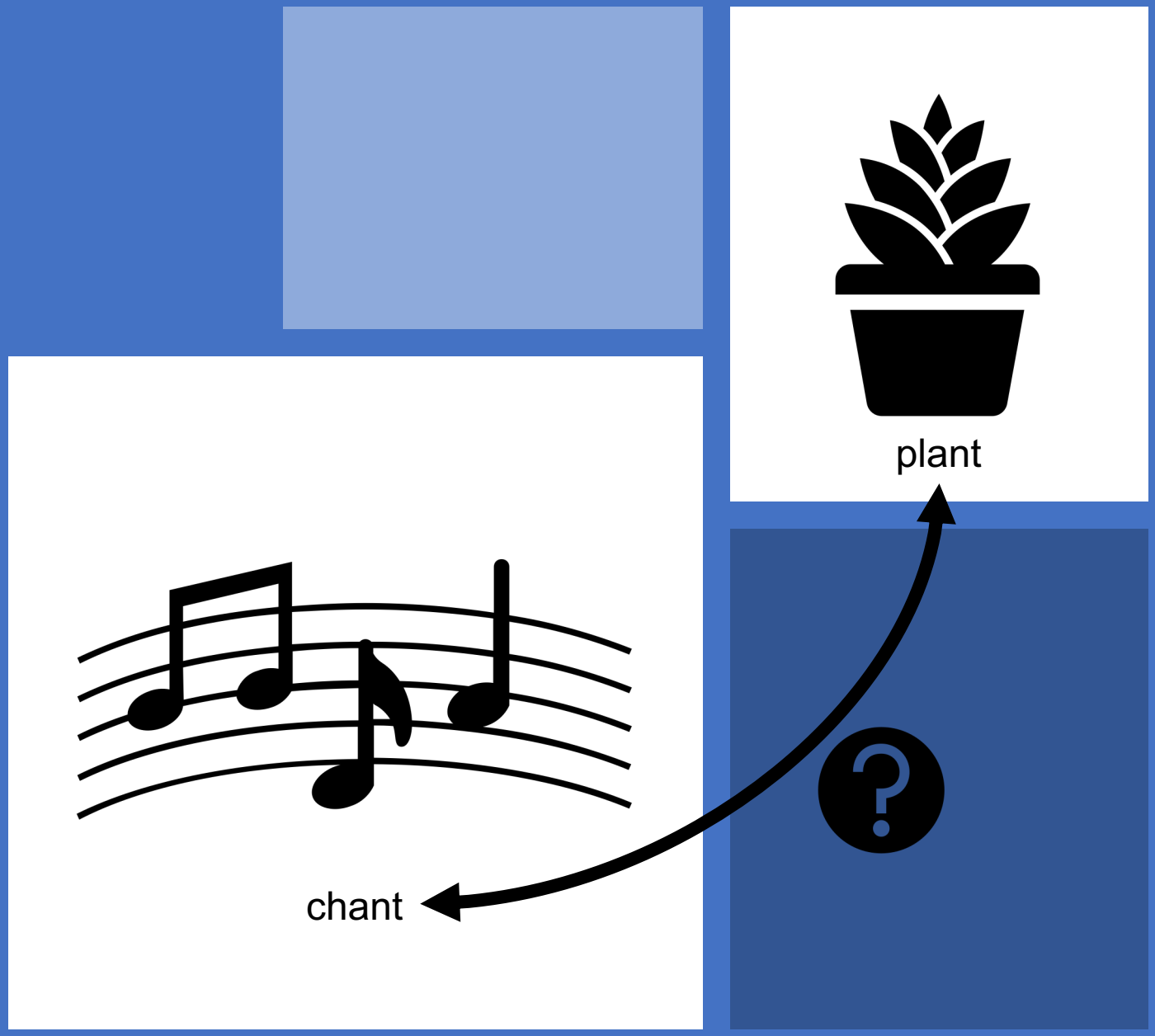
Edit Distance

Natalie Parde
UIC CS 421

