CS 341 – Algorithms

Lecture 12 – Dynamic Programming II

25 June 2021

Today's Plan

1. Longest Increasing Subsequence (LIS)
2. Faster Algorithm for LIS
3. Longest Common Subsequence (LCS)
4. Edit Distance

Longest Increasing Subsequence

Given n numbers $a_1, \dots, a_n$, a subsequence is a subset $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ with $i_1 < i_2 < \cdots < i_k$.

A subsequence is *increasing* if $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$.

Input: n numbers $a_1, \dots, a_n$.

Output: an increasing subsequence of maximum length.

5, $\underline{1}$, 9, 8, 8, 8, $\underline{4}$, $\underline{5}$, $\underline{6}$, $\underline{7}$

$$\mathrm{LIS}(i, a)$$

$$\|$$

$$\max\left\{ \begin{array}{l} \mathrm{LIS}(i+1, a), \\ \mathrm{LIS}(i+1, a_i) \text{ if } a_i > a \end{array} \right\}$$

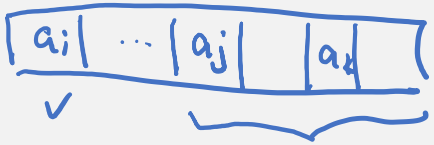
Recurrence

Subproblems: Let $L(i)$ be the length of a longest increasing subsequence starting at a_i and using the numbers in $\{a_i, a_{i+1}, \dots, a_n\}$ only.

answer: $\max\limits_{1 \le i \le n} \{L(i)\}$.

base case: $L(n) = 1$.

recurrence:

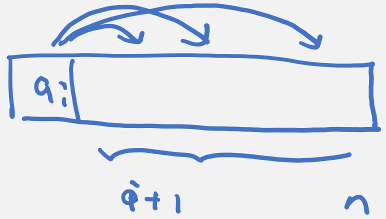


$$L(i) = \max_{i+1 \le j \le n} \{ 1 + L(j) \mid a_j > a_i \}$$

Bottom-Up Implementation

```
L(i) = 1    ∀ 1 ≤ i ≤ n    // initialization
for i from n downto 1 do
    for j from i+1 to n do
        if a_j > a_i and L(j)+1 > L(i)
            then update L(i) ← L(j)+1 .
```

$L(n+1) = 0$



$$\max_{i+1 \le j \le n} \{ 1 + L(j) \mid a_j > a_i \}$$

Time complexity: $\Theta(n^2)$

Example: 3, 8, 7, 2, 6, 4, 12, 14, 9

4, 3, 3, 4, 3, 3, 2, 1, 1   L-values

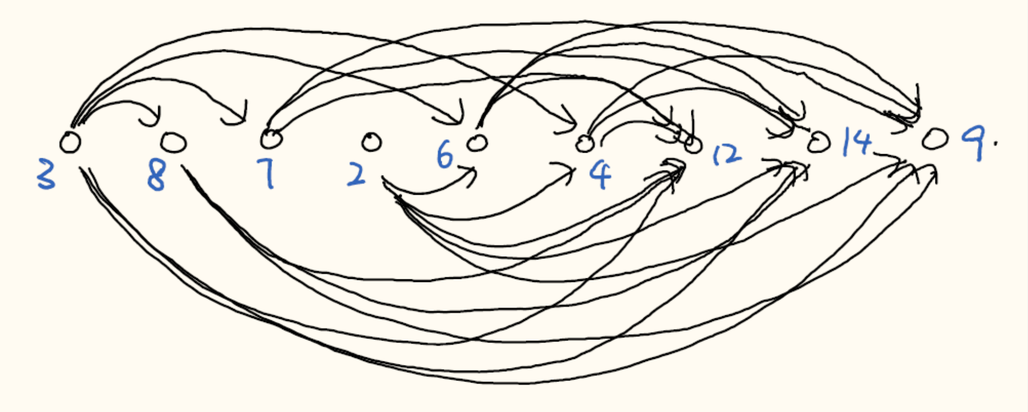
Longest Path in Directed Acyclic Graphs

Printing a solution:

① parent

② look at table

The longest increasing subsequence can be reduced to finding a longest path in a directed acyclic graph.



