The Longest Increasing Subsequence Problem

Given a string $X = \langle x_1, x_2, \ldots, x_n \rangle$ of n characters drawn from a totally ordered alphabet, an *Increasing Subsequence* (IS) of X is a subsequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ of X (that is, $Z = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_k} \rangle$, with $1 \leq i_1 < i_2 \ldots < i_k \leq n$) with $z_i < z_{i+1}$ for $1 \leq i < k$. Our goal is to design and analyze a dynamic programming algorithm that returns a *Longest* Increasing Subsequence (LIS) of the input string X, denoted $\mathrm{LIS}(X)$.

We will develop two strategies, which are both based on subproblem spaces which solve a more constrained problem on prefixes of X. The first subproblem space contains n suproblems (one for each prefix) and yields a quadratic-time algorithm. In contrast, the second subproblem space contains a quadratic number of subproblems, but yields an algorithm whose running time is $O(n \log n)$.

1 A Quadratic Algorithm

For $1 \leq i \leq n$, define $\overline{\mathrm{LIS}}(X_i)$ as the longest among all those increasing subsequences of prefix X_i *which end with* x_i. (Observe that $\overline{\mathrm{LIS}}(X_i)$ may be much shorter than $\mathrm{LIS}(X_i)$.) Moreover, let $\ell[i]$ be the length of a $\overline{\mathrm{LIS}}(X_i)$. Clearly, the length of $\mathrm{LIS}(X)$ is $\max\{\ell[i] : 1 \leq i \leq n\}$, since any $\overline{\mathrm{LIS}}(X_i)$ is just an increasing subsequence of X, hence no larger than $\mathrm{LIS}(X)$, but also, if $Z = \mathrm{LIS}(X) = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_k} \rangle$, then Z must be a $\overline{\mathrm{LIS}}(X_{i_k})$.

Let us now derive an optimal subtstructure property for the space $\{\overline{\mathrm{LIS}}(X_i) : 1 \leq i \leq n\}$. For $1 \leq i \leq n$, define S_i as the set $\{j : 1 \leq j < i \text{ and } x_j < x_i\}$. Then, the following recurrence must hold:

$$\ell[i] = \begin{cases} 1 & \text{if } S_i = \emptyset, \\ 1 + \max\{\ell[j] : j \in S_i\} & \text{otherwise.} \end{cases}$$

To show that the above relation holds, note that if $S_i = \emptyset$, then all elements x_j with $1 \leq j < i$ are no smaller than x_i, hence the only IS of X_i with last element x_i is $\langle x_i \rangle$. When $S_i \neq \emptyset$, consider a given $\overline{\mathrm{LIS}}(X_i)$. Such a $\overline{\mathrm{LIS}}$ has clearly length at least 2. Now, if the penultimate element of such subsequence is x_k, then $k < i$ and $x_k < x_i$ (by the definition of increasing subsequence), therefore $k \in S_i$. Moreover, all the first $k-1$ elements of $\overline{\mathrm{LIS}}(X_i)$ must form a $\overline{\mathrm{LIS}}(X_k)$. If this were not the case, we could find an increasing subsequence ending with x_i longer than the $\overline{\mathrm{LIS}}(X_i)$ itself, a contradiction. Needless to say, $\overline{\mathrm{LIS}}(X_k)$ must be the longest among all the $\overline{\mathrm{LIS}}$ ending with elements x_j with $j \in S_i$, or, again, we could obtain a longer $\overline{\mathrm{LIS}}(X_i)$.

Observe that $S_1 = \emptyset$, hence we have $\ell[1] = 1$ as a base case. The above relation immediately yields a dynamic programming algorithm for the LIS problem. In the algorithm, we maintain as additional information needed to reconstruct the LIS, a variable *end*, storing the ending point of the longest $\overline{\mathrm{LIS}}(X_i)$ found so far, a variable *len*, storing the length of such subsequence, and finally a vector $prev[i]$, $1 \leq i \leq n$ containing the following information:

$$prev[i] = \begin{cases} 0 & \text{if } S_i = \emptyset, \\ k & \text{if } \ell[k] = \max\{\ell[j] : j \in S_i\}. \end{cases}$$

The pseudocode of the algorithm is the following:

```
LIS(X)
n ← length(X)
len ← 1
end ← 1
ℓ[1] ← 1
prev[1] ← 0
for i ← 2 to n do
    ℓ[i] ← 1
    prev[i] ← 0
    for j ← 1 to i − 1 do
        if (x_j < x_i) then
            if (ℓ[i] < 1 + ℓ[j]) then
                ℓ[i] ← 1 + ℓ[j]
                prev[i] ← j
    if (len < ℓ[i]) then
        len ← ℓ[i]
        end ← i
return len, end, prev
```

