

# Programming and Data Structures with Python

Madhavan Mukund

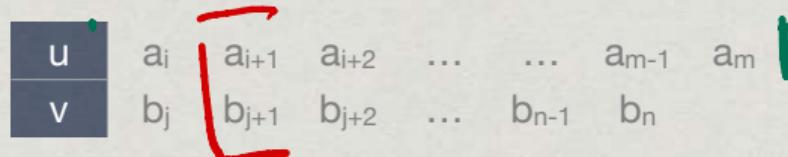
<https://www.cmi.ac.in/~madhavan>

18 March, 2021

# Edit distance

- \* Minimum number of editing operations needed to transform one document to the other
  - \* Insert a character
  - \* Delete a character
  - \* Substitute a character by another one
- \* In our example,  
28 characters inserted, 18 **deleted**, 2 **substituted**
- \* Edit distance is at most 48

# Inductive structure



$a_i = b_j$  by  
subst.

Drop  $a_i \sim b_j$

- \*  $\text{ED}(i,j)$  stands for  $\text{ED}(a_i a_{i+1} \dots a_m, b_j b_{j+1} \dots b_n)$
- \* If  $a_i = b_j$ ,  $\text{ED}(i,j) = \text{ED}(i+1,j+1)$  ✓
- \* If  $a_i \neq b_j$ ,  $\text{LCS}(i,j) =$   
$$1 + \min(\underbrace{\text{ED}(i+1,j+1)}, \underbrace{\text{ED}(i+1,j)}, \underbrace{\text{ED}(i,j+1)})$$
- \* As with LCS/LCW, extend positions to  $m+1, n+1$ 
  - \*  $\text{ED}(m+1,j) = n-j+1$  for all  $j$  # Insert  $b_j b_{j+1} \dots b_n$  in  $u$
  - \*  $\text{ED}(i,n+1) = m-i+1$  for all  $i$ , # Insert  $a_i a_{i+1} \dots a_m$  in  $v$

# Subproblem dependency

- \* Like LCS,  $ED(i,j)$  depends on  $ED(i+1,j+1)$ ,  $ED(i+1,j)$  and  $ED(i,j+1)$
- \* Dependencies for  $ED(m,n)$  are known
- \* Start at  $ED(m,n)$  and fill by row, column or diagonal

	0	1	2	3	4	5	6
	s	e	c	r	e	t	.
0	b						
1	i						
2	s						
3	e						
4	c						
5	t						
6	.						

Base

The diagram shows a 7x8 grid representing the Edit Distance table. The columns are labeled 0 through 6, and the rows are labeled 0 through 6. The first row and column are for the empty string. The second row contains 's', the third 'e', the fourth 'c', the fifth 'r', the sixth 'e', and the seventh '.'. The first column contains 'b', 'i', 's', 'e', 'c', 't', and '.'. A green box highlights the cell at (3,3) containing 'e'. Yellow arrows point to this cell from three other cells: (2,2), (3,2), and (2,3). A green line traces a path from the bottom-left corner (0,0) towards the highlighted cell. A green bracket is placed under the row 'e' to indicate it is the current row being processed.

$v \rightarrow u$     $t$   
 $t$

## Subproblem dependency

- \* Like LCS,  $ED(i,j)$  depends on  $ED(i+1,j+1)$ ,  $ED(i+1,j)$  and  $ED(i,j+1)$
- sect*  
*sect*

- \* Dependencies for  $ED(m,n)$  are known
- \* Start at  $ED(m,n)$  and fill by row, column or diagonal

$t \rightarrow w$   
*bisect*  
 $u \rightarrow v$    *bisect*  
*bisect*

	0	1	2	3	4	5	6
	s	e	c	r	e	t	.
0	b						5 6
1	i						4 5
2	s						3 4
3	e						2 3
4	c						1 2
5	t	5	4	3	2	1	0 1
6	.	6	5	4	3	2	1 0

$t$   
↓  
bisect

~~Secret~~

~~bisect~~

- \* Trace back the path
- \* Transforming “secret” to “bisect”
  - \* Del “b” :  $(0,0) \rightarrow (1,0)$
  - \* Del “i” :  $(1,0) \rightarrow (2,0)$
  - \* Ins “r” :  $(5,3) \rightarrow (5,4)$
  - \* Ins “e”:  $(5,4) \rightarrow (5,5)$

~~Del r~~  
~~Del e~~

## Recovering the solution

- modulo your expectation

	0	1	2	3	4	5	6
s	e	c	r	e	t	.	
0	b	4	5	5	5	5	6
1	i	3	4	4	4	4	5
2	s	2	3	3	3	3	4
3	e	3	2	3	2	2	3
4	c	4	3	2	2	1	2
5	t	5	4	3	2	1	0
6	.	6	5	4	3	2	1

# ED(u,v), DP

```
function ED(u,v) # u[0..m], v[0..n]
    for r = 0,1,...,m+1 { ED[r][n+1] = m-r+1 }
    for c = 0,1,...,m+1 { ED[m+1][c] = n-c+1 }

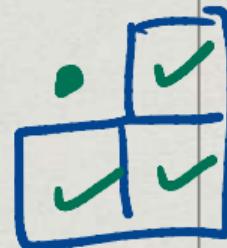
    for c = n,n-1,...,0
        for r = m,m-1,...0
            if (u[r] == v[c])
                ED[r][c] = ED[r+1][c+1]
            else
                ED[r][c] = 1 + min(ED[r+1][c+1],
                                     ED[r+1][c],
                                     ED[r][c+1])

    return(ED[0][0])
```

# Complexity

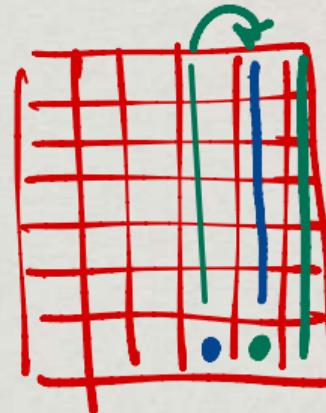


- \* Again  $O(mn)$  using dynamic programming (or memoization)
  - \* Need to fill an  $O(mn)$  size table
  - \* Each table entry takes constant time to compute



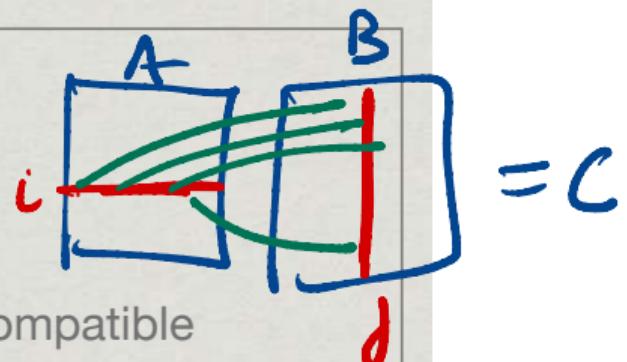
# Space complexity

- \* For LCW, LCS, ED
  - \* Need to fill an  $O(mn)$  size table
  - \* Do we need to store the entire table?
- \* Filling column by column, only need next column and current column
  - \* Or next row and current row
- \* Reduce space to  $O(n)$ , assuming  $m \geq n$



$2 \times n$   
 $2 \times m$

# Multiplying matrices



- To multiply matrices A and B, need compatible dimensions

\* A of dimension  $m \times n$ , B of dimension  $n \times p$

\*  $AB$  has dimension  $mp$

- Each entry in  $\underline{\underline{AB}}$  takes  $O(n)$  steps to compute

\*  $AB[i,j] = A[i,1]B[1,j] + A[i,2]B[2,j] + \dots + A[i,n]B[n,j]$

- Overall, computing  $AB$  is  $O(mnp)$

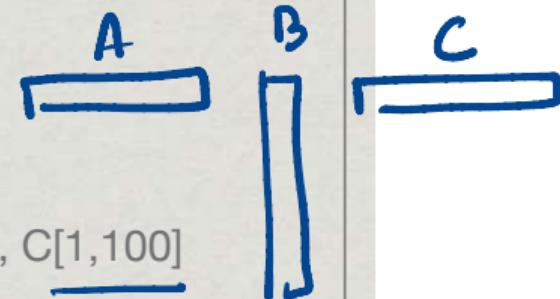
$$\begin{aligned} O[mn] &= O[np] \\ C[i,j] &= \sum_{k=1}^n A[i,k]B[k,j] \end{aligned}$$

*mp entries in C  
each takes  $O(n)$  time*

# Multiplying matrices

- \* Matrix multiplication is associative
  - \*  $ABC = (AB)C = A(BC)$
  - \* Bracketing does not change the answer ...
  - \* ... but can affect the complexity of computing it!

# Multiplying matrices



- \* Suppose dimensions are A[1,100], B[100,1], C[1,100]
  - \* Computing A(BC)
    - \* BC is [100,100],  $100 \times 1 \times 100 = 10000$  steps
    - \* A(BC) is [1,100],  $1 \times 100 \times 100 = 10000$  steps
  - \* Computing (AB)C
    - \* AB is [1,1],  $1 \times 100 \times 1 = 100$  steps
    - \* (AB)C is [1,100],  $1 \times 1 \times 100 = 100$  steps
  - \* A(BC) takes 20000 steps, (AB)C takes 200 steps!

$(M_1, N_2)$   $(M_3, 4)$

## Multiplying matrices

$((M_1(M_2 \quad M_3))M_4)_{L_1})$

- \* Given matrices  $M_1, M_2, \dots, M_n$  of dimensions  $[r_1, c_1], [r_2, c_2], \dots, [r_n, c_n]$

$m \cdot n \cdot p$   
operations

minimize  
total  
ops

$M_1 M_4$   $M_2 M_3$

- \* Dimensions match, so  $M_1 \times M_2 \times \dots \times M_n$  can be computed

$c_i = r_{i+1}$  for  $1 \leq i < n$

- \* Find an optimal order to compute the product



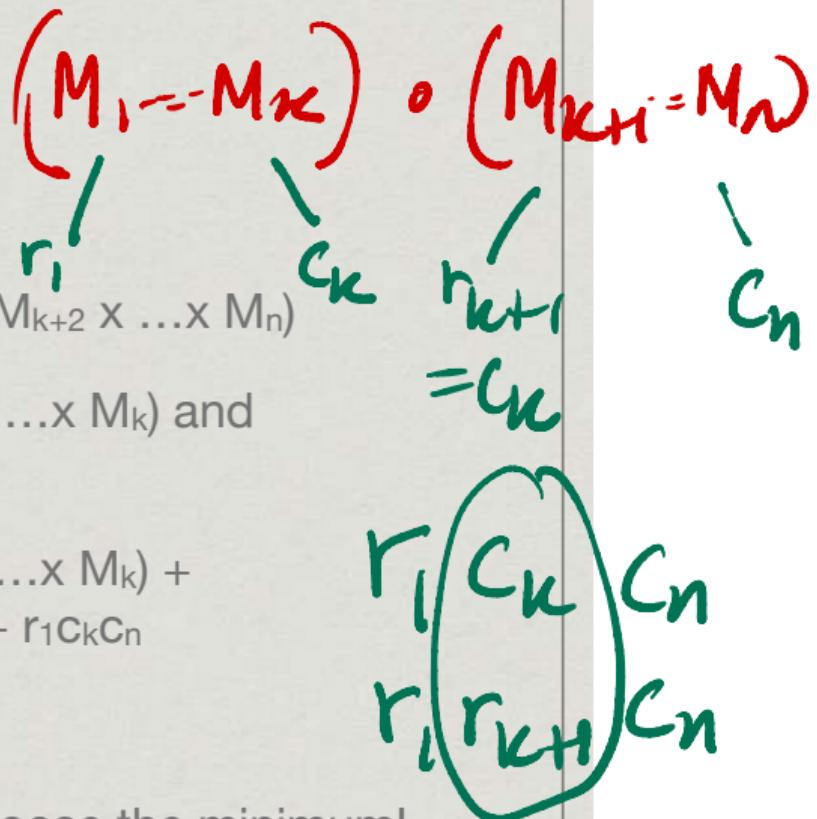
- \* That is, bracket the expression optimally

# Inductive structure

- \* Product to be computed:  $M_1 \times M_2 \times \dots \times M_n$
- \* Final step would have combined two subproducts
  - \*  $(M_1 \times M_2 \times \dots \times M_k) \times (M_{k+1} \times M_{k+2} \times \dots \times M_n)$ , for some  $1 \leq k < n$
  - \* First factor has dimension  $(r_1, c_k)$ , second  $(r_{k+1}, c_n)$
  - \* Final multiplication step costs  $O(r_1 c_k c_n)$
  - \* Add cost of computing the two factors

# Subproblems

- \* Final step is  $(M_1 \times M_2 \times \dots \times M_k) \times (M_{k+1} \times M_{k+2} \times \dots \times M_n)$
- \* Subproblems are  $(M_1 \times M_2 \times \dots \times M_k)$  and  $(M_{k+1} \times M_{k+2} \times \dots \times M_n)$
- \* Total cost is  $\text{Cost}(M_1 \times M_2 \times \dots \times M_k) + \text{Cost}(M_{k+1} \times M_{k+2} \times \dots \times M_n) + r_1 c_k c_n$
- \* Which k should we choose?
- \* No idea! Try them all and choose the minimum!



$i \in \{1, \dots, n\}$   $j \in \{i, \dots, n\}$

$O(n^2)$  problem

Inductive formulation

$M_1 M_2 M_3$

$M_4$

\* Cost( $M_1 \times M_2 \times \dots \times M_n$ ) =

minimum value, for  $1 \leq k \leq n$ , of

Cost( $M_1 \times M_2 \times \dots \times M_k$ ) +

Cost( $M_{k+1} \times M_{k+2} \times \dots \times M_n$ ) +

$r_1 c_k c_n$

\* When we compute Cost( $M_1 \times M_2 \times \dots \times M_k$ ) we will get subproblems of the form  $M_j \times M_{j+1} \times \dots \times M_k$

$M_L M_{L+1} \dots M_j$

$M_1 - M_j | M_{j+1} - M_K$

$M_{K+1} - M_n | M_{n+1} - M_n$

$M_1 M_2 M_3$   
+  
 $M_4$ ?  
 $+ r_1 c_3 c_4$

0

||

✓

Cost( $M_1 \times M_2 \times \dots \times M_k$ ) +

✓

Cost( $M_{k+1} \times M_{k+2} \times \dots \times M_n$ ) +

# In general ...

- \*  $\text{Cost}(M_i \times M_{i+1} \times \dots \times M_j) =$   
minimum value, for  $i \leq k < j$ , of
  - $\text{Cost}(M_i \times M_{i+1} \times \dots \times M_k) +$
  - $\text{Cost}(M_{k+1} \times M_{k+2} \times \dots \times M_j) +$
  - $r_i c_k c_j$
- \* Write  $\text{Cost}(i,j)$  to denote  $\text{Cost}(M_i \times M_{i+1} \times \dots \times M_j)$

# Final equation

- \*  $\text{Cost}(i,i) = 0$  — No multiplication to be done
- \*  $\text{Cost}(i,j) = \min$  over  $i \leq k < j$   
[  $\text{Cost}(i,k)$  +  $\text{Cost}(k+1,j)$  +  $r_i c_k c_j$  ]
- \* Note that we only require  $\text{Cost}(i,j)$  when  $i \leq j$

~~Cost~~  $\text{Cost}[i,j]$

# Subproblem dependency

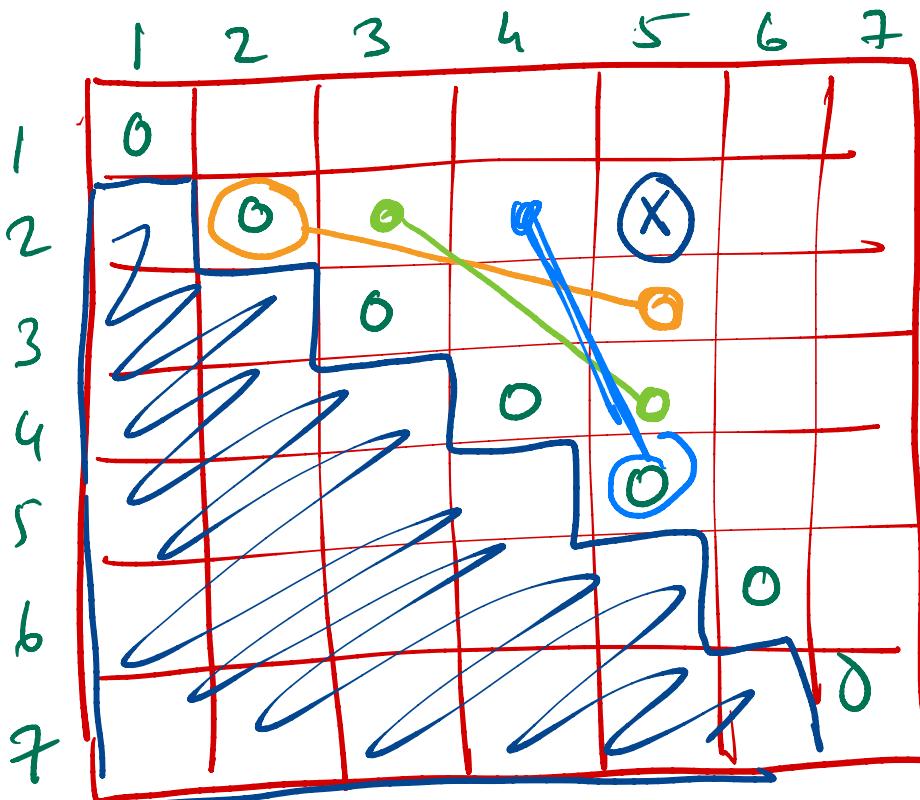
- \*  $\text{Cost}(i,j)$  depends on  $\text{Cost}(i,k)$ ,  $\text{Cost}(k+1,j)$  for all  $i \leq k < j$
- \* Can have  $O(n)$  dependent values, unlike LCS, LCW, ED
- \* Start with main diagonal and fill matrix by columns, bottom to top, left to right

	1	...	i	...	j	...	n
1	0						
...		0					
i			0				
...				0			
j					0		
...						0	
n							0

Diagram illustrating subproblem dependency in a dynamic programming matrix. The matrix has rows labeled 1, ..., i, ..., j, ..., n and columns labeled 1, ..., i, ..., j, ..., n. A blue shaded area highlights the submatrix from (i, i) to (j, j). Handwritten green annotations include:

- A wavy line starting from the cell (i, i) and ending at the cell (j, j), labeled "j < i".
- A red dot at the cell (j, i).
- Orange arrows pointing from (i, i) to (j, i) and from (j, i) to (j, j).
- Handwritten zeros in various cells, such as (i, 1), (j, 1), and (j, n).

$$\text{Cost}(M_i \cdot M_{i+1} \cdots M_j) = \text{Cost}(i, j)$$



$$\text{Cost}(i, i) \approx 0$$

$$\text{Cost}(2, 5)$$

$k=2$  |  $\text{Cost}(2, 2) +$   
 $\text{Cost}(3, 5)$   
 $+ r_2 c_2 c_5$

$k=3$  |  $\text{Cost}(2, 3) +$   
 $\text{Cost}(4, 5)$   
 $+ r_2 c_3 c_5$

$k=4$  |  $\text{Cost}(2, 4) +$   
 $r_2 c_4 c_5 + \text{Cost}(5, 5)$

O	X	✓	*	~	X	O
O	X	✓	*	~	X	
O	X	✓	*	~		
O	X	✓	*			
O	X	✓				

# MMCost(M1, ..., Mn), DP

```
function MMC(R,C)
# R[1..n],C[1..n] have row/column sizes
```

```
for r = 1,...,n
```

```
    MMC[r][r] = 0 ✓
```

```
for c = 2,3,...,n+1
```

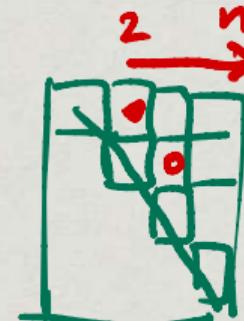
```
    for r = c,c-1,...,1 ||
```

```
        MMC[r][c] = infinity
```

```
        for k = r,r+1,...,c-1
```

```
            subprob = MMC[r][k] + MMC[k][c] +
                    R[r]C[k]C[c]
```

```
            if (subprob < MMC[r][c])
                MMC[r][c] = subprob
```



$$\begin{aligned}2-1 &= 1 \\3-2 &= 2\end{aligned}$$

|| 0(n)

# Complexity

$M_1 \ M_2 \ \dots \ M_n$

table

$n^2$

- \* As with LCS, ED, we have to fill an  $O(n^2)$  size table
- \* However, filling  $MMC[i][j]$  could require examining  $O(n)$  intermediate values
- \* Hence, overall complexity is  $O(n^3)$

$M_1 - M_3$   $M_4 - M_5$   
 $\underbrace{\qquad\qquad}_{M_1 M_2}$

$M_1 - M_4$   $M_5$   
 $M_1 M_2 \sim M_1 M_3$

$M_1 - M_5$   
 $k=4$   
 $k=3$

# Dynamic Programming - Intelligent brute force

Never overlook a subproblem

Never recompute a subproblem

Cost: # subproblems  $\times$  Cost to combine  
subproblems

↓  
If this is exponential,  
DP is also exponential

# Bin Packing

Fill a container with furniture

Every subset of furniture is a subproblem

Exponentially many subproblems

P vs NP