Claim a directed path

⟺ an increasing subsequence

Exercise: Design a dynamic programming algorithm to find a longest path in a directed acyclic graph.

Today's Plan

1. Longest Increasing Subsequence (LIS)
2. Faster Algorithm for LIS
3. Longest Common Subsequence (LCS)
4. Edit Distance

Hw3 deadline

Wed July 7 , 11pm

midterm

Faster Algorithm: Overview

Now we present a clever implementation that solves LIS in $O(n \log n)$ time.

The bottleneck is to find an increasing subsequence to extend a_i, which takes $O(n)$ times.

The observation is that we don't need to store all the subproblems for future computation,
because some subproblems are "dominated" by other subproblems.

For each length k, we will only store the "best" position to start an increasing subsequence of length k.

Then, these "best subproblems" satisfy a monotone property, which allows us to use binary search.

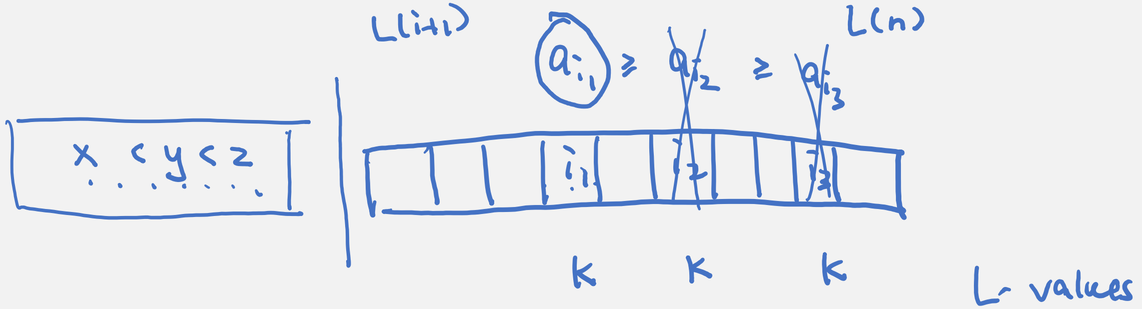
For example, given …………, 3, 8, ~~7~~, ~~2~~, ~~6~~, ~~4~~, 12, 14, ~~9~~

4, 3, 3, 4, 3, 3, 2, 1, 1 L-values

Best Subproblems

Suppose we have already computed $L(i+1), \dots, L(n)$, and we would like to compute $L(i)$.

Suppose $L(i_1) = \cdots = L(i_l) = k$, what is the best subproblem to keep for computation of $L(1), \dots, L(i)$?



Define $pos[k] = \arg\max_{j>i}\{a_j \mid L(j) = k\}$ to be the best position to extend a subsequence of length k.

Let $m = \max_{i+1 \leq j \leq n}\{L(j)\}$ be the length of a longest increasing subsequence we have computed so far.

Then we will only store the subproblems $L(pos[1]), L(pos[2]), \dots, L(pos[m])$ for future computations.

For example, ……,

↓	↓				↓		
2,	7,	6,	1,	4,	8,	5,	3
3	2	2	3	2	1	1	1

pos[1] = n-2

pos[2] = n-7

pos[3] = n-8

Monotonicity

Once we only keep the best subproblems, then we have the following useful property.

Claim. $a(pos[1]) > a(pos[2]) > \cdots > a(pos[m])$.

6, 7, 8

$a(pos[3]) = 10$

$a(pos[4]) = 2$

Intuition: A longer increasing subsequence is harder to be extended than a shorter increasing subsequence.

proof suppose by contradiction $a(pos[j]) \geq a(pos[j-1])$

$$\underset{\uparrow}{a_{i_1}} < \underline{\underline{a_{i_2}}} < \dots < a_{i_j}$$

$i_1 = pos[j]$

a position to start an increasing subseq of length j-1

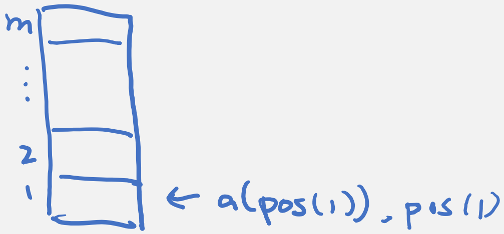
$$a_{i_2} > a_{i_1} = a(pos[j]) \geq a(pos[j-1])$$

Contradiction, because i_2 is a better place to start an inc subseq of length j-1.

