Longest Increasing Subsequences (LIS) & Longest Decreasing Subsequences (LDS) – Dãy Con Tăng Dài Nhất & Dãy Con Giảm Dài Nhất

Nguyễn Quản Bá Hồng*

Ngày 21 tháng 7 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:

- *Longest Increasing Subsequences & Longest Decreasing Subsequences (LDS) – Dãy Con Tăng Dài Nhất & Dãy Con Giảm Dài Nhất.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/LIS/NQBH_LIS.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/LIS/NQBH_LIS.tex.
- .
  PDF: URL: .pdf.
  TEX: URL: .tex.

Bài viết này giới thiệu về dãy con tăng dài nhất (longest increasing subsequence, abbr., LIS) & dãy con giảm dài nhất (longest decreasing subsequence, abbr., LDS) cùng các bài toán Lập Trình Thi Đấu (Competitive Programming, abbr., CP) liên quan đến 2 khái niệm này.

Mục lục

1 Longest Increasing Subsequence Problem – Bài Toán Dãy Con Tăng Dài Nhất

Resources – Tài nguyên.

1. Wikipedia/longest increasing subsequence.
2. Geeks4Geeks/Longest Increasing Subsequence (LIS).
3.

In computer science, the *longest increasing subsequence* problem aims to find a subsequence of a given sequence $\{a_i\}_{i=1}^n$ in which the subsequence's elements are sorted in an ascending order & in which the subsequence is as long as possible. This subsequence is not necessarily contiguous or unique. The longest increasing subsequence are studied in the context of various disciplines related to mathematics, including algorithmics, random matrix theory, representation theory, & physics. The longest increasing subsequence problem is solvable in time $O(n \log n)$, where n denotes the length of the input sequence.

*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.

– Trong khoa học máy tính, bài toán *longest increasing subsequence* nhằm mục đích tìm một dãy con của một dãy $\{a_i\}_{i=1}^n$ cho trước, trong đó các phần tử của dãy con được sắp xếp theo thứ tự tăng dần & sao cho dãy con này dài nhất có thể. Dãy con này không nhất thiết phải liền kề hoặc duy nhất. Dãy con tăng dài nhất được nghiên cứu trong bối cảnh của nhiều ngành toán học khác nhau, bao gồm thuật toán, lý thuyết ma trận ngẫu nhiên, lý thuyết biểu diễn và vật lý. Bài toán dãy con tăng dài nhất có thể giải được trong thời gian $O(n \log n)$, với n biểu thị độ dài của dãy đầu vào.

Definition 1 (Numerical sequence – dãy số, [WS10], p. 25)**.** *A* sequence *is a set of numbers* $u_1, u_2, \ldots$ *in a definite order of arrangement (i.e., a* correspondence *with the natural numbers or a subset thereof) & formed according to a definite rule. Each number in the sequence is called a* term*;* u_n *is called the* nth term*. The sequence is called* finite *or* infinite *according as there are or are not a finite number of terms. The sequence* $u_1, u_2, \ldots$ *is also designated briefly by* $\{u_n\}$*.*

Có thể hiểu khái niệm dãy (sequence) ở đây 1 cách tổng quát hơn là 1 dãy các đối tượng Toán học hoặc Tin học, e.g., dãy số phức $\{a_n\}_{n=1}^{\infty}$ là 1 dãy gồm các số $a_n \in \mathbb{C}$, $\forall n = 1, 2, \ldots$, dãy các hàm số thực $\{f_n\}_{n=1}^{\infty}$ là 1 dãy gồm các hàm số $f_n : \mathbb{R} \to \mathbb{R}$, $\forall n = 1, 2, \ldots$, hay dãy các dãy $\{\{a_{m,n}\}_{n=1}^{\infty}\}_{m=1}^{\infty}$ tức 1 dãy gồm các phần tử của dãy lại là các dãy số $\{a_{m,n}\}_{n=1}^{\infty}$, $\forall m = 1, 2, \ldots$ Trước hết, ta tập trung vào khái niệm dãy đơn giản nhất: dãy số – numerical sequence, trước khi đến với khái niệm *hội tụ đều* của dãy hàm (uniform convergence of sequences of functions).

