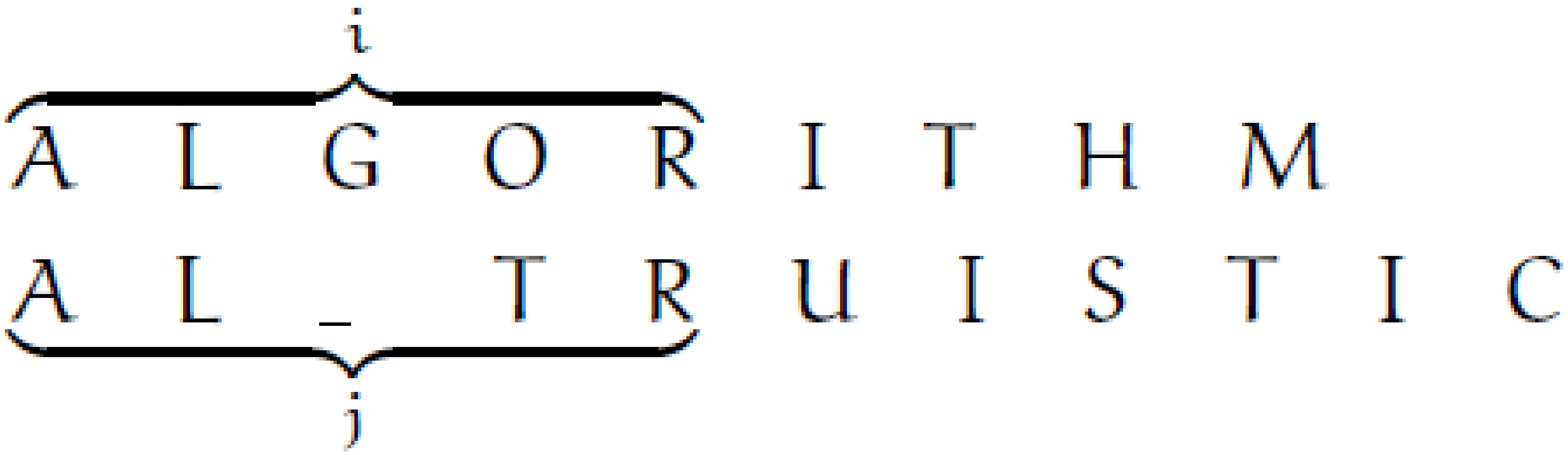
Dynamic Programming
Edit Distance Algorithm

(Class 18)

Edit Distance: Dynamic Programming Algorithm

- Suppose we have an m-character string A and an n-character string B.
- Define $E(i, j)$ to be the edit distance between the first i characters of A and the first j characters of B. For example:



- The edit distance between entire strings A and B is $E(m, n)$.
- The gap representation for the edit sequences has a crucial "*optimal substructure*" property.
- If we remove the last column, the remaining columns must represent the shortest edit sequence for the remaining substrings.

- The edit distance is 6 for the following two words.

A	L	G	O	R	_	I	_	T	H	M
A	L	_	T	R	U	I	S	T	I	C

- If we remove the last column, the edit distance reduces to 5.

A	L	G	O	R	_	I	_	T	H
A	L	_	T	R	U	I	S	T	I

- We can use the optimal substructure property to devise a recursive formulation of the edit distance problem.

- There are a couple of obvious base cases:
- The only way to convert an empty string ($i = 0$) into a string of j characters is by doing j insertions. Thus:

$$E(0, j) = j$$

- The only way to convert a string of i characters into the empty $j = 0$ string is with i deletions:

$$E(i, 0) = i$$

- There are four possibilities for the last column in the shortest possible edit sequence:
- **Deletion:** Last entry in bottom row is empty.

$$\begin{array}{lllllllll}
\overbrace{A\quad L\quad G}^{i=3} & O & R & I & T & H & M & & \\
\underbrace{A\quad L\quad _}_{j=2} & T & R & U & I & S & T & I & C
\end{array}$$