$\square$

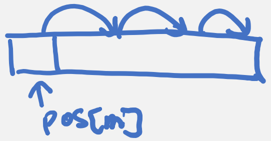
Updating the Best Subproblems by Binary Search

Claim. $a(pos[m]) < a(pos[m-1]) < \cdots < a(pos[2]) < a(pos[1])$.



consider the element a_i

① $a_i < a(pos[m])$ $a_i <$



great, form longer increasing subseq

$m \leftarrow m+1$ $pos[m] \leftarrow i$

② $a(pos[j]) \leq a_i < a(pos[j-1])$

cannot use a_i to form an increasing subseq of length $\geq j+1$

but can use a_i . . . . . of length j

i would at least as good a starting number for length j $\Rightarrow$ $pos[j] = i$

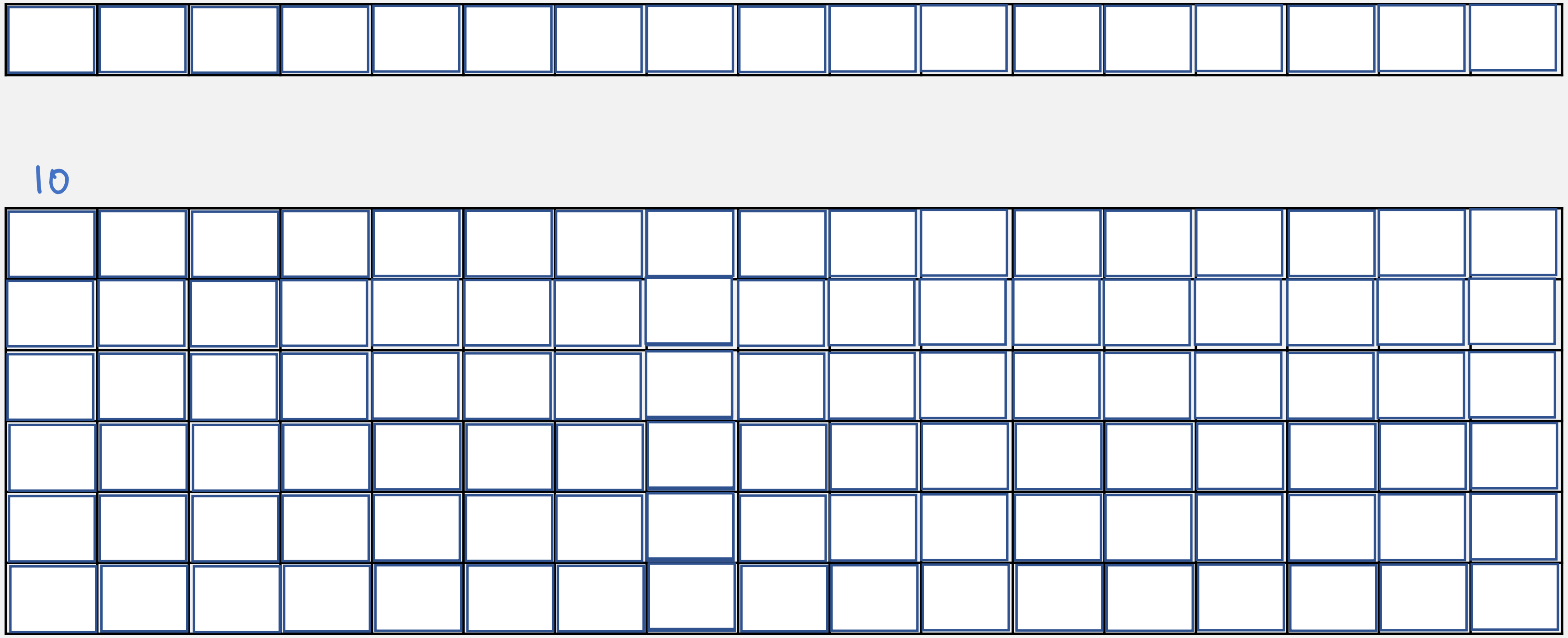
③ $a(pos[1]) < a_i$ $pos[1] = i$

Fast Algorithm

```
$m=1$ ,    $pos[1] = n$ .        // base case.
for $i$ from $n-1$ downto 1 do
    if $a_i < a[pos[m]]$ , then set $m \leftarrow m+1$ and $pos[m] = i$ .   // longer increasing subsequence
                                                                        $< a[pos[j-1]])$
    else use binary search to find the smallest $j$ so that $a[pos[j]] \leq a_i$ , then set $pos[j] = i$ .
return $m$ .
```

Example

10, 18, 25, 6, 70, 32, 2, 40, 11, 38, 21, 33, 86, 17, 51, 24, 57



L-values

10

← a(pos(2))

← a(pos(1))

Example

10, 18, 25, 6, 70, 32, 2, 40, 11, 38, 21, 33, 86, 17, 51, 24, 57

7	6	5	6	2	4	6	3	5	3	4	3	1	3	2	2	1

L- values

10

18	18	6	6	2	2	2										
25	25	25	11	11	11	11	11	11								
32	32	32	32	32	32	21	21	21	21	21						
40	40	40	40	40	40	40	40	38	38	33	33	17	17			
