

Longest increasing subsequence

From Wikipedia, the free encyclopedia

The **longest increasing subsequence** problem is to find a subsequence of a given [sequence](#) in which the subsequence elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous. Longest increasing subsequences are studied in the context of various disciplines related to [mathematics](#), including [algorithmics](#), [random matrix theory](#), [representation theory](#), and [physics](#). The longest increasing subsequence problem is solvable in time $O(n \log n)$, where n denotes the length of the input sequence.^[1]

Example

In the binary [Van der Corput sequence](#)

0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15

a longest increasing subsequence is

0, 2, 6, 9, 13, 15.

This subsequence has length six; the input sequence has no seven-member increasing subsequences. The longest increasing subsequence in this example is not unique: for instance,

0, 4, 6, 9, 11, 15

is another increasing subsequence of equal length in the same input sequence.

Relations to other algorithmic problems

The longest increasing subsequence problem is closely related to the [longest common subsequence problem](#), which has a quadratic time [dynamic programming](#) solution: the longest increasing subsequence of a sequence S is the longest common subsequence of S and T , where T is the result of [sorting](#) S . However, for the special case in which the input is a permutation of the integers $1, 2, \dots, n$, this approach can be made much more efficient, leading to time bounds of the form $O(n \log \log n)$.^[2]

The largest [clique \(graph theory\)](#) in a [permutation graph](#) is defined by the longest decreasing subsequence of the permutation that defines the graph; the longest decreasing subsequence is equivalent, by negation of all numbers, to the longest increasing subsequence. Therefore, longest increasing subsequence algorithms can be used to solve the [clique problem](#) in permutation graphs.^[3]

The longest increasing subsequence problem can also be related to finding the [longest path](#) in a [directed acyclic graph](#) derived from the input sequence.

Efficient algorithms

The algorithm outlined below solves the longest increasing subsequence problem efficiently, using only arrays and [binary searching](#). For the first time this algorithm was developed by M. L. Fredman in 1975. It processes the sequence elements in order, maintaining the longest increasing subsequence found so far. Denote the sequence values as $X[1]$, $X[2]$, etc. Then, after processing $X[i]$, the algorithm will have stored values in two arrays:

- $M[j]$ — stores the position k of the smallest value $X[k]$ such that $k \leq i$ and there is an increasing subsequence of length j ending at $X[k]$
- $P[k]$ — stores the position of the predecessor of $X[k]$ in the longest increasing subsequence ending at $X[k]$.

In addition the algorithm stores a variable L representing the length of the longest increasing subsequence found so far.

Note that, at any point in the algorithm, the sequence

$$X[M[1]], X[M[2]], \dots, X[M[L]]$$

is nondecreasing. For, if there is an increasing subsequence of length i ending at $X[M[i]]$, then there is also a subsequence of length $i-1$ ending at a smaller value: namely the one ending at $P[M[i]]$. Thus, we may do binary searches in this sequence in logarithmic time.

The algorithm, then, proceeds as follows.

```
L = 0
for i = 1, 2, ... n:
    binary search for the largest positive j ≤ L such that X[M[j]] < X[i] (or
set j = 0 if no such value exists)
    P[i] = M[j]
    if j == L or X[i] < X[M[j+1]]:
        M[j+1] = i
        L = max(L, j+1)
```

The result of this is the length of the longest sequence in L . The actual longest sequence can be found by backtracking through the P array: the last item of the longest sequence is in $X[M[L]]$, the second-to-last item is in $X[P[M[L]]]$, etc. Thus, the sequence has the form

$$\dots, X[P[P[M[L]]]], X[P[M[L]]], X[M[L]].$$

Because the algorithm performs a single binary search per sequence element, its total time is $O(n \log n)$.

