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.

Contents

Overview

Formally, the problem is as follows:

Given a sequence $a_1, a_2, \ldots, a_n$, 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 $a_1, a_2, \ldots, a_n$. 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 0 to n:
        max := 0;
        for j from 0 to k, if a[k] > a[j]:
            if q[j] > max, then set max = q[j].
        q[k] := max + 1;
    max := 0
    for i from 0 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 $a_1, a_2, a_3, \ldots, a_i$.

Observe that, for any particular i, $A_{i,1} < A_{i,2} < \ldots < A_{i,j}$. 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 $A_{i,j} < a_{i+1} <= A_{i,j+1}$ 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 $a_1, a_2, \ldots, a_n$.

Further explain:--GONG Zhi Tao 11:19, 1 August 2012 (EDT)

We have elements: $a_1, a_2, a_3, \ldots, a_i$.

And we have a longest increasing subsequences of them: $A_{i,1} < A_{i,2} < \ldots < A_{i,j}$, for any $A_{i,k}(1 <= k <= j)$ you could not find a smaller alternative.

Now we have a new element: a_{i+1}

What we can do about it:

1. insert it at the back if $A_{i,j} < a_{i+1}$, where we will have a longer one;

2. make it an alternative for $A_{i,k}$ if $A_{i,k-1} < a_{i+1}\ AND\ a_{i+1} <= A_{i,k}$

Alternative means that we MIGHT get longer ones if using the new element.

Implementation

- C
- C++ ($O(n \log n)$ algorithm - output sensitive - $O(n \log k)$)
- Python ($O(n^2)$)

Other Resources

- "Patience Sorting To Find Longest Increasing Subsequence" on PerlMonks (http://www.perlmonks.org/?node_id=547199)

Categories: Dynamic Programming | Longest Increasing Subsequence | Graph Theory