Sliding Window Technique – Kỹ Thuật Cửa Sổ Trượt

Nguyễn Quản Bá Hồng*

Ngày 16 tháng 8 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Sliding Window Technique – Kỹ Thuật Cửa Sổ Trượt.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/sliding_window/NQBH_sliding_window.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/sliding_window/NQBH_sliding_window.tex.
- *Olympiad in Informatics & Association for Computing Machinery–International Collegiate Programming Contest – Olympic Tin Học Sinh Viên OLP & ICPC.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/NQBH_OLP_ICPC.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/NQBH_OLP_ICPC.tex.
- Codes:
  - Input: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/input.
  - Output: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/output.
  - C: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/C.
  - C++: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/C++.
  - C#: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/C%23.
  - Java: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/Java.
  - JavaScript: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/JavaScript.
  - Python: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/Python.
  - Resources: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/resource.

Mục lục

1 Introduction to Sliding Window Technique – Giới Thiệu Kỹ Thuật Cửa Sổ Trượt

Resources – Tài nguyên.

1. Benjamin Qi. USACO Guide/sliding window. Maintaining data over consecutive subarrays.
2. Geeks for Geeks/sliding window technique.
3. Geeks for Geeks/top problems on sliding window technique for interviews.
4. Geeks for Geeks/practice questions on sliding window.

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: https://nqbh.github.io/. GitHub: https://github.com/NQBH.

The *sliding window* is a powerful algorithmic technique used to optimize problems involving arrays or strings. It helps reduce the time complexity of problems that require checking or computing results over contiguous subarrays or substrings.

– *Cửa sổ trượt* là một kỹ thuật thuật toán mạnh mẽ được sử dụng để tối ưu hóa các bài toán liên quan đến mảng hoặc chuỗi. Nó giúp giảm độ phức tạp về thời gian của các bài toán đòi hỏi phải kiểm tra hoặc tính toán kết quả trên các mảng con hoặc chuỗi con liền kề.

Instead of repeatedly iterating over the same elements, the sliding window maintains a range (or "window") that moves step-by-step through the data, updating results incrementally.

– Thay vì lặp lại nhiều lần các phần tử giống nhau, cửa sổ trượt duy trì một phạm vi (hay "cửa sổ") di chuyển từng bước qua dữ liệu, cập nhật kết quả theo từng bước.

Question 1. *When to use sliding window? – Khi nào nên sử dụng cửa sổ trượt?*

Answer. You can apply the sliding window technique when:

- The problem involves a contiguous sequence (subarray or substring).
- You need to find: maximum/minimum sum in a fixed-size window, longest/shortest subarray with certain conditions, or count of distinct elements in a range.
- Brute force would involve nested loops, leading to higher complexity.

– Bạn có thể áp dụng kỹ thuật cửa sổ trượt khi:

- Bài toán liên quan đến một chuỗi liên tiếp (mảng con hoặc chuỗi con).
- Bạn cần tìm: tổng tối đa/tối thiểu trong một cửa sổ có kích thước cố định, mảng con dài nhất/ngắn nhất với các điều kiện nhất định, hoặc số lượng phần tử riêng biệt trong một phạm vi.
- Thử nghiệm vét cạn sẽ liên quan đến các vòng lặp lồng nhau, dẫn đến độ phức tạp cao hơn.

& đặc biệt trong xử lý ảnh (Image Processing) & thị giác máy tính (Computer Vision), người ta thường dùng kỹ thuật cửa sổ trượt để xử lý từng cụm pixel. □

Problem 1 (Maximum sum of subarrays with k elements)**.** *Given an array of integers $\{a_i\}_{i=0}^{n-1} \subset \mathbb{Z}$, calculate the maximum sum of a subarray having size exactly k.*

Bài toán 1 (Tổng lớn nhất của một mảng con có k phần tử)**.** *Cho một mảng số nguyên $\{a_i\}_{i=0}^{n-1} \subset \mathbb{Z}$, tính tổng lớn nhất của một mảng con có kích thước chính xác là k.*

Input. *Dòng 1 chứa số bộ test t. Mỗi trong t cặp dòng tiếp theo: Dòng 1 chứa 2 số $n, k \in \mathbb{N}^\star$: số lượng phần tử của mảng & của mảng con. Dòng 2 chứa n phần tử của mảng a.*

Output. *Tổng lớn nhất có thể của 1 mảng con kích thước bằng k của mảng a.*

max_sum_subarray_k_element.inp	max_sum_subarray_k_element.out
5 3 5 2 −1 0 3	6
9 4 1 4 2 10 23 3 1 0 20	4