70	70	70	70	70	51	51	51	51	51	51	51	51	51	51	24	
86	86	86	86	86	86	86	86	86	86	86	86	86	57	57	57	57

← a(pos(2))

← a(pos(1))

Today’s Plan

1. Longest Increasing Subsequence (LIS)
2. Faster Algorithm for LIS
3. Longest Common Subsequence (LCS)
4. Edit Distance

Longest Common Subsequence

Input: Two string $a_1 a_2 \ldots a_n$ and $b_1 b_2 \ldots b_m$, where each a_i, b_j is a symbol.

Output: The largest k such that there exist $i_1 < \cdots < i_k$ and $j_1 < \cdots < j_k$ so that $a_{i_l} = b_{j_l}$ for $1 \le l \le k$.

One example is that we are given two DNA sequences and want to identify common structure.

$S_1 = AAACCGTGAGTTATTCGTTCTAGAA$

$S_2 = CACCCCTAAGGTACCTTTGGTTC$

$ACCTAGTACTTTG$

Note that longest increasing subsequence (LIS) is a special case of longest common subsequence (LCS).

3, 1, 2, 8, 9, 7

(LIS)

→

3, ①, ②, 8, 9, ⑦

①, ②, 3, ⑦, 8, 9 sorted

(LCS)

Recurrence

<u>Subproblems</u>: Let $C(i, j)$ be the length of a longest common subsequence of $a_i \ldots a_n$ and $b_j \ldots b_m$.

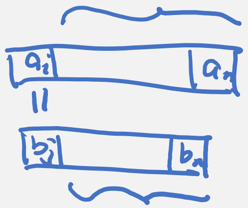
answer: $C(1,1)$

base cases: $C(n+1, j) = 0 \quad \forall j \qquad C(i, m+1) = 0 \quad \forall i$

recurrence: $C(i, j)$

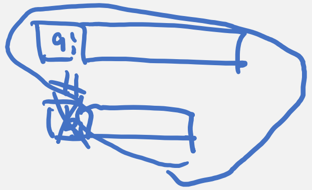
① if $a_i = b_j$, then $sol_1 = 1 + C(i+1, j+1)$

else, then $sol_1 = 0$



② drop a_i, then $sol_2 = C(i+1, j)$

③ drop b_j, then $sol_3 = C(i, j+1)$



$C(i,j) = \max\{sol_1, sol_2, sol_3\}$.