The running time $T(n)$ of the above algorithm is upperbounded by the number of iterations of the two nested loops needed to compute vectors ℓ and $prev$. Therefore

$$\begin{aligned} T(n) &= \Theta\left(\sum_{i=2}^{n}\sum_{j=1}^{i-1} 1\right) \\ &= \Theta\left(\sum_{i=2}^{n}(i-1)\right) \\ &= \Theta\left(n^2\right). \end{aligned}$$

In order to print any $\overline{\mathrm{LIS}}(X_i)$ using the information returned by the above algorithm, we can proceed recursively as follows:

```
PRINT_LIS(i, prev, X)
if (prev[i] ≠ 0)
    then PRINT_LIS(prev[i], prev, X)
print(x_i)
return
```

The LIS of X can be printed by invoking PRINT_LIS($end, prev, X$). The running time of the printing algorithm is clearly upperbounded by the length of the LIS, which is in turn at most n.

2 A $\Theta(n \log n)$ Algorithm

The above algorithm is based on a subproblem space which, given a prefix X_i, constrains the head of the optimal increasing subsequence to end with x_i. Knowing the head of the subsequence is important to be able to set up an optimal substructure property. We

wonder whether a less stringent constraint on the head of the subsequence may yield a better algorithm. For instance, suppose that, for each i, with $1 \leq i \leq n$, we set out to compute the LIS(X_i) ending with the *smallest last character* (called *head* in what follows) . In other words, we strengthen the subproblems by seeking a *specific* LIS(X_i), namely, the one which is most likely to be extended further. Unfortunately, it is easy to convince oneself that the resulting subproblem space does not feature a substructure property that allows us to obtain the solution for X_{i+1} given the solution for X_i. To see this, assume that we have computed the smallest-head LIS $Z = \langle z_1, z_2, \ldots, z_k \rangle$ for X_i. If $z_k < x_{i+1}$ then we can easily argue that the string $\langle Z, x_{i+1} \rangle$ is the smallest-head LIS for prefix X_{i+1}. However, if this is not the case, we cannot be sure that Z is also the smallest-head LIS for X_{i+1}, since an increasing subsequence Z' of X_i of length $k-1$ may exist with $z'_{k-1} < x_{i+1}$, whence $\langle Z', x_{i+1} \rangle$ would be the smallest-head LIS for X_{i+1}.

The above line of reasoning suggests that in order to be able to set up an optimal substructure property based on increasing subsequences of smallest head, we need to compute, for each prefix X_i, $1 \leq i \leq n$, its increasing subsequences of smallest head *of any length*. More formally, given a string X of length n, and a prefix X_i, with $1 \leq i \leq n$, for $1 \leq k \leq n$ we say that a subsequence $Z^{k,i}$ of X_i is a *Smallest-Head Subsequence of X_i of length k*, denoted $Z^{k,i} = \mathrm{SH}(k,i)$, if $Z^{k,i} = \langle z_1^{k,i}, z_2^{k,i}, \ldots, z_k^{k,i} \rangle$ is an increasing subsequence of X_i of length k with smallest value of $z_k^{k,i}$, if one such subsequence exists. In case no such subsequence exists, we set conventionally $Z^{k,i} = \epsilon$ and $z_k^{k,i} = \infty$. Our subproblem space will then be $\{\mathrm{SH}(k,i) : 1 \leq k, i \leq n\}$. Observe that $LIS(X) = \mathrm{argmax}\{|SH(k,n)| : 1 \leq k \leq n\}$ and that the subproblem space features n^2 subproblems.

Let us fix a given value i, with $1 \leq i \leq n$. One important property of the subproblem space defined above is the following. Let $Z^{k,i} = \mathrm{SH}(k,i)$ and $Z^{k+1,i} = \mathrm{SH}(k+1,i)$, with $Z^{k,i}, Z^{k+1,i} \neq \epsilon$. Then, $z_k^{k,i} < z_{k+1}^{k+1,i}$. Indeed, if this were not the case, the sequence $\langle z_1^{k+1,i}, z_2^{k+1,i}, \ldots, z_k^{k+1,i} \rangle$ would be an increasing sequence of length k with head $z_k^{k+1,i}$ smaller than $z_k^{k,i}$, which is impossible since $Z^{k,i} = \mathrm{SH}(k,i)$. In words, the heads of the nonempty SH sequences for a prefix X_i are increasing with k. We will crucially exploit this property to obtain a faster algorithm, regardless of the large number of subproblems.