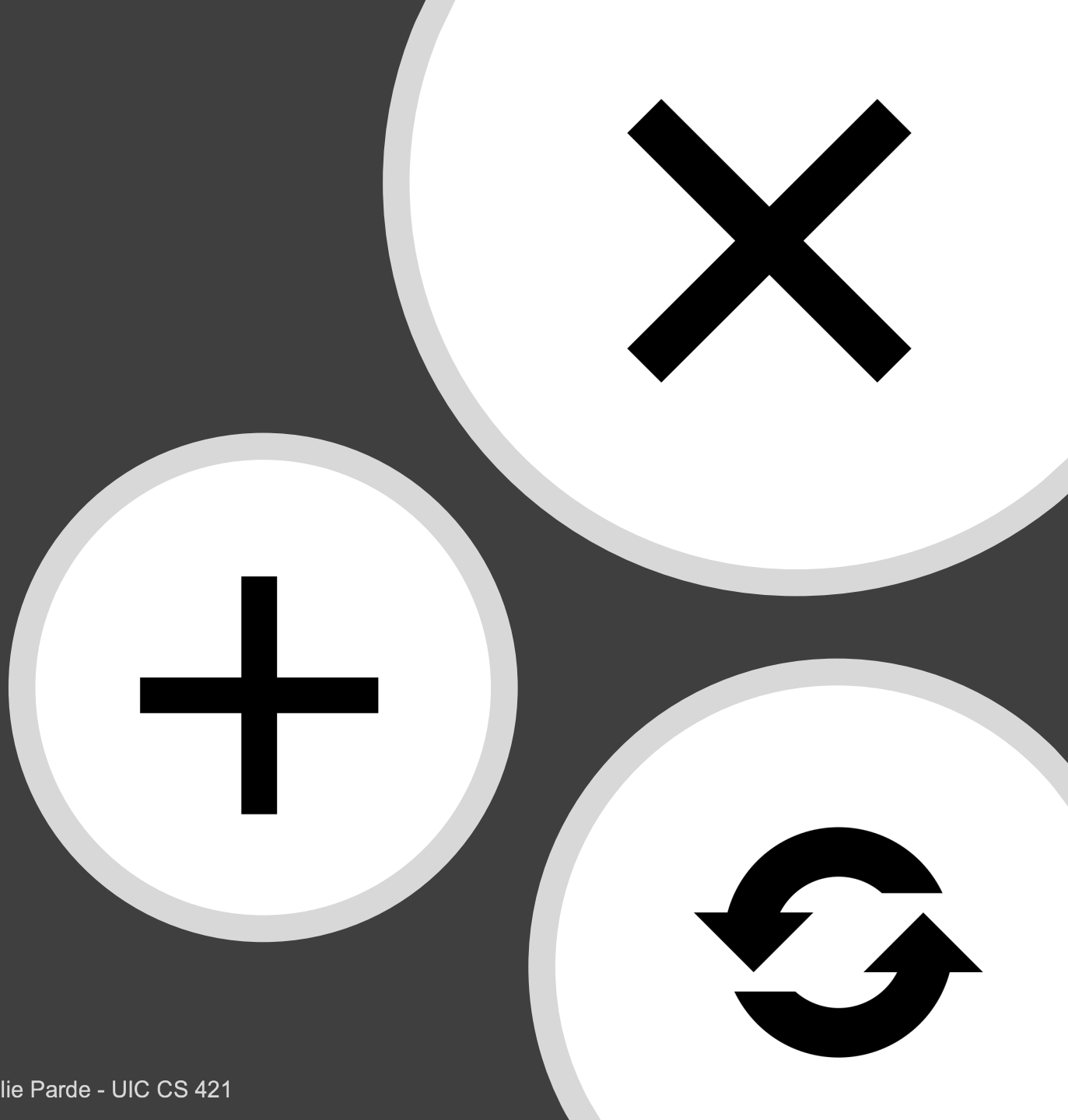


Edit Distance

Simple way to answer the question: How similar are two strings?