See also

- [Erdős–Szekeres theorem](#), showing that any sequence of n^2+1 distinct integers has either an increasing or a decreasing subsequence of length $n+1$.
- [Patience sorting](#), another efficient technique for finding longest increasing subsequences
- [Plactic monoid](#)
- [Anatoly Vershik](#), a Russian mathematician who studied applications of group theory to longest increasing subsequences
- [Tracy–Widom distribution](#), the distribution of the largest eigenvalue of a random matrix in the [Gaussian unitary ensemble](#). In the limit as n approaches infinity, the length of the longest increasing subsequence of a randomly permuted sequence of n items has a distribution approaching the Tracy–Widom distribution.^[4]

References

1. [^] [*Schensted, C.](#) (1961), "[Longest increasing and decreasing subsequences](#)", *Canadian Journal of Mathematics* **13**: 179–191, [MR0121305](#), [ISSN 0008-414X](#), <http://books.google.com/books?id=G3sZ2zG8AiMC&pg=PA179>
2. [^] [Hunt, J.](#); [Szymanski, T.](#) (1977). "A fast algorithm for computing longest common subsequences". *Communications of the ACM* **20**: 350–353. [doi:10.1145/359581.359603](#).
3. [^] [Golumbic, M. C.](#) (1980), *Algorithmic Graph Theory and Perfect Graphs*, Computer Science and Applied Mathematics, Academic Press, p. 159 .
4. [^] [Baik, Jinho](#); [Deift, Percy](#); [Johansson, Kurt](#) (1999), "On the distribution of the length of the longest increasing subsequence of random permutations", *Journal of the American Mathematical Society* **12** (4): 1119–1178, [doi:10.1090/S0894-0347-99-00307-0](#), [arXiv:math/9810105](#) .

External links

- [Algorithmist's Longest Increasing Subsequence](#)

Retrieved from "http://en.wikipedia.org/wiki/Longest_increasing_subsequence"

Categories: [Algorithms on strings](#) | [Combinatorics](#) | [Formal languages](#) | [Dynamic programming](#)

- This page was last modified on 17 December 2009 at 06:34.
 - Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. See [Terms of Use](#) for details.
- Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

Longest Increasing Subsequence

From Algorithmist

The **Longest Increasing Subsequence** problem is to find the longest increasing subsequence of a given sequence. It also reduces to a [Graph Theory](#) problem of finding the longest path in a [Directed acyclic graph](#).

Overview

Formally, the problem is as follows:

Given a sequence a_1, a_2, \dots, a_n , find the largest subset such that for every $i < j$, $a_i < a_j$.

Techniques

Longest Common Subsequence

A simple way of finding the longest increasing subsequence is to use the [Longest Common Subsequence \(Dynamic Programming\)](#) algorithm.

1. Make a [sorted](#) copy of the sequence A , denoted as B . $O(n \log(n))$ time.
2. Use [Longest Common Subsequence](#) on with A and B . $O(n^2)$ time.

Dynamic Programming

There is a straight-forward [Dynamic Programming](#) solution in $O(n^2)$ time. Though this is asymptotically equivalent to the [Longest Common Subsequence](#) version of the solution, the constant is lower, as there is less overhead.

Let A be our sequence a_1, a_2, \dots, a_n . Define q_k as the length of the longest increasing subsequence of A , subject to the constraint that the subsequence must end on the element a_k . The longest increasing subsequence of A must end on *some* element of A , so that we can find its length by searching for the maximum value of q . All that remains is to find out the values q_k .

But q_k can be found recursively, as follows: consider the set S_k of all $i < k$ such that $a_i < a_k$. If this set is null, then all of the elements that come before a_k are greater than it, which forces $q_k = 1$.

Otherwise, if S_k is not null, then q has some distribution over S_k . By the general contract of q , if we maximize q over S_k , we get the length of the longest increasing subsequence in S_k ; we can append a_k to this sequence, to get that:

$$q_k = \max(q_j | j \in S_k) + 1$$

If the actual subsequence is desired, it can be found in $O(n)$ further steps by moving backward through the q -array, or else by implementing the q -array as a set of stacks, so that the above "+ 1" is accomplished by "pushing" a_k into a copy of the maximum-length stack seen so far.

Some pseudo-code for finding the length of the longest increasing subsequence:

```
function lis_length( a )
  n := a.length
  q := new Array(n)
  for k from 1 to n:
    max := 0;
    for j from 1 to k-1, if a[k] > a[j]:
      if q[j] > max, then set max = q[j].
    q[k] := max + 1;
  max := 0
  for i from 1 to n:
    if q[i] > max, then set max = q[i].
  return max;
```

Faster Algorithm

There's also an $O(n \log n)$ solution based on some observations. Let $A_{i,j}$ be the smallest possible tail out of all increasing subsequences of length j using elements $a_1, a_2, a_3, \dots, a_i$.