Let us now derive an optimal substructure property for the newly defined subproblem space. As a base case, observe that $Z^{1,1} = \mathrm{SH}(1,1) = \langle x_1 \rangle$, while $Z^{k,1} = \epsilon$ for $2 \leq k < n$. For $1 \leq i < n$, and $1 \leq k \leq n$, let $Z^{k,i} = \mathrm{SH}(k,i)$, and let $z_k^{k,i}$ denote its head (recall that we set $z_k^{k,i} = \infty$ if $Z^{k,i} = \epsilon$). We have:

$$Z^{k,i+1} = \mathrm{SH}(k,i+1) = \begin{cases} \mathrm{SH}(k,i) & \text{if } z_k^{k,i} \leq x_{i+1} \\ \langle \mathrm{SH}(k-1,i), x_{i+1} \rangle & \text{if } (z_k^{k,i} > x_{i+1}) \wedge (z_{k-1}^{k-1,i} < x_{i+1}) \\ \mathrm{SH}(k,i) & \text{if } (z_k^{k,i} \geq x_{i+1}) \wedge (z_{k-1}^{k-1,i} \geq x_{i+1}) \end{cases}$$

The proof of the above property is very simple. If $z_k^{k,i} \leq x_{i+1}$, then the extra character x_{i+1} is of no use to create a smaller-head $\mathrm{SH}(k,i+1)$, and the same argument applies to the case $(z_k^{k,i} \geq x_{i+1})$ and $(z_{k-1}^{k-1,i} \geq x_{i+1})$, since any increasing subsequence of length $k-1$ has a head no smaller than x_{i+1}, hence none can be appended with x_{i+1} to create a better increasing sequence of length k for X_{i+1}. However, if $z_k^{k,i} > x_{i+1}$ and $z_{k-1}^{k-1,i} < x_{i+1}$, then we can construct the sequence $\langle \mathrm{SH}(k-1,i), x_{i+1} \rangle$ which has a smaller head than $\mathrm{SH}(k,i)$ and is indeed optimal. Notice that the above property also

encompasses the case where $\mathrm{SH}(k,i) = \epsilon$ and hence $z_k^{k,i} = \infty$. When this extreme case applies, we have that $|LIS(X_{i+1}| = |LIS(X_i)| + 1$.

Observe that under the above substructure property, since the heads of the SH sequences are ordered, when switching from prefix X_i to prefix X_{i+1} *only a single subproblem SH*$(\bar{k}, i+1)$ *may have a different optimal solution from SH*$(\bar{k}, i)$. Also, such a subproblem can be spotted in a logarithmic number of comparisons by maintaining the heads of the SH's and performing a binary search of the value x_{i+1} on such heads.

The code below implements the above strategy. In the code, we use vector $Z[1..n]$ to keep the heads of the SH's. In particular, after iteration i, $Z[k]$ contains $z_k^{k,i}$. The code utilizes subroutine BIN_SEARCH(Z, x) (whose code is omitted for brevity), which returns the (only) index $\bar{k}$ such that $Z[\bar{k}-1] < x$ and $Z[\bar{k}] \geq x$ (where, for uniformity, we set $Z[0] = -\infty$).

```
LIS(X)
n ← length(X)
Z[1] ← x_1
for k ← 2 to n do
    Z[k] ← ∞
for i ← 2 to n do
    k̄ ← BIN_SEARCH(Z, x_i)
    Z[k̄] ← x_i
ℓ ← 0
while ((ℓ < n) and (Z[ℓ + 1] < ∞)) do
    ℓ ← ℓ + 1
return ℓ
```

Observe that the above algorithm, whose correctness follows from the optimal substructure property proved beforehand, only computes the length ℓ of $\mathrm{LIS}(X)$. The reader can easily modify the code to keep additional information needed to reconstruct the sequence. Specifically, we need to keep, at each iteration i and for each value $Z[k] < \infty$, the index i_k such that $Z[k] = z_k^{k,i} = x_{i_k}$ (these can be easily stored in a vector $index[1..n]$). Then, it suffices to keep an extra vector $prev[1..n]$ which, after iteration i, carries the following information: if $j = index[k]$, then $prev[j]$ is the index of the character preceding the head x_j in $\mathrm{SH}(k,i)$. Clearly, at iteration i, $prev$ is updated by setting $prev[i] = index[\bar{k}-1]$. Then, at the end of the n-th iteration, if $index[\ell] = t$, then the LIS can be obtained as $\langle \ldots x_{prev[prev[t]]}, x_{prev[t]}, x_t \rangle$.

The running time of the above algorithm is clearly dominated by the n calls to subroutine BIN_SEARCH, for a total running time $T(n) = O(n \log n)$.