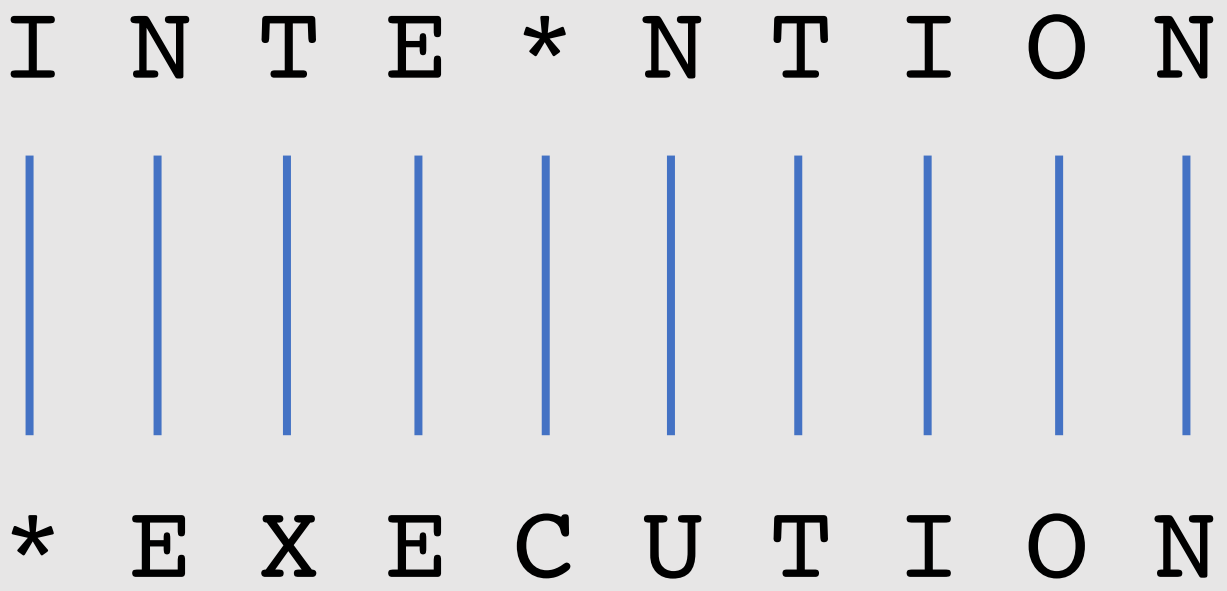
Minimum Edit Distance

- Minimum number of editing operations needed to transform one string into another
- Possible editing operations:
  - Insertion
  - Deletion
  - Substitution



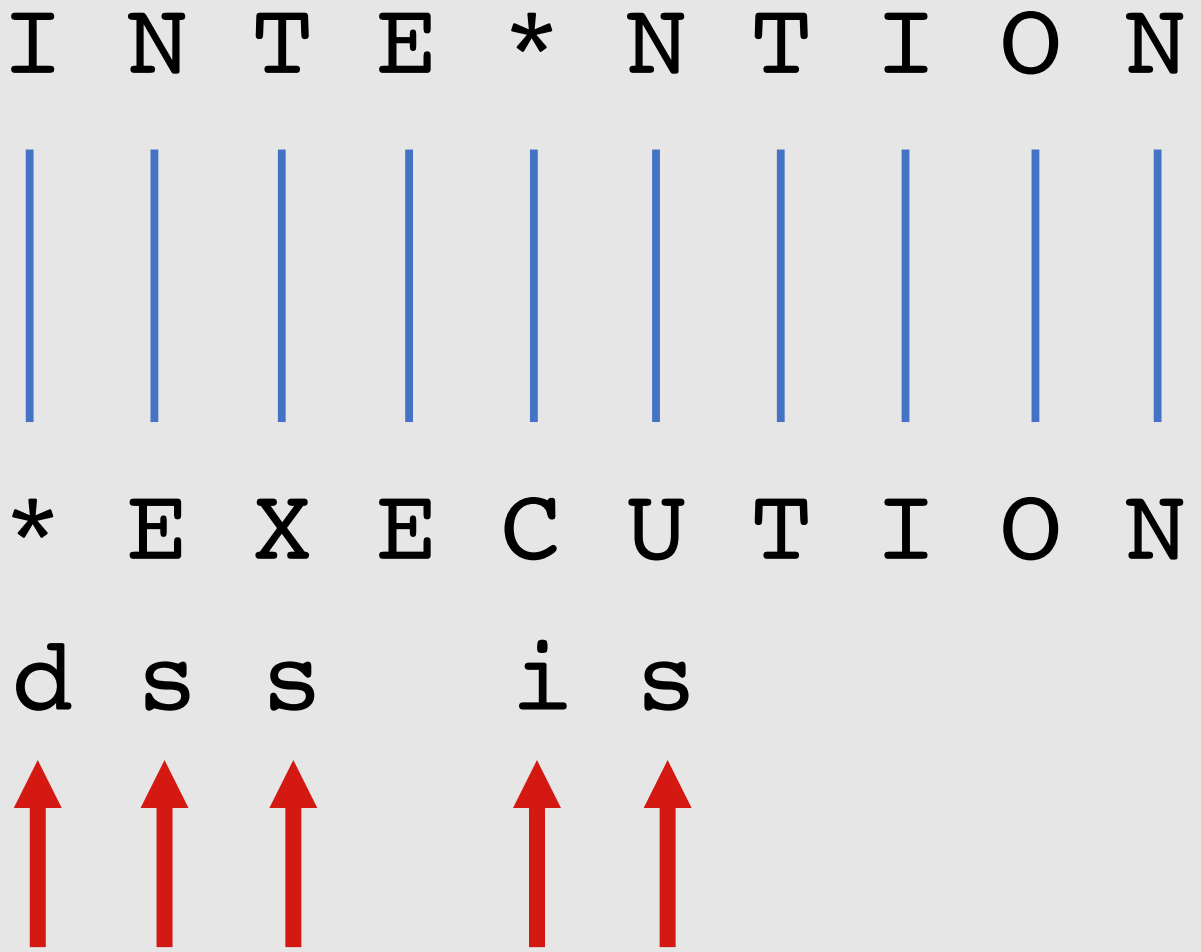
Minimum Edit Distance

- Two strings and their **alignment**:



Minimum Edit Distance

- If each operation has a cost of 1 (Levenshtein distance)
  - Distance between these is 5
- If substitutions cost 2 (alternative also proposed by Levenshtein)
  - Distance between them is 8



Other Uses of Edit Distance in NLP

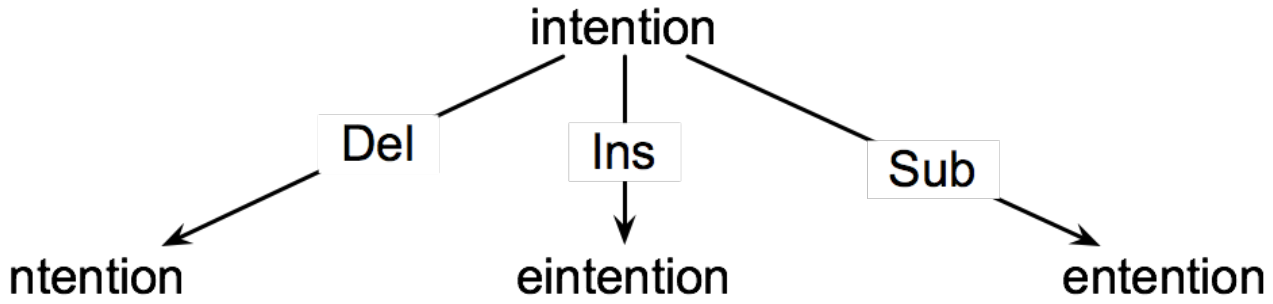
- Evaluating Machine Translation and speech recognition

```
Spokesman confirms    senior government adviser was shot
Spokesman said    the senior           adviser was shot dead
            S       I              D                    I
```

- Named Entity Extraction and Entity Coreference
  - **IBM Inc.** announced today
  - **IBM** profits

How to find the minimum edit distance?

- Search for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize (the number of edits)



However, the search space of all edit sequences is huge!

- We can’t afford to navigate naïvely
- Lots of distinct paths wind up at the same state
  - We don’t have to keep track of all of them (just the shortest paths)

Formal Definition: Minimum Edit Distance

- For two strings
  - X of length *n*
  - Y of length *m*
- We define D(*i*,*j*) as the edit distance between X[1..*i*] and Y[1..*j*]
  - X[1..*i*] = the first *i* characters of X
- The edit distance between X and Y is thus D(*n*,*m*)

Intuition: Dynamic Programming

- Minimum edit distance can be solved using **dynamic programming**
  - Stores intermediate outputs in a table
  - Intuition: If some string B is in the optimal path from string A to string C, then that path must also include the optimal path from A to B
- D(n,m) is computed tabularly, combining solutions to subproblems
- Bottom-up
  - We compute D(i,j) for small *i,j*
  - And compute larger D(i,j) based on previously computed smaller values
    - i.e., compute D(*i,j*) for all *i* ($0 < i < n$) and *j* ($0 < j < m$)

Formal Definition: Minimum Edit Distance

- Initialization

  $D(i,0) = i$

  $D(0,j) = j$

- Recurrence Relation:

  For each $i = 1 \ldots n$

  For each $j = 1 \ldots m$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

  $D(N,M)$ is distance

The Edit Distance Table

N										
O										
I										
T										
N										
E										
T										
N										
I										
#										
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2								
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2	3	4	5	6	7			
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2	3	4	5	6	7	6		
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

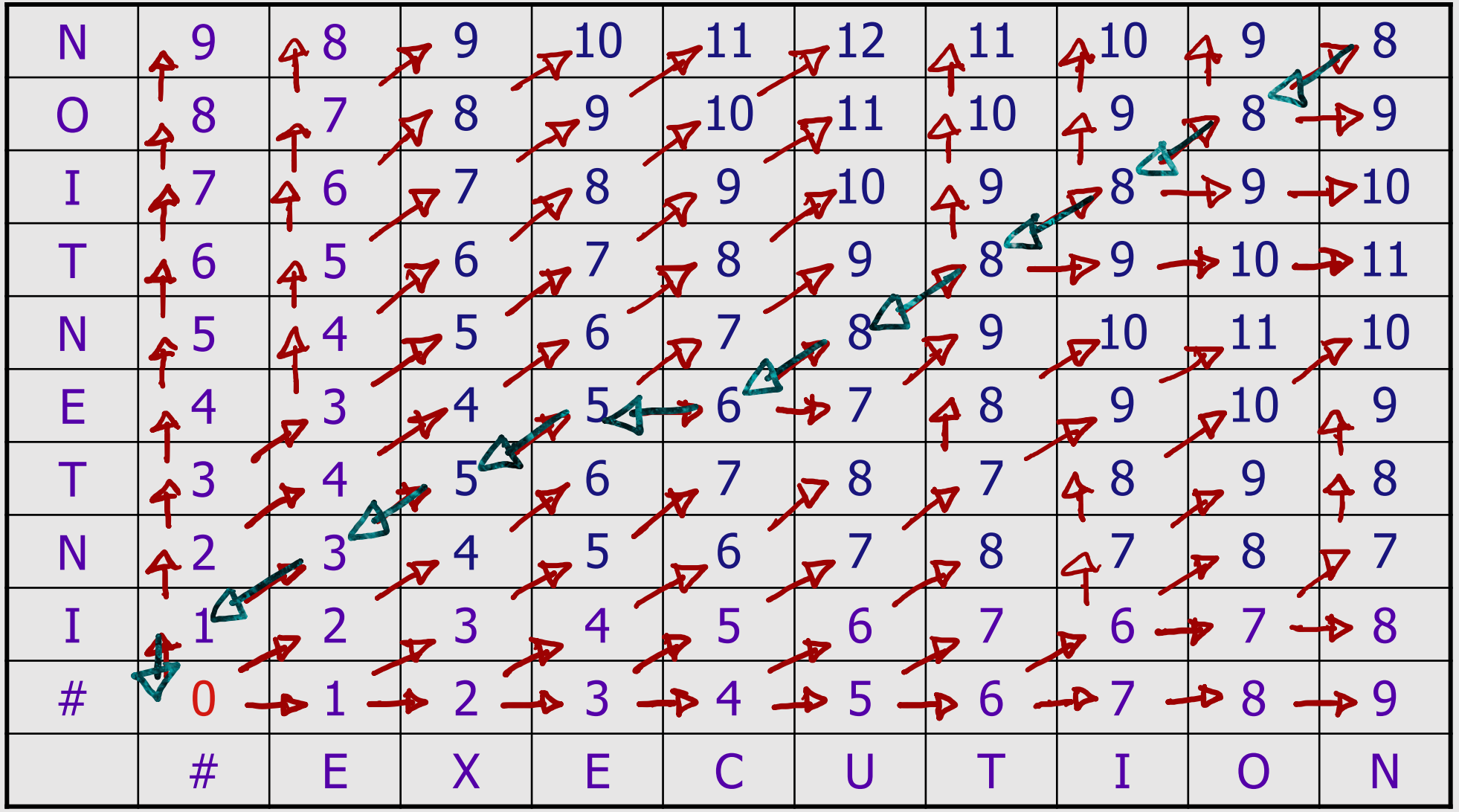
Backtrace for Computing Alignments

- We know the minimum edit distance now …but what is the alignment between the two strings?
- We can figure this out by maintaining a **backtrace**
  - For each new cell, remember where we came from!
    - D(i-1,j) ?
    - D(i,j-1) ?
    - D(i-1,j-1) ?
- Once we reach the end of the table (upper right corner), we can trace backward using these pointers to figure out the alignment

The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table



Formal Definition: Minimum Edit Distance with Backtrace

- Base conditions:

  $D(i,0) = i$ $\quad$ $D(0,j) = j$

  Termination: $D(N,M)$ is distance

- Recurrence Relation:

For each $i = 1 \ldots n$

For each $j = 1 \ldots m$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

$$ptr(i,j) = \begin{cases} \text{LEFT} \\ \text{DOWN} \\ \text{DIAG} \end{cases}$$