- In this case

$$E(i,j) = E(i-1,j) + 1$$

- **Insertion:** The last entry in the top row is empty.

$$\overbrace{A\quad L\quad G\quad O\quad R\quad \_}^{i=5}\quad I\quad T\quad H\quad M$$

$$\underbrace{A\quad L\quad \_\quad T\quad R\quad U}_{j=5}\quad I\quad S\quad T\quad I\quad C$$

- In this case

$$E(i,j) = E(i,j-1) + 1$$

- **Substitution:** Both rows have characters in the last column.

$$\overbrace{A\quad L\quad G\quad O}^{i=4}\quad R\quad I\quad T\quad H\quad M$$

$$\underbrace{A\quad L\quad \_\quad T}_{j=3}\quad R\quad U\quad I\quad S\quad T\quad I\quad C$$

- If the characters are different, then:

$$E(i,j) = E(i-1, j-1) + 1$$

- **Maintenance:** Both rows have same characters in the last column.

$$
\begin{array}{l}
\overbrace{\mathrm{A\quad L\quad G\quad O\quad R}}^{i=5}\quad \mathrm{I\quad T\quad H\quad M} \\
\underbrace{\mathrm{A\quad L\quad \_\quad T\quad R}}_{j=4}\quad \mathrm{U\quad I\quad S\quad T\quad I\quad C}
\end{array}
$$

- If characters are same, no substitution is needed:

$$E(i,j) = E(i-1, j-1)$$

- Thus, the edit distance $E(i,j)$ is the smallest of the four possibilities:

$$E(i,j) = \min \begin{pmatrix} E(i-1,j)+1 & \\ E(i,j-1)+1 & \\ E(i-1,j-1)+1 & \text{if } A[i] \neq B[j] \\ E(i-1,j-1) & \text{if } A[i] = B[j] \end{pmatrix}$$

- Consider the example of edit between the words “ARTS” and “MATHS”:

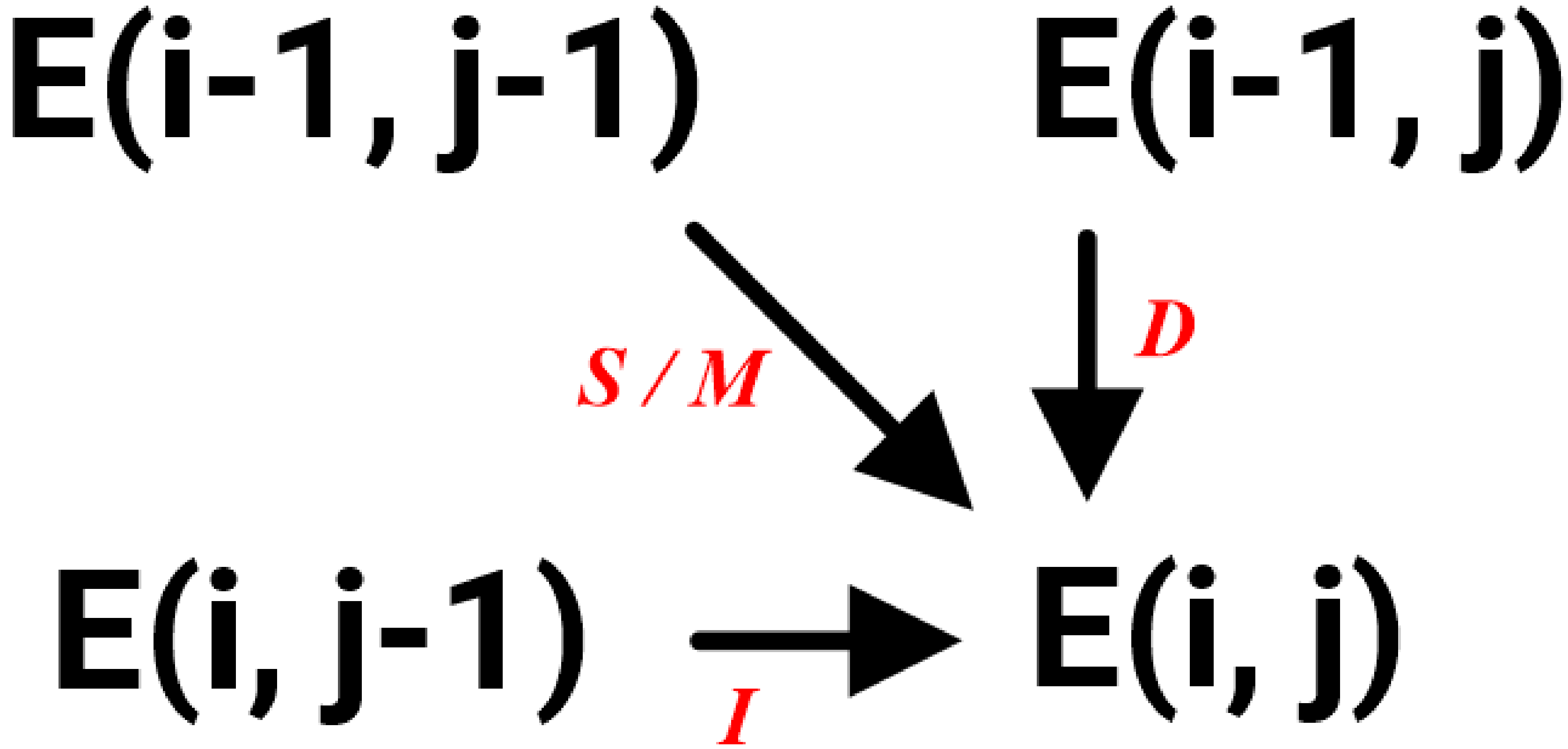
A	R	T	S	
M	A	T	H	S

- The edit distance would be in $E(4,5)$.
- If we recursion to compute, we will have:

$$\mathrm{E}(4,5) = \min \begin{pmatrix} \mathrm{E}(3,5)+1 & \\ \mathrm{E}(4,4)+1 & \\ \mathrm{E}(3,4)+1 & \text{if } \mathrm{A}[4] \neq \mathrm{B}[5] \\ \mathrm{E}(3,4) & \text{if } \mathrm{A}[4] = \mathrm{B}[5] \end{pmatrix}$$

- Recursion clearly leads to the same repetitive call pattern that we saw in Fibonacci sequence.
- To avoid this, we will use the dynamic programming approach.
- We will build the solution bottom-up.

Pattern of Building E(i, j) using Table



- We will build a matrix of 6x4.
- Use the base case $E(0, j)$ to fill first row.
- Use the base case $E(i, 0)$ to fill first column.
- We will fill the remaining E matrix row by row.

		A	R	T	S
	0	→1	→2	→3	→4
M					
A					
T					
H					
S					

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1				
A	↓ 2				
T	↓ 3				
H	↓ 4				
S	↓ 5				

First row and first column entries using the base cases

- We can now fill the second row.
- The table not only shows the values of the cells E[i, j] but also arrows that indicate how it was computed using values in E[i − 1, j], E[i, j − 1] and E[i − 1, j − 1].
- Thus, if a cell E[i,j] has a down arrow from E[i − 1, j] then the minimum was found using E[i − 1, j].
- For a right arrow, the minimum was found using E[i, j − 1].

- For a diagonal down right arrow, the minimum was found using $E[i - 1, j - 1]$.
- There are certain cells that have two arrows pointed to it.
- In such a case, the minimum could be obtained from the diagonal $E[i - 1, j - 1]$ and either of $E[i - 1, j]$ and $E[i, j - 1]$.
- We will use these arrows later to determine the edit script.

