ALGORITHM DESIGN USING DYNAMIC PROGRAMMING METHOD

Partha P Chakrabarti

Indian Institute of Technology Kharagpur

Overview of Algorithm Design

1. Initial Solution
   a. Recursive Definition – A set of Solutions
   b. Inductive Proof of Correctness
   c. Analysis Using Recurrence Relations
2. Exploration of Possibilities
   a. Decomposition or Unfolding of the Recursion Tree
   b. Examination of Structures formed
   c. Re-composition Properties
3. Choice of Solution & Complexity Analysis
   a. ~~Balancing the Split~~, Choosing Paths
   b. Identical Sub-problems
4. Data Structures & Complexity Analysis
   a. Remembering Past Computation for Future
   b. Space Complexity
5. Final Algorithm & Complexity Analysis
   a. Traversal of the Recursion Tree
   b. Pruning
6. Implementation
   a. Available Memory, Time, Quality of Solution, etc

1. Core Methods
   a. Divide and Conquer
   b. Greedy Algorithms
   c. Dynamic Programming
   d. Branch-and-Bound
   e. Analysis using Recurrences
   f. Advanced Data Structuring
2. Important Problems to be addressed
   a. Sorting and Searching
   b. Strings and Patterns
   c. Trees and Graphs
   d. Combinatorial Optimization
3. Complexity & Advanced Topics
   a. Time and Space Complexity
   b. Lower Bounds
   c. Polynomial Time, NP-Hard
   d. Parallelizability, Randomization

Basics of Dynamic Programming Method

1. Recursive Decomposition → optimal sub-structure (optimization)
2. HANDLING IDENTICAL SUB-PROBLEMS ⟸ exponential time complexity
3. MEMOIZATION & REUSE — Remember → Data Storage
4. Evaluation
   - A) Top-down ✓ done[]
   - B) Iterative Bottom up ⟸ Acyclic structure
5. Data Structures → preprocessing

a) Fibonacci (Pingala) ✓

b) Matrix Chain Multiplication ✓

c) String Related
   - Longest Common Subsequence
   - Sequence Alignment
   - NLP related problems

d) Matrix operations

e) Graph Algorithms

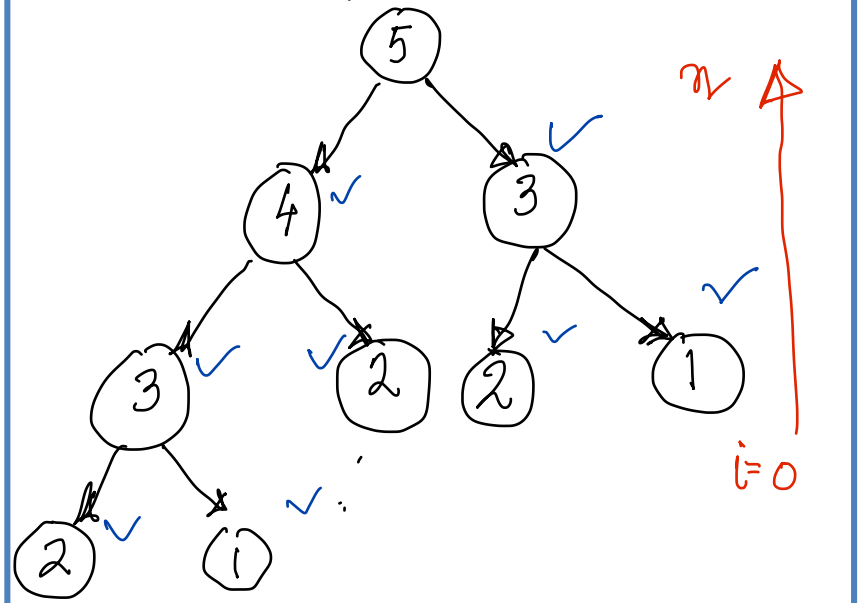
f) Coins / Knapsack

g) optimal BST

Revising Fibonacci-like Structures

$$f(n) = f(n-1) + f(n-2), \quad n \geq 2$$
$$= 0, \quad n = 0$$
$$= 1, \quad n = 1$$

$n-k$ constant k



F[ ], Done[ ]

Done[0] = Done[1] = 1
All other Done[i] = 0
F[0] = 0 F[1] = 1

```
eval-f(n)
{ if (Done[n] = 1) return (F[n])
  m = eval-f(n-1) + eval-f(n-2)
  {Done[n] = 1
  F[n] = m
  return(F[n])
}
```

```
F[0]=0, F[1]=1
for i = 2 to n do
  F[i] = F[i-1]
       + F[i-2]
```

only 2 addnl variables

Fibonacci-like Structures (cntd.)