Explanation. *For test case 1, a satisfying subarray is $[5, 2, -1]$. For test case 2, a satisfying subarray is $[4, 2, 10, 23]$.*

1st solution: Naive approach: $O(nk)$ time & $O(1)$ space. A mathematical formalization of this problem can be given by

$$\max_{i \in \overline{0,n-k}} \sum_{j=i}^{i+k-1} a_j = \max_{i \in \overline{0,n-k}} a_i + a_{i+1} + \cdots + a_{i+k-1}.$$

1. NQBH's C++: maximum sum of subarrays with k elements: naive approach:

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

int main() {
    int t, n, k, sum, ans = INT_MIN;
    cin >> t;
    while (t--) {
        cin >> n >> k;
        int a[n];
        for (int i = 0; i < n; ++i) cin >> a[i];
```

```
        for (int i = 0; i < n - k + 1; ++i) {
            sum = 0;
            for (int j = i; j < i + k; ++j) sum += a[j];
            ans = max(ans, sum);
        }
        cout << ans << '\n';
    }
}
```

2. Geeks for Geeks's maximum sum of subarrays with k elements: naive approach:

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

int max_sum(vector<int>& arr, int k) {
    int n = arr.size();
    int max_sum = INT_MIN;
    for (int i = 0; i <= n - k; ++i) { // consider all blocks starting with i
        int current_sum = 0;
        // calculate sum of current subarray of size k
        for (int j = 0; j < k; ++j) current_sum += arr[i + j];
        max_sum = max(current_sum, max_sum); // update result if required
    }
    return max_sum;
}

int main() {
    int t, n, k;
    cin >> t;
    while (t--) {
        cin >> n >> k;
        vector<int> a(n);
        for (int& i : a) cin >> i;
        cout << max_sum(a, k) << '\n';
    }
}
```

□

2nd solution: Sliding window technique: $O(n)$ time & $O(1)$ space. We compute the sum of the 1st k elements out of n terms using a linear loop & store the sum in variable `window_sum`. Then we will traverse linearly over the array till it reaches the end & simultaneously keep track of the maximum sum. To get the current sum of a block of k elements just subtract the 1st element from the previous block & add the last element of the current block.

– Chúng ta tính tổng của k phần tử đầu tiên trong số n số hạng bằng cách sử dụng vòng lặp tuyến tính & lưu tổng vào biến `window_sum`. Sau đó, chúng ta sẽ duyệt tuyến tính trên mảng cho đến khi đến cuối & đồng thời theo dõi tổng lớn nhất. Để tính tổng hiện tại của một khối k phần tử, chỉ cần trừ phần tử đầu tiên khỏi khối trước & cộng phần tử cuối cùng của khối hiện tại.

C++:

1. NQBH's C++:

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

int main() {
    int t, n, k, sum;
    cin >> t;
    while (t--) {
        sum = 0;
        cin >> n >> k;
        int a[n];
        for (int i = 0; i < n; ++i) cin >> a[i];
        for (int i = 0; i < k; ++i) sum += a[i]; // sum of 1st k elements
        int ans = sum; // init answer as sum of 1st k elements
        for (int i = k; i < n; ++i) {
```

```
            sum += a[i] - a[i - k];
            ans = max(ans, sum);
        }
        cout << ans << '\n';
    }
}
```

2. Geeks for Geeks's maximum sum of subarrays with k elements: sliding window:

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

int max_sum(vector<int>& arr, int k) {
    int n = arr.size();
    if (n <= k) return -1; // invalid size
    // compute sum of 1st window of size k
    int max_sum = 0;
    for (int i = 0; i < k; ++i) max_sum += arr[i];
    // compute sums of remaining windows by removing 1st element of previous window
    // & adding last element of current window
    int window_sum = max_sum;
    for (int i = k; i < n; ++i) {
        window_sum += arr[i] - arr[i - k];
        max_sum = max(max_sum, window_sum);
    }
    return max_sum;
}

int main() {
    int t, n, k;
    cin >> t;
    while (t--) {
        cin >> n >> k;
        vector<int> a(n);
        for (int& i : a) cin >> i;
        cout << max_sum(a, k) << '\n';
    }
}
```

□

Question 2 (How). *How to use sliding window technique? – Làm thế nào để sử dụng kỹ thuật cửa sổ trượt?*

Answer. There are basically 2 types of sliding window:

- *Fixed size sliding window.* The general steps to solve these equations by following below steps:
  1. Find the size of the window required, say k.
  2. Compute the result for the 1st window, i.e., the window consisting of the 1st k elements of the data structure.
  3. Then use a loop to slide the window by 1 & keep computing the result window by window.
- *Variable size sliding window.* The general steps to solve these questions by the following steps:
  1. In this type of sliding window problem, we increase our right pointer 1 by 1 till our condition is true.
  2. At any step if our condition does not match, we shrink the size of our window by increasing left pointer.
  3. Again, when our condition satisfies, we start increasing the right pointer & follow step 1.
  4. We follow these steps until we reach to the end of the array.

– Về cơ bản có 2 loại cửa sổ trượt:

- *Cửa sổ trượt kích thước cố định.* Các bước chung để giải các phương trình này bằng cách làm theo các bước dưới đây:
  1. Tìm kích thước của cửa sổ cần thiết, giả sử là k.
  2. Tính kết quả cho cửa sổ thứ nhất, tức là cửa sổ bao gồm k phần tử đầu tiên của cấu trúc dữ liệu.
  3. Sau đó, sử dụng vòng lặp để trượt cửa sổ đi 1 & tiếp tục tính kết quả theo từng cửa sổ.

- *Cửa sổ trượt kích thước thay đổi.* Các bước chung để giải các bài toán này bằng các bước sau:

  1. Trong loại bài toán cửa sổ trượt này, chúng ta tăng con trỏ phải lên 1 cho đến khi điều kiện đúng.
  2. Tại bất kỳ bước nào, nếu điều kiện không khớp, chúng ta sẽ thu nhỏ kích thước cửa sổ bằng cách tăng con trỏ trái.
  3. Một lần nữa, khi điều kiện của chúng ta thỏa mãn, chúng ta bắt đầu tăng con trỏ phải & làm theo bước 1.
  4. Chúng ta làm theo các bước này cho đến khi đến cuối mảng.

□

Question 3. *How to identify "sliding window problems", i.e., problems that can be solved by applying sliding window technique?*

Answer. Here are some signs or criteria that a problem can be & should be solved by sliding window technique:

1. These problems generally require finding maximum/minimum subarrays, substrings which satisfy some specific condition.
2. The size of the subarray or substring k will be given in some of these problems.
3. These problems can easily (i.e., naively but slowly) be solved in $O(n^2)$ time complexity using nested loops, however, we can solve these problems in $O(n)$ time complexity by using sliding window technique.
4. Required time complexity is $O(n)$ or $O(n \log_2 n)$.
5. Constraints: $n \in [10^6]$.

– Dưới đây là một số dấu hiệu hoặc tiêu chí cho thấy một bài toán có thể được giải quyết bằng kỹ thuật cửa sổ trượt:

1. Những bài toán này thường yêu cầu tìm các mảng con tối đa/tối thiểu, các chuỗi con thỏa mãn một số điều kiện cụ thể.
2. Kích thước của mảng con hoặc chuỗi con k sẽ được cung cấp trong một số bài toán này.
3. Những bài toán này có thể dễ dàng (tức là, một cách ngây thơ nhưng chậm chạp) được giải quyết với độ phức tạp thời gian $O(n^2)$ bằng cách sử dụng các vòng lặp lồng nhau, tuy nhiên, chúng ta có thể giải quyết những bài toán này với độ phức tạp thời gian $O(n)$ bằng cách sử dụng kỹ thuật cửa sổ trượt.
4. Độ phức tạp thời gian yêu cầu là $O(n)$ hoặc $O(n \log_2 n)$.
5. Ràng buộc: $n \in [10^6]$.

□

2 Sáng Tác Bài Toán Sử Dụng Kỹ Thuật Cửa Sổ Trượt

Question 4. *Can sliding window technique be extended to more complex data structures e.g. 2D matrix, 2D vector, or even graphs?*
– Kỹ thuật cửa sổ trượt có thể được mở rộng sang các cấu trúc dữ liệu phức tạp hơn như ma trận 2D, vectơ 2D hoặc thậm chí là đồ thị không?

Bài toán 2. *Cho 1 ma trận $A = \{a_{ij}\}_{i,j=1}^{m,n} \in \mathbb{R}^{m \times n}$. Tìm ma trận con của A có: (a) Tổng lớn nhất. (b) Tổng nhỏ nhất.*

Có thể mở rộng ra cho mảng nhiều chiều, i.e., tensor nhiều chiều.

Bài toán 3. *Cho 1 đồ thị đơn hữu hạn $G = (V, E)$ có trọng số ở các đỉnh. Tìm đồ thị con $G' = (V', E')$ của G sao cho: (a) G' liên thông & có tổng các trọng số ở đỉnh nhỏ nhất, lớn nhất.*

3 Miscellaneous