Bài toán 1 (Programming: Nhập xuất dãy số)**.** *Viết chương trình* C/C++, Pascal, Python, Java *để nhập vào 1 dãy số nguyên, thực, phức.*

Solution. C: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/C/basic_1D_array.c.

```
#include <stdio.h>
#define N 50 // const int N = 50

void array_int_input(int a[], int& n) {
    while (1) {
        printf("The number of elements needed to use (<= %d): ", N);
        scanf("%d", &n);
        if ((n < 0) || (n > N)) printf("Wrong input, re-input ...\n");
        else break;
    }
    for (int i = 0; i < n; ++i) {
        printf("a[%d] = ", i);
        scanf("%d", &a[i]);
    }
}

void array_int_output(int a[], int n) {
    for (int i = 0; i < n; ++i) printf("%d ", a[i]);
}

bool insert(int a[], int &n, int x, int k) {
    int (k < 0 || k > n) return false;
    for (int i = n - 1; i >= k; --i) a[i + 1] = a[i];
    a[k] = x;
    ++n;
    return true;
}

void remove(int a[], int &n, int k) {
    int (k < 0 || k > n) return;
    a[k] = a[n - 1];
    --n;
}

void remove_preserving_order(int a[], int &n, int k) {
    if (k < 0 || k > n) return;
    for (int i = k; i < n - 1; ++i) a[i] = a[i + 1];
    --n;
}

void split_even_odd(int a[], int n, int even[], int &k, int odd[], int &h) {
    k = h = 0;
    for (int i = 0; i < n; ++i)
    if (a[i] % 2 == 0) even[k++] = a[i]; // <=> even[k] = a[i]; ++k;
```

```
45      else odd[h++] = a[i]; // <=> odd[h] = a[i]; ++h;
46
47  }
48
49  void split_half(int a[], int n, int b[], int &k, int c[], int &h) {
50      int i;
51      h = k = 0;
52      for (i = 0; i < n / 2; ++i) b[k++] = a[i];
53      for (i = n / 2; i < n; ++i) c[h++] = a[i];
54  }
55
56  void array_int_concatenation(int a[], int &n, int b[], int k, int c[], int h) {
57      int i;
58      n = 0;
59      for (i = 0; i < k; ++i) a[n++] = b[i];
60      for (i = 0; i < h; ++i) a[n++] = c[i];
61  }
62
63  void array_int_concatenation_zigzag(int a[], int &n, int b[], int k, int c[], int h) {
64      int i, j;
65      i = j = n = 0;
66      while (i < k && j < h) {
67          if (n % 2 == 0) a[n++] = b[i++];
68          else a[n++] = c[j++];
69      }
70      while (i < k) a[n++] = b[i++];
71      while (j < h) a[n++] = c[j++];
72  }
73
74  int main() {
75      // initialization 1D array
76      /*
77      int a[5] = {1, 3, 5, 7, 9};
78      int a[5] = {1, 3, 5};
79      int a[5] = {0};
80      int a[] = {1, 3, 5}, n = sizeof(a) / sizeof(a[0]);
81      */
82      int b[N], m;
83      array_int_input(b, m);
84      array_int_output(b, m);
85      return 1;
86  }
```

□

Định nghĩa 1 (Dãy con). *Cho 1 dãy số* $\{a_i\}_{i=1}^n$*, 1* dãy con *của dãy số* $\{a_i\}_{i=1}^n$ *là 1 dãy số có dạng* $\{a_{i_j}\}_{j=1}^k$ *với* $k \in [n]$*,* $i_j \in [n]$*,* $\forall j \in [k]$*, & dãy chỉ số* $\{i_j\}_{j=1}^k$ *là dãy tăng ngặt, i.e.,* $1 \le i_1 < i_2 < \cdots < i_k \le n$*.*

Bài toán 2 (Programming: Trích dãy con). *Viết chương trình* C/C++, Pascal, Python, Java *để nhập vào 1 dãy số nguyên, thực, hoặc phức* $\{a_i\}_{i=1}^n$ *& 1 dãy chỉ số* $\{i_j\}_{j=1}^k$ *tăng ngặt, rồi xuất dãy con* $\{a_{i_j}\}_{i=1}^n$ *tương ứng của dãy* $\{a_i\}_{i=1}^n$*.*

Solution. C++: NQBH's C++: subsequence:

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       int n, k; // n: length of original sequence, k: length of a subsequence
6       cin >> n >> k;
7       int a[n], b[k]; // a: original sequence of length n, b: a subsequence of length k
8       for (int i = 0; i < n; ++i) cin >> a[i];
9       for (int i = 0; i < k; ++i) cin >> b[i];
10      cout << "Original sequence a:";
11      for (int i = 0; i < n; ++i) cout << " " << a[i];
12      cout << '\n';
```

```
    cout << "Index sequence b:";
    for (int i = 0; i < k; ++i) cout << " " << b[i];
    cout << '\n';
    cout << "Subsequence a[b[i]]:";
    for (int i = 0; i < k; ++i) cout << " " << a[b[i]];
}
```

□

Định nghĩa 2 (Dãy con tăng (ngặt)). *Cho 1 dãy số* $\{a_i\}_{i=1}^n$.

(a) 1 dãy con tăng (chặt/ngặt) *của* $\{a_i\}_{i=1}^n$ *là 1 dãy số có dạng* $\{a_{i_j}\}_{j=1}^k$ *với* $k \in [n]$, $i_j \in [n]$, $\forall j \in [k]$, *& cả dãy chỉ số* $\{i_j\}_{j=1}^k$ *lần dãy* $\{a_{i_j}\}_{j=1}^k$ *đều là dãy tăng ngặt, i.e.,* $1 \le i_1 < i_2 < \cdots < i_k \le n$, *&* $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$.

(b) 1 dãy con giảm (chặt/ngặt) *của* $\{a_i\}_{i=1}^n$ *là 1 dãy số có dạng* $\{a_{i_j}\}_{j=1}^k$ *với* $k \in [n]$, $i_j \in [n]$, $\forall j \in [k]$, *dãy chỉ số* $\{i_j\}_{j=1}^k$ *lần dãy* $\{a_{i_j}\}_{j=1}^k$ *là 1 dãy tăng ngặt, i.e.,* $1 \le i_1 < i_2 < \cdots < i_k \le n$, *nhưng* $\{a_{i_j}\}_{j=1}^k$ *là dãy giảm, i.e.,* $a_{i_1} > a_{i_2} > \cdots > a_{i_k}$.

Ký hiệu $\mathrm{IS}(a), \mathrm{DS}(a), \mathrm{NIS}(a), \mathrm{NDS}(a)$ *lần lượt là 4 tập hợp các dãy con tăng ngặt, giảm ngặt, không tăng, không giảm của dãy số thực* $a = \{a_i\}_{i=1}^n$.

Định nghĩa 3 (Dãy con tăng dài nhất). Dãy con tăng dài nhất *của 1 dãy số thực* $a = \{a_i\}_{i=1}^n$ *là dãy số có dạng* $\{a_{i_j}\}_{j=1}^k$, *với dãy chỉ số* $\{i_j\}_{j=1}^k \subset \mathbb{N}^\star$ *là 1 dãy tăng ngặt, sao cho k lớn nhất có thể.*

Problem 1. *Given an array* `arr[]` $= \{a_i\}_{i=1}^n$ *of size* $n \in \mathbb{N}^\star$, *the task is to find the length of the Longest Increasing Subsequence(s) (LIS), i.e., the longest possible subsequence in which the elements of the subsequence are sorted in increasing order.*

– Với một mảng `arr[]` $= \{a_i\}_{i=1}^n$ *có kích thước* $n \in \mathbb{N}^\star$, *nhiệm vụ là tìm độ dài của Dãy con tăng dài nhất (LIS), tức là dãy con dài nhất có thể trong đó các phần tử của dãy con được sắp xếp theo thứ tự tăng dần.*

Lemma 1. *(a) If a sequence* $a = \{a_i\}_{i=1}^n$ *is strictly increasing, then its longest increasing subsequence is itself, i.e.,* $\mathrm{LIS}(a) = a$.

(b) If a sequence $a = \{a_i\}_{i=1}^n$ *is non-increasing, then its longest increasing subsequences are each of its elements.*

Example 1 (Binary Van de Corput sequence – Chuỗi Van de Corput nhị phân). *In the 1st* 16 *terms of the binary Van de Corput sequence* $0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$, *1 of the longest increasing subsequence is* $0, 2, 6, 9, 11, 15$. *This subsequence has length* 6, *the input sequence has no* 7*-member increasing subsequences. The longest increasing subsequence in this example is not the only solution, e.g.,* $0, 4, 6, 9, 11, 15$; $0, 2, 6, 9, 13, 15$, *&* $0, 4, 6, 9, 13, 15$ *are other increasing subsequences of equal length in the same input sequence.*

– Trong 16 *số hạng đầu tiên của dãy Van de Corput nhị phân* $0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$, *1 trong các dãy con tăng dài nhất là* $0, 2, 6, 9, 11, 15$. *Dãy con này có độ dài* 6, *dãy đầu vào không có dãy con tăng* 7 *phần tử nào. Dãy con tăng dài nhất trong ví dụ này không phải là nghiệm duy nhất, ví dụ:* $0, 4, 6, 9, 11, 15$; $0, 2, 6, 9, 13, 15$, *&* $0, 4, 6, 9, 13, 15$ *là các dãy con tăng khác có độ dài bằng nhau trong cùng dãy đầu vào.*

1.1 Solve LIS using naive approach: Recursion – exponential time & linear space

The idea is to traverse the input array from left to right & find the length of Longest Increasing Subsequences (LISs) ending with every element `arr[i]`. Let the length found for `arr[i]` be $L[i]$. At the end we return maximum of all $L[i]$ values. To compute $L[i]$, we use *recursion*, we consider all *smaller elements* on the left of `arr[i]`, recursively compute LIS value for all the smaller elements on the left, take the maximum of all & add 1 to it. If there is no smaller element on the left of an element, we return 1 (the length of the sequence contains only that element).

Lemma 2. *Let* $L(i)$ *be the length of the LIS ending at index i s.t.* `arr[i]` *is the last element of the LIS. Then,* $L(i)$ *can be recursively written as:*

$$L(i) = \begin{cases} 1 + \max(L(\text{prev})) & \textit{where } 0 < \text{prev} < i,\ \texttt{arr[prev]} < \texttt{arr[i]}, \\ 1 & \textit{if no such prev exists.} \end{cases}$$

Formally, the length of LIS ending at index i, is 1 greater than the maximum of lengths of all LIS ending at some index `prev` *s.t.* `arr[prev] < arr[i]` *where* prev $< i$. *After we fill the L array, we find LIS as maximum of all in L[]. Overall,* $\mathrm{LIS} = \max_{0 \le i < n} L[i]$.

The above recurrence relation follows the optimal substructure property. Draw recursive tree to see overlapping subproblems.
– Quan hệ đệ quy ở trên tuân theo tính chất cấu trúc con tối ưu. Hãy vẽ cây đệ quy để xem các bài toán con chồng chéo.
G4G's C++: LIS.

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

// return LIS of subarray ending with index i: time complexity: exponential & space complexity: linear
int lis_end_at_idx(vector<int>& arr, int idx) { // compute L[i]
    if (idx == 0) return 1; // base case
```

```
    // consider all elements on the left of i
    // recursively compute LISs ending with them & consider the largest
    int mx = 1;
    for (int prev = 0; prev < idx; ++prev)
        if (arr[prev] < arr[idx])
            mx = max(mx, lis_end_at_idx(arr, prev) + 1);
    return mx;
}

int lis(vector<int>& arr) {
    int n = arr.size();
    int res = 1;
    for (int i = 1; i < n; ++i) res = max(res, lis_end_at_idx(arr, i));
        return res;
}

int main() {
    int n; cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) cin >> a[i];
    cout << lis(a);
}
```