Ackermann's Function

$f(n) = r(n)$ if $c(n)$ is true

$= f(g(n)) + f(h(n))$ if $c(n)$ is false

where $c(n)$, $r(n)$, $g(n)$, $h(n)$ are not recursive and can be computed deterministically

F[]

Done[i]
- 0 if it has not been evaluated
- 1 if evaluation has begun but not completed
- 2 if final value is computed

check for cycles

Top-down

```
eval-f(n)
{ if (Done[n]=2) return (F[n])
  if (c(n)=true)                    (2) Base
  { Done[n]=2, F[n]=r(n)
    return (F[n]) }
  if (Done[n]=1) return ("CYCLE")   (3) cycle detection
  Done[n]=1                         (4) Recursive
  x=g(n), y=h(n)
  z=eval-f(x)+eval-f(y)
  F[n]=z
  Done[n]=2
  return (F[n])
}
```

(1)

$M_1 \times M_2 \times \cdots \times M_n$

MATRIX CHAIN MULTIPLICATION Problem

$M_1 \times M_2 \times \cdots \times M_n$

(M1 X (M2 X (M3 X M4))) = ((M1 x M2) X (M3 X M4)) = (((M1 x M2) X M3) X M4) = (M1 X (M2 X M3)) X M4) $\Leftarrow$ $M_1 \times (M_2 \times M_3) \times M_4$

BUT THE NUMBER OF MULTIPLICATIONS TO GET THE ANSWER DIFFER !!

Let A be a [p by q] Matrix and B be a [q by r] Matrix. The number of multiplications needed to compute A X B = p*q*r

$C(1) = \emptyset\ 1$
$C(2) = 1$

$M_1 \times M_2 \times M_3 \times M_4$
$[P_0 P_1]\ [P_1 P_2]\ [P_2 P_3]\ [P_3 P_4]$

associative

Thus if M1 be a [10 by 30] Matrix, M2 be a [30 by 5] Matrix and M3 be a [5 by 60] Matrix

Then the number of computations for

$$C(n) = \sum_{i=1}^{n-1} C(i) * C(n-i) = C_1 * C_1$$

5 ways

$$\frac{n(n-1)}{2} - 1$$

P

(M1 X M2) X M3 = 10*30*5 for P = (M1 X M2) and 10*5*60 for P X M3. Total = 4500

M1 X (M2 X M3) = 30*5*60 for Q = (M2 X M3) and 10*30*60 for M1 X Q. Total = 27000

$$C(n) = \sum C(i) * C(n-i)$$

Matrix Chain Multiplication: Recursive Definition

$M_1 \times M_2 \times M_3 \times \cdots \times M_n$

$[p_0, p_1]\ [p_1, p_2]\ [p_2, p_3] \quad [p_{n-1}, p_n]$

m_{ij} = optimal multiplications for multiplying $M_i \times M_{i+1} \cdots \times M_j$

M_i to M_j

$$m_{ij} = \begin{cases} 0 & \text{if } i = j \\ \min_{k=i}^{j-1} \{ m_{ik} + m_{k+1,j} + p_{i-1} * p_k * p_j \} & \text{if } i < j \end{cases}$$

