Design and Analysis of Algorithms — Lab

R S Milton, C Aravindan, T T Mirnalinee
Department of CSE, SSN College of Engineering

Session 5: Dynamic Programming
30 January 2020

1 Longest Increasing Sequence

The input is a sequence of elements $A = [A_0, A_1, \ldots, A_{n-1}]$. A subsequence of A is another sequence obtained from A by deleting zero or more elements, without changing the order of the remaning elements. The output should be the longest subsequence whose elements are in increasing order.

Example: `A = [5, 2, 8, 6, 3, 6, 9, 7]`.

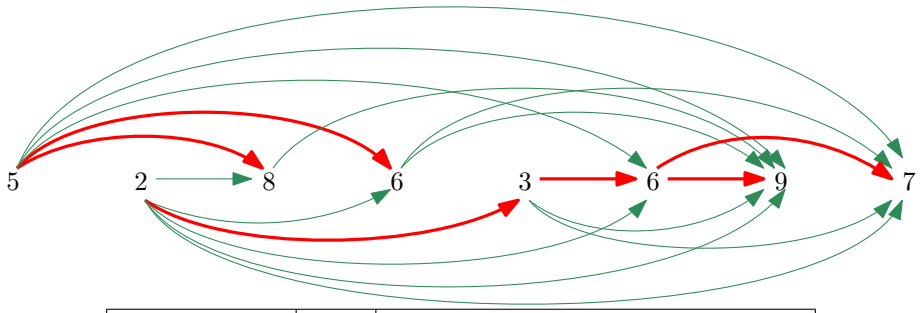
5	2	8	6	3	6	9	7
5	2	8	6	3	6	9	7
5	2	8	6	3	6	9	7
5	2	8	6	3	6	9	7
5	2	8	6	3	6	9	7

`[2, 3, 6, 9]`, `[5, 8, 9]` and `[6, 7]` are increasing subsequences. `[2, 3, 6, 9]` is a longest increasing subsequence (LIS) of A. Yet another LIS is `[2, 3, 6, 7]`.

Let `L(j)` be the length of the LIS that ends at `A[j]`. For every item `A[i]` in `A[0:j]`, which is smaller than `A[j]`, the LIS that ends at `A[i]` can be extended with `A[j]`. Out of all these extensible LISes, select the LIS with the maximum length. This is the LIS that should be extended with `A[j]` to get the LIS that ends at `A[j]`.

$$L(j) = 1 + \max_{0 \le i < j} \{L(i) | A[i] < A[j]\}$$

Subproblems and their dependent subproblems are illustrated. Directed edges indicate the "smaller" subproblem to "larger" subproblem relation. For each item, edge from its predecessor in the LIS is shown in red. (The LIS, in fact, is the longest path in this directed acyclic graph.)



Index	j	0	1	2	3	4	5	6	7	8
Item	A[j]	5	2	8	6	3	6	9	7	∞
Length	L[j]	1	1	2	2	2	3	4	4	5
Predecessor	P[j]	-	-	0	0	1	4	5	5	6

The Table shows, for each item `j`, the length `L[j]` of the LIS that ends with item `j`. We have also shown the predecessor of each item `j` in the LIS that ends with that item.

Exhaustive Search

1. Design an algorithm to find `LengthES(j)`, the length of `LIS(j)`, the longest increasing sequence that ends at `A[j]`. Implement the algorithm `LengthES(j)`.
2. Using `LengthES(j)`, find the lenth of the LIS of the array `A[0:n]`. Test it.
   - ▷ `[2, 4, 3, 5, 1, 7, 6, 9, 8]`
   - ▷ `[5, 1, 5, 7, 2, 4, 9, 8]`
   - ▷ `[3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3, 2, 3, 8, 4, 6, 2, 6]`
3. Calculate the emprical running time of `LengthES(n)`.

Dynamic Programming

1. Using Dynamic Programming, design an algorithm `LengthDP(j)` to find the length of the `LIS(j)`, longest increasing sequence that ends at `A[j]` for `j = 0,1,2,...,n-1`. Implement the algorithm `LengthDP(j)`.
2. How will you find the length the LIS of the array `A[0:n]` using `LengthDP()`?
3. Introduce `A[n]` $= \infty$, and modify `LengthDP()` to return the length of the LIS of `A[0:n]`.
4. Algorithm `LengthDP()` maintains `L[j]`, the length of the LIS that ends at `A[j]`. In addition, now we also want to maintain the predecessor `P[j]` of `A[j]` in the LIS that ends at `A[j]`. Modify `LengthDP(j)` to maintain both `L[j]` and `P[j]`.

5. Write a function `TraceLIS(n)` that returns the LIS of `A[0:n]` as a list. This function constructs the LIS of `A[0:n]` by tracing pedecessor array `P[]` starting from `n`.

    ▷ Implement `TraceLIS` as a recursive function.

    ▷ Implement `TraceLIS` as an iterative function.

6. Calculate the emprical running time of `LengthDP(n)`.