Dynamic Programming

Rossano Venturini[1]

1 **Computer Science Department, University of Pisa, Italy**
rossano.venturini@unipi.it

Notes for the course *"Competitive Programming and Contests"* at Department of Computer Science, University of Pisa.
Web page: `https://github.com/rossanoventurini/CompetitiveProgramming`

These notes sketch the content of the 15th lecture.

1 Longest increasing subsequence

▶ **Problem 1** (Longest increasing subsequence)**.** *Given a sequence S of n numbers, find the lenght of its longest increasing subsequence.*

The longest increasing subsequence (LIS) means to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible.

As an example consider the sequence $S = 10, 22, 9, 21, 33, 50, 41, 60, 80$. The lenght of LIS is 6 and $10, 22, 33, 50, 60, 80$ is a LIS. In general, LIS is not unique, e.g. $9, 21, 33, 50, 60, 80$ is another LIS.

Consider the sequence $S[1, n]$ and let $\mathsf{LIS}(i)$ be the LIS of the prefix $S[1, i]$ whose last element is $S[i]$.

$$\mathsf{LIS}(i) = \begin{cases} 1 + \max(\mathsf{LIS}(j) \mid 1 \leq j < i \text{ and } \mathsf{S[j]} < \mathsf{S[i]}) \\ 1 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if such } j \text{ does not exist} \end{cases}$$

It should be clear that the above recurrence can be solved in time $\Theta(n^2)$.

2 Dynamic programming as shortest/longest path on a DAG

Often it is convenient to reduce a problem to a (single source) longest path computation on a suitable DAG.

Let's consider again the LIS problem. Our DAG G has a vertex for each element of the sequence, plus a dummy vertex that marks the end of the sequence. Let us use v_i to denote the vertex corresponding to element $S[i]$, v_{n+1} denotes the dummy vertex.

Edges are as follows. Every vertex has an edge to v_{n+1}. Moreover, a vertex v_j is connected to v_i iff $j < i$ and $S[j] < S[i]$. Thus, we have an edge (v_j, v_i) iff $S[i]$ can follow $S[j]$ in a increasing subsequence.

By construction, it should be clear that there exists a one-to-one correspondance between increasing subsequences of S and paths from v_1 to v_{n+1} on G. Thus, any longest path on G corresponds to a LIS on S.

A longest path on a DAG G, even weighted ones, can be computed in time proportional to the number of edges in G. The DAG above has at most n^2 edges and, thus, the reduction gives an algorithm with the same time complexity of the previous solution.

This reduction is always possible and sometimes it helps in reasoning about properties of the problems to get faster solutions.

3 Speeding up LIS

Take a look to the tutorial here `http://www.geeksforgeeks.org/longest-monotonically-increasing-subseq` for a $\Theta(n \log n)$ time algorithm.

Intuition. For any position i in S, consider the tuple $\langle i, \mathsf{LIS}(i), S[i] \rangle$.

We say that position i *dominate*s position j iff $\mathsf{LIS}(i) \geq \mathsf{LIS}(j)$ and $S[i] < S[j]$.

Considering only dominant positions suffices to find a LIS.

Given a new position k, what's the lenght of the LIS of the subsequence up to k which ends at position k?

We search for dominant position i such that $S[i]$ is the predecessor of $S[k]$. Then, $\mathsf{LIS}(k) = 1 + \mathsf{LIS}(i)$.

We can use a BST to store dominant positions.

4 Minimum number of jumps

▶ **Problem 2** (Minimum number of jumps)**.** *Given an array of integers where each element represents the max number of steps that can be made forward from that element. Write a function to return the minimum number of jumps to reach the end of the array (starting from the first element). If an element is* 0*, then cannot move through that element.*

As an example, consider the array $1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9$. The minimum number of jumps is 3, i.e., $1, 3, 9$.

Think about the reduction to a SSSP on a DAG. This gives a $\Theta(n^2)$ time solution.

A linear time solution is here `http://www.geeksforgeeks.org/minimum-number-jumps-reach-endset-2on-`