$M_{ik} \times (M_{k+1}, j)$

m_{ij} → No. of computations to multiply $M_i \cdots M_j$

$M_1 = 10 \times 30$

$M_2 = 30 \times 5$

$M_3 = 5 \times 60$

$M_4 = 60 \times 4$

$p_0 = 10$

$p_1 = 30$

$p_2 = 5$

$p_3 = 60$

$p_4 = 4$

$[p_0 p_1]\ [p_1 p_2]\ [p_2 p_3]$

$M_1 \times M_2 \times M_3$

$[p_1, p_2]\ [p_2, p_3]\ [p_3, p_4]$

$M_2 \rightarrow 0$

$M_3 \times M_4$

$[p_2, p_3]\ [p_3, p_4]$

$p_2 * p_3 * p_4$

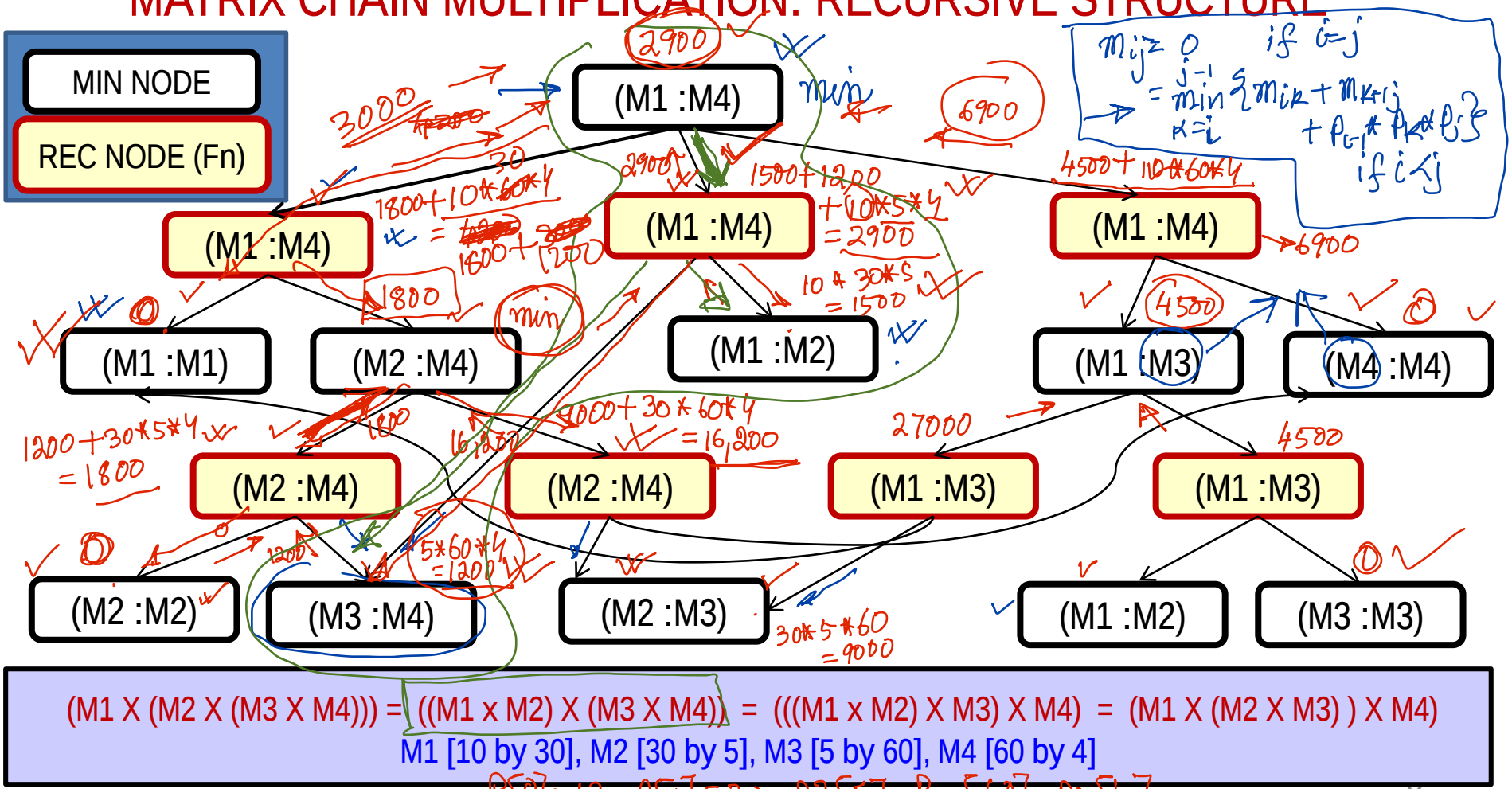
$(M_i \cdots M_k) \times (M_{k+1} \cdots M_j)$

m_{ik} $\quad$ $m_{k+1,j}$

$[p_{i-1}\ p_k]\ [p_k\ p_j]$

$p_{i-1} * p_k * p_j$

MATRIX CHAIN MULTIPLICATION: RECURSIVE STRUCTURE



Matrix Chain Multiplication: Top-Down Evaluation

M[i,j], Done[i,j] = 0

```
eval-m (i,j)
{ if (Done[i,j] = 1) return (M[i,j])
  if (i=j) { Done[i,j]=1;
             M[i,j]=0;
             return (M[i,j]) }
  val = ∞
  for (k = i to j-1)
  { v_k = eval-m (i,k) +
          eval-m (k+1, j)
          + p[i-1] * p[k] * p[j]
    if (v_k < val) val = v_k
  }
```