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1	↘ 1			
A	↓ 2				
T	↓ 3				
H	↓ 4				
S	↓ 5				

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1	↘ 1	↘ →2		
A	↓ 2				
T	↓ 3				
H	↓ 4				
S	↓ 5				

Computing $E[1,1]$ and $E[1,2]$

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1	↘ 1	↘ →2	↘ →3	
A	↓ 2				
T	↓ 3				
H	↓ 4				
S	↓ 5				

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1	↘ 1	↘ →2	↘ →3	↘ →4
A	↓ 2				
T	↓ 3				
H	↓ 4				
S	↓ 5				

Computing $E[1,3]$ and $E[1,4]$

- An edit script can be extracted by following a unique path from E[0, 0] to E[4, 5].
- There are three possible paths in the current example.
- Let us follow these paths and compute the edit script.
- In an actual implementation of the dynamic programming version of the edit distance algorithm, the arrows would be recorded using an appropriate data structure.
- For example, each cell in the matrix could be a record with fields for the value (numeric) and flags for the three incoming arrows.

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1	↘ 1	↘ →2	↘ →3	↘ →4
A	↓ 2	↘ 1	↘ →2	↘ →3	↘ →4
T	↓ 3	↓ 2	↘ 2	↘ 2	→3
H	↓ 4	↓ 3	↘↓ 3	↘↓ 3	↘ 3
S	↓ 5	↓ 4	↘↓ 4	↘↓ 4	↘ 3

The final table with all E[i, j] entries computed

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1	↘ 1	↘ →2	↘ →3	↘ →4
A	↓ 2	↘ 1	↘ →2	↘ →3	↘ →4
T	↓ 3	↓ 2	↘ 2	↘ 2	→3
H	↓ 4	↓ 3	↘↓ 3	↘↓ 3	↘ 3
S	↓ 5	↓ 4	↘↓ 4	↘↓ 4	↘ 3

Solution path 1:

1+	0+	1+	1+	0	= 3
D	**M**	**S**	**S**	**M**	
M	A	T	H	S	
_	A	R	T	S	

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1	↘ 1	↘ →2	↘ →3	↘ →4
A	↓ 2	↘ 1	↘ →2	↘ →3	↘ →4
T	↓ 3	↓ 2	↘ 2	↘ 2	→3
H	↓ 4	↓ 3	↘↓ 3	↘↓ 3	↘ 3
S	↓ 5	↓ 4	↘↓ 4	↘↓ 4	↘ 3

Possible edit scripts. The red arrows from $E[0,0]$ to $E[4,5]$ show the paths that can be followed to extract edit scripts.

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1	↘ 1	↘ →2	↘ →3	↘ →4
A	↓ 2	↘ 1	↘ →2	↘ →3	↘ →4
T	↓ 3	↓ 2	↘ 2	↘ 2	→3
H	↓ 4	↓ 3	↘↓ 3	↘↓ 3	↘ 3
S	↓ 5	↓ 4	↘↓ 4	↘↓ 4	↘ 3

Solution path 2:

1+	1+	0+	1+	0	= 3
S	**S**	**M**	**D**	**M**	
M	A	T	H	S	
A	R	T	_	S	

		A	R	T	S
	0	→1	→2	→3	→4
M	↓ 1	↘ 1	↘ →2	↘ →3	↘ →4
A	↓ 2	↘ 1	↘ →2	↘ →3	↘ →4
T	↓ 3	↓ 2	↘ 2	↘ 2	→3
H	↓ 4	↓ 3	↘↓ 3	↘↓ 3	↘ 3
S	↓ 5	↓ 4	↘↓ 4	↘↓ 4	↘ 3

Solution path 3:

1+	0+	1+	0+	1+	0	= 3
D	**M**	**I**	**M**	**D**	**M**	
M	A	_	T	H	S	
_	A	R	T	_	S	

Analysis of DP Edit Distance

- There are $O(n^2)$ entries in the matrix.
- Each entry $E(i,j)$ takes $O(1)$ time to compute.
- The total running time is $O(n^2)$.

- This approach is suitable for small number of inputs.
- But what if we want to solve the DNA sample with millions of characters as input?