- Dòng đầu tiên: hai số nguyên  $n$  và  $k$  ( $1 \leq n \leq 10^5$ ,  $1 \leq k \leq 10^9$ ).
- Dòng thứ hai:  $n$  số nguyên (mỗi số không âm và  $\leq 10^9$ ).

#### Output

- Một số nguyên: độ dài lớn nhất của đoạn thỏa mãn điều kiện.

## Ví dụ

### Input:

```
5 7
2 1 5 1 3
```

### Output:


```
3
```

**Giải thích:** Đoạn `[2, 1, 5]` không thỏa mãn vì tổng  $> 7$ , nhưng `[5, 1, 3]` tổng = 9 không được. Đoạn `[2, 1, 5]` hoặc `[1, 5, 1]` tổng 8 hoặc 7 đều chấp nhận, chọn dài nhất là 3.

## 2. Gợi ý tư duy

### a) Cách Brute-force

- Duyệt tất cả các đoạn `[i, j]`, tính tổng từng đoạn.
- Nếu tổng  $\leq k$ , cập nhật độ dài lớn nhất.

 Độ phức tạp:  $O(n^2)$  — không khả thi với `n` lớn.

### b) Cách tối ưu bằng Two Pointers (Sliding Window)

- Giữ cửa sổ `[left, right]` sao cho tổng trong cửa sổ  $\leq k$ .
- Khi tổng vượt quá `k`, co `left`.
- Cập nhật kết quả trong quá trình.


 Độ phức tạp:  $O(n)$

## 3. Code mẫu C++

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n;
    long long k;
    cin >> n >> k;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
    }
    int left = 0, max_len = 0;
    long long sum = 0;
```

```
for (int right = 0; right < n; ++right) {
    sum += a[right];
    while (sum > k) {
        sum -= a[left++];
    }
    max_len = max(max_len, right - left + 1);
}
cout << max_len << endl;
}
```

 Sliding Window giúp ta luôn giữ một đoạn thỏa mãn điều kiện tổng liên tiếp mà không phải tính tổng lại từ đầu!

## Danh sách bài tập luyện tập thêm

### 1. [Two Sum II - Input Array Is Sorted \(LeetCode\)](#)

- **Mô tả:** Tìm hai phần tử trong mảng đã sắp xếp sao cho tổng đúng bằng `target`.
- **Gợi ý:** Two pointers đối đầu (`left`, `right`) di chuyển tùy theo tổng.

### 2. [3Sum \(LeetCode\)](#)

- **Mô tả:** Tìm tất cả bộ ba số có tổng bằng 0, không trùng lặp.
- **Gợi ý:** Sắp xếp mảng, fix một phần tử, two pointers tìm bộ đôi còn lại.

### 3. [Container With Most Water \(LeetCode\)](#)

- **Mô tả:** Tìm hai cột nước tạo container lớn nhất.
- **Gợi ý:** Two pointers đối đầu, tiến con trỏ thấp hơn.

### 4. [Longest Subarray with Sum \$\leq K\$ \(CSES\)](#)

- **Mô tả:** Đoạn con dài nhất có tổng  $\leq k$ .
- **Gợi ý:** Sliding Window, co giãn cửa sổ.

### 5. [Subarray Sum \(CSES\)](#)

- **Mô tả:** Đếm số đoạn con tổng đúng bằng `x`.
- **Gợi ý:** Sliding Window khi phần tử không âm.



## 6. [Minimum Size Subarray Sum \(LeetCode\)](#)

- **Mô tả:** Tìm đoạn con ngắn nhất có tổng  $\geq$  `target`.
  - **Gợi ý:** Co giãn cửa sổ khi đủ điều kiện.
- 

## 7. [Count Number of Nice Subarrays \(LeetCode\)](#)

- **Mô tả:** Đếm đoạn con chứa đúng `k` số lẻ.
  - **Gợi ý:** Two pointers + đếm số lượng đoạn hợp lệ.
- 

## 8. [Maximum Erasure Value \(LeetCode\)](#)


- **Mô tả:** Tổng giá trị lớn nhất của đoạn không trùng phần tử.
  - **Gợi ý:** Two pointers + set theo dõi phần tử.
- 

## 9. [Boats to Save People \(LeetCode\)](#)

- **Mô tả:** Tối thiểu số thuyền cần dùng để chở hết người.
  - **Gợi ý:** Sắp xếp + two pointers ghép người nhẹ và nặng.
- 

## 10. [Two Sum Less Than K \(LeetCode\)](#)

- **Mô tả:** Tìm tổng hai số lớn nhất nhỏ hơn `k`.
  - **Gợi ý:** Two pointers đối đầu chọn cặp tối ưu.
- 

 Luyện tập đa dạng các dạng bài Two Pointers giúp học sinh thành thạo tư duy tối ưu hóa trên dãy số và đoạn con! 