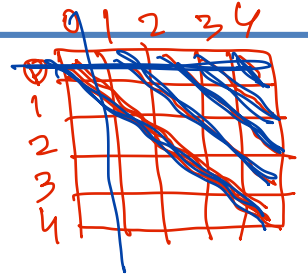
REUSE

BASE

RECURSIVE

(1, n)

Done[i,j] = 1
M[i,j] = val
return (M[i,j])



$n^2 \rightarrow M[i,j]$

n

$O(n^3)$

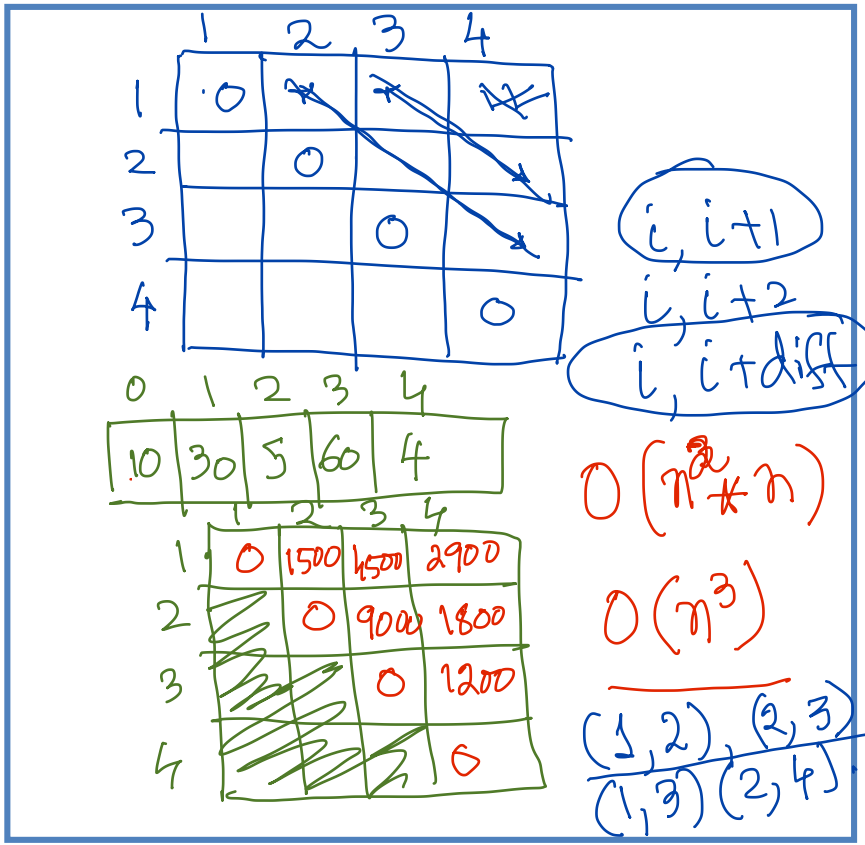
I can decipher my acyclic dependency structure

(i, i)
(i, i+1)
(i, i+2) (i, i+3) .... (i, n)

$O(n^2 * n) = O(n^3)$

Bottom-up iterative algorithm

Matrix Chain Multiplication: Iterative Evaluation




```
iterativeEval()
{for (i = 1 to n) M[i,i] = 0
 for (diff = 1 to n-1)
  for (i = 1 to n-diff)
   { j = i + diff
     M[i,j] = ∞
     for (k = i to j-1)
     { q = M[i,k] + M[k+1,j]
           + p[i-1] * p[k] * p[j]
       if (q < M[i,j]) M[i,j] = q
     }
   }
```

$O(n \cdot n \cdot n) = O(n^3)$

String Matching Problems

1. Pattern Matching in a Text

S: string of characters
P: string of characters
Find occurrences of P in S

(a) Exact Match
(b) Approximate match

S: a a b a c a a b a b a c a a
P: a b a b a c

S: a b a b a b a c
P: a b a

2. Sequence Alignment Problem

X: INTERACTION
Y: CONTRADICT

NTRACT CTI NTRAI

a) All matches of length $\geq k$
b) Longest match
↳ Longest Common Subsequence

Protein Alignment

A T C G

X: C G A T A A T T G A G A
Y: G T T C C T A A T A

EDIT DISTANCE PROBLEMS

Exact Pattern Matching in a String

$S = \{s_1, s_2, \ldots, s_n\}$

$P = \{p_1, p_2, \ldots, p_m\}$

```
match (S, P)
{ if (|S| < |P|) return 0;
  if (|P| = 0) return 1;
     x = 0; y = 0
  if (s1 = p1)
     {x = match_exact(S - {s1}, P - {p1})}
     y = (match (S - {s1}, P))
  return (x ∨ y)
}
```

