Dynamic Programming Part 2

Arnav Gupta

April 9, 2024

Contents

1 Longest Increasing Subsequence

Input: an array A of n integers

Output: a longest increasing subsequence of A that doe not need to be contiguous

Remark: there are 2^n subsequences

1.1 Idea

A longest increasing subsequence S ending at $A[i]$ looks like $S = [..., A[j], A[i]] = s' + [A[i]]$.

S' is a longest increasing subsequence ending at $A[j]$ or empty.

Don't know j but can try all $j < i$ for which $A[j] < A[i]$.

```
LongestIncreasingSubsequence(A[1..n]):
  L[1] = 1
  for i in range(2, n):
      L[i] = 1
      for j in range(1, i-1):
```

```
        if A[j] < A[i]:
            L[i] = max(L[i], L[j] + 1)
return max entry in L
```

Runtime: $\Theta(n^2)$

Remark: the algorithm does not return the sequence itself, but could be modified to do so

2 Longest Common Subsequence

Input: arrays A and B of length n and m characters respectively

Output: the max length k of a common subsequence to A and B (no need to be contiguous)

Remark: there are 2^n subsequences in A and 2^m subsequences in B

2.1 Bivariate Recurrence

Definition: let $M[i,j]$ be the longest subsequence between $A[1..i]$ and $B[1..j]$

- $M[0,j] = 0$ for all j
- $M[i,0] = 0$ for all i
- $M[i,j]$ is the max of up to 3 values:
  - $M[i, j-1]$ (don't use $B[j]$)
  - $M[i-1, j]$ (don't use $A[i]$)
  - $1 + M[i-1, j-1]$ if $A[i] = B[j]$

The algorithm computes all $M[i,j]$ using 2 nested loops so runtime $\Theta(mn)$