Python:

```
def lis_end_at_idx(arr, idx):
    if idx == 0: return 1 # base case
    # consider all elements on the left of i
    # recursively compute LISs ending with them & consider the largest
    mx = 1
    for prev in range(idx):
        if arr[prev] < arr[idx]:
            mx = max(mx, lis_end_at_idx(arr, prev) + 1)
    return mx

def lis(arr):
    n = len(arr)
    res = 1
    for idx in range(1, n):
        res = max(res, lis_end_at_idx(arr, idx))
    return res

if __name__ == "__main__":
    n = int(input())
    arr = list(map(int, input().split()))
    print(lis(arr))
```

1.2 Using memoization – $O(n^2)$ time & $O(n)$ space

1.3 Using DP (bottom up tabulation) – $O(n^2)$ time & $O(n)$ space

1.4 Using binary search – $O(n \log_2 n)$ time & $O(n)$ space

1.5 Problems based on LIS

1.6 Relations to Other Algorithmic Problems – Mối Quan Hệ với Các Vấn Đề Thuật Toán Khác

The longest increasing subsequence problem is closely related to the longest common subsequence problem (LCS), which has a quadratic time dynamic programming solution (i.e., time complexity $O(n^2)$): the longest increasing subsequence of a sequence s is the longest common subsequence of s & t, where t is the result of sorting s, i.e., $t = \texttt{sort}(s)$. However, for the special case in which the input is a permutation of $[n] = \{1, 2, \ldots, n\}$, this approach can be made much more efficient, leading to time bounds of the form $O(n \log \log n)$.