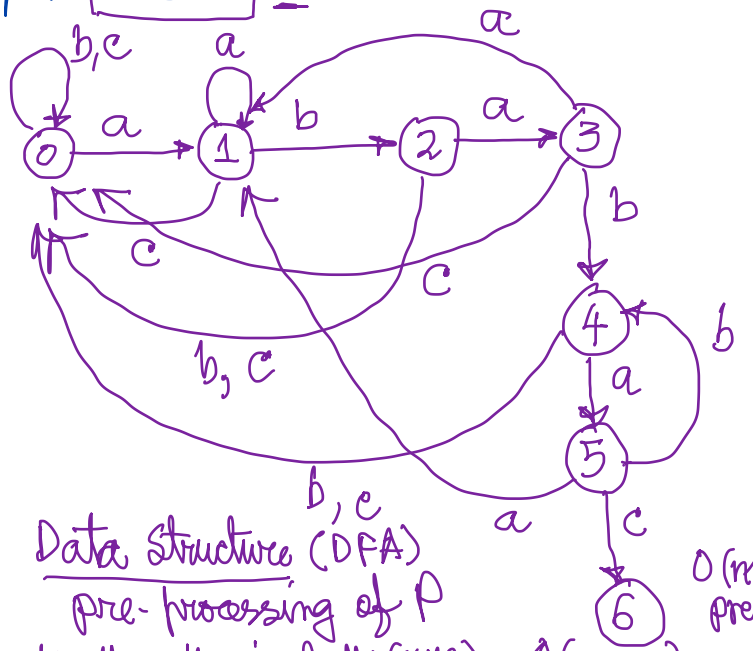
match_exact (S, P) has to be written separately

$O(n.m)$

easily converted into iteration

S: a a b a c a a b a b a c a a

P: a b a b a c



Data Structure (DFA)

pre-processing of P

Knuth - Morris - Pratt (KMP)

$O(n+m)$

$O(m)$ preprocessing

Longest Common Subsequence (LCS)

X: INTERACTION
Y: CONTRADICT
Z_1: NTRACT ←
Z_2: NTRAI
Z_3: CTI

S1: C G A T A A T T G A G A
S2: G T T C C T A A T A

a) Longest Common Subsequence
b) All Common Subsequences of
length $\geq k$
or length $\geq (1-\varepsilon)$ LCS

- associative
- mutations / commutativity

More General Formulation
↳ EDIT Distance

LCS: Recursive Formulation

```
LCS(X, Y)
{ if (X = NULL or Y = NULL)
        return (NULL);
  Let X = {x1, x2, ..., xn}
      Y = {y1, y2, ..., ym}
  if (x1 = y1)
  { Z1 = x1 || LCS(X - {x1}, Y - {y1})
    return (Z1)
  }
  else
{
  Z2 = LCS(X - {x1}, Y);
  Z3 = LCS(X, Y - {y1});
  Z = max(Z2, Z3)
  return (Z)
}
}
```