Analysis

correctness : recurrence

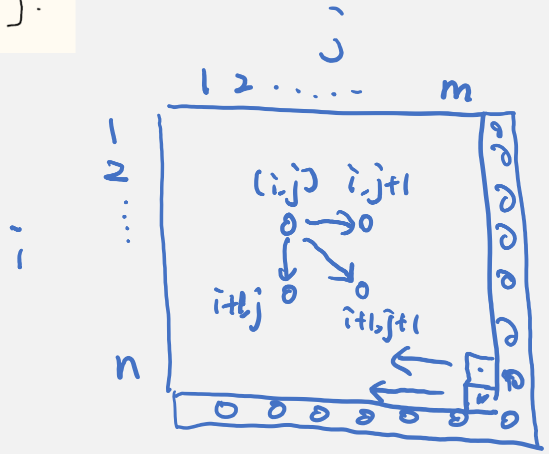
time complexity : $O(n \cdot m)$ subproblems

each takes $O(1)$ time

total $O(n \cdot m)$

Bottom-Up Implementation

```
$C(i, m+1) = 0 \quad \forall\ 1 \le i \le n$ ,   $C(n+1, j) = 0 \quad \forall\ 1 \le j \le m$ .     // base cases
for i from n downto 1 do
    for j from m downto 1 do
        if $a_i = b_j$ , set $SoL \leftarrow 1 + C(i+1, j+1)$ , else $SoL \leftarrow 0$.
        $C(i,j) = \max\{ SoL, C(i+1, j), C(i, j+1) \}$.
```



Today’s Plan

1. Longest Increasing Subsequence (LIS)
2. Faster Algorithm for LIS
3. Longest Common Subsequence (LCS)
4. Edit Distance

Edit Distance

Input: Two string $a_1 a_2 \dots a_n$ and $b_1 b_2 \dots b_m$, where each a_i, b_j is a symbol.

Output: The minimum k s.t. we can do k add/delete/change operations to transform $a_1 \dots a_n$ to $b_1 \dots b_m$.

S	—	N	O	W	Y
S	U	N	N	—	Y

Cost: 3

—	S	N	O	W	—	Y
S	U	N	—	—	N	Y

Cost: 5

Recurrence

Subproblems: Let $D(i, j)$ be the edit distance of $a_i \ldots a_n$ and $b_j \ldots b_m$.

answer : $D(1,1)$

base case : $D(n+1, m+1) = 0$

recurrence: $D(i,j)$

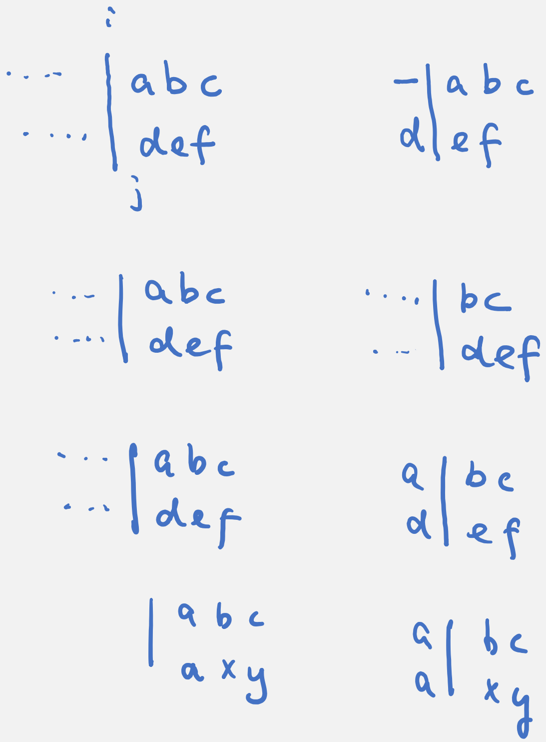
① ADD if $j \leq m$, $SOL_1 = 1 + D(i, j+1)$
else, $SOL_1 = \infty$

② DELETE if $i \leq n$, $SOL_2 = 1 + D(i+1, j)$
else $SOL_2 = \infty$

③ CHANGE if $i \leq n$ & $j \leq m$, $SOL_3 = 1 + D(i+1, j+1)$
else $SOL_3 = \infty$

④ MATCH if $i \leq n$ & $j \leq m$ & $a_i = b_j$, $SOL_4 = D(i+1, j+1)$
else $SOL_4 = \infty$

$D(i,j) = \min(SOL_1, SOL_2, SOL_3, SOL_4)$.