– Bài toán dãy con tăng dài nhất có liên quan chặt chẽ đến bài toán dãy con chung dài nhất (LCS), có lời giải quy hoạch động thời gian bậc hai (tức là độ phức tạp thời gian $O(n^2)$): dãy con tăng dài nhất của dãy s là dãy con chung dài nhất của s &

t, trong đó t là kết quả của việc sắp xếp s, tức là $t = \texttt{sort}(s)$. Tuy nhiên, đối với trường hợp đặc biệt mà đầu vào là một hoán vị của $[n] = \{1, 2, \ldots, n\}$, cách tiếp cận này có thể hiệu quả hơn nhiều, dẫn đến giới hạn thời gian ở dạng $O(n \log \log n)$.

The largest clique in a permutation graph corresponds to the longest decreasing subsequence of the permutation that defines the graph (assuming the original non-permuted sequence is sorted from lowest value to highest). Similarly, the maximum independent set in a permutation graph corresponds to the longest non-decreasing susbequence. Therefore, longest increasing subsequence algorithms can be used to solve the clique problem efficiently in permutation graphs.

– Clique lớn nhất trong đồ thị hoán vị tương ứng với dãy con giảm dài nhất của hoán vị định nghĩa đồ thị (giả sử dãy không hoán vị ban đầu được sắp xếp từ giá trị thấp nhất đến cao nhất). Tương tự, tập độc lập lớn nhất trong đồ thị hoán vị tương ứng với dãy con không giảm dài nhất. Do đó, các thuật toán dãy con tăng dài nhất có thể được sử dụng để giải quyết bài toán clique một cách hiệu quả trong đồ thị hoán vị.

In the Robinson–Schensted correspondence between permutations & Young tableaux, the length of the 1st row of the tableau corresponding to a permutation equals the length of the longest increasing subsequence of the permutation, & the length of the 1st column equals the length of the longest decreasing subsequence.

– Trong sự tương ứng Robinson–Schensted giữa các hoán vị & bảng Young, độ dài của hàng thứ nhất của bảng tương ứng với một hoán vị bằng độ dài của dãy con tăng dài nhất của hoán vị, & độ dài của cột thứ nhất bằng độ dài của dãy con giảm dài nhất.

1.7 Efficient algorithms for LIS – Các thuật toán hiệu quả cho LIS

The algorithm outlined below solves the longest increasing subsequence problem efficiently with arrays & binary searching. It processes the sequence elements in order, maintaining the longest increasing subsequence found so far. Denote the sequence values as $\{x[i]\}_{i=0}^{n} = x[0], x[1], \ldots$ (0-based indexing). Then, after processing $x[i]$, the algorithm will have stored an integer L & values in 2 arrays:

- L stores the length of the longest increasing subsequence found so far, i.e., `curr_longest_increasing_subsequence_length`.
- $m[l]$ stores the index k of the smallest value $x[k]$ s.t. there is an increasing subsequence of length l ending at $x[k]$ in the range $k \leq i$. Explicitly, suppose that $K_{i,l}$ denotes the set of all indices j s.t. $j \leq i$ & there exists an increasing subsequence of length l ending at $x[j]$. Then $k = m[l]$ is the index in $K_{i,l}$ for which $x[m[l]]$ is minimized; i.e., $m[l] \in k_{i,l}$ & $x[m[l]] = \min_{j \in K_{i,l}} x[j]$ (or equivalently, $m[l] \in K_{i,l}$ & $x[m[l]] \leq x[j]$, $\forall j \in K_{i,l}$); if multiple indices satisfy this condition then $m[l]$ is the largest one.

2 Miscellaneous

Tài liệu

[WS10] Robert Wrede and Murray R. Spiegel. *Advanced Calculus*. 3rd edition. Schaum's Outline Series. McGraw Hill, 2010, p. 456. ISBN: 978-0071623667.