

COMP3600 Assignment 2

Felix Andrews (3285754)

September 13, 2002

1 Dynamic Programming Algorithms

1.1 The Structure of an Optimal Solution

Let $X_n = \langle x_1, x_2, \dots, x_n \rangle$, a given sequence of positive integers.

1.1.1 The Structure of an LDS

Assume we have $Z_k = \langle z_1, z_2, \dots, z_k \rangle$ whose elements are a subset of the indices of X_n , such that the elements referenced by Z_k form a *Longest Decreasing Subsequence* (LDS) of X_n . The LDS results from the function $LDS(X_n, z_k)$, which returns the longest decreasing subsequence up to and including the index z_k (this sequence must have a final element x_{z_k}). Note that the way the function is defined, $LDS(X_n, n) \neq LDS^*$ (where LDS^* is the LDS overall) in the general case, since the $LDS(X_n, n)$ function only considers sequences which have final index n . The overall LDS^* does not necessarily end with the final element of X_n . Rather, LDS^* will be the maximum sized set from $LDS(X_n, 1..n)$, i.e. $LDS(X_n, z_k)$.

For any LDS that has a final element x_i , the previous element in the LDS could have been any of the $1..i - 1$ elements of X_{i-1} , or alternatively x_i could stand on its own. If it came from some element x_j , then the remaining prefix of the LDS must be the LDS which has final element x_j . Why? If there was longer strictly decreasing subsequence with x_j as final element, we could substitute it into the original LDS to obtain a longer sequence: a contradiction. Thus the problem has optimal substructure:

$$LDS(X_n, z_k) \equiv LDS(X_n, z_{k-1}) \cup \{x_{z_k}\}$$

1.1.2 The Structure of an IWS

The same property applies to an *Increasing Weighted Subsequence* (IWS) which has the maximum sum of elements of X_n .

$$IWS(X_n, z_k) \equiv IWS(X_n, z_{k-1}) \cup \{x_{z_k}\}$$

1.2 A Recursive Solution

1.2.1 Definition of a Longest Decreasing Subsequence

- Let $c[i]$ be the length of the LDS which has final element x_i .

- Let $p[i]$ be the index of the penultimate element in the LDS which has final element x_i .
- Let c^* be the length of LDS* (the longest decreasing subsequence overall).
- Let p^* be the index of the last element of LDS* in X_n .

The final solution, as defined before, is the maximum length over all computed subproblems:

$$c^* = \max_{1 \leq k \leq n} c[k] \qquad \Rightarrow p^* = p[k]$$

Any subproblem solution can be defined recursively as follows. Each element up to the current one is considered and the maximum subsequence chosen. If there are no earlier elements then the subsequence is simply this element; as such it is length 1. The same applies if we cannot extend the sequence from a previous point (since it must be decreasing). If we can extend it, the subsequence length is the length up to the previous element plus 1. The index of the last element in the sequence we are extending is also recorded.

$$c[i] = \max_{1 \leq k \leq i} \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_i \geq x_k \\ c[k] + 1 & \text{if } x_i < x_k \end{cases} \Rightarrow \begin{cases} p[i] = 0 \\ p[i] = 0 \\ p[i] = k \end{cases}$$

1.2.2 Definition of an Increasing Weighted Subsequence

A recursive definition of an IWS is similar and can be expressed simply with the following formulae (where the definitions of $c[i]$, $p[i]$, c^* and p^* are analogous to the LDS case).

$$c^* = \max_{1 \leq k \leq n} c[k] \qquad \Rightarrow p^* = p[k]$$

$$c[i] = \max_{1 \leq k \leq i} \begin{cases} x_1 & \text{if } i = 1 \\ x_i & \text{if } x_i \leq x_k \\ c[k] + x_i & \text{if } x_i > x_k \end{cases} \Rightarrow \begin{cases} p[i] = 0 \\ p[i] = 0 \\ p[i] = k \end{cases}$$

1.3 Computing an Optimal Solution

1.3.1 LDS-length

It would now be easy to write a brute force search over all possible subsequences that runs in exponential time, but there are in fact only n overlapping subproblems, so we can apply dynamic programming.

```
LDS-LENGTH( $X$ )
1    $n \leftarrow \text{length}(X)$ 
2    $c^* \leftarrow 1$ 
3    $z \leftarrow 1$ 
4   for  $i \leftarrow 1$  to  $n$ 
5       do  $c[i] \leftarrow 1$ 
6            $p[i] \leftarrow 0$ 
```

```

7   for  $i \leftarrow 1$  to  $n$ 
8       do for  $k \leftarrow 1$  to  $i - 1$ 
9           do if  $x_i < x_k$ 
10              then if  $c[k] + 1 > c[i]$ 
11                  then  $c[i] \leftarrow c[k] + 1$ 
12                       $p[i] \leftarrow k$ 
13                          if  $c[i] > c^*$ 
14                              then  $c^* \leftarrow c[i]$ 
15                                   $z \leftarrow i$ 
16   return  $c$  and  $p$  and  $z$ 

```

The length of the LDS* is given by $c[z]$.

1.3.2 IWS-sum

Similar! See the recursive definition in section (1.2.2).

1.4 Constructing an Optimal Sequence

To print the optimal sequence we can make use of the p array which was developed as part of the main algorithm in the previous section. It shows us the path of indices to follow to construct an LDS or IWS. A function such as the following could extract it (using the output of the main algorithm):

```

PRINT-LDS( $X, m, z$ )
1   if  $m[z] > 0$ 
2       then PRINT-LDS( $X, m, m[z]$ )
3   print  $x_z$ 

```

2 Time Complexity Analysis

Consider the pseudocode for LDS-LENGTH. The first 3 lines take constant time. The loop in lines 4-6 must run in $\Theta(n)$ time. The next outer loop (line 7) executes n times, and the inner loop (line 8) executes at most $n - 1$ times. Note that the block within these (lines 9-15) takes only $O(1)$ time. Thus the dominant running time complexity of this algorithm is $O(n(n - 1)) = O(n^2)$. The same analysis can be applied to the algorithm for calculating an IWS.

3 Implementation in C

See the associated files `decreasing_subseq.c` and `increasing_weight_subseq.c`.