X: YET
Y: YES
↳ ET, ES

INTERACTION
CONTRADICT

Z_2: X = NTERACTION
Y = CONTRADICT

Z_3: X = INTERACTION
Y = ONTRADICT

LCS: Dynamic Programming

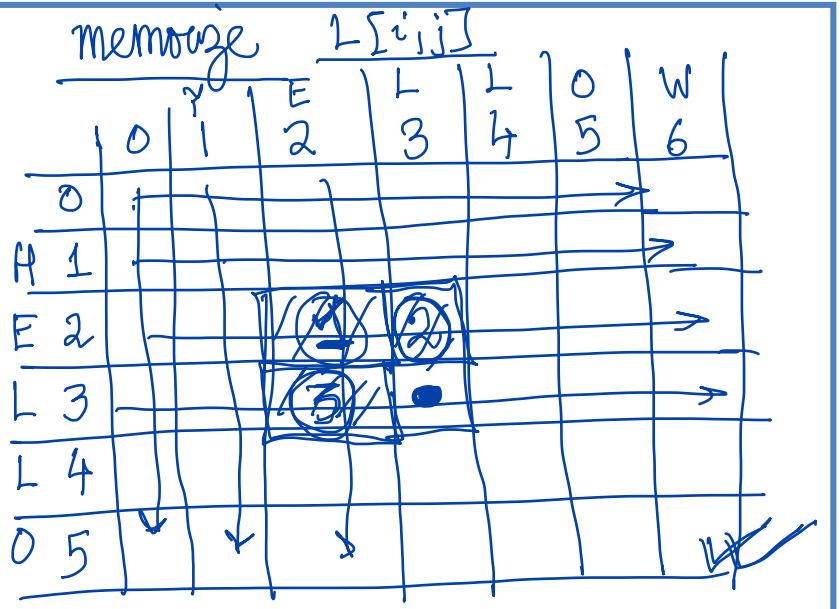
$$LCS(X_i, Y_j) = 0 \quad \text{if } i=0 \text{ or } j=0$$

$$= LCS(X_{i-1}, Y_{j-1}) + 1 \quad \text{if } x_i = y_j \ \& \ i>0, j>0$$

$$= \max\{ LCS(X_{i-1}, Y_j), LCS(X_i, Y_{j-1}) \}$$

X = HELLO
Y = YELLOW

} ELLO



LCS(X_i, Y_j)

Top-down , Bottom-up

LCS: Dynamic Programming Implementation

L[i,j]

```
LCS_iter( )
{  for (i = 1 to n)   L[i,0] = 0
   for (j = 1 to m)   L[0,j] = 0

   for (i = 1 to n)
     for (j = 1 to m)
     { if (X[i] = Y[j])
           L[i,j] = L[i-1,j-1] + 1
       else
           L[i,j] = max{ L[i-1,j],
                         L[i,j-1] }
     }
}
```

O(n.m)

HELLO , YELLOW

P[i,j]

		Y 1	E 2	L 3	L 4	O 5	W 6
0	0	0	0	0	0	0	0
H 1	0	0	0	0	0	0	0
E 2	0	0	1	1	1	1	1
L 3	0	0	1	2	2	2	2
L 4	0	0	1	2	3	3	3
O 5	0	0	1	2	3	4	4

O(nm) space

ELLO

Edit Distance

X: HELLO
Y: YELLOW

Spelling errors
Letter switches
Typos

General Approximate Match

S1: HELLO
S2: YELLOW

Transforming S1 to S2 using ins, del, subst

→ subst (H, Y)
→ delete (H) or insert (Y)

costs for each of
insert
delete
substitute

match: cost = 0

ex: ins: 1, del: 1, subs: 2

ins(Y), del(H)

	HELLO	(YELLOW)
(ins Y)	YHELLO	(ELLOW)
del (H)	YELLO	(ELLOW)
match	YELLO	(LLOW)
match	YELLO	(LOW)
match	YELLO	(OW)

match (YELLO_) (W)
ins(W)
YELLOW (Ø)

cost 3

Edit Distance: Formulation

$$d[i,0] = \sum_{k=1}^{i} del_i$$

$$d[0,j] = \sum_{k=1}^{j} ins_j$$

$$d[i,j] = d[i-1,j-1] \quad \text{if } x_i = y_j$$

$$= \min \begin{cases} d[i-1,j] + del_i, \\ d[i,j-1] + ins_j, \\ d[i-1,j-1] + sub_{i,j} \end{cases}$$

costs: ins = 1, del = 1
subst = 2

	0	Y_1	E_2	L_3	L_4	O_5	W_6
0	0	1	2	3	4	5	6
H 1	1	2	3	4	5	6	7
E_2	2	3	2	3	4	5	6
L_3	3	4	3	2	3	4	5
L_4	4	5	4	3	2	3	4
O_5	5	6	5	4	3	2	3

$O(m,n)$ time
$O(m \cdot n)$ space ← memorized space

Edit Distance using Dynamic Programming

As a practice write down

a) Top-down algorithm with memorization

b) Bottom-up iterative algorithm

Variations

1. More operators like exchanges in rows (commutativity)

   thouhg ~~s~~ through

2. Various kinds of approximate matches

3. Multidimensional matching or alignment

→ HASH TABLES

Dynamic Programming

1. Knapsack Problem
2. Matrix, Graph Path
3. Optimal Weighted BST

Thank you

Any Questions?