Observe that, for any particular i , $A_{i,1} < A_{i,2} < \dots < A_{i,j}$. This suggests that if we want the longest subsequence that ends with a_{i+1} , we only need to look for a j such that $A_{i,j} < a_{i+1} \leq A_{i,j+1}$ and the length will be $j + 1$.

Notice that in this case, $A_{i+1,j+1}$ will be equal to a_{i+1} , and all $A_{i+1,k}$ will be equal to $A_{i,k}$ for $k \neq j + 1$.

Furthermore, there is at most one difference between the set A_i and the set A_{i+1} , which is caused by this search.

Since A is always ordered in increasing order, and the operation does not change this ordering, we can do a [binary search](#) for every single a_1, a_2, \dots, a_n .

Implementation

- [C](#)
- [C++](#) ($O(n \log n)$ algorithm - output sensitive - $O(n \log k)$)

Other Resources

- ["Patience Sorting To Find Longest Increasing Subsequence" on PerlMonks](#)

Retrieved from "http://www.algorithmist.com/index.php/Longest_Increasing_Subsequence"

Longest Increasing Subsequence.c

From Algorithmist

This is an implementation of [Longest Increasing Subsequence](#) in C.

```
// Returns the length of the longest increasing subsequence.
// Note that this is looking for the longest strictly increasing subsequence.
// This can be easily modified for other situations.
int lcs( int* a, int N ) {
    int *best, *prev, i, j, max = 0;
    best = (int*) malloc ( sizeof( int ) * N );
    prev = (int*) malloc ( sizeof( int ) * N );

    for ( i = 0; i < N; i++ ) best[i] = 1, prev[i] = i;

    for ( i = 1; i < N; i++ )
        for ( j = 0; j < i; j++ )
            if ( a[i] > a[j] && best[i] < best[j] + 1 )
                best[i] = best[j] + 1, prev[i] = j; // prev[] is for backtracking
the subsequence

    for ( i = 0; i < N; i++ )
        if ( max < best[i] )
            max = best[i];

    free( best );
    free( prev );

    return max;
}

// Sample usage.
int main(){
    int b[] = { 1, 3, 2, 4, 3, 5, 4, 6 };
    // the longest increasing subsequence = 13456?
    // the length would be 5, as well lcs(b,8) will return.
    printf("%d\n", lcs( b, 8 ) );
}
```

Retrieved from "http://www.algorithmist.com/index.php/Longest_Increasing_Subsequence.c"

Longest Increasing Subsequence.cpp

From Algorithmist

This is an implementation of [Longest Increasing Subsequence](#) in [C++](#).

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

/* Finds longest strictly increasing subsequence. O(n log k) algorithm. */
vector<int> find_lis(vector<int> &a)
{
    vector<int> b, p(a.size());
    int u, v;

    if (a.size() < 1) return b;

    b.push_back(0);

    for (size_t i = 1; i < a.size(); i++) {
        if (a[b.back()] < a[i]) {
            p[i] = b.back();
            b.push_back(i);
            continue;
        }

        for (u = 0, v = b.size()-1; u < v;) {
            int c = (u + v) / 2;
            if (a[b[c]] < a[i]) u=c+1; else v=c;
        }

        if (a[i] < a[b[u]]) {
            if (u > 0) p[i] = b[u-1];
            b[u] = i;
        }
    }

    for (u = b.size(), v = b.back(); u--; v = p[v]) b[u] = v;
    return b;
}

/* Example of usage: */
#include <cstdio>
int main()
{
    int a[] = { 1, 9, 3, 8, 11, 4, 5, 6, 4, 19, 7, 1, 7 };
    vector<int> seq(a, a+sizeof(a)/sizeof(a[0]));
    vector<int> lis = find_lis(seq);

    for (size_t i = 0; i < lis.size(); i++)
        printf("%d ", seq[lis[i]]);
    printf("\n");

    return 0;
}
```

Retrieved from "http://www.algorithmist.com/index.php/Longest_Increasing_Subsequence.cpp"