Longest Increasing Subsequence

Longest Increasing Subsequence

Problem Statement

- Given a sequence of n numbers, determine the largest increasing subsequence
  - The elements of the subsequence do not have to occur consecutively
  - E.g. 2 3 1 7 4 6 9 5
  - Answer: 2 3 4 6 9 (5 elements)

Longest Increasing Subsequence

Greedy Solutions

- Choosing smallest number, then smallest larger number following that
- Choosing first number, then next available
- No known greedy solution

Backtracking

- For each index, can branch for yes (including) or no (not including)
- Many branches will be quickly pruned when we choose to include a number that is too small
- Has the usual problem of repeating sub-problems
  - E.g. 3 1 4 2 5 ….
    - The branch that includes 3 4 5
    - The branch that includes 1 2 5
    - Duplicate calculations from that point

Longest Increasing Subsequence

Dynamic Programming

- We need the usual components:
  - Define the subproblems (i.e what does each element in the matrix represent exactly)
  - A recurrence relation to indicate how one subproblem can be computed based on the solution to other smaller subproblems

Getting Started

- Suppose S[k] := the longest increasing subsequence considering the elements 1…k
- What is the recurrence relation?
  - Consider the two cases: element k is part of the answer, or not
- S[k] = max(solution when k is included, solution when k is not included)
- S[k] = max(1 + S[j], S[k-1]),
  - Where $j<k$ is the index that maximizes S under the constraint that the j^{th} element is smaller than the k^{th} element (or 0 if no such element)

Longest Increasing Subsequence

Being a little loose with notation

- $S[k] = \max(1 + S[j], S[k-1])$,
  - Where $j<k$ is the index that maximizes S under the constraint that the j^{th} element is smaller than the k^{th} element (or 0 if no such element)
  - Really we should explicitly write in our equation that in the case we "include" element k, we are taking the maximum of S[j] over all indices $j < k$, and such that element j is less than element k, instead of just describing it below.

Longest Increasing Subsequence

Dynamic Programming

- S[k] = max(1 + S[j], S[k-1]),
  - Where j<k is the index that maximizes S under the constraint that the j^{th} element is smaller than the k^{th} element (or 0 if no such element)

Let's Step Through an Example

- 2 4 7 1 5
- S[0] = 0 by definition with no elements
- S[1] = 1
- S[2] = max(1+S[1], S[1]) = 2
- S[3] = max(1+S[2], S[2]) = 3
- S[4] = max(1+0, S[3]) = 3
- S[5] = max(1+S[4], S[4]) = 4

WRONG ANSWER! What went wrong?

Longest Increasing Subsequence

Dynamic Programming

- $S[k] = \max(1 + S[j], S[k-1])$,
  - Where $j<k$ is the index that maximizes S under the constraint that the j^{th} element is smaller than the k^{th} element (or 0 if no such element)

What went wrong?

- $S[k]$ didn't provide any information as to whether or not the k^{th} element is part of the solution (or more generally what the last included element is), so we don't know whether or not we can add on to that sequence.

Longest Increasing Subsequence

Dynamic Programming – Second Try

- S[k] := largest subsequence that can be formed from the first k elements, such that element k is included in the subsequence
- We can find the overall solution when complete by taking the maximum of S[k] over all k
- The recurrence relation now requires k to be included
  - S[k] = 1 + S[j]
  - Where j<k is the index that maximizes S under the constraint that the j^{th} element is smaller than the k^{th} element (or 0 if no such element)

Longest Increasing Subsequence

Dynamic Programming – Second Try

- S[k] := largest subsequence that can be formed from the first k elements, such that element k is included in the subsequence
- S[k] = 1 + S[j]
  - Where $j<k$ is the index that maximizes S under the constraint that the j^{th} element is smaller than the k^{th} element (or 0 if no such element)

Let's Step Through an Example

- 2 4 7 1 5
- S[0] = 0 by definition with no elements
- S[1] = 1
- S[2] = 1 + S[1] = 2
- S[3] = 1 + S[2] = 3
- S[4] = 1 + 0 = 1
- S[5] = 1 + 2 = 3
- Answer is $\max_k$ (S[k]) = 3

Longest Increasing Subsequence

Dynamic Programming – Second Try

- S[k] := largest subsequence that can be formed from the first k elements, such that element k is included in the subsequence
- S[k] = 1 + S[j]
  - Where j<k is the index that maximizes S under the constraint that the j^{th} element is smaller than the k^{th} element (or 0 if no such element)

Tracing back to determine the actual sequence

- Find an index that maximizes S[k], include that element
- Work backwards to an index j such that S[j] = S[k] – 1 and element j is smaller than element k, and include that element
- Repeat until entire sequence is determined
- There may be multiple equivalent longest increasing subsequences

Longest Increasing Subsequence

Iterative Solution: "Bottom-Up"

- Start with S[1] and compute each successive value

Complexity

- $O(n^2)$

```
LONGEST-INCR-SUBSEQ (a_1,…,a_n)
1. for k=1 to n do
2.     S[k] = 1
3.     for j=1 to k-1 do
4.         if a_j<a_k and S[k]<S[j]+1 then
5.             S[k] = S[j]+1
6. return max_k S[k]
```

Example

Given X = {2, 4, 7, 1, 5}

What does S[j] stand for?

What is a recurrence relation tying S[j] to smaller problems?

What is the solution to the original problem expressed in terms of S?

Fill in the S matrix.

S[0]	S[1]	S[2]	S[3]	S[4]	S[5]