Analysis

Correctness : recurrence

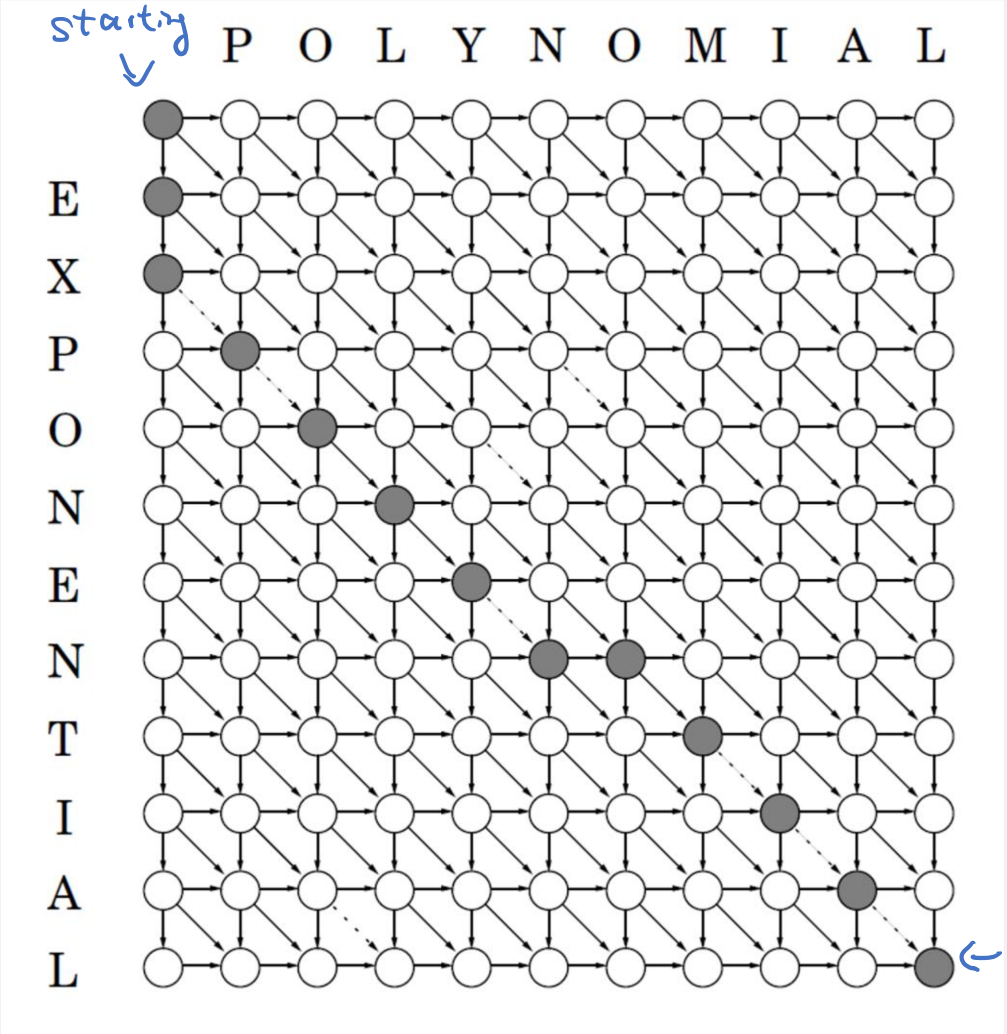
time : $O(nm)$ subproblems, each $O(1)$ time

total $O(nm)$.

Important Exercise: Bottom-up implementation.

Recent Result: Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)

Graph Searching



starting ↓

← target

↓ add

→ delete

⇢ match

↘ change