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

u	a_i	a_{i+1}	a_{i+2}	…	…	a_{m-1}	a_m
v	b_j	b_{j+1}	b_{j+2}	…	b_{n-1}	b_n	

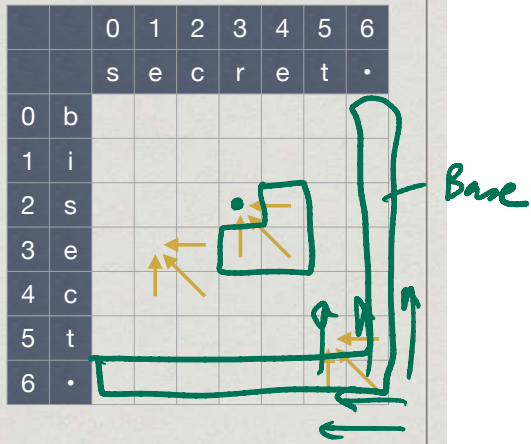
$a_i = b_j$ by subst.

Drop a_i & b_j

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



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						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

Recovering the solution

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

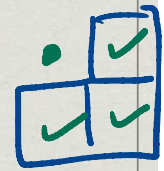
		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	1	5	5	5	5	5	6
1	i	1	4	4	4	4	4	5
2	s	2	3	3	3	3	3	4
3	e	3	2	3	2	2	2	3
4	c	4	3	2	2	1	1	2
5	t	5	4	3	2	1	0	1
6	•	6	5	4	3	2	1	0

ED(u,v), DP

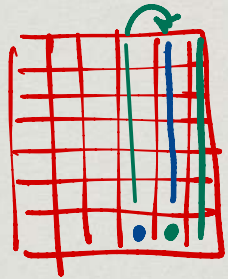
```
function ED(u,v) # u[0..m], v[0..n]

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

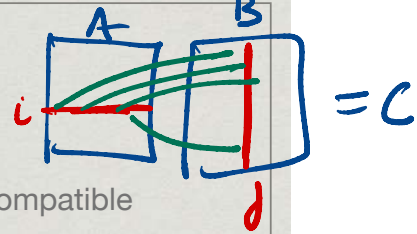


- For LCW, LCS, ED
  - Need to fill an O(mn) size table
  - Do we need to store the entire table?
- Filling column by column, only need next column and current column
  - Or next row and current row
- Reduce space to O(n), assuming m ≥ n

2 x n

2 x m

Multiplying matrices



- To multiply matrices A and B, need compatible dimensions
  - A of dimension m x n, B of dimension n x p
  - AB has dimension mp
- Each entry in AB take O(n) steps to compute
  - AB[i,j] is A[i,1]B[1,j] + A[i,2]B[2,j] +…+ A[i,n]B[n,j]
- Overall, computing AB is O(mnp)

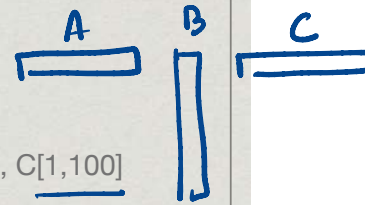
$$C[i,j] = \sum_{k=1}^{n} A[i,k]B[k,j]$$

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 x 1 x 100 = 10000 steps
    - A(BC) is [1,100],1 x 100 x 100 = 10000 steps
  - Computing (AB)C
    - AB is [1,1], 1 x 100 x 1 = 100 steps
    - (AB)C is [1,100], 1 x 1 x 100 = 100 steps
- A(BC) takes 20000 steps, (AB)C takes 200 steps!

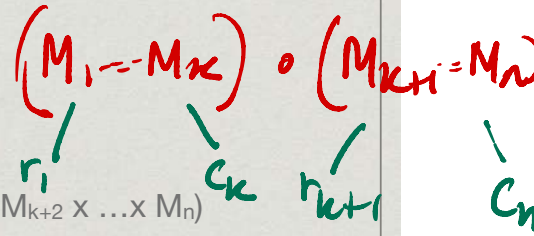
Multiplying matrices

- Given matrices M_1, M_2,…, M_n of dimensions $[r_1,c_1]$, $[r_2,c_2]$, …, $[r_n,c_n]$
  - Dimensions match, so $M_1 \times M_2 \times \ldots \times M_n$ can be computed
  - $c_i = r_{i+1}$ for $1 \le i < n$
  - Find an optimal order to compute the product
  - That is, bracket the expression optimally

Inductive structure

- Product to be computed: $M_1 \times M_2 \times \ldots \times M_n$
- Final step would have combined two subproducts
  - $(M_1 \times M_2 \times \ldots \times M_k) \times (M_{k+1} \times M_{k+2} \times \ldots \times M_n)$, 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
  $(M_1 \times M_2 \times \ldots \times M_k) \times (M_{k+1} \times M_{k+2} \times \ldots \times M_n)$
- Subproblems are $(M_1 \times M_2 \times \ldots \times M_k)$ and $(M_{k+1} \times M_{k+2} \times \ldots \times M_n)$
- Total cost is $\text{Cost}(M_1 \times M_2 \times \ldots \times M_k) + \text{Cost}(M_{k+1} \times M_{k+2} \times \ldots \times M_n) + r_1 c_k c_n$
- Which k should we choose?
- No idea! Try them all and choose the minimum!

Inductive formulation

- Cost($M_1 \times M_2 \times \ldots \times M_n$) =

  minimum value, for $1 \le k < n$, of

  - Cost($M_1 \times M_2 \times \ldots \times M_k$) +
  - Cost($M_{k+1} \times M_{k+2} \times \ldots \times M_n$) +
  - $r_1 c_k c_n$

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

In general …

- Cost(M_i x M_{i+1} x …x M_j) =

  minimum value, for $i \leq k < j$, of

  Cost(M_i x M_{i+1} x …x M_k) +

  Cost(M_{k+1} x M_{k+2} x …x M_j) +

  $r_i c_k c_j$

- Write Cost(i,j) to denote Cost(M_i x M_{i+1} x …x M_j)

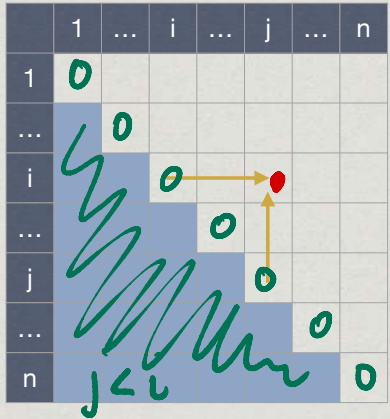
Final equation

- Cost(i,i) = 0 — No multiplication to be done
- Cost(i,j) = min over $i \leq k < j$

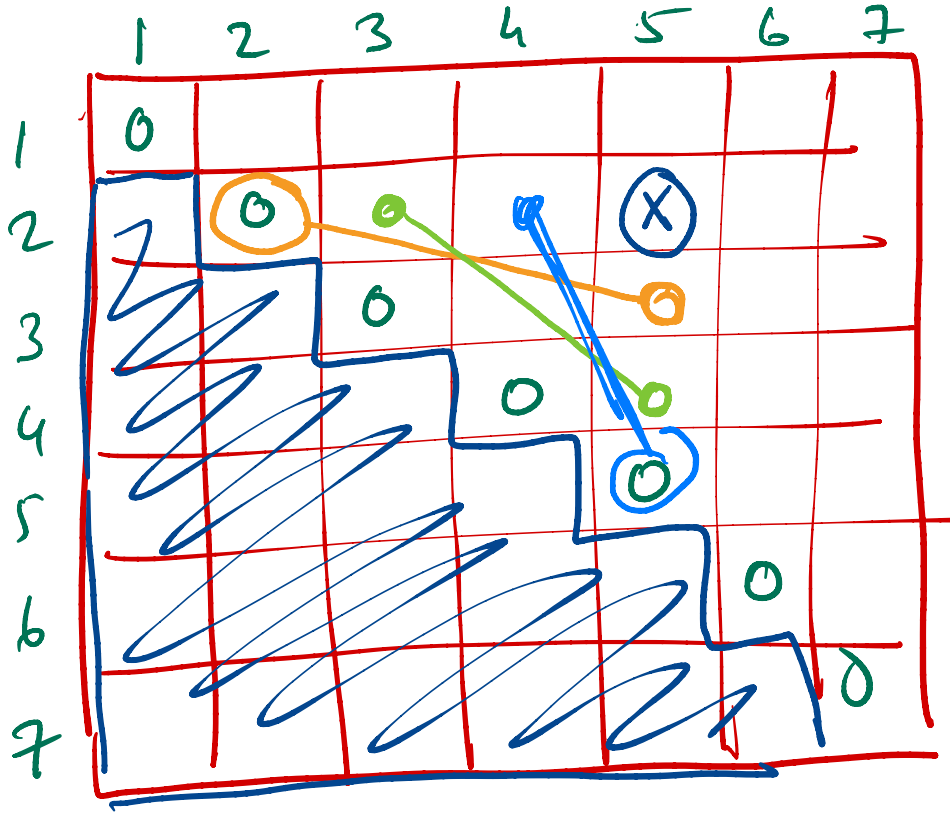
  $[\ \text{Cost}(i,k) + \text{Cost}(k+1,j) + r_i c_k c_j\ ]$
- Note that we only require Cost(i,j) when $i \leq j$

Subproblem dependency

- Cost(i,j) depends on Cost(i,k), Cost(k+1,j) for all $i \le k < j$
- 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



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



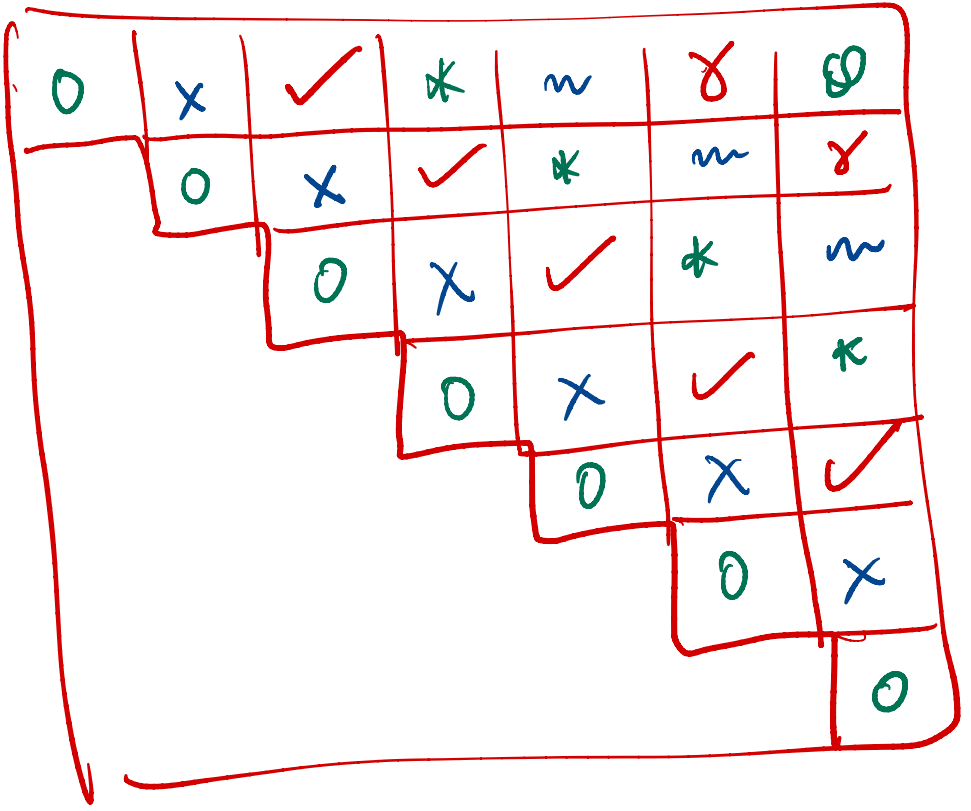
$Cost(i,i) = 0$

$Cost(2,5)$! Min

$k=2$ | $Cost(2,2) + Cost(3,5) + r_2 c_2 c_5$

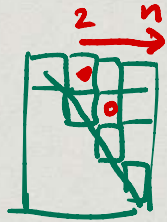
$k=3$ | $Cost(2,3) + Cost(4,5) + r_2 c_3 c_5$

$k=4$ | $Cost(2,4) + r_2 c_4 c_5 + Cost(5,5)$


0 x ✓ * ~ γ ∞
0 x ✓ * ~ γ
0 x ✓ * ~
0 x ✓ *
0 x ✓
0 x
0

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
  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
```



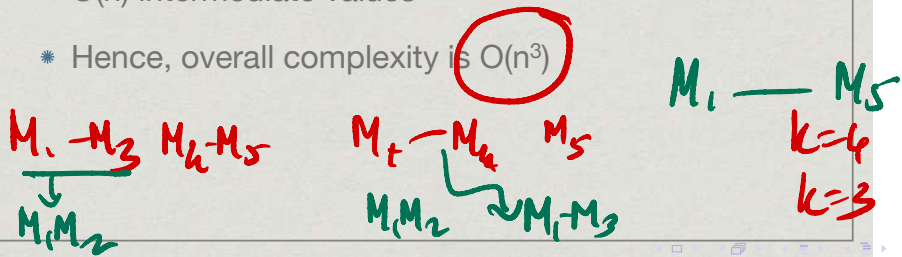
2-1=1
3-2=2

O(n)

Complexity

$M_1\ M_2\ \cdots\ M_n$

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



Dynamic Programming – Intelligent brute force

Never overlook a subproblem

Never recompute a subproblem

Cost: # subproblems × 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