2 Pointers Method – Phương Pháp 2 Con Trỏ

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:

- *2 Pointers Method – Phương Pháp 2 Con Trỏ*.
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/2_pointers/NQBH_2_pointers.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/2_pointers/NQBH_2_pointers.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.

List of classical problems: subarray sum on $\mathbb{N}, \mathbb{Z}, \mathbb{R}$, 2SUM, 3SUM, k-sum, .

Mục lục

1 Introduction to 2 Pointers Method – Giới Thiệu Phương Pháp 2 Con Trỏ

Resources – Tài nguyên.

1. ANTTI LAAKSONEN. *Competitive Programmer's Handbook*. Chap. 8: Amortized Analysis. Sect. 8.1: 2 pointers method, p. 77.
2. DARREN YAO, QI WANG, RYAN CHOU. USACO Guide/2 pointers.

   Abstract. Iterating 2 monotonic pointers across an array to search for a pair of indices satisfying some condition in linear time. – Lặp lại 2 con trỏ đơn điệu trên 1 mảng để tìm kiếm 1 cặp chỉ số thỏa mãn 1 số điều kiện trong thời gian tuyến tính.
3. Geeks for Geeks/2 pointers technique.

*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.

4. Phan Đình Khôi. VNOI/kỹ thuật 2 con trỏ.

The 2 pointers method iterates 2 pointers across an array, to track the start & end of an interval. It can also be used to track 2 values in an array.

– Phương thức 2 con trỏ lặp lại 2 con trỏ trên 1 mảng để theo dõi điểm bắt đầu và kết thúc của 1 khoảng. Phương thức này cũng có thể được sử dụng để theo dõi 2 giá trị trong 1 mảng.

General ideas of 2 pointers method. In the 2 pointer method, 2 pointers are used to iterate through the array values. Both pointers can move to 1 direction only, which ensures that the algorithm works efficiently.

– *Ý tưởng tổng quan của phương pháp 2 con trỏ.* Trong phương pháp 2 con trỏ, 2 con trỏ được sử dụng để lặp qua các giá trị mảng. Cả hai con trỏ chỉ có thể di chuyển theo 1 hướng, điều này đảm bảo thuật toán hoạt động hiệu quả.

Cho 1 mảng $a = \{a[i]\}_{i=0}^{n-1}$. Gọi l, r lần lượt là con trỏ trái & con trỏ phải (left- & right pointers, respectively). Thay đổi l, r để tìm kiếm trên mảng a, có thể kết hợp với sắp xếp trước đó, yêu cầu của bài toán.

Analysis of Algorithm Complexity . The running time of the 2 pointers algorithm depends on the number of steps the right pointer moves. While there is no useful upper bound on how many steps the pointer can move on a *single* turn, we know that the pointer moves a *total* of $O(n)$ steps during the algorithm, because it only moves to the right. Since both the left & right pointers move $O(n)$ steps during the algorithm, the algorithm works in $O(n)$ time.

– *Phân tích Độ phức tạp Thuật toán.* Thời gian chạy của thuật toán 2 con trỏ phụ thuộc vào số bước mà con trỏ phải di chuyển. Mặc dù không có giới hạn trên hữu ích nào về số bước con trỏ có thể di chuyển trong 1 lượt, chúng ta biết rằng con trỏ di chuyển tổng cộng $O(n)$ bước trong suốt thuật toán, vì nó chỉ di chuyển sang phải. Vì cả con trỏ trái và phải đều di chuyển $O(n)$ bước trong suốt thuật toán, thuật toán hoạt động trong thời gian $O(n)$.

Problem 1 (Codeforces/books)**.** *When Valera has got some free time, he goes to the library to read some books. Today he's got t free minutes to read. That's why Valera took n books in the library & for each book he estimated the time he is going to need to read it. Let's number the books by integers from 1 to n. Valera needs a_i minutes to read the ith book.*

Valera decided to choose an arbitrary book with number i & read the books 1 by 1, starting from this book. I.e., he will 1st read book number i, then book number $i+1$, then book number $i+2$ & so on. He continues the process until he either runs out of the free time or finishes the nth book. Valera reads each book up to the end, i.e., he does not start reading the book if he doesn't have enough free time to finish reading it. Print the maximum number of books Valera can read.

Input. *The 1st line contains 2 integers $n \in [10^5], t \in [10^9]$ – the number of books & the number of free minutes Valera's got. The 2nd line contains a sequence of n integers $a_1, a_2, \ldots, a_n$, $a_i \in [10^4]$, $\forall i \in [n]$, where number a_i shows the number of minutes that the boy needs to read the ith book.*

Output. *Print a single integer – the maximum number of books Valera can read.*

Sample.

book.inp	book.out
4 5 3 1 2 1	3
3 3 2 2 3	1

Bài toán 1 (Sách)**.** *Khi Valera có chút thời gian rảnh, anh ấy đến thư viện để đọc sách. Hôm nay anh ấy có t phút rảnh để đọc. Đó là lý do tại sao Valera đã mượn n cuốn sách trong thư viện & với mỗi cuốn sách, anh ấy ước tính thời gian cần thiết để đọc hết. Hãy đánh số các cuốn sách theo số nguyên từ 1 đến n. Valera cần a_i phút để đọc cuốn sách thứ i.*

Valera quyết định chọn 1 cuốn sách bất kỳ có số i & đọc từng cuốn sách một, bắt đầu từ cuốn sách này. Tức là, anh ấy sẽ đọc cuốn sách số i trước, sau đó là cuốn sách số $i+1$, rồi cuốn sách số $i+2$ & cứ tiếp tục như vậy. Anh ấy tiếp tục quá trình này cho đến khi hết thời gian rảnh hoặc đọc hết cuốn sách thứ n. Valera đọc hết mỗi cuốn sách cho đến hết, tức là anh ấy sẽ không bắt đầu đọc cuốn sách nếu không có đủ thời gian rảnh để đọc hết. In ra số lượng sách tối đa mà Valera có thể đọc.

Đầu vào. *Dòng đầu tiên chứa 2 số nguyên $n \in [10^5], t \in [10^9]$ – số lượng sách & số phút miễn phí mà Valera có. Dòng thứ 2 chứa 1 dãy n số nguyên $a_1, a_2, \ldots, a_n$, $a_i \in [10^4]$, $\forall i \in [n]$, trong đó a_i cho biết số phút mà cậu bé cần để đọc cuốn sách thứ i.*

Đầu ra. *In ra 1 số nguyên duy nhất – số lượng sách tối đa mà Valera có thể đọc.*

1st solution: 2 pointers method. We want to find the longest contiguous segment of books that can be read within t minutes. To accomplish this, we can define `left, right` to represent the beginning & end of the segment. Both will start at the beginning of the array. These numbers can be thought of as pointers, hence the name "2 pointers".

– Chúng ta muốn tìm đoạn sách liền kề dài nhất có thể đọc được trong vòng t phút. Để thực hiện điều này, chúng ta có thể định nghĩa `left, right` để biểu diễn điểm đầu & điểm cuối của đoạn sách. Cả hai đều bắt đầu từ đầu mảng. Những con số này có thể được coi là các con trỏ, do đó có tên là "2 con trỏ".

For every value of `left` in increasing order, let's increase `right` until the total time for the segment is maximized while being $\leq t$. `ans` will store the maximum value of `right` $-$ `left` $+ 1$ (segment size) that we have encountered so far. After increasing `left` by 1, the time used decreases, hence the right pointer never has to move leftwards. Since both pointers will move at most n times, the overall time complexity is $O(n)$.

– Với mỗi giá trị của `left` theo thứ tự tăng dần, tăng `right` cho đến khi tổng thời gian cho đoạn thẳng đạt giá trị cực đại trong khi vẫn giữ nguyên $\leq t$. `ans` sẽ lưu trữ giá trị tối đa của `right` $-$ `left` $+ 1$ (kích thước đoạn thẳng) mà chúng ta đã gặp

cho đến nay. Sau khi tăng `left` thêm 1, thời gian sử dụng sẽ giảm xuống, do đó con trỏ phải không bao giờ phải di chuyển sang trái. Vì cả hai con trỏ sẽ di chuyển tối đa n lần, nên độ phức tạp thời gian tổng thể là $O(n)$.

C++:

1. USACO Guide's C++: book: sliding window idea, time complexity $O(n)$.

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, t;
    cin >> n >> t;
    vector<int> a(n);
    for (int& v : a) cin >> v;
    int r = -1, sum = 0, ans = 0;
    for (int l = 0; l < n; sum -= a[l++]) {
        while (r + 1 < n && sum + a[r + 1] <= t) sum += a[++r];
        ans = max(ans, r - l + 1);
    }
    cout << ans << '\n';
}
```

2. NQBH's C++: book: TLE test 9 CodeForces

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

int main() {
    int n, t, count, sum, l, r, ans = 0;
    cin >> n >> t;
    int a[n];
    for (int i = 0; i < n; ++i) cin >> a[i];
    for (l = 0; l < n; ++l) { // count max number of books that can be read from l-th book
        count = 0; // reset count
        sum = 0; // reset sum from point l to pointer r
        for (r = l; r < n; ++r) {
            sum += a[r];
            if (sum > t) break;
            else ++count;
        }
        ans = max(ans, count);
    }
    cout << ans;
}
```

□

Remark 1 (Cf. 2 pointers method vs. sliding method method)**.** *You can visualize these pointers as maintaining a sliding window of books for this problem. – Có thể hình dung những con trỏ này như đang duy trì 1 cửa sổ sách trượt cho vấn đề này.*

Phương pháp 2 con trỏ rất liên quan với phương pháp cửa sổ trượt vì 2 con trỏ này đánh dấu biên trái & biên phải của cửa sổ trượt, còn cửa sổ trượt thì nắm tất cả thông tin về mảng hoặc chuỗi nằm giữa 2 con trỏ & bao gồm cả ở vị trí 2 con trỏ. 1 cách nom na, có thể hiểu phương pháp 2 con trỏ là phần biên của phần nội dung của phương pháp cửa sổ trượt.

Bài toán 2 (Monotone sequence concatenation – Ghép dãy đơn điệu)**.** *Cho 2 dãy số nguyên đã được sắp xếp không giản* $a = \{a_i\}_{i=1}^m, b = \{b_i\}_{i=1}^n$ *lần lượt có* $n, m \in \mathbb{N}$ *phần tử. Ghép chúng thành dãy* $c = \{c_i\}_{i=1}^{m+n}$ *được bố trí theo thứ tự không giảm.*

Constraint. $m, n \in [10^5], a_i, b_j \in \overline{0, 10^9}$, $\forall i \in [m], j \in [n]$.

Sample.

monotone_sequence_concatenation.inp	monotone_sequence_concatenation.out
5 6 1 3 6 8 10 2 6 7 12 14 15	1 2 3 6 6 7 8 10 12 14 15

Solution. Vì 3 dãy a, b, c không giảm, i.e., $a_1 \le a_2 \le \cdots a_m, b_1 \le b_2 \le \cdots \le b_n, c_1 \le c_2 \le \cdots \le c_{m+n}$, nên $c_1 = \min\{a_1, b_1\}$. Tại mỗi thời điểm, phần tử tiếp theo của dãy c sẽ là phần tử nhỏ nhất trong các phần tử chưa được đưa vào dãy c, nên bằng cách so sánh phần tử nhỏ nhất chưa được chọn ở dãy a & phần tử nhỏ nhất chưa được chọn ở dãy b, phần tử nhỏ hơn sẽ được chọn vào dãy c. Ban đầu, lúc dãy c chưa có phần tử nào, i.e., $c = \emptyset$ thì a_1, b_1 lần lượt là phần tử nhỏ nhất chưa được chọn của 2 dãy a, b. Tại mỗi thời điểm, nếu đưa phần tử a_i vào dãy c thì phần tử nhỏ nhất chưa được chọn của dãy a sẽ là a_{i+1}, còn nếu đưa phần tử b_j vào dãy c thì phần tử nhỏ nhất chưa được chọn của dãy b sẽ là b_{j+1}.

C++:

1. NQBH's C++: monotone sequence concatenation:

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

int main() {
    int m, n;
    cin >> m >> n;
    int a[m], b[n], c[m + n];
    for (int i = 0; i < m; ++i) cin >> a[i];
    for (int i = 0; i < n; ++i) cin >> b[i];
    int idx_a = 0, idx_b = 0; // 2 pointers
    for (int i = 0; i < m + n; ++i) {
        if (a[idx_a] <= b[idx_b]) {
            c[i] = a[idx_a];
            ++idx_a;
        }
        else {
            c[i] = b[idx_b];
            ++idx_b;
        }
    }
    for (int i = 0; i < m + n; ++i) cout << c[i] << ' ';
}
```

or shorter:

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

int main() {
    int m, n;
    cin >> m >> n;
    int a[m], b[n], c[m + n];
    for (int i = 0; i < m; ++i) cin >> a[i];
    for (int i = 0; i < n; ++i) cin >> b[i];
    int idx_a = 0, idx_b = 0; // 2 pointers
    for (int i = 0; i < m + n; ++i)
        if (a[idx_a] <= b[idx_b]) c[i] = a[idx_a++];
        else c[i] = b[idx_b++];
    for (int i = 0; i < m + n; ++i) cout << c[i] << ' ';
}
```

□

1.1 Subarray sum – Tổng các phần tử của mảng con

Resources – Tài nguyên.

1. Geeks for Geeks/subarray with given sum.

Problem 2 (Subarray sum on $\mathbb{N}$ – Tổng mảng con trên $\mathbb{N}$)**.** *Given an array $\{a_i\}_{i=1}^n \subset \mathbb{N}^\star$ (1-based indexing) of n positive integers & a target sum $x \in \mathbb{N}^\star$. Find a subarray whose sum is x or report that there is no such subarray.*

– Cho 1 mảng $\{a_i\}_{i=1}^n \subset \mathbb{N}^\star$ (đánh chỉ số dựa trên 1) gồm n số nguyên dương & 1 tổng đích x. Tìm 1 mảng con có tổng là x hoặc báo cáo rằng không có mảng con nào như vậy.

We can reformulate the problem mathematically as follows:

$$\text{Find } l, r \in [n],\ l \le r \text{ s.t. } \sum_{i=l}^{r} a_i = x.$$

Solution. This problem can be solved in $O(n)$ time (i.e., linear time) by using the 2 pointers method. The idea is to maintain pointers that point to the 1st & last value of a subarray. On each turn, the left pointer moves 1 step to the right, & the right pointer moves to the right as long as the resulting subarray sum is at most x. If the sum becomes exactly x, a solution has been found.

– Bài toán này có thể được giải quyết trong thời gian $O(n)$ (tức là thời gian tuyến tính) bằng cách sử dụng phương pháp 2 con trỏ. Ý tưởng là duy trì các con trỏ trỏ đến giá trị đầu tiên & cuối cùng của 1 mảng con. Mỗi lần dịch chuyển, con trỏ trái di chuyển 1 bước sang phải, & con trỏ phải di chuyển sang phải miễn là tổng mảng con thu được không quá x. Nếu tổng bằng đúng x, thì đã tìm được nghiệm.

C++:

1. NQBH's C++: subarray sum:

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

int main() {
    int n, x;
    cin >> n >> x;
    int a[n];
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
        if (a[i] == x) {
            cout << i << ' ' << i;
            return 0;
        }
    }
    int l = 0, r = 1; // left- & right pointers
    int sum = a[0] + a[1];
    while (sum != x) {
        if (sum < x && r == n - 1) {
            cout << "Impossible";
            return 0;
        }
        if (sum < x && r < n - 1) sum += a[++r];
        if (sum > x && l < n - 1) sum -= a[l++];
    }
    cout << l << ' ' << r;
}
```

□

A little bit different version of the above problem:

Problem 3 (Subarray sum – Tổng các phần tử của mảng con)**.** *Given a* 1*-based indexing array* `arr[]` *of nonnegative integers & an integer* x*. Return the left & right indexes (*1*-based indexing) of a subarray whose sum of its elements is equal to* x*. In case of multiple satisfying subarrays, return the subarray indexes which come 1st on moving from left to right. If no such subarray exists, return* -1*.*

– Cho 1 mảng lập chỉ mục dựa trên 1 `arr[]` *gồm các số nguyên không âm & 1 số nguyên* x*. Trả về các chỉ mục trái & phải (lập chỉ mục dựa trên* 1*) của 1 mảng con có tổng các phần tử bằng* x*. Trong trường hợp có nhiều mảng con thỏa mãn, trả về các chỉ mục mảng con đứng đầu tiên khi di chuyển từ trái sang phải. Nếu không tồn tại mảng con nào như vậy, trả về* -1*.*

Sample.

subarray_sum.inp	subarray_sum.out
8 23 15 2 4 8 9 5 10 23	2 5
6 7 1 10 4 0 3 5	3 5
2 0 1 4	-1

1st solution. **Naive approach: using nested loop $O(n^2)$ time & $O(1)$ space**: Use a nested loop where the outer loop picks a starting element, & the inner loop calculates the cumulative sum of elements starting from this element. For each starting element, the inner loop iterates through subsequent elements & adding each element to the cumulative sum until the given sum is found or the end of the array is reached. If at any point the cumulative sum equals the given sum, then return starting & ending indices (1-based). If no such sub-array is found after all iterations, then return -1.

– **Cách tiếp cận ngây thơ: sử dụng vòng lặp lồng nhau $O(n^2)$ thời gian & $O(1)$ không gian**: Sử dụng vòng lặp lồng nhau, trong đó vòng lặp ngoài chọn 1 phần tử bắt đầu, & vòng lặp trong tính tổng tích lũy của các phần tử bắt đầu từ phần tử này. Với mỗi phần tử bắt đầu, vòng lặp trong sẽ lặp qua các phần tử tiếp theo & cộng từng phần tử vào tổng tích lũy cho đến khi tìm được tổng cho trước hoặc đến cuối mảng. Nếu tại bất kỳ điểm nào, tổng tích lũy bằng tổng cho trước, thì trả về chỉ số bắt đầu & kết thúc (dựa trên 1). Nếu không tìm thấy mảng con nào như vậy sau tất cả các lần lặp, thì trả về -1.

C++:

1. G4G's C++: https://www.geeksforgeeks.org/dsa/find-subarray-with-given-sum/:

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

// function to find a continuous subarray whose sum = a given number
vector<int> subarray_sum_naive(vector<int> arr, int target) {
    vector<int> res;
    int n = arr.size();
    for (int s = 0; s < n; ++s) { // pick a staarting point for a subarray
        int curr = 0; // current sum from starting & ending indices
        // consider all ending points for the picked starting point
        for (int e = s; e < n; ++e) {
            curr += arr[e];
            if (curr == target) {
                res.push_back(s + 1);
                res.push_back(e + 1);
                return res;
            }
        }
    }
    return {-1}; // if no subarray is found
}

int main() {
    int n, x;
    cin >> n >> x;
    vector<int> a(n);
    for (int& v : a) cin >> v;
    vector<int> res = subarray_sum_naive(a, x);
    for (int ele : res) cout << ele << " ";
}
```

C#: G4G's C#: subarray sum: https://www.geeksforgeeks.org/dsa/find-subarray-with-given-sum/:

```cpp
using System;
using System.Collections.Generic;

class GfG {
    // function to find a continuous subarray which adds up to a given number
    static List<int> subarray_sum(int[] arr, int target) {
        List<int> res = new List<int>();
        int n = arr.Length;
        for (int s = 0; s < n; ++s) { // pick a starting point for a subarray
            int curr = 0;
            // consider all ending points for the picked starting point
            for (int e = s; e < n; ++e) {
                curr += arr[e];
                if (curr == target) {
                    res.Add(s + 1);
                    res.Add(e + 1);
                    return res;
                }
            }
        }
        res.Add(-1); // if no satisfying subarray is found
        return res;
```

```
    }

    static void Main() {
        int[] arr = {15, 2, 4, 8, 9, 5, 10, 23};
        int target = 23;
        List<int> res = subarray_sum(arr, target);
        foreach (var ele in res)
        Console.Write(ele + " ");
    }
}
```

Java: G4G's Java: subarray sum: https://www.geeksforgeeks.org/dsa/find-subarray-with-given-sum/:

```
import java.util.ArrayList;
import java.util.List;

class GfG {
    // function to find a continuous subarray whose sum = a given number
    static ArrayList<Integer> subarray_sum(int[] arr, int target) {
        ArrayList<Integer> res = new ArrayList<>();
        int n = arr.length;
        for (int s = 0; s < n; ++s) { // pick a starting point for a subarray
            int curr = 0;
            // consider all ending points for the picked starting point
            for (int e = s; e < n; ++e) {
                curr += arr[e];
                if (curr == target) {
                    res.add(s + 1);
                    res.add(e + 1);
                    return res;
                }
            }
        }
        res.add(-1); // if no satisfying subarray is found
        return res;
    }

    public static void main(String[] args) {
        int[] arr = {15, 2, 4, 8, 9, 5, 10, 23};
        int target = 23;
        ArrayList<Integer> res = subarray_sum(arr, target);
        for (int ele : res)
        System.out.print(ele + " ");
    }
}
```

Java Script: G4G's Java Script: subarray sum: https://www.geeksforgeeks.org/dsa/find-subarray-with-given-sum/:

```
// function to find a continuous subarray whose sum = a given number
function subarraySum(arr, target) {
    let res = [];
    let n = arr.length;

    // Pick a starting point for a subarray
    for (let s = 0; s < n; s++) {
        let curr = 0;

        // Consider all ending points
        // for the picked starting point
        for (let e = s; e < n; e++) {
            curr += arr[e];
            if (curr === target) {
                res.push(s + 1);
                res.push(e + 1);
                return res;
```

```
            }
        }
    }
    // If no subarray is found
    return [-1];
}

// Driver Code
let arr = [15, 2, 4, 8, 9, 5, 10, 23];
let target = 23;
let res = subarraySum(arr, target);

console.log(res.join(' '));
```

Python:

1. G4G's Python: subarray sum: https://www.geeksforgeeks.org/dsa/find-subarray-with-given-sum/:

```
# function to find a continuous subarray which adds up to a given number
def subarray_sum(arr, target):
    res = []
    n = len(arr)
    for s in range(n): # pick a starting point for a subarray
        curr = 0 # initialize current calculated sum
        for e in range(s, n): # consider all ending points for the picked starting point
            curr += arr[e]
            if curr == target:
                res.append(s + 1)
                res.append(e + 1)
                return res
    return [-1] # if no subarray is found

if __name__ == "__main__":
    n, x = map(int, input().split())
    arr = list(map(int, input().split()))
    res = subarray_sum(arr, x)
    for ele in res:
        print(ele, end = " ")
```

□

2nd solution. **Sliding Window: $O(n)$ time & $O(1)$ space.** As we know that all the elements in the given subarray are positive, if a subarray has sum greater than the given sum then there is no possibility that adding elements to the current subarray will be equal to the given sum. Hence the idea is to use a similar approach to a sliding window.

- Start with an empty window.
- Add elements to the window while the current sum is less than the target sum.
- If the sum is greater than the target sum, remove elements from the start of the current window.
- If the current sum is equal to the target sum, return the result.

– **Cửa sổ trượt: $O(n)$ thời gian & $O(1)$ không gian.** Như chúng ta đã biết, tất cả các phần tử trong mảng con đã cho đều dương, nếu 1 mảng con có tổng lớn hơn tổng đã cho thì không có khả năng việc thêm các phần tử vào mảng con hiện tại sẽ bằng tổng đã cho. Do đó, ý tưởng là sử dụng 1 cách tiếp cận tương tự như cửa sổ trượt.

- Bắt đầu với 1 cửa sổ trống.
- Thêm các phần tử vào cửa sổ khi tổng hiện tại nhỏ hơn tổng mục tiêu.
- Nếu tổng lớn hơn tổng mục tiêu, xóa các phần tử khỏi đầu cửa sổ hiện tại.
- Nếu tổng hiện tại bằng tổng mục tiêu, trả về kết quả.

C++:

1. G4G's C++: subarray sum by sliding window:

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

vector<int> subarray_sum_sliding_window(vector<int>& arr, int target) {
    int s = 0, e = 0; // initialize window
    vector<int> res;
    int curr = 0; // current calculated sum
    for (size_t i = 0; i < arr.size(); ++i) {
        curr += arr[i];
        // if current sum becomes >=, set end & try adjusting start
        if (curr >= target) {
            e = i;
            // while current is greater, remove starting elements of current window
            while (curr > target && s < e)
            curr -= arr[s++];
            if (curr == target) { // if found a satisfying subarray
                res.push_back(s + 1);
                res.push_back(e + 1);
                return res;
            }
        }
    }
    return {-1}; // if no satisfying subarray is found
}

int main() {
    int n, x;
    cin >> n >> x;
    vector<int> a(n);
    for (int& v : a) cin >> v;
    vector<int> res = subarray_sum_sliding_window(a, x);
    for (int ele : res) cout << ele << " ";
}
```

□

Problem 4 (Subarray sum on $\mathbb{Z}$)**.** *Given an unsorted array of integers $\{a_i\}_{i=1}^n \subset \mathbb{Z}$, find a subarray that adds to a given number $x \in \mathbb{Z}$. (a) If there is more than 1 satisfying subarray, print any of them. (b) Print all satisfying subarrays.*

Sample.

subarray_sum_negative.inp	subarray_sum_negative.out
6 33 1 4 20 3 10 5	2 4
5 −10 2 12 −2 −20 10	1 3
6 20 −10 0 2 −2 −20 10	−1

Resources – Tài nguyên.

- Geeks for Geeks/subarray with given sum – handles negative numbers.

1st solution: Naive approach: using nested loop: $O(n^2)$ time & $O(1)$ space. □

2nd solution: Prefix Sum & Hash Map: $O(n)$ time & $O(n)$ space. □

3rd solution: Hashing + prefix sum: $O(n)$ time & $O(n)$ space – Băm + tiền tố tổng: $O(n)$ thời gian & $O(n)$ không gian. The above sliding-window solution does not work for arrays with negative numbers. To handle all cases, we use hashing & prefix sum. The idea is to store the sum of elements of every prefix of the array in a hashmap, i.e., every index stores the sum of elements up to that index hashmap. So to check if there is a subarray with sum equal to the target sum, check for every index $i \in [n]$, & sum up to that index as `curr_sum`. If there is a prefix with a sum equal to `curr_sum - target`, then the subarray with the given sum is found.

– Giải pháp cửa sổ trượt ở trên không hoạt động với các mảng có số âm. Để xử lý mọi trường hợp, chúng tôi sử dụng băm & tiền tố tổng. Ý tưởng là lưu trữ tổng các phần tử của mọi tiền tố của mảng trong 1 hashmap, tức là, mỗi chỉ mục lưu trữ tổng

các phần tử lên đến hashmap chỉ mục đó. Vì vậy, để kiểm tra xem có mảng con nào có tổng bằng tổng mục tiêu hay không, kiểm tra mọi chỉ mục $i \in [n]$, & tổng đến chỉ mục đó là `curr_sum`. Nếu có 1 tiền tố có tổng bằng `curr_sum - target`, thì mảng con có tổng đã cho sẽ được tìm thấy. □

Remark 2 (Tương quan của bài toán subarray sum với các thuật toán tìm đường đi ngắn nhất trên đồ thị (algorithms for shortest path problems on graphs)). *Thuật toán Dijkstra cũng chỉ hoạt động được cho các đồ thị có trọng số không âm, không hoạt động được với các đồ thị có lẫn trọng số âm & trọng số không âm. Chỉ có thuật toán Bellman–Ford mới hoạt động cho đồ thị với trọng số thực tùy ý.*

Another problem that can be solved using the 2 pointers method is the following 2SUM problem:

1.2 2SUM & 3SUM problem – Bài toán tổng 2 & 3 phần tử

2 pointers is really an easy & effective technique that is typically used for 2 sum in sorted arrays, closest 2 sum, 3 sum, 4 sum, trapping rain water & many other popular interview questions. E.g., given a non-sorted or sorted array `arr` (w.l.o.g., sorted in ascending order by default) & a `target`, find if there exists any pair of elements $(\texttt{arr}[i], \texttt{arr}[j])$ such that their sum is equal to the target.

– 2 con trỏ thực sự là 1 kỹ thuật dễ dàng & hiệu quả, thường được sử dụng cho tổng 2 trong các mảng đã sắp xếp, tổng 2 gần nhất, tổng 3, tổng 4, bẫy nước mưa & nhiều câu hỏi phỏng vấn phổ biến khác. Ví dụ: cho 1 mảng `arr` đã sắp xếp hoặc chưa sắp xếp (không mất tính tổng quát, mặc định được sắp xếp theo thứ tự tăng dần) & 1 `target`, hãy tìm xem có tồn tại cặp phần tử nào $(\texttt{arr}[i], \texttt{arr}[j])$ sao cho tổng của chúng bằng với mục tiêu hay không.

Question 1 (Output of 2sum problem). *How deeply do we want to know or obtain when we solve* 2SUM *problem?*

Bài toán 2SUM có nhiều cách phát biểu. Sau đây là 2 cách: 1 kiểu thuật toán & 1 kiểu toán học.

Problem 5 (2SUM). *Given an array of $n \in \mathbb{N}^\star$ numbers & a target sum $x \in \mathbb{N}$, find 2 array values such that their sum is x, or report that no such values exist.*

– Cho 1 mảng gồm $n \in \mathbb{N}^\star$ số & tổng mục tiêu $x \in \mathbb{N}$, tìm 2 giá trị mảng sao cho tổng của chúng là x hoặc báo cáo rằng không tồn tại giá trị nào như vậy.

$$\boxed{\text{Given } x \in \mathbb{N} \text{ \& an array } \{a_i\}_{i=1}^n \subset \mathbb{N}, \text{ find } i, j \in [n],\ i \neq j \text{ s.t. } a_i + a_j = x.}$$

Problem 6 (CSES Problem Set/sum of 2 values). *You are given an array of $n \in \mathbb{N}^\star$ integers. Find 2 values (at distinct positions) whose sum is x.*

Input. *The 1st input line has 2 integers $n, x \in \mathbb{N}^\star$: the array size & the target sum. The 2nd line has n integers $a_1, a_2, \ldots, a_n$: the array values.*

Output. *Print 2 integers: the positions of the values. If there are several solutions, you may print any of them. If there are no solutions, print* `IMPOSSIBLE`.

Constraints. $n \in [2 \cdot 10^5], x, a_i \in [10^9]$, $\forall i \in [n]$.

Sample.

sum_2_value.inp	sum_2_value.out
4 8 2 7 5 1	2 4

Bài toán 3 (Tổng 2 giá trị). *Bạn được cho 1 mảng gồm $n \in \mathbb{N}^\star$ số nguyên. Tìm 2 giá trị (ở các vị trí phân biệt) có tổng bằng x.*

Đầu vào. *Dòng đầu vào thứ nhất chứa 2 số nguyên $n, x \in \mathbb{N}^\star$: kích thước mảng & tổng mục tiêu. Dòng thứ hai chứa n số nguyên $a_1, a_2, \ldots, a_n$: các giá trị mảng.*

Đầu ra. *In 2 số nguyên: vị trí của các giá trị. Nếu có nhiều nghiệm, bạn có thể in bất kỳ nghiệm nào. Nếu không có nghiệm nào, in* `IMPOSSIBLE`.

Ràng buộc. $n \in [2 \cdot 10^5], x, a_i \in [10^9]$, $\forall i \in [n]$.

1st solution: Naive approach $O(n^2)$ time. C++:

1. NQBH's C++: sum of 2 values: naive approach:

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

int main() {
    int n, x;
    cin >> n >> x;
    int a[n];
```

```
    for (int i = 0; i < n; ++i) cin >> a[i];
    for (int i = 0; i < n - 1; ++i)
        for (int j = i + 1; j < n; ++j)
            if (a[i] + a[j] == x) {
                cout << i + 1 << " " << j + 1;
                return 0;
            }
    cout << "IMPOSSIBLE";
}
```

This naive program only passes 18/27 AC, 9/27 TLE on CSES.

Worst-case analysis. +++

□

Solution. To solve the problem, we 1st sort the array values in increasing order. After that, we iterate through the array using 2 pointers. The left pointer starts at the 1st value & moves 1 step to the right on each turn. The right pointer begins at the last value & always moves to the left until the sum of the left & right value is at most x. If the sum is exactly x, a solution has been found.

– Để giải bài toán, trước tiên chúng ta sắp xếp các giá trị mảng theo thứ tự tăng dần. Sau đó, chúng ta duyệt qua mảng bằng 2 con trỏ. Con trỏ bên trái bắt đầu từ giá trị đầu tiên & dịch chuyển 1 bước sang phải mỗi lần. Con trỏ bên phải bắt đầu từ giá trị cuối cùng & luôn dịch chuyển sang trái cho đến khi tổng của giá trị bên trái & bên phải lớn nhất là x. Nếu tổng bằng đúng x, thì bài toán đã tìm được nghiệm.

The running time of the algorithm is $O(n \log_2 n)$, because it 1st sorts the array in $O(n \log_2 n)$ time, & then both pointers move $O(n)$ steps.

It is also possible to solve the problem in another way in $O(n \log_2 n)$ time using binary search. In such a solution, we iterate through the array & for each array value, we try to find another value that yields the sum x. This can be done by performing n binary searches, each of which takes $O(\log_2 n)$ time. □

A more difficult problem is the 3SUM problem that asks to find 3 array values whose sum is x. Using the idea of the above algorithm, this problem can be solved in $O(n^2)$ time.

– 1 bài toán khó hơn là bài toán 3SUM, yêu cầu tìm 3 giá trị mảng có tổng bằng x. Sử dụng ý tưởng của thuật toán trên, bài toán này có thể được giải trong thời gian $O(n^2)$.

Problem 7 (3SUM)**.** *Given an array of $n \in \mathbb{N}^\star$ numbers & a target sum $x \in \mathbb{N}$, find 3 array values such that their sum is x, or report that no such values exist.*

– Cho 1 mảng gồm $n \in \mathbb{N}^\star$ số & tổng mục tiêu $x \in \mathbb{N}$, tìm 3 giá trị mảng sao cho tổng của chúng là x hoặc báo cáo rằng không tồn tại giá trị nào như vậy.

1st solution: Naive approach. C++:

1. NQBH's C++: 3sum: naive approach:

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

int main() {
    int n, x;
    cin >> n >> x;
    int a[n];
    for (int i = 0; i < n; ++i) cin >> a[i];
    for (int i = 0; i < n - 2; ++i)
        for (int j = i + 1; j < n - 1; ++j)
            for (int k = j + 1; k < n; ++k)
                if (a[i] + a[j] + a[k] == x) {
                    cout << i + 1 << " " << j + 1 << " " << k + 1;
                    return 0;
                }
    cout << "IMPOSSIBLE";
}
```

Worst-case analysis. +++

□

Problem 8 (k-sum, $\star$)**.** *Given an array of $n \in \mathbb{N}^\star$ numbers & a target sum $x \in \mathbb{N}$, find $k \in \mathbb{N}^\star$ array values such that their sum is x, or report that no such values exist.*

– Cho 1 mảng gồm $n \in \mathbb{N}^\star$ số & tổng mục tiêu $x \in \mathbb{N}$, tìm $k \in \mathbb{N}^\star$ giá trị mảng sao cho tổng của chúng là x hoặc báo cáo rằng không tồn tại giá trị nào như vậy.

Solution: Use bitmask. +++ □

2 Problems: 2 Pointers Method – Bài Tập: Phương Pháp Phương Pháp 2 Con Trỏ

Problem 9 (Subarray Sums I).

Problem 10 (Sum of Three Values).

Problem 11 (Paired Up).

Problem 12 (Cellular Network).

Problem 13 (They Are Everywhere).

Problem 14 (Quiz Master).

Problem 15 (Diamond Collector).

Problem 16 (Sleepy Cow Herding).

Problem 17 (An impassioned circulation of affection).

3 Some Extensions of 2 Pointers Method – Vài Mở Rộng của Phương Pháp 2 Con Trỏ